M.Sc. (IT) Previous Year
MIT-06

ADVANCED DBMS

मध्यप्रदेश भोज (मुक्त) विश्वविद्यालय – भोपाल
MADHYA PRADESH BHOJ (OPEN) UNIVERSITY - BHOPAL

COURSE WRITERS

Yogita Baveja Khatri, Assistant Professor, J.S.S., Noida
Meena Arora, Assistant Professor, J.S.S., Noida
Unit (1.0-1.2.3, 1.5-1.11)
Manas Ghosh, Lecturer, R.C.C. Institute of Information Technology, Kolkata
Sudipta Pathak, System Consultant, Cognizant
Units (1.2.4, 1.2.5, 1.3, 2.0-2.2.1, 2.3, 2.4, 2.5, 2.7-2.11, 3.3, 4.0-4.2.1, 4.5, 4.7, 4.8-4.14)
Gagan Varshney, Professor and H.O.D., I.M.S. Engineering College, Ghaziabad
Rajneesh Agrawal, Senior Scientist, Department of I.T., Government of India
Unit (4.4)
Dr. Preety Khatri, Assistant Professor, Computer Science, S.O.I.T., I.M.S., Noida
Units (1.4, 2.2.2-2.2.3, 2.3.1, 2.6, 3.0-3.2, 3.3.1, 3.4, 3.5, 3.6-3.13, 4.2.2, 4.3, 4.6, 4.7.1, 5)

Published by Registrar, MP Bhoj (Open) University, Bhopal in 2020

VIKAS®

Vikas® is the registered trademark of Vikas® Publishing House Pvt. Ltd.

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

SYLLABI-BOOK MAPPING TABLE

Advanced DBMS

Syllabi	Mapping in Book
Unit - I Query Processing: Query Execution Algorithms, Set Operations, Pipelining, Query Optimization: Using Heuristics in Query Optimization, Basic Algorithms for Executing Query Operations.	**Unit-1:** Query Processing and Optimization **(Pages 3-34)**
Unit - II Database Tuning: Location of Bottlenecks, Tunable Parameters, Extended Relational Model: Relational Model Concepts, Referential Integrity Constraints, Structured Query Language (SQL), Querying Data with Multiple Conditions, Basic Relation algebra Operations.	**Unit-2:** Database Tuning and Extended Relational Model **(Pages 35-154)**
Unit - III Object Oriented Database Systems: Characteristics of an object-relation Database Management System (ORDBMS), Complex Objects, Inheritance, Function Overloading, Rules, Distributed Database: Distributed Database System, Distributed Database Design, Data Fragmentation, Data Replication, Data Allocation, Distributed Database Query Processing.	**Unit-3:** Object Oriented Database Systems and Distributed Database **(Pages 155-189)**
Unit - IV Database Security: Security and Integrity Threats, International or Malicious Threats, Defense Mechanisms, Security Policies, Authorization, Objects, View as objects, Granularity, Subject, Access Types, Database Operating Systems: Features of a Database OS, Concurrency Control Model, Theory of Serializability, Concurrency Control Algorithms,	**Unit-4:** Database Security and Authorization **(Pages 191-270)**
Unit - V Multimedia Databases, Impact of IT on Libraries, Multimedia Data Formats, Continuous-Media Data, Similarity- Based Retrieval, Mobility and Personal Database, Database Technologies, Serving Database on the Web, Applying Databases to the Internet, Deductive Databases, Logic Concepts, Deductive Database Notations, Interpretation of Rules, Inference Mechanism, Deductive Database Systems, Deductive Object Oriented Databases	**Unit-5:** Multimedia and Deductive Database **(Pages 271-312)**

CONTENTS

INTRODUCTION

NOTES

A Database Management System or DBMS is a software package including specific computer programs that control the creation, maintenance and use of a database. It provides an environment that is both convenient and efficient to use. A DBMS allows different user application programs to concurrently access the same database. It allows organizations to conveniently develop databases for various applications through database administrators (DBAs) and other specialists. The American National Standards Institute/Standards Planning and Requirements Committee Architecture or ANSI/SPARC is referred as an abstract design standard for a DBMS. Technically, a database is referred as an integrated collection of data records, files and other objects. DBMSs may use a variety of database models to conveniently describe and support applications. Database languages also simplify the database organization as well as retrieving and presenting information from it.

A database modeling language is a data modeling language which is used to define the schema of each database hosted in the DBMS, according to the DBMS database model. Thus, a database model is a theory or specification describing how a database is structured and used. Common models include hierarchical model, network model and relational model. In hierarchical model, data is organized into a tree like structure, implying a single upward link in each record to describe the nesting and a sort field to keep the records in a particular order in each same level list. Network model organizes data using two fundamental constructs, called records and sets. Records contain fields and sets define one-to-many relationships between records. Relational model is a database model which describes constraints on the possible values and combinations of values. SQL is a widely used database language providing means of data manipulation (store, retrieve, update, delete) and database creation. The SELECT statement in SQL is used to select queried data from a database in SQL. The result is stored in a result table, called the result-set. Modern RDBMS, such as MS SQL Server, Microsoft Access, MSDE, Oracle, DB2, Sybase, MySQL, Postgres and Informix, use SQL as standard database language. The database transaction must be Atomic, Consistent, Isolated and Durable, i.e., ACID. Atomicity states that if one part of the transaction fails, the entire transaction fails and the database state is left unchanged. Consistency states that only consistent data will be written to the database. Isolation states that no transaction should be able to interfere with another transaction at all. Durability means that every committed transaction is protected against power loss, crash or errors. Hence, a DBMS provides facilities for controlling data access, enforcing data integrity, managing concurrency control and recovering the database after failures and restoring it from backup files, as well as maintaining database security.

This book, *Advanced DBMS* is divided into five units that follow the self-instruction mode with each unit beginning with an Introduction to the unit, followed by an outline of the Objectives. The detailed content is then presented in a simple but structured manner interspersed with Check Your Progress Questions to test the student's understanding of the topic. A Summary along with a list of Key Terms and a set of Self-Assessment Questions and Exercises is also provided at the end of each unit for recapitulation.

UNIT 1 QUERY PROCESSING AND OPTIMIZATION

Structure

1.0 INTRODUCTION

Query processing is the procedure of selecting the best strategic plan to be used in response to a database request. A response is then generated by executing the query. The component of the DBMS (Database Management System) responsible for generating this strategy is called a query processor. The query optimizer in the database which is a software component in the RDBMS that carries out analysis of SQL statement for finding the best way for its execution. Modern optimizers are cost-based, and they estimate the cost of every possible way for executing a statement and choosing that with the lowest cost.

Heuristics query optimization is based method of producing an efficient query execution plan in the form of a query tree or a query graph data structure. A query tree is a tree data structure that corresponds to a relational algebra expression. It represents relational algebra operations as internal nodes and input relations of the query as leaf nodes of the tree. An execution of the query tree consists of executing an internal node operation (addition, deletion or updation etc.) whenever its operands are available and then replacing that internal node by the relation that results from executing the operation. The execution terminales when the root node is executed and a result.

In this unit, you will study about the query processing, query execution algorithms, significance of set operations, pipelining, heuristics in query optimization and basic algorithms for executing query operations.

NOTES

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basics of query processing
- Discuss the query execution algorithms
- State the significance of set operations
- Describe about the pipelining
- Analyse the heuristics in query optimization
- Explain the basic algorithms for executing query operations

1.2 QUERY PROCESSING

We can identify four broad stages in query processing as follows:

1. Cast the query into internal form.
2. Convert it to canonical form.
3. Choose candidate low level procedures.
4. Generate query plans and choose the cheapest of them.

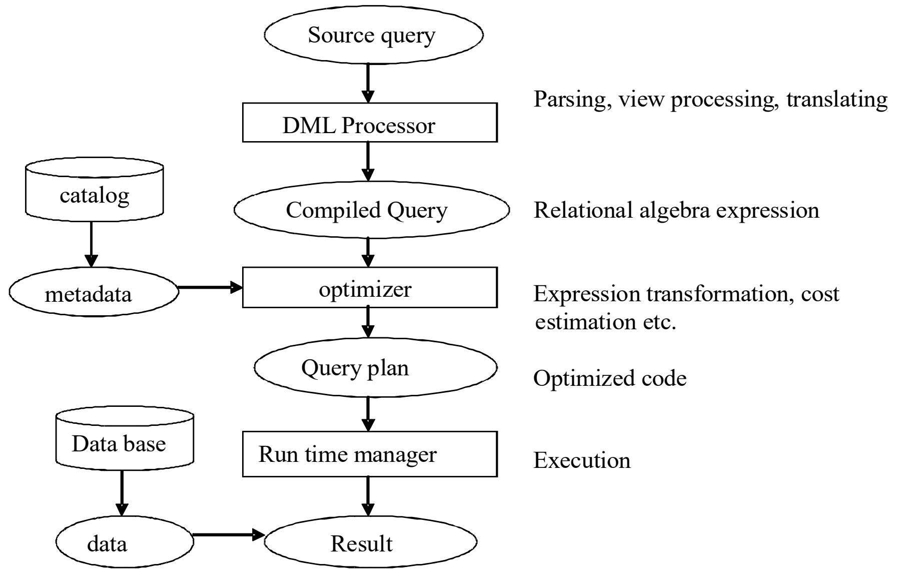


Fig. 1.1 *Query Processing Overview*

Query processing is the procedure of selecting the best strategic plan to be used in response to a database request. A response is then generated by executing the query. The component of the DBMS (DataBase Management System) responsible for generating this strategy is called a query processor.

Query processing is also referred to in database literature as query optimization. However, optimization here is mostly in the form of improvement of status of database in light of the inexact knowledge. Figure 1.1 shows the query processing overview.

Query processing is a process involving several steps.

These steps are:

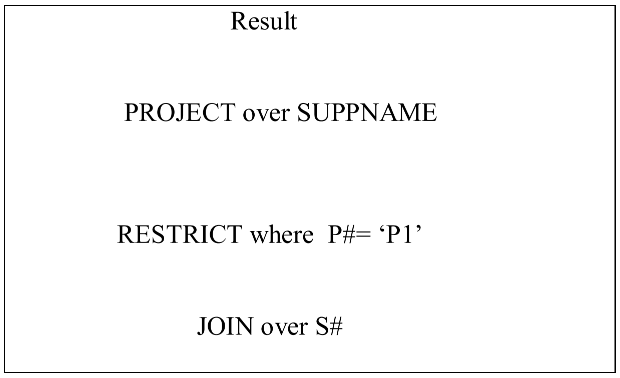
1.2.1 Parsing and Translation

The first step is that the query is transformed in a standard form.

For instance, a query expressed in QBE (Query By Example) is translated into SQL (Structured Query Language) and subsequently into a relational algebraic expression. The parser portion of query processor checks the syntax and verifies the relations. The attributes used in query are defined in the database during this transformation process.

The following table shows a query tree representation:

Query 'Get names of Suppliers who supply part P1'



((SP JOIN S) WHERE P# = P# ('P2')) { SNAME }

1.2.2 Optimization

The next step is query optimization. Having translated the query into a particular form, such as, a relational algebraic expression, an equivalent expression for those in the query are substituted by performing optimization.

The transformation is also based on factors like, whether the files are sorted or not and also on the presence of creation database structures like, indexes and so on.

Example: In order to transform the output form into one of its equivalent forms effectively, certain transformation rules are used. For instance,

(B JOIN C) WHERE restriction on A

can be transformed into following form using optimization

(B WHERE restriction on B) JOIN C

1.2.3 Evaluation and Query Execution Algorithms

In the next step, transformed query is evaluated by a number of strategies called access plans. The physical characteristics of the physical storage and data are taken into account while generating access plans.

Query execution plan is a sequence of important operations that are used for evaluation of a query. It is also called query evaluation plan. Different evaluation plans have different costs.

NOTES

Example: Consider the query to be as follows:

'Display the balances of all accounts which are greater than 4500.'

Its corresponding SQL expression will be:

SELECT balance FROM account WHERE balance > 4500

Now the algebraic expression will be:

- $S_{\text{balance} > 4500}(P_{\text{balance}}(\text{ account }))$
- $P_{\text{balance}}(S_{\text{balance} > 4500}(\text{ account }))$

The following figure illustrates an evaluation plan for the above given query:

$$\begin{tikzcd}[row sep=large]
\Pi_{\text{balance}} \arrow[d, dash] \\
(\sigma_{\text{balance} > 4500 \text{ (account)}}) \arrow[d, dash] \\
\text{account}
\end{tikzcd}$$

Improving query processing strategy or more accurately, query optimization helps in reducing query execution time and overhead.

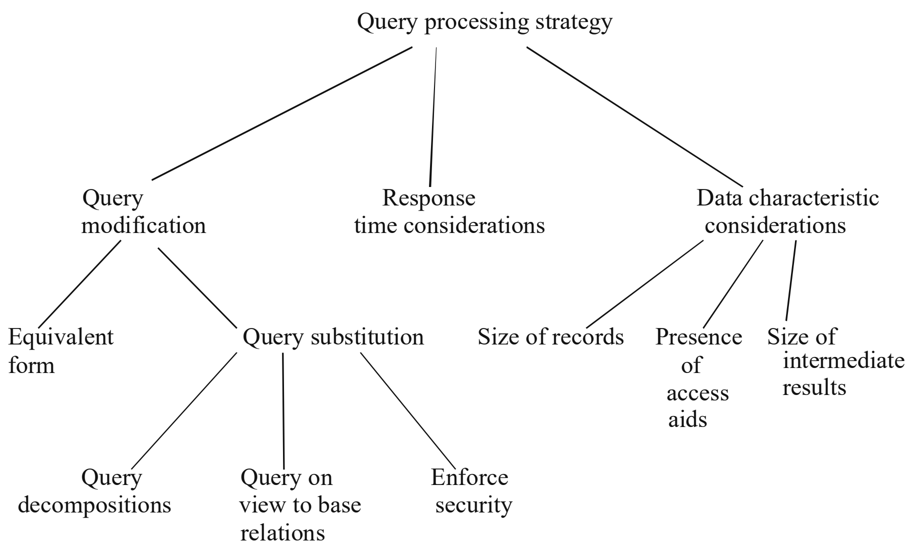

Query processing strategy
Query modification
Response time considerations
Data characteristic considerations
Equivalent form
Query substitution
Size of records
Presence of access aids
Size of intermediate results
Query decompositions
Query on view to base relations
Enforce security

Fig 1.2 *Query Processing Strategy*

Query modification is another aspect of query processing (Refer Fig. 1.2). This is called for when query is based on a view. Such queries have to be replaced by appropriate queries on the base relations.

Stage 1: Cast the query into internal form.

It involves the elimination of purely external considerations by transformation of the original query into some internal representation that is more suitable for machine manipulation . Thus, the internal form typically chosen is a kind of query tree or abstract syntax tree.

Example: Get names of suppliers who supply part 'P3' (query tree).

$$\begin{tikzcd}
\text{SP} \arrow[dr] & & \text{S} \arrow[dl] \\
& \text{JOIN over S\#} \arrow[d] & \\
& \text{RESTRICT where P\# =''P3''} \arrow[d] & \\
& \text{PROJECT OVER SNAME} \arrow[d] & \\
& \text{Final Result} &
\end{tikzcd}$$

Thus, we can assume that the internal representation of the query of above illustration is as the algebraic expression shown as follows:

((SP Join S) WHERE P# =P#('P3')) {SNAME}

Stage 2: Convert to canonical form.

In this , a number of optimizations that are 'guaranteed to be good' are performed by optimizer, regardless of the physical access paths and actual data values that exist in the stored database.

Canonical Forms:

For given set X of objects (say queries), equivalence is assumed among the objects. Subset S of X is said to be a set of canonical forms for X, under the stated definition of equivalence, only if every object x in X is equivalent to the single object s in X. The object s is said to be the canonical form for object x.

Example: (P JOIN Q)WHERE restrictions on P can be transformed into an equivalent but more efficient expression.

(P WHERE restriction on P) JOIN Q

Stage 3: Choose Candidate Low-level Procedures.

At this stage, it should be decided how to execute the transformed query represented by that converted form by the optimizer.

Also, considerations such as other physical access paths or the existence of indexes, physical clustering of stored data, distribution of data values, etc come into play.

Basic strategy is to consider the expressions of query as specifying a series of 'low level' operations (restrict, join, summarize, etc) with certain interdependene among them. For each low level operation, the optimizer will have a set of predefined implementation procedures available to it and each procedure will also have a cost formula, indicating the cost associated with it.

Stage 4: Generate Query plans and choose the cheapest of them.

The final stage in optimization process involves the set of candidate query plans to be constructed, followed by a choice of the cheapest (i.e., the best) of those plans.

Choosing the cheapest plan naturally requires a method for assigning a cost to any given plan. The cost for a given plan is the sum of all costs associated with individual procedures.

The following are the key components of query processing. Their roles and functions have been described here.

NOTES

Scanner/Parser: Scanner identifies the language, such as, SQL, keywords, relation names and attributes names in text of the query.The parsing process has two functions, i.e., 1) breaking down the syntax into constituent parts that can be understood by RDBMS (Relational DataBase Management System) and 2) checking the correct syntax of incoming query.

Standardization/Relational Algebraic Expression: Capability to accept high level dynamic queries from users who have no knowledge of underlying data structures and to convert them in a useful format for optimization are some of the important advantages of RDBMS. It rearranges the query tree into a more standardized canonical format known as relational algebraic expression, for easy use by query optimizer.

Code Generator: A code generator converts a physical query plan into an executable code.

Query Optimization: Its basic goal is to produce an efficient plan for processing the query represented by a standardized canonical query tree. Query optimizer is the main component of RDBMS, enabling it to work efficiently and intelligently.

Query Execution/Tree Manager: It runs the query code, whether in compiled or interpreted mode, to produce query results.

It is the process of executing the plan chosen during query optimization. The objective is to spend the least amount of time in executing the plan by quickly returning the answer to the user. Different areas of query processor rely on the sort algorithms e.g., index creations, merge joins, stream aggregation and so on.

1.2.4 General Strategies for Query Processing

Data is stored in database, but query is made to retrieve data from the database. There is always a problem with performance that is related to query. More than 80 percent of problems related to database query performance can be solved by writing efficient SQL statements.

One may make a strategy for retrieving data and should prepare a query *execution plan* that describes the way RDBMS will process a particular query. This includes index usage, joins logic, and estimated resource cost.

The *query optimizer* in the database which is a software component in the RDBMS that carries out analysis of SQL statement for finding the best way for its execution. Modern optimizers are cost-based, and they estimate the cost of every possible way for executing a statement and choosing that with the lowest cost. Statistics gathered from the database are important component of cost-based optimizers. Number of rows in a table and number of unique values in each indexed column, keeps such statistic. Some optimizers are rule-based, and they can apply a set of rules to SQL statement instead of cost estimates while deciding for its best execution.

Order of table names: Ideally, the DBMS must access the most selective table that keep a sizable amount of records eliminating most rows from the result set.

- **Order of search predicates:** Ideally, the most restrictive predicate in which most number of rows has been eliminated has to be evaluated first.

- **Query rewrites:** Queries should be rewritten in an efficient form by the optimizer. Many optimizers automatically rewrite subselects into equivalent joins for simplifying the subsequent processing. For this purpose, certain DBMS options has to be enabled to allow optimizer for rewriting queries.
- **Other criteria:** One has to find other criteria that affect optimizer.

It is always advised to avoid scanning of large tables. Tables having large number of rows, say more than over 1000 rows, scanning all these rows in the table takes lot of time. It is better to use indexing.

A database table containing many columns consumes more memory for intermediate operations, such as sorting and putting data in a result set. It is always advisable to keep fewer columns in a database table. Keeping fewer tables results in more efficient query.

Hints are special syntax in SQL which are placed in the SQL statement that directs optimizer to take certain actions, such as directing for the use of a particular index or a particular method to join tables. But hint should not be used frequently because hints are not portable.

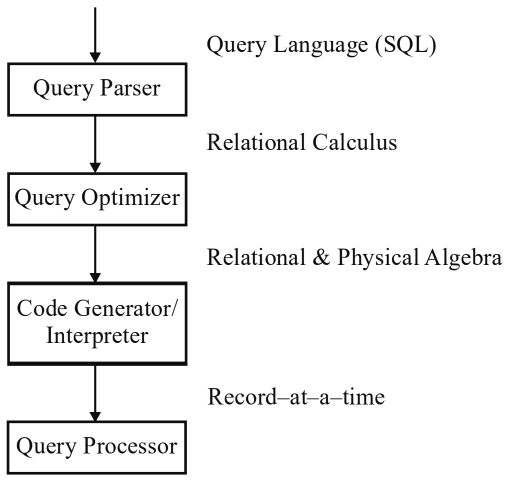
The use of temporary tables also improves query processing such as, in assembling large result sets with addition of an index for supporting multiple queries against the temporary table.

Views helps: Views also helps in improving query since they hide complex operations such as nested aggregate functions. Normal DBMSs do not have SQL statement cache. Views may process with more efficiency than ordinary queries since the SQL statement defining the view are already parsed and optimized. This means that this work is not to be done every time the view is accessed.

Indexes greatly improve data access times but consume overheads and take storage space and there should be a trade off.

Query execution

Query execution can be defined as the method in which the selected plan is executed at the query optimization stage. The methods accessible for the query optimizer are determined by the query execution component. For instance, algorithms accessible to the query optimizer like merge join and hash join are applied by the SQL Server.



The query optimizer acts as the intelligence of the relational database system and enables it to work effectively. The query is bound to get completed at a faster rate if the database has a complex query optimizer rather than a normal optimizer, especially when the query is difficult.

The main aim of query execution is to execute the plan as early as possible and provide the answer the user or the program run by him in minimum time which is unlike the plan processing by using limited resources such as, memory, I/O and CPU. For instance, usually, a parallel query utilizes a greater variety of resources unlike a nonparallel query; however, due to its ability of returning decisions faster, it is preferred more than the others.

The phase of query execution comes before query optimization since the methods of processing that are accessible determine the series of options provided to the optimizer. The methods available involve operations such as hash operations, join, sorting and disk I/O. These methods are briefly discussed as follows:

(i) Disk I/O: Proper and effective data transfer between disks and the memory is the basic foundation for an efficient query processing. Many improvements in disk I/O have been implemented by SQL Server 7.0.

Random I/O vs sequential I/O: Disk I/O is a costly computer operation of two types: sequential I/O, that processes data in a similar arrangement as saved on the disk and random I/O, that processes the data in a unsystematic way, leaping to any file on any disk from one file to another disk. If the data involved in the file is of a large amount, sequential I/O may become cheaper than random I/O.

The on-disk structure of Microsoft SQL Server 7.0 decreases random I/O allowing brisk scans of outsized heap tables. Since, no specific arrangement is given to the stored data rows; it makes these tables without a clustered index which is a vital feature for decision support queries. If the sorting of index order for the successive processing steps is not required, then the same may be also used for grouped and non-grouped indexes.

If possible, the server will scan an index in place of reading the table which is beneficial if the index is a covering index where one possesses every field compulsory to gratify the query. Reading the B-tree index in the disk order may solve a query rather than in sort order. Henceforth, it would be in a sequential I/O and would perform in a greater pace.

(ii) Merge joins, hash joins, and hash teams: SQL Server 6.5 makes use of cased loops iteration to navigate from one to another row like shifting to three or four order-line items from an order record, is utilized by the. Although, such type of iteration is not suggested for joins of more than one record like typical data warehouse queries.

Merge joins, hash joins, and hash teams are methods introduced by SQL Server 7.0 out of which, hash teams is an important improvement not accessible in any other relational database.

Merge joins: For processing outer joins, inner joins, semi-joins, unions and intersections, a merge join concurrently passes over two sorted inputs. It takes the help of sorted scans of B-tree indexes. This is the technique that chooses whether join fields are indexed and columns represented in the index cover the query.

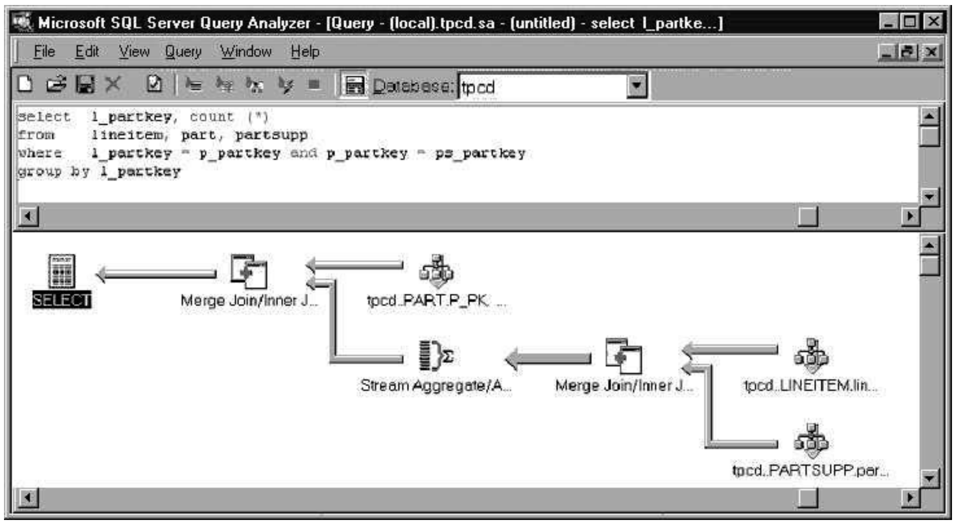
Hash joins: Such joins mix input values. These input values are based on randomizing function that is replicable and runs values for comparison in the hash table to check for similarities. The inputs that are smaller than memory that is available have the hash table remaining in the memory. On the other hand, where larger inputs are involved, overflow files on disk are employed. Choosing for large, non-indexed tables, specifically for intermediate results can be termed as hashing.

SQL Server 7.0 uses well known hashing techniques such as cache-optimized in-memory hashing, recursive partitioning, hybrid hashing, large memory, role reversal and bit-vector filtering.

Hash teams: It is an improvement in SQL Server 7.0. The query optimizer takes the advantage of various queries consisting of multiple processing phases. In such processing, the query optimizer takes the benefit of related processes along the multiple phases. For example, if one wants to procure information about the number of order-line items introduced to every part number and for every vendor, given in the following SQL code:

```
    SELECT l_partkey, count (*)
    FROM lineitem, part, partsupp
    WHERE  l_partkey  =  p_partkey  and  p_partkey  =
ps_partkey
    GROUP BY l_partkey
```

The query processor generates the following execution plan in response to the code:



```
select   l_partkey, count (*)
from     lineitem, part, partsupp
where    l_partkey = p_partkey and p_partkey = ps_partkey
group by l_partkey
```

As a result, the query executes merge of inner join between the table named `lineitem` and the `table partsupp`. It computes `count` which is aggregate for stream aggregate, and subsequently unites the outcome with part table. This query does not require a sort operation. It starts with the recovery of records in sorted order from tables, `lineitem` and `partsupp` making use of sorted scans on an index.

1.2.5 Types of Query Optimizers

(i) Syntax-based query optimizers: In such optimizers, a technical plan for attaining the response to an SQL query is created depending on the precise syntax of the query and the arrangement of the sections in the query. The same plan is executed each time in spite of the apt change in the number/composition of records, in such type of query optimizer. Statistics about the database are neither preserved nor considered contrasting to other optimizers.

(ii) Cost-based query optimizers: For answering an SQL query, the cost-based query optimizer chooses amid different alternative plans, the best cost effective plan. The plan is decided by emphasizing on various factors such as the number of I/O actions, the amount of time taken by the CPU, etc. Through recording statistics related to the quantity and composition of records in a table or index, the cost-based query optimizer estimates the costs and is not reliant on the query's accurate syntax or the arrangement of clauses inside the query.

1.3 SET OPERATIONS

Oracle SQL supports the following four set operations:

- UNION
- UNION ALL
- INTERSECT
- MINUS
- DIVIDE

Two SELECT statements can be combined into a compound query by a set operation only if they satisfy the following two conditions:

- The result sets of both the queries must have the same number of columns.
- The data type of each column in the second result set must match the data type of its corresponding column in the first result set. Data types, such as VARCHAR, VARCHAR2 and CHAR, are compatible. When this occurs, the SELECT statements are said to be row-compatible.

These conditions are also referred to as *union compatibility* conditions. It should be noted that both the individual queries cannot have ORDER BY CLAUSES but ORDER BY clause may be applied to the entire result of the UNION.

Restrictions on Set Operators

- The set operations cannot be valid with columns of type BLOB, CLOB, BFILE, VARRAY or nested table.
- The UNION, INTERSECT and MINUS operations cannot be performed on LONG columns.
- If the SELECT list preceding the set operator contains an expression, then a column alias must be provided for the expression in order to refer to it in the ORDER BY clause.

- The ORDER BY clause cannot be specified in both the subqueries of these operators.

The general syntax for a query involving a set operation is as follows:

```
<component_query>
{UNION | UNION ALL | MINUS | INTERSECT}
<component_query>
```

The keywords UNION, UNION ALL, MINUS and INTERSECT are set operators.

UNION ALL	Combines the results of two SELECT statements into one result set.
UNION	Combines the results of two SELECT statements into one result set, and then eliminates any duplicate rows from that result set.
MINUS	Takes the result set of one SELECT statement, and removes those rows that are also returned by a second SELECT statement.
INTERSECT	Returns only those rows that are returned by each of two SELECT statements.

1. UNION

The UNION, as defined in relational algebra, is the set union of two relations. UNION returns all the distinct rows returned by either of the queries it is applied to. Since a mathematical set contains no duplicates, the set union contains no duplicates, and any row appearing in both results must appear but once in the result.

As the UNION operation eliminates duplicates, DISTINCT, while syntactically correct, is not necessary.

Query: List all employees who are working in department 'D01' or 'D03'.

```
SELECT * FROM emp WHERE dno='D01'
UNION
SELECT * FROM emp WHERE dno='D03'
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97

6 rows selected.

This UNION could have been done more concisely by using an IN clause.

```
SELECT * FROM emp WHERE dno IN('D01','D03');
```

However, this is not easy if the two component queries are from different tables.

2. UNION ALL

NOTES

The UNION ALL operator merges the result sets of two component queries. This operation returns rows retrieved by either of the component queries without eliminating duplicates.

Query: List the code, name and designation of all employees who are working in the project 'P01' or 'P03'.

```
SELECT ecode, ename, desg FROM emp WHERE ecode IN (SELECT ecode FROM assign WHERE pid='P01')
UNION ALL
SELECT ecode, ename, desg FROM emp WHERE ecode IN (SELECT ecode FROM assign WHERE pid='P03');
```

Output:

ECODE	ENAME	DESG
E01	KOUSHIK GHOSH	SYSTEM ANALYST
E02	JAYANTA DUTTA	PROGRAMMER
E08	GOUTAM DEY	PROGRAMMER
E01	KOUSHIK GHOSH	SYSTEM ANALYST
E03	HARI NANDAN TUNGA	PROGRAMMER
E06	JISHNU BANERJEE	SYSTEM MANAGER
E09	PINAKI BOSE	PROGRAMMER

7 rows selected.

Notice that because the ALL clause was used, the duplicates were retained in the result. The UNION eliminates the duplicates.

```
SELECT ecode, ename, desg FROM emp WHERE ecode IN (SELECT ecode FROM assign WHERE pid='P01')
UNION
SELECT ecode, ename, desg FROM emp WHERE ecode IN (SELECT ecode FROM assign WHERE pid='P03');
```

Output:

ECODE	ENAME	DESG
E01	KOUSHIK GHOSH	SYSTEM ANALYST
E02	JAYANTA DUTTA	PROGRAMMER
E03	HARI NANDAN TUNGA	PROGRAMMER
E06	JISHNU BANERJEE	SYSTEM MANAGER
E08	GOUTAM DEY	PROGRAMMER
E09	PINAKI BOSE	PROGRAMMER

6 rows selected.

3. INTERSECT

The INTERSECT, as defined in relational algebra, is the set intersection of two relations. It returns only those rows returned by both component queries. It is the logical product of two sets. Since a mathematical set contains no duplicates, the set intersection contains no duplicates, and any element appearing in both the sets must appear but once in the result.

Query: Retrieve code of the employees who have been assigned a project.

```
SELECT ecode FROM emp
INTERSECT
SELECT DISTINCT ecode FROM assign;
```

Output:

```
ECODE
-----
 E01
 E02
 E03
 E06
 E07
 E08
 E09
```

7 rows selected.

The INTERSECT operation can be implemented using the EXISTS clause as follows:

```
SELECT ecode FROM emp
WHERE EXISTS (SELECT * FROM assign WHERE
emp.ecode=assign.ecode);
```

4. MINUS

MINUS, as defined in relational algebra, is the set difference of two relations. It returns only rows returned by the first component query but not by the second. It is the difference between two sets. Since a mathematical set contains no duplicates, the set difference contains no duplicates. Since the MINUS operation eliminates duplicates, DISTINCT, while syntactically correct, is not necessary.

Query: Retrieve code of the employees who are not assigned any project.

```
SELECT ecode FROM emp
MINUS
SELECT DISTINCT ecode FROM assign;
```

Output:

```
ECODE
-----
 E04
 E05
```

The above query can be used in a subquery to list the employee details.

NOTES

Query: Retrieve employee details who are not assigned in any project.

```
SELECT * FROM emp
WHERE ecode IN (SELECT ecode FROM emp
MINUS
SELECT DISTINCT ecode FROM assign);
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95

The MINUS operation can be implemented using the NOT EXISTS clause as follows:

```
SELECT ecode FROM emp
WHERE NOT EXISTS (SELECT * FROM assign WHERE
emp.ecode=assign.ecode);
```

Output:

ECODE
E04
E05

5. DIVIDE

In SQL, the division operation is not directly supported. It is, however, defined as a component of RELATIONAL ALGEBRA. According to Date, 'DIVIDE takes two relations, one binary and one unary, and builds a relation consisting of all values of one attribute of the binary relation that match (in the other attribute) all values in the unary relation.' By the use of subquery with the existence test, the DIVISION operation can be implemented. Consider the following query:

Query: List the employees who are working on all the projects.

```
SELECT * FROM emp E WHERE NOT EXISTS(SELECT * FROM project P
WHERE NOT EXISTS(SELECT * FROM assigns A
WHERE A.pid=P.pid AND E.ecode=A.ecode));
```

Alternatively, it can be written as follows:

```
SELECT * FROM emp WHERE ecode IN( SELECT ecode FROM
assign GROUP BY ecode HAVING COUNT(ecode)=(SELECT
COUNT(pid) FROM project));
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D0	SYSTEM ANALYST	10-MAR-93

ORDER BY clause in set operation

If it is needed to use ORDER BY in a query involving set operations, then it must be placed at the end of the entire statement. The ORDER BY clause should appear only once at the end of the compound query. The component queries cannot have individual ORDER BY clauses.

```
SELECT dno, dname FROM dept WHERE city='KOLKATA'
UNION
SELECT ecode, ename FROM emp WHERE dno='D01'
ORDER BY dno;
```

Output:

```
ECODE     ENAME
D01       PROJECT
D03       PERSONNEL
E01       KOUSHIK GHOSH
E02       JAYANTA DUTTA
E07       RANI BOSE
E08       GOUTAM DEY
```

6 rows selected.

Notice that the column name used in the ORDER BY clause of this query is taken from the first component query. Specifying columns from the second component query will generate an error, as in the following example:

```
SELECT dno, dname FROM dept WHERE city='KOLKATA'
UNION
SELECT ecode, ename FROM emp WHERE dno='D01'
ORDER BY ecode;
```

Output:

```
ORDER BY ecode
 *
ERROR at line 4:
ORA-00904: invalid column name
```

The solution is that the column should be specified in the ORDER BY clause by column position as follows:

```
SELECT dno, dname FROM dept WHERE city='KOLKATA'
UNION
SELECT ecode, ename FROM emp WHERE dno='D01'
ORDER BY 1;
```

Output:

ECODE	ENAME
D01	PROJECT
D03	PERSONNEL
E01	KOUSHIK GHOSH
E02	JAYANTA DUTTA
E07	RANI BOSE
E08	GOUTAM DEY

6 rows selected.

Precedence of Set Operators

If more than two component queries are combined using the set operators, then the expression is evaluated from left to right.

```
SELECT ecode, ename FROM emp WHERE dno='D01'
UNION
SELECT ecode, ename FROM emp WHERE ecode IN (SELECT ecode
FROM assign
WHERE pid='P02')
INTERSECT
SELECT ecode, ename FROM emp WHERE desg='PROGRAMMER';
```

Output:

ECODE	ENAME
E02	JAYANTA DUTTA
E08	GOUTAM DEY

Parentheses may be used to impose a particular order of evaluation on the set operators. The above query gives different result if a parenthesis is used. In that case, the query within the parentheses is evaluated first. The result is then combined with the component queries outside the parentheses.

```
SELECT ecode, ename FROM emp WHERE dno='D01'
UNION
(SELECT ecode, ename FROM emp WHERE ecode IN (SELECT
ecode FROM assign
WHERE pid='P02')
INTERSECT
SELECT ecode, ename FROM emp WHERE desg='PROGRAMMER');
```

Output:

ECODE	ENAME
E01	KOUSHIK GHOSH
E02	JAYANTA DUTTA
E07	RANI BOSE
E08	GOUTAM DEY

1.4 PIPELINING

Pipelining is a method of decomposing a sequential process into suboperations. Every subprocess is executed in a devoted segment, which operates parallely with all other segments.

Pipelining is an effective method of increasing the execution speed of the processor, provided the following conditions are satisfied:

- It is possible to break up an instruction into a number of independent paths, each part taking nearly equal time to execute.
- There must be locality in instruction execution, i.e., instructions are executed in a sequence one after the other in the order in which they are written. If instructions are not executed in a sequence but 'jump around' due to many branch instructions, then pipelining is not effective.
- Sufficient resources are available in a processor so that if a resource is required by the successive instructions in the pipeline, it is readily available.

A pipeline is essentially a collection of dispensed parts from which binary data pours. Every section carries out partial processing, marked in the way the task is divided. The outcome gained from the calculation in every part is moved to the next sections in pipeline. The final outcome comes once data has gone through all parts. it is specialty of pipelines that lot of calculations could be in way of different sections in the same durations.

The foremost way to view the pipeline framework is to think that each segment comprises an input register that holds the data information followed by a combinational circuit. The combinational circuit in the particular segment performs the suboperation. The result of the combinational circuit in a given segment is assigned to the input register of the next segment. A clock is assigned to every register after sufficient time has gone by to do all segment activity. Thus, in this manner, the information runs through the pipeline one step at a time.

Suppose we want to perform the combined multiply and add operations with a stream of numbers.

$$A_i * B_i + C_i \text{ for } i = 1, 2, 3, 4, ..., 7.$$

Each suboperation is to be executed in a dedicated segment in a pipeline structure. Each segment consists of one or two registers and a combinational circuit to carry out the operation as shown in Figure 1.3. $R1$, R2, R3, R4 and $R5$ are registers that receive new information with every clock pulse. The suboperations done in every segment of the pipeline are as follows:

$R1 \leftarrow Ai$	$R2 \leftarrow Bi$	Input Ai and Bi
$R3 \leftarrow R1 * R2,$	$R4 \leftarrow Ci$	Multiply and input Ci
$R5 \leftarrow R3 + R4$		Add Ci to product

NOTES

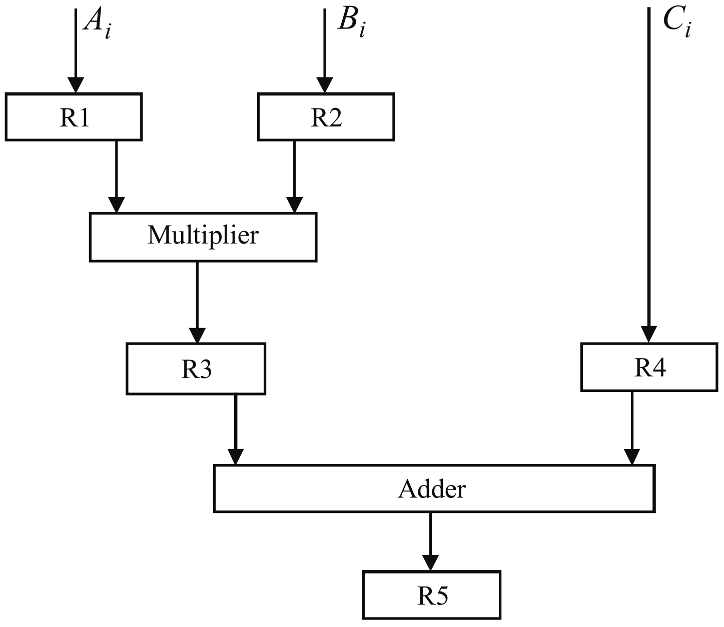


Fig. 1.3 *Pipeline Processing*

Table 1.1 *Content of Registers in a Pipeline*

Clock pulse Number	**Segment 1**		**Segment 2**		**Segment 3**
	R1	**R2**	**R3**	**R4**	*R5*
1	A_1	B_1	–	–	–
2	A_2	B_2	$A_1 * B_1$	C_1	–
3	A_3	B_3	$A_2 * B_2$	C_1	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_1	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_1	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_1	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_1	$A_5 * B_5 + C_5$
8	–	–	$A_7 * B_7$	C_1	$A_6 * B_6 + C_6$
9	–	–	– –	–	$A_7 * B_7 + C_7$

The five registers are loaded with new data at every clock pulse, as illustrated in Table 1.1. With the first clock pulse, the value of A1 and B1 transfers into register R1 and R2, respectively. The product of R1 and R2 will transfer into R3 with the second clock pulse. At the same clock pulse, the value of *C*1 will transfer into register R4 and the value of A2 and B2 will transfer into register R1 and R2. The third clock pulse operates on all three segments simultaneously. It places A3 and B3 into R1 and R2, sends the product of R1 and R2 into R3, transfers C2 into R4 and places the sum of R3 and R4 into R5. It takes three clock pulses to fill the pipe and retrieve the first output from R5. From there on, each clock pulse produces a new output and moves the data one step down the pipeline. This will continue as long as the new input data flows into the system. When no additional data is accessible, the process continues till the last result comes out of the pipeline.

The general structure of a three-segment pipeline is shown in Figure 1.4. Each segment consists of a combinational circuit through which operands pass in a specified sequence. The combinational circuit *Ci* performs the required suboperation over the data flowing through the pipe. *Ri* is the register that will hold the intermediate results between the stages. A common clock is applied to all registers. Information flows through adjacent stages under the control of a common clock. Also, registers are placed in between two segments to separate them.

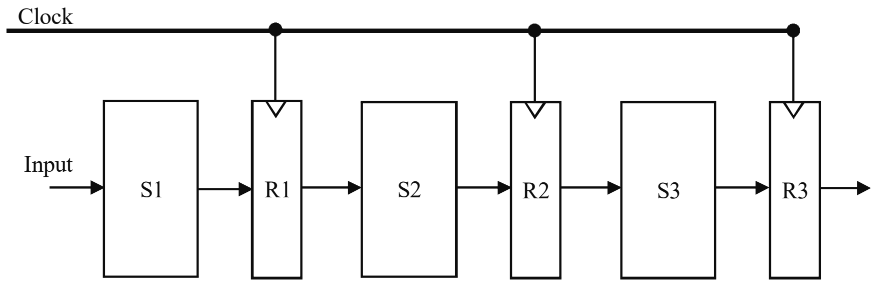


Fig. 1.4 *Three-Segment Pipeline*

Space-Time Diagram for Pipeline

The behaviour of a pipeline can be illustrated with a space–time diagram. Figure 1.5 shows the segment utilization as a function of time. The time in clock cycles, is given along the horizontal axis and the vertical axis gives the segment number.

Segment	1	2	3	4	5	6	7	8	9 → Clock cycles
1	T1	T2	T3	T4	T_5	T_6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T5	
4				T1	T2	T3	T4	T5	T6

Fig. 1.5 *Space–Time Diagram*

The diagram shown in Figure 1.5 shows six tasks T1 through T6 executed in four segments. Initially, task T1 is handled by segment 1. After the first clock, segment 2 is busy with T1, while segment 1 is busy with task T2. Continuing in this manner, the first task T1 is completed after the fourth clock cycle. From then on, the pipe completes a task with every clock cycle. No matter how many segments there are in the system, once the pipeline is full, it takes only one clock period to obtain an output.

Consider a case where a *k*-segment pipeline with a clock-cycle time t_p is used to execute *n* tasks.

The first task T1 needs the same time as kt_p to finish its operation since there are *k*-segments in the pipe. The remaining $(n-1)$ task comes out of the pipe at the speed of one task per clock cycle. They will complete all the tasks after a time equivalent to $(n-1)\ t_p$.

Thus, to finish n tasks using a k-segment pipeline, the time needed will be $k + (n - 1)$ clock cycles.

As for instance, to complete 6 tasks using the 4-segment pipeline, the time needed to finish all operations is $4 + (6 - 1) = 9$ clock cycles.

Consider a non-pipeline unit that performs the same operation and takes time equal to t_n to complete each task. The total time required for n tasks is nt_n.

The speedup of a pipeline processing over an equivalent non-pipeline processing is defined by the ratio:

$$S = nt_n/(k + (n - 1))t_p$$

Therefore, as the amount of tasks increases, n becomes much bigger than $k - 1$, and $k + n - 1$ reaches the value of n. Thus, speedup becomes:

$$S = t_n/t_p$$

Check Your Progress

1. Write the four broad stages that constitute query processing.
2. What does a scanner do in query processing?
3. Define the term UNION.
4. What is pipelining?

1.5 USING HEURISTICS IN QUERY OPTIMIZATION

It is a rule based method of producing an efficient query execution plan in the form of a query tree or a query graph data structure.

The basic heuristic rule is to perform SELECT and PROJECT operations on the file before subjecting it to JOIN operations. There are several other heuristic execution strategies.

These strategies are as follows:

(a) Perform selection operation as early as possible.
(b) Perform projection as easily as possible.
(c) Compute one common expression.
(d) Combine the Cartesian product with a subsequent selection operation.
(e) Use associativity of binary operations to rearrange leaf nodes.

Notation for query trees and query graphs: A query tree is a tree data structure that corresponds to a relational algebra expression. It represents relational algebra operations as internal nodes and input relations of the query as leaf nodes of the tree.

An execution of the query tree consists of executing an internal node operation (addition, deletion or updation etc.) whenever its operands are available and then replacing that internal node by the relation that results from executing the operation. The execution terminales when the root node is executed and a result. relation for the query is obtained.

Now we will do a query transformation using transformation rules and strategies.

Example: Consider the following query :

For query project located in 'Bombay' relations, the project number, the controlling department numbers and the department manager's last name, address, and date of birth are provided by the following schemas.

PROJECT

Pname	Pnumber	Plocation	Dname

DEPARTMENT

Dname	Dnumber	mgereno	Mgrstardate

EMPLOYEE

Fname	Initial	Lname	Eno	DOB	Add	Sex	Salary	Dno

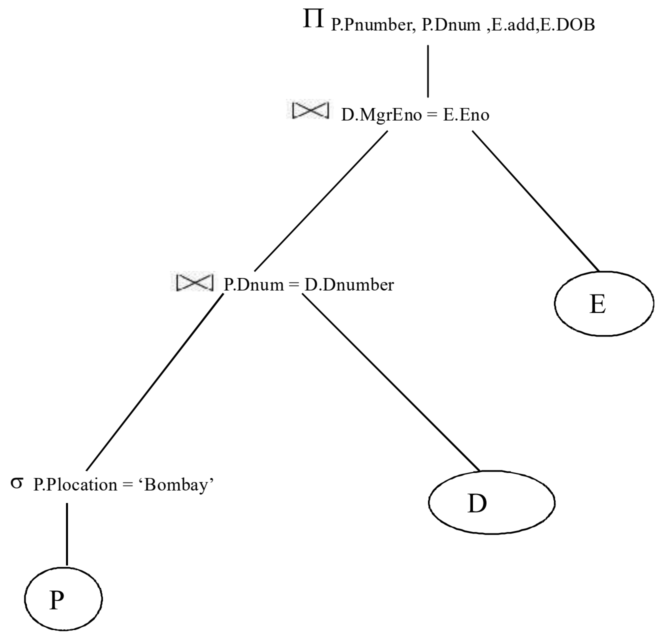
Solution:

$\pi_{Pnumber,Dnum,Lname,Add,DOB}(((\sigma_{Plocation=\text{ 'Bombay'}(Project)})))\ _{Dnum=Dnumber}(DEPARTMENT)\ _{manger=eno}(EMPLOYEE)$

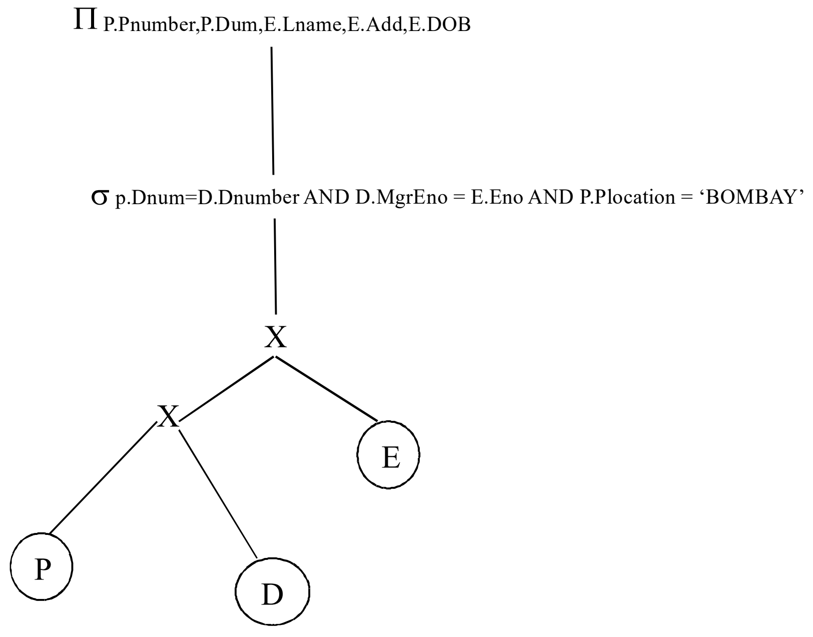
The equivalent SQL query is:

SELECT p. Pnumbner, P.Dnum, E.Lname, E.add, E.DOB FORM PROJECT As P, DEPERTMENT As D, Employee As E WHERE P.Dnum=D.Number AND D.Mgreno=E.Eno AND P.Plocation= 'BOMBAY';

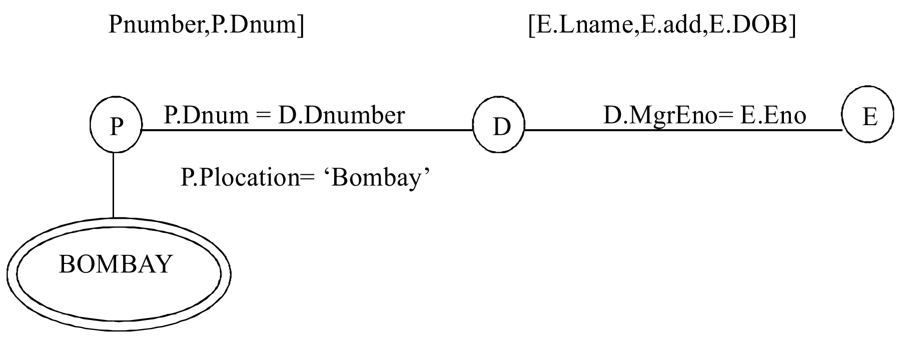
(a) Query tree corresponding to the relational algebra expression:



(b) Initial (canonical) query tree for SQL query


Π P.Pnumber,P.Dum,E.Lname,E.Add,E.DOB
σ p.Dnum=D.Dnumber AND D.MgrEno = E.Eno AND P.Plocation = 'BOMBAY'
X
X
E
P
D

(c) Query graph for query:



Heuristic Optimization of Query Trees

In general, many different query trees or, in other words, many different relational algebra expressions can be equivalent; it means that they can correspond to the same query. A standard initial query tree corresponding to an SQL query can be typically generated by a query parser and such a canonical tree represents a relational algebra expression that is very inefficient if executed directly. Now it becomes the job of heuristic query optimizer to transform this initial query tree to a final query tree which is then ready for execution. The rules for equivalence among relational algebra expressions must be included in an optimizer that can be applied to the initial tree.

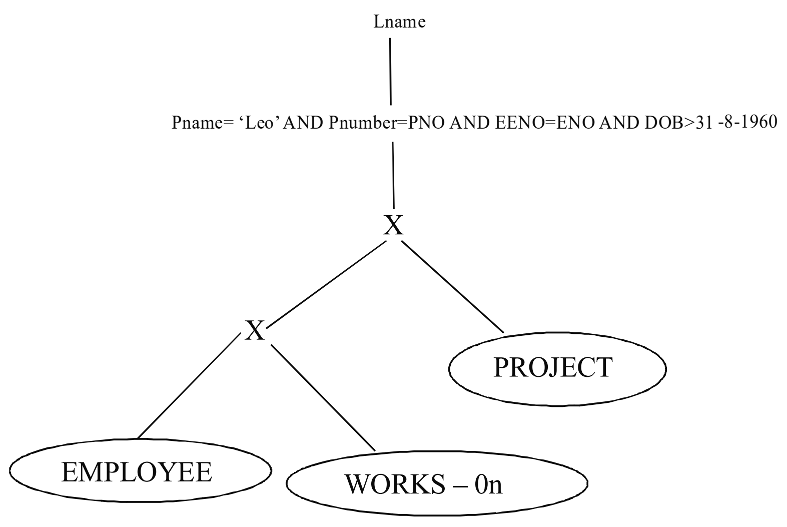
***Example*:** Consider the following query:

'Find the last names of all employees born after 1960 who work on a project "Leo".'

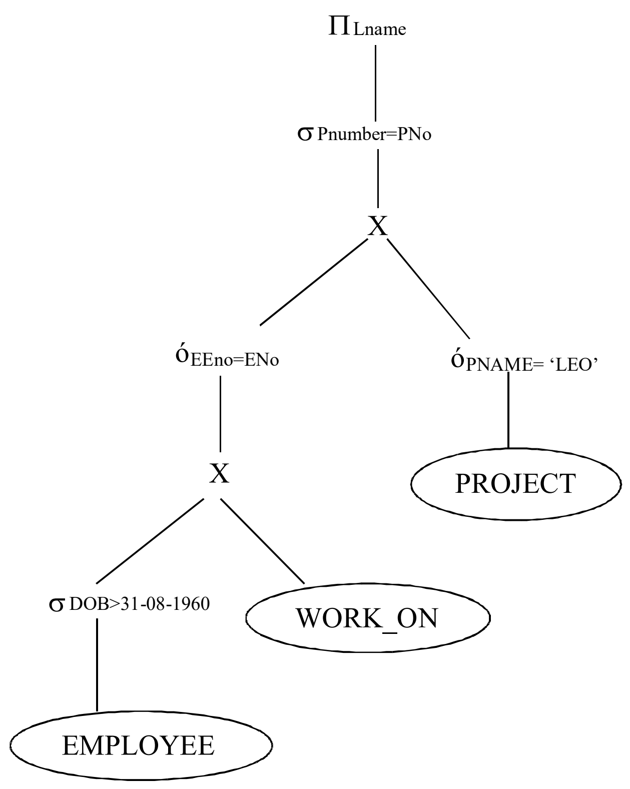
Solution: A query tree can be transformed step by step into another query tree that is more efficient. However, it is to be noted that the transformation steps always lead to an equivalent query tree.

Steps in converting a query tree during heuristic optimization

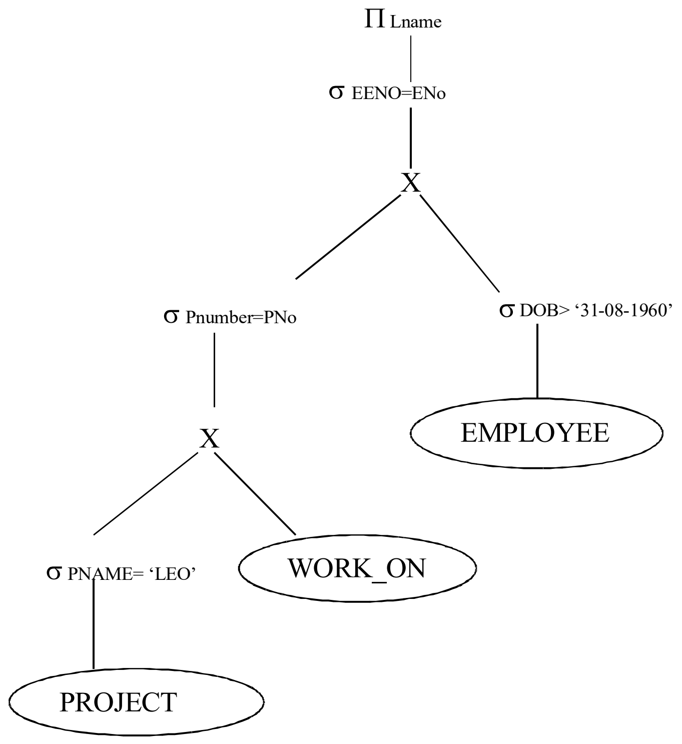
(a) Initial query tree for SQL query


Lname
Pname= 'Leo' AND Pnumber=PNO AND EENO=ENO AND DOB>31 -8-1960
X
X
PROJECT
EMPLOYEE
WORKS – 0n

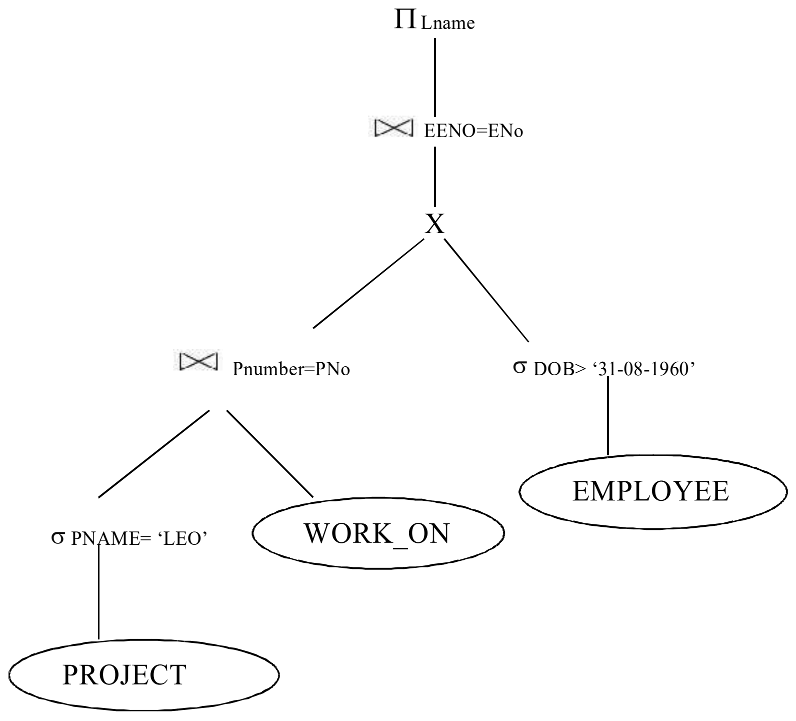
(b) Moving SELECT operations down the query tree.



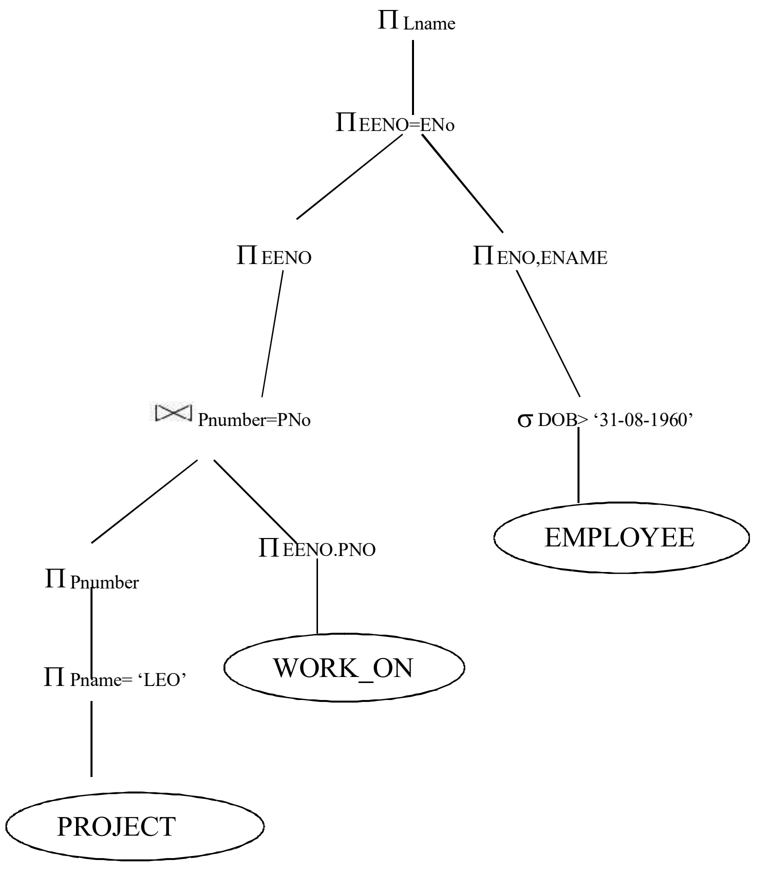
(c) Applying the more restrictive SELECT operation first



(d) Replacing CARTESIAN PRODUCT SELECT with JOIN operation



(e) Moving PROJECT operations down the query tree:



Converting Query Trees into Query Execution Plans

An execution plan for a relational algebra expression represented as a query tree includes the algorithms to be used in computing the relational operators represented in the tree as well as the information about the access methods available for each relation.

For example, consider the following query:

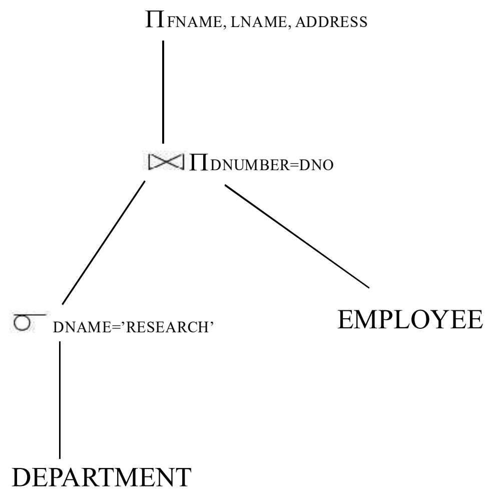
‘Retrieve the name and address of all employees who work for the “Research” department.’

Its corresponding algebra expression is:

$\Pi_{\text{FNAME, LNAME,ADDRESS}}(\sigma_{\text{DNAME='RESEARCH'}}(\text{DERPARTMENT}) \bowtie \Pi_{\text{DNUMBER=DNO}}$ EMPLOYEE)

NOTES

The corresponding query tree is shown as follows:



To convert this into an execution plan, the optimizer might choose an index search for the SELECT operation, a nested loop join algorithm for the join, a table scan as an access method for EMPLOYEE, and a scan of the JOIN result for the PROJECT operator. Moreover, the approach taken for executing the query may specify a pipelined or materialized evaluation. The advantage of pipelining is the cost saving in not having to read the intermediate results back for next operation and also in not having to write the intermediate results to disk.

The cost for query execution has the following components:

(i) Access cost to secondary storage
(ii) Storage cost
(iii) Computation cost
(iv) Memory usage cost
(v) Communication cost

Cost functions for SELECT

The costs are given in terms of number of block transfers between the memory and secondary storage. The other costs, such as, storage cost, computation time, and so on are ignored, as they are not very significant.

Cost functions for JOIN

The JOIN operation is the most time consuming operation in query processing. To develop a reasonably accurate cost function for JOIN operations, we need to have an estimate of the size of the file that is generated as a result of the JOIN operation. It can be considered as a ratio of the size of the join file to the size of the Cartesian product file, if both are applied to the same input files.

1.6 BASIC ALGORITHMS FOR EXECUTING QUERY OPERATIONS

Consider a selection operation on a relation whose tuples are stored together in one file. Two scan algorithms to implement the selection operation are explained in the following section.

Basic Algorithms

1. **A1 (linear search)**: In this type of search, the system scans each file block and tests all records to check if they satisfy selection condition. For a selection on a key attribute, if the required record is found without looking at the other records of the relation, the system can terminate the scan.

    The cost of linear search in terms of the number of I/O operations is b_r, where b_r denotes the number of blocks in the files. Selections on key attributes have an average cost of $b_r/2$ and may have a maximum cost of b_r.

    Advantage: The linear search algorithms can be applied to any file.

    Disadvantage: It is slower.

2. **A2 (binary search):** We can use a binary search to locate records that satisfy the selection criteria, if the file is ordered on an attribute and the selection condition is an equality comparison on the attribute.

    The system performs the binary search on the block of the file.The number of blocks that need to be examined to find a block containing the required records is $(\log_2(b_r))$, where b_r denotes the number of blocks in the file.

    More than one block may contain required records and the cost of reading the extra blocks has to be added to the cost estimate if the selection is on a nonkey attribute.

Selections Using Indices

Search algorithms that use an index are referred to as index scans. Search algorithms that use an index are:

1. **A3 (Primary index equality on key)**: We can use the index to retrieve a single record that satisfies the corresponding equality condition for an equality comparison on a key attribute with a primary index.

    If a B^+ tree is used, the cost of the operation in term of I/O operations is equal to the height of the tree plus one I/O to fetch the record.

2. **A4 (Primary index equality on nonkey):** When the selection condition specifies an equality proportional to the height of the tree plus the number of blocks containing records with the specified search key, we can retrieve multiple records by using a primary index.

3. **A5 (Secondary index equality):** This strategy can retrieve a single record if the equality condition is on a key; multiple records may be retrieved if the indexing field is not a key. Selection specifying an equality condition can use a secondary index.

    In the first case, only one record is retrieved and the cost is equal to the height of the tree plus one I/O operation to fetch the record. In the second case, each record may be resident on a different block, which may result in one I/O operation per retrieved record.

Selections Involving Comparisons

Consider a selection of the form $ó_{A<v(r)}$. We can implement the selection either by using a liner or binary search or by using indices in one of the following ways:

1. **A6 (Primary index comparison):** A primary ordered index (for example, a primary B^+ tree index) can be used when the selection condition is a comparison. For comparison conditions of the form A>v or A>v, a primary index on A can be used to direct the retrieval of tuples. For A>v, we look up the value v in the index to find the first tuple in the file that has a value of A=v. A file scan, starting from that tuple to the end of the file, returns all tuples that satisfy the condition. For A>v, the file scan starts with first tuple such that A>v.

   For A<v, we use a simple file scan starting from the beginning of the file, and continuing up to the first tuple with attribute A=v. For comparison of the form A<v or A<v, an index lookup is not required. The case A<v is similar, except the scan continues up to the first tuple with attribute A>v. In either case, index is not useful.
2. **A7 (secondary index comparison):** We can use a secondary ordered index to guide retrieval for comparison conditions involving <, > or =. The lowest level index blocks are scanned either from the smallest value up to v (for <and = <) or from v up to the maximum value (for > and > =)

Implementation of Complex Selections

We now consider selection conditions with more complex selection predicates.

These can be in the following forms:

- **Disjunction**
- **Conjunction**
- **Negation**

  In negation, the result of selection is the set of tuples of r for which the condition is evaluated to be false.We can implement a selection operation involving either a disjunction or a conjunction of simple conditions by using one of the following algorithms:

1. **A8 (Conjunctive selection using one index):** We first determine whether an access path is available for an attribute in one of the simple conditions. If there is a one, then one of the selection algorithms (A2 through A7) can retrieve records by satisfying that condition.
2. **A9 (Conjunction selection using composite index):** An appropriate composite index may be available for some conjunctive selections. If the selection specifies an equality condition on two or more attributes, and a composite index exists on these combined attribute fields, then the index can be directly searched. The type of index determines which of the algorithms—A3, A4, or A5—will be most suitable.
3. **A10 (Conjunctive selection by the intersection of identifiers):**Another alternative approach for implementing conjunctive selection operations involves the use of record pointer for on the fields individual conditions. The algorithm scans each index for pointers to tuples that satisfy an individual condition. The intersection of all the retrieved pointers is the set of pointers to tuples that satisfies the conjunctive condition. The pointers are used to

retrieve the actual records. The cost of algorithm A10 is the sum of the costs of retrieving the records at the intersection of retrieved lists of pointers, plus the cost of individual index scans. This cost can be reduced by sorting the list of pointers first and then retrieving records in the sorted order. Thereby,

(i) Blocks are read in sorted order, minimizing disk arm movement.

(ii) All pointers to records in block come together, hence all selected records in the block can be retrieved using a single I/O operation.

4. **A11 (Disjunctive selection by the union of identifiers)**: Each index is scanned for pointers to tuples that satisfy the individual condition, if access paths are available on all the conditions of a disjunctive selection. The union of all retrieved pointers yields the set of pointers to all tuples that satisfy the disjunctive condition. The actual records can then be retrieved by using the pointers. However, we will have to perform a linear scan of the relation to find tuples that satisfy the condition if even one of the conditions does not have an access path.

Check Your Progress

5. Write the heuristic execution strategies.
6. Write the one advantage of A1 (linear search) algorithms.
7. State various complex selection predicates.

1.7 ANSWERS TO 'CHECK YOUR PROGRESS'

1. We can identify four broad stages in query processing as follows:
   - Cast the query into internal form.
   - Convert it to canonical form.
   - Choose candidate low level procedures.
   - Generate query plans and choose the cheapest of them.
2. Scanner identifies the language, such as, SQL, keywords, relation names and attribute names in the text of the query.
3. The UNION, as defined in relational algebra, is the set union of two relations. UNION returns all the distinct rows returned by either of the queries it is applied to. Since a mathematical set contains no duplicates, the set union contains no duplicates, and any row appearing in both results must appear but once in the result. As the UNION operation eliminates duplicates, DISTINCT, while syntactically correct, is not necessary.
4. Pipelining is a method of decomposing a sequential process into suboperations. Every subprocess is executed in a devoted segment, which operates parallely with all other segments.
5. The heuristics execution strategies are as follows:
   - Perform selection operation as early as possible.
   - Perform projection as easily as possible.

NOTES

- Compute one common expression.
- Combine the Cartesian product with a subsequent selection operation.
- Use associativity of binary operations to rearrange leaf nodes.

6. One advantage of A1 (linear search) algorithms is that they can be applied to any file.
7. Some of the complex selection predicates are as follows:
    - Disjunction
    - Conjunction
    - Negation

1.8 SUMMARY

- Query processing is the procedure of selecting the best strategic plan to be used in response to a database request. A response is then generated by executing the query. The component of the DBMS (Database Management System) responsible for generating this strategy is called a query processor.
- Query processing is also referred to in database literature as query optimization. However, optimization here is mostly in the form of improvement of status of database in light of the inexact knowledge.
- The parser portion of query processor checks the syntax and verifies the relations.
- Scanner identifies the language, such as, SQL, keywords, relation names and attributes names in text of the query.
- The UNION, as defined in relational algebra, is the set union of two relations. UNION returns all the distinct rows returned by either of the queries it is applied to. Since a mathematical set contains no duplicates, the set union contains no duplicates, and any row appearing in both results must appear but once in the result.
- The UNION ALL operator merges the result sets of two component queries.

  This operation returns rows retrieved by either of the component queries without eliminating duplicates.
- The INTERSECT, as defined in relational algebra, is the set intersection of two relations. It returns only those rows returned by both component queries. It is the logical product of two sets. Since a mathematical set contains no duplicates, the set intersection contains no duplicates, and any element appearing in both the sets must appear but once in the result.
- MINUS, as defined in relational algebra, is the set difference of two relations. It returns only rows returned by the first component query but not by the second. It is the difference between two sets. Since a mathematical set contains no duplicates, the set difference contains no duplicates. Since the MINUS operation eliminates duplicates, DISTINCT, while syntactically correct, is not necessary.

- In SQL, the division operation is not directly supported. It is, however, defined as a component of RELATIONAL ALGEBRA. According to Date, 'DIVIDE takes two relations, one binary and one unary, and builds a relation consisting of all values of one attribute of the binary relation that match (in the other attribute) all values in the unary relation.
- Pipelining is a method of decomposing a sequential process into suboperations. Every subprocess is executed in a devoted segment, which operates parallely with all other segments.
- The basic heuristic rule is to perform SELECT and PROJECT operations on the file before subjecting it to JOIN operations.
- In general, many different query trees or, in other words, many different relational algebra expressions can be equivalent; it means that they can correspond to the same query.
- An execution plan for a relational algebra expression represented as a query tree includes the algorithms to be used in computing the relational operators represented in the tree as well as the information about the access methods available for each relation.
- The linear search algorithms can be applied to any file.

1.9 KEY TERMS

- **Query Processing:** Query processing is the procedure of selecting the best strategic plan to be used in response to a database request. A response is then generated by executing the query. The component of the DBMS (Database Management System) responsible for generating this strategy is called a query processor.
- **UNION ALL:** The UNION ALL operator merges the result sets of two component queries. This operation returns rows retrieved by either of the component queries without eliminating duplicates.
- **Pipelining:** Pipelining is a method of decomposing a sequential process into suboperations.

1.10 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is query processing?
2. List the set operations supported by Oracle SQL.
3. Write the satisfied conditions of pipelining.
4. Write the basic rule of heuristic.

Long-Answer Questions

1. Explain the query processing strategy with the help of appropriate examples.
2. Explain the set operators in SQL by giving example programs.
3. Discuss the pipeline processing by giving appropriate examples.
4. Describe the heuristic optimization of query trees by giving example programs.
5. Describe the cost functions of join operation.

1.11 FURTHER READING

Navathe, S.B. and R. Elmasri. 1997. *Database System Concepts*, Third edition. New York: McGraw-Hill.

Korth, Henry F., Avi Silberschatz and S. Sudarshan. 2010. *Database System Concepts*, 6th edition. New York: McGraw-Hill.

Elmasri, Ramez and Shamkant B. Navathe. 2007. *Fundamentals of Database Systems*, 5th edition. New Jersey: Pearson Education.

Martin, James. 2007. *Principles of Data Base Management*. New Jersey:

Pearson Education.

Martin, James. 1975. *Computer Data-Base Organization*. New Jersey: Prentice Hall.

Jackson, Glenn A. 1988. *Relational Database Design With Microcomputer Applications*. New Jersey: Prentice Hall.

Date, C.J. 2000. *An Introduction to Database System*, Seventh edition. Boston: Addison Wesley.

UNIT 2 DATABASE TUNING AND EXTENDED RELATIONAL MODEL

Structure

2.0 INTRODUCTION

DataBase Administration (DBA) installs, configures, troubleshoots and maintains a database system and also separates the function of resource management from database services and technologies for managing data. Data integration means combining data from various sources and providing a single unified view of such data to users. It plays a vital role when two alike organizations require to combine their databases or merge research outcomes from different research organizations.

SQL is a language that facilitates interaction with relational database management systems. In 1979, ORACLE, an SQL product, was released. Today, SQL has become an important relational database management system. SQL does not depend on the underlying database structure and many different versions of SQL exist. However, it is the current industry standard query language for organizing, managing and retrieving data/information from databases. It is not just a query language for retrieving data but is also used for managing all the tasks of

DBMS, including data definitions, data manipulation, access control and data sharing.

Relational algebras received attention after relational model of database was published in 1970, by Codd, who proposed this algebra as a foundation for database query languages. Relational algebra, is widely used in, and now, is a part of computer science. It is based on algebra of sets and is an extension of first-order logic. It is concerned with a set of relations, closed under operators, which operate on one or more relations, yielding another relation.

In this unit, you will study about the database tuning, location of bottlenecks, tunable parameters and relational model concepts, referential integrity constraints, Structured Query Language (SQL), querying data with multiple conditions and basic relation algebra operations.

2.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basics of database tuning
- Discuss the location of bottlenecks
- Describe the significance of tunable parameters
- Explain the concepts of relational model
- Describe the referential integrity constraints
- Explain the Structured Query Language (SQL)
- Discuss the querying data with multiple conditions
- Explain the basic relation algebra operations

2.2 DATABASE ADMINISTRATION AND DATABASE TUNING

DataBase Administration (DBA) installs, configures, troubleshoots and maintains a database system and also separates the function of resource management from database services and technologies, for managing data. The database administrator is responsible for monitoring and optimizing the performance of a system through disk optimization, index tuning, etc. It is also responsible for understanding the complete scenario of business lexicon where it is translated into a logical data model. DBA is administered to analyse the design phase that collects data for the design, development, testing and operational phases. DBA sets user privileges in the database environment, which affects two areas — Internet-enabled databases and storage of procedural logic in the DBMS — which are incorporated with user-defined functions and stored procedures. A comprehensive approach is used with database administration which helps the system programmers to develop a DBA interface that explains its functioning. Sometimes, the DBA functions, such as planning, coordinating and implementing the security measures to safeguard the computer databases, are used by the system programmers when more than one

DBMS product is used. In such a case, the DBA performs completes the following tasks:

- Identifies and catalogs the required data for business users
- Produces conceptual and logical data models to depict accurately the relationship among various business processes of the organization
- Sets data policies for the organization
- Identifies data owners, such as administrative staff and local users
- Sets standards for central use of data
- Focusses on the DBMS and physical databases
- Opposes metadata but deals with data

For example, a value represents number 5. But, without metadata, it cannot be specifically defined. It can be analysed as follows:

- It can represent a month day, the 5th month of the year.
- It can be the 5th day of any month.
- It can be an age of 5 years.
- It can be the number of books.

Analysing with metadata, value 5 can lead to the following questions or queries:

- Is 5 a smaller number or larger number than any other number?
- What is its domain?
- What is its data type?
- Is it an integer or a decimal number?
- Is it the mantle of modular and data planner?

These queries can be solved by DBA. DBA is incorporated to create an appropriate physical database that is transformed by a logical data model. It is prepared to recover data in a reusable point. Basically, DBA is involved with the actual implementation of databases. It can also modify complex databases. In the database world, DBA is responsible for the following complex tasks:

(*i*) Designing a Physical and Conceptual Schema

DBA interacts with system users to understand what type of data is to be used. It designs the conceptual schema to determine the type of relationships, such as 1:1, 1: M, M:1 or M:M, which will be maintained between various tables and attributes in databases. The physical schema determines how databases will be stored. Basically, it is used for external schema which is created by additional views.

(*ii*) Tuning a Database

The DBA is responsible for modifying the database to ensure and register adequate performances as per the changing requirement. It can alter data to be imported from Sequential Query Language (SQL) and Comma Separated Values (CSV) and also to export data to various formats, such as SQL, CSV, Xtensible Markup Language (XML), Portable Document Format (PDF), Word and Excel. It is featured to successfully tune with multiple database servers. Creating PDF graphics to database layout, transforming stored data using a set of predefined functions, such as Binary Large OBjects (BLOB) data or download link, are maintained by DBA.

Making data available to and recovering data users.

(*iii*) **Data Availability and Recovery for Users**

In case of any system failure, the user will not be disturbed and can continue to work with uncorrupted data because DBA provides the restored data as a consistent state. Implementing the procedures to take the backup of data at regular intervals is possible through a DBA because it maintains the log of the system activities, which facilitates recovery from system crisis.

(*iv*) **Working with Database Models**

DBA transforms a logical data model into a physical database design and incorporates the knowledge of DBMS that creates an appropriate and efficient physical database design from a logical model.

(*v*) **Performing Implementation and Operations for DBMS**

The systems programming role creates an impact on DBMS implementations and operations. Sometimes, DBA plays a proxy role of system administration that is responsible for the installation and setup of DBMS. Enabling useful databases for applications and clients and designing optimal databases to administer and tune the databases are the activities performed by DBA. DBA is required to design the database so that it can be maintained with minimal disruptions.

(*vi*) **Enabling System Locking and Log Files**

Locking the system; maintaining log errors, such as update log, binary update log, slow queries log; and monitoring startup messages and files are activities done by the DBA. Sometimes, username and timestamp are logged for queries. It supports and helps in repair works if the data file is changed during recovery. DBA changes the user file *tbl_file.MYD* as *tbl_file-datetime.bak*, if the backup file is required by the system.

(*vii*) **Security and Authorization**

Access of unauthorized data can also be prevented by DBA. In DBMS, users are granted permission to access the data which maintains certain views and relations. For example, for an online air ticket booking system, a traveller is allowed to find the details of his/her reservation and updates. However, if any user wants to see the details of others, then it is not possible because viewing important credentials and information is restricted by the database administration side. The DBA enforces this policy by granting travellers the permission to view in the read only mode, the traveller's information. The Oracle database administrator can be taken as an example. All the above functions of DBA can be performed with reference to Oracle database system whose functions are as follows:

- It is responsible to install and upgrade the application and server tools of Oracle.

- It allocates the system planning and storage management to the database systems.
- It creates primary storage space, known as tablespaces.
- It creates primary objects, such as tables, views and indexes, if application developers design an application.
- It modifies the database structure which is specified by application developers.
- It enrolls users to maintain the system security.
- It controls and monitors user access to databases.
- It plays an important role for backup and recovery database information.
- It also maintains archived data on the tape.
- It takes the backs up and restores the database.
- It monitors to optimize the performance of databases.
- It contacts Oracle Corporation to take technical support.

2.2.1 Tuning Operations

A system, on installation, is expected to perform well. Considering many aspects of operations on the system, some tuning has to be established with the system and its users. There are many features available with the system and a normal user is not aware of all these. He or she is mostly concerned with jobs that are done repeatedly. Depending on the nature of work of an employee, the level of work is also decided. For this reason, all do not have the same privilege of use. An RDBMS user, normally, uses queries using SQL to retrieve data/information from the system. Database administrator does his best to keep the database ready for use, at all times. Hence, the idea is to make most efficient use of the system. This requires tuning.

Tuning Considerations

Performance problems result from poorly written SQL statements. This can be rectified by adopting a better practice for writing statements in SQL. But, this is not the only factor related to performance. Role of the Database Administrator (DBA) is vital in tuning the operation. Other things that can be done to improve performance and tune it are stated as follows:

Minimize Disk Reads and Writes

Every computer user, developer in particular, is aware that I/O (input/output) operations are slowest on most computer systems. This relates to operations such as reading data from the storage system (hard disk, memory chips, CD-ROM, pen-drives, etc.) and writing data to these storage devices. Other operations such as movement of data in system memory and carrying out different types of calculations on data and the process of transformations are much faster than these I/O operations. So, keeping input/output at the minimum, is one aspect of tuning. There are some ways to achieve it. These are:

- **Allocation of buffers of correct size:** Buffer is an area of memory that holds data, recently read from or is required for writing to storage system.

When the proper size of the buffer is used, recently read data stays in the memory for some time, with a probability of receiving a new query. If the new query needs data, it is already available in a buffer. This makes retrieval fast. In case of output, if there is buffer space, RDBMS writes the data in buffer and then copies the buffer from the memory to the storage system for final storage later. This fundamental feature of RDBMS is known as asynchronous I/O.

- **Spreading of disk I/O:** These days, systems have multiple disk drives. Spreading database file to other disk drives allow parallel I/O operation. If the same drive is partitioned as more drives, these are not considered as different drives. A disk drive accesses only one spot on the drive at a time, and multiple I/O operations on a single drive are handled serially. If multiple I/O operations are spread to other drives, the operation can be made faster.

Tuning SQL Queries

By writing better SQL statements, more than 80 percent of database query performance can be improved. Putting SQL statements inside stored procedures yields good improvement in performance in Microsoft SQL Server and Sybase. But it may not be true for every database. Oracle is one such database.

Some ways to ensure a good query design:

- **Knowledge of data:** Before writing statement in SQL, know about the number of rows in each table. Also, know about the selectivity of WHERE predicates and number of rows that is expected in the result set. If rows are more, thinking time for writing SQL statement is more.
- **Do not scan large tables:** If a table has more than 1000 rows scanning of all rows will consume more resources and hence, expensive. Using an index is better in such cases.
- **Avoid unnecessary columns:** Intermediate operations take more time if data in each row becomes wider in the result set. More disk space and memory are required for intermediate operations.
- **Avoid unnecessary tables:** Keep table at optimum level. Fewer the tables, more efficient will be the query.
- **Avoid sorting on large result sets:** Sorting consumes resources and hence, is expensive. There may be cases when rows under sorting, do not fit the memory. Most optimizers use index to avoid sorting operation, but its use should be judged by a trade-off since overheads are required to maintain the index.
- **Matching data types in predicates:** When a comparison is done by predicate in two column values, or a column value and a literal in altering rows, data types should match. In case data types do not match, it should be converted to one of them before comparing.
- **Use IN instead of OR when possible:** IN operator can be rewritten as a JOIN. OR operator needs running of multiple queries with the results combined by the DBMS. IN operator is far more efficient.

Indexes improve data access time. But these consume storage space that needs to be maintained. Use of index for improving performance of query statements is being given below:

- **Avoid indexes on frequently updated columns:** When column data is updated, indexes containing that column should also be updated. So, creation of an index on a column that is frequently updated, demands more write operations.
- **Create selective indexes:** Index selectivity is a term, defined as the ratio of the number of distinct values in a column per row in a table. For example, if there is a table having 500 rows and a column containing 400 distinct values, the index selectivity is 0.8. This value is considered good. Columns like gender having either of the two distinct values (M and F) has very poor selectivity (.002 in this case). An index which is unique has a selectivity ratio of 1.0. This is the best possible. As a thumb rule, avoid indexes having selectivity less than 0.33. But there are indexes especially designed with low index selectivity. This is done is setting bitmap indexes.
- **Foreign key indexes improve joins:** Putting an index on a foreign key column improves performance of join operation to a great extent. Additional join methods enable the use by an optimizer.
- **Avoid Over-indexing:** Use of more than three or four indexes for any table, is not advised. Too many indexes on a table badly affect operation of INSERT or UPDATE, issued against that table.

Like these, there are many others things related to the use of system that needs considerations. A good DBA is required to fine tune all requirements for users. As system expands with use, DBA has to constantly work for maintaining the system in perfect running condition with no downtime, except for planned housekeeping.

2.2.2 Location of Bottlenecks

In a data communications context, the bottleneck is defined as a point where the flow of data is either impaired or stopped completely. Essentially, a bottleneck results when there is not enough data handling capacity for accommodating the current volume of data traffic.

The term ‘Bottleneck’ in ‘SQL Server’ refers to the congestion of data traffic on the data routing route, this congestion reduces the flow of data causing the bottleneck at the SQL Server. Alternatively, the SQL Server bottleneck means reduction in the performance of SQL Server. This situation generally happens when any shared database resources, such as SQL database is concurrently accessed by too many people. When any SQL Server is bottlenecked then the SQL database users may attempt to use another SQL Server in order to access the required database which possibly will slow down the SQL Server performance experiencing the effects of one or more SQL Server bottlenecks. The SQL Server bottlenecks happen for various reasons, but it basically happens as a result of problems with memory, I/O (Input/Output) or CPU (Central Processing Unit).

However, every time it is not easy to determine whether the server performance problems is from a SQL Server bottleneck or from any another source, hence the database administrator or database user can look for the following common indications/signs of bottlenecks to narrow the search for the source of the issue.

Common Indications/Signs of SQL Server Bottlenecks

- SQL Server Hogging the Processor
- Longer Execution Times on Queries
- Excessive I/O
- Application Log showing Out-of-Memory Messages
- Extreme Activity on the Disks
- Long Wait Times Per I/O

The sudden appearance of one or more of the above mentioned indications/ signs is a good signal that there is a SQL Server bottleneck somewhere in the computer system, which can be identified and rectified.

Types of SQL Server Bottlenecks

There are three main types of SQL Server bottlenecks, namely the memory, I/O, and CPU. It is essential and significant for DBAs (Database Administrators) to be familiar with the causes and symptoms of the SQL Server bottlenecks, and to fix each of the issues so that they can identify and remove the bottlenecks quickly in order to minimize the impact of reduced performance. Following are some of the best performance counters to monitor that will help identify performance bottlenecks without any delay.

Memory Bottlenecks: Memory bottlenecks are generally a result of insufficient memory resources or SQL Server activities consuming up available memory. The indications/signs that must be paid attention includes longer query execution times, excessive I/O, out-of-memory messages in the application log, and frequent system crashes.

The best techniques to avoid memory-related bottlenecks are to use a query optimizer to discard unnecessary or obsolete queries and to configure page life expectancy in conjunction with buffer cache hit ratio to limit visits to the data disk. When it is too late to avoid the bottleneck, then try reviewing and tuning queries, reconfigure memory, or add physical memory.

I/O Bottlenecks: When there is not enough storage available on the disk to support regular database operations, such as TempDB (Temporary DataBase), then I/O bottlenecks occur. Long response times, application slowdowns, and frequent task time-outs are all key indicators/signs that can cause an I/O bottleneck. The I/O bottleneck type can be avoided by setting up monitoring alarms and threshold alerts to identify which activities use excessive amounts of storage. When an I/O bottleneck occur, then the database user must limit reading and writing of database pages to and from the disk.

CPU Bottlenecks: The prominent cause of CPU bottlenecks is insufficient hardware resources. It is comparatively easy to identify the CPU bottlenecks by checking the data logs to determine whether SQL Server is using excessive CPU time. The CPU bottlenecks could be avoided by having a dedicated SQL Server, i.e., the server with only SQL commands if possible.

To solve CPU bottlenecks, the first step is to identify the CPU hogs. Once the problem area is identified, the queries can be tuned, execution plans can be improved, or the system can be reconfigured.

2.2.3 Tunable Parameters

Database tuning describes a group of activities used to optimize and homogenize the performance of a database. It usually overlaps with query tuning, but refers to design of the database files, selection of the DataBase Management System (DBMS) application, and configuration of the database's environment, such as Operating System (OS), CPU, etc.

Database tuning aims to maximize use of system resources to perform the task efficiently and quickly.

DBMS tuning refers to tuning of the DBMS and the configuration of the memory and processing resources of the computer running the DBMS. This is typically done through configuring the DBMS, but the resources involved are shared with the host system.

Following are the tunable parameters and methods to optimize performance.

1. shared_buffer
2. wal_buffers
3. effective_cache_size
4. work_mem
5. synchronous_commit
6. max_connections
7. max_prepared_transactions
8. max_worker_processes
9. max_parallel_workers_per_gather
10. max_parallel_workers
11. effective_io_concurrency
12. random_page_cost
13. min_ and max_wal_size
14. max_fsm_pages

The SQL is considered as a very versatile database system, capable of running efficiently in low-resource environments and also in the environments shared with various other applications. In terms of databases, following are the two different specifications of 'Fast or Quick Data Processing':

1. Number of Transactions Per Second
2. Throughput or Amount of Data Processing

Even though these specifications are interrelated but are not the same. Both have completely different requirements in terms of I/O operations. Each database engine has a specific memory layout and handles different memory areas for different purposes.

All of the configuration parameters can be edited in the 'sql.conf' database configuration file. This is a regular text file and can be edited using Notepad or any other text editor. The changes will occur when the server is restarted.

Microsoft SQL Server

The following guidelines pertain to performance tuning parameters for Microsoft SQL Server databases which are documented the Microsoft SQL Server documentation.

- Store tempDB on a fast I/O device.
- Increase the recovery interval if perfmon shows an increase in I/O.
- Use an I/O block size larger than 2 KB.

2.3 EXTENDED RELATIONAL MODEL

The Entity-Relationship or E-R analysis is beneficial for a database designer in various ways. The following are the features of the E-R analysis:

- The constructs used in the E-R model can easily be transformed into relational tables.
- The E-R model is simple and easy to understand and is, therefore, used by a database designer to communicate a database design to users accessing the database.
- It is used as a design plan by a database developer to implement a data model in a DBMS.

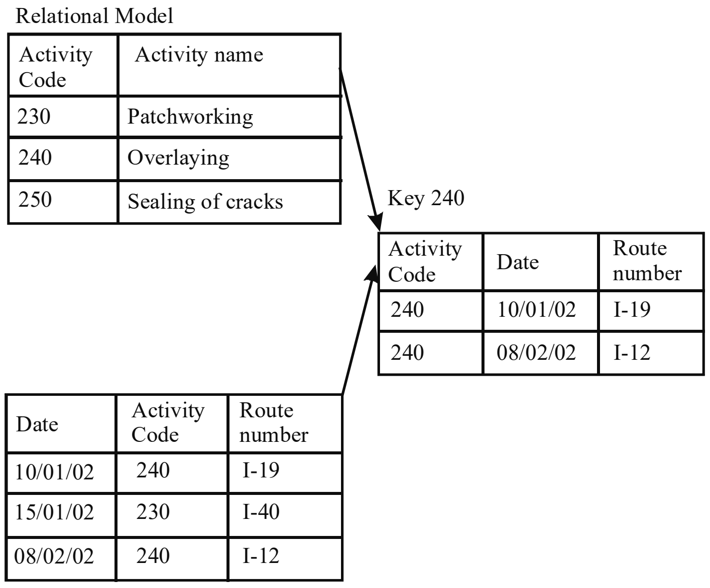
2.3.1 Relational Model Concepts

Relational model of a database is based on set theory of mathematics. Relation is a central concept in relational model that is borrowed from the concept of set theory. Such concepts are widely applied in designing relational model of a database. Viewing from the application side, this model is based on first-order predicate logic. E.F. Codd first proposed the concept of this model in the year 1969.

Access to data in relational model is made via relations. Relations storing data are known as base relations; 'table' is an implementation of this base relation. There are other relations too that are derived using operations such as selection, projection, union, intersection etc., and hence, they are known as 'derived relations'.

The following Figure shows a relation (table) derived from two tables. First table shows few activities coded for easy structuring of work. Every activity has few routes which started on some specific date. This is shown in another table, showing date, activity code and route number. Here, overlaying is done by following two routes I – 19 and I – 12. Third table is a result of query that displays the required route number, activity code and the date.

NOTES



Activity Code	Activity name
230	Patchworking
240	Overlaying
250	Sealing of cracks

Activity Code	Date	Route number
240	10/01/02	I-19
240	08/02/02	I-12

Date	Activity Code	Route number
10/01/02	240	I-19
15/01/02	230	I-40
08/02/02	240	I-12

Fig. 2.1 *Relational Model*

Thus, relational model describes database as a set of predicates having a finite set of variables, known as predicate variables. Model also describes constraints for possible values of their combinations. Database content is thought to be as a logical model of the database, containing a set of relations, with one relation for every predicate variable, satisfying all predicates. Queries from such a database are made as predicates and hence, request for information is also a predicate.

The two principal rules for the relational model are known as entity integrity and referential integrity.

The Model

In a relational model, data is presented by mathematical n-ary relation that is a subset of the Cartesian product that has n number of domains. While dealing with a mathematical model, such data is analysed with two-valued predicate logic in which any proposition has only two possible outcomes, either *true* or *false.* There is no place for a third outcome such as 'not applicable', or 'not available'. Some favour use of such a two-valued logic for relational model, whereas others favour three-valued logic and this will be still called relational. In such a model, relational algebra or relational calculus is used for operation on data.

The relational model is consistent, which is achieved by using constraints while designing database. This design is also known as logical schema.

Concept of a relational model is shown by the following figure. Attributes in a relational model are put as columns of a table in this model. Name of relation is shown as R, but it can assume any name according to the context. For example, if we are dealing with details of employees, we may write the relation name as Emp or EMP, as may appeal to one who designs the database. Row is known as tuple. A relation is stored as a table. Attributes may be in any order in the table. A set of attribute is heading entries which are put under rows and columns for the table body. An attribute must have some specific values.

NOTES

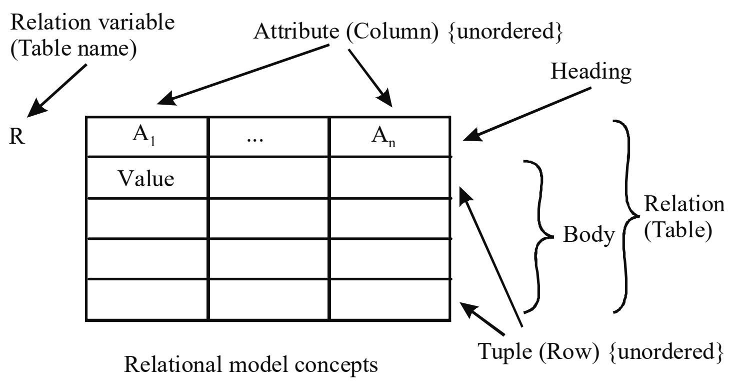

Relation variable (Table name)
Attribute (Column) {unordered}
Heading
R
A1
...
An
Value
Body
Relation (Table)
Tuple (Row) {unordered}
Relational model concepts

Fig. 2.2 *Relational Model Concepts*

Thus, a **relation** has a tabular structure with definition for every column and data put in this structure. The structure is defined by the **heading** and the data is entered in the **body** containing a set of rows. In a database, a **relvar** stands for a named variable of a specific relation type in which some relation of that type has been assigned. A database **relvar** that stands for relational variables is called a **base table**. **Update operators** namely, INSERT, DELETE or UPDATE are used to make changes in the database entry. To retrieve data some queries are made using expression according to definitions of the operators. In making queries using Structural Query Language (SQL), heading may not always be taken as a set of column definitions always. This is so since a column may not have any name in certain cases and also, the same name may appear in two or more columns. Also, body may not always be a set of rows since the same row may appear in the same body more than once.

SQL and the Relational Model

SQL is the standard language for making queries in relational databases but it deviates from this model in many cases. The present standard of ISO SQL has no mention of terms or concepts used in the relational model. Even then, a database can be created that conforms to a relational model by use of SQL if certain features of SQL are not used.

Deviations in SQL from Relational Model

Deviations of SQL from relational model have been found which are mentioned below. There are some database servers that implement the entire SQL standard and do not permit some of these deviations. NULL is generally allowed but duplicate column names or anonymous columns is not common in a table.

Duplicate rows: In an SQL table**,** a row may appear more than once, but the same tuple is not allowed to occur more than once in a relation.

Anonymous columns: In a SQL table, there may be an unnamed column that can not be referenced in expressions. But relational model has every attribute, named as well as referenceable.

Duplicate column names: In a SQL table, columns may have the same name and due to ambiguity, it cannot be referenced. But every attribute of a relational model is referenceable.

NOTES

Column order significance: In a SQL table, column-order is significant and defined. Also in SQL, union and Cartesian product are not commutative. The relational model has no compulsion on ordering of the attributes in a relation. It can be unordered.

Views without CHECK OPTION: Views in DBMS can be created without using CHECK OPTION key work. The view can be updated without effecting the large database view. For example, if an INSERT has been made and accepted, it is not essential that the row, thus inserted, will make its appearance in the view. It may not appear at all. On the other hand, in case of relational model updates to a view will have the same effect as if it (the view) was a base relvar.

NULL: In SQL table, NULL can appear in place of a column value in some rows. This deviation comes due to the fact that this ad hoc concept is implemented in SQL using three-valued logic where comparison of NULL with itself does not result in *true* but gives a third truth value as *unknown*. In a similar way, comparing NULL with a value other than itself will not give *false* but will show *unknown*. For this reason NULL is taken as a mark instead of a value. But in a relational model, anything not true is taken as false and vice versa. This requires that every tuple in a relation body must have a value for each attribute in that relation. Such a deviation has some dispute.

Relational Operations

Data or information is retrieved from a relational database by means of a query, written in some type of query language, which is mostly some dialect of SQL. SQL queries are embedded in a software providing a user friendly interface. There are web sites that perform SQL queries while generating pages. Wikipedia is one such example. When a query is made, a set of results is returned containing a list of rows with the required information. A most simple query returns all rows of a table, but in actual practice rows are filtered in a way that narrows down the search and returns only those which are queried.

As a result of a query, data is usually extracted from more than one table and is merged into one and this operation is known as 'join'. A join operation takes every possible combination of rows as Cartesian product, and then filters out other things to narrow down the search criteria. In actual implementation, a query optimizer rewrites to perform faster.

Relational databases are flexible and queries can be written by programmers that were earlier unanticipated by designers of the database. This makes relational databases capable of multiple applications, not foreseen by the original designers.

Database Normalization

Classification of relations is made on the basis of anomalies in the database relations. Relational database do not allow data redundancy which is an anomaly usually found in databases. A database in the first normal form has many anomalies. There are many normal forms of relations such as first normal form, second normal form, third normal form, Boyce-Codd normal form, fourth normal form and fifth normal form. These are usually denoted as 1NF, 2NF…. BCNF and 5NF. 1NF is the lowest level. The process of normalization converts a relation form lower form to higher form.

Database

We can take some simple examples of how relations are created in a database design. Let's say we give some relvars (relation variables) with attributes. We are using six relvars Customer, Oder, Order Line, Invoice, Invoice Line and Product. Attributes of these relations and their relationships form the design of a relational model. The design has relation having attributes as mentioned below:

- Customer (**<u>Customer ID</u>**, Sales Tax ID, Name, Address, City, State, PIN, Phone)
- Order (**<u>Order No</u>**, <u>Customer ID</u>, <u>Invoice No</u>, Date Placed, Date Promised, Terms, Status)
- Order Line (**<u>Order No</u>**, **<u>Order Line No</u>**, <u>Product Code</u>, Qty)
- Invoice (**<u>Invoice No</u>**, <u>Customer ID</u>, <u>Order No</u>, Date, Status)
- Invoice Line (**<u>Invoice No</u>**, **<u>Invoice Line No</u>**, <u>Product Code</u>, Qty Shipped)
- Product (**<u>Product Code</u>**, Product Description)

Attributes that are in bold as well as underlined are known as *candidate keys*. Attributes that are underlined but not bold are known as *foreign keys*.

From amongst candidate keys, one is arbitrarily chosen as primary key and has preference over candidate keys and these other candidate keys are known as alternate keys. A candidate key serves the purpose of a unique identifier that enforces 'no duplication' of any tuple. Superkeys that also include candidate keys and foreign keys can be composed of many attributes. They are thus, called composite keys.

Foreign keys show integrity constraints that enforces that value of the attribute set has been taken from a candidate key in another relation. For example, we take the relation Order and **<u>Customer ID</u>** as foreign key. A join operation draws information from many relations at once. Query can be made by joining relvars in the example given above from database for all; Customers, Orders, and Invoices.

2.4 REFERENTIAL INTEGRITY CONSTRAINTS

Let us start with constraints.

Constraints

Constraints define a condition, which needs to be satisfied while storing data in a database. DBMS allows you to define and implement the constraints for a database object. For example, you can specify a constraint that each field in the employee_id column of the Employee table must contain a unique value. The database designers need to identify the constraints during database design. The various types of constraint are:

- Referential integrity
- Entity integrity

Referential Integrity

Referential integrity ensures that the relationship between tables remain preserved when you insert, delete, and modify data. In SQL Server 2000, referential integrity is based on relationship between foreign keys and primary keys or between foreign keys and unique keys. Referential integrity ensures that key values are consistent across the related tables. When referential integrity is enforced, SQL Server 2000 prevents users from adding records to a related table if there is no associated record in the primary table. Users are also prevented from changing values in a primary table or deleting records from the primary table if there are related records in the related table.

Foreign Key constraint prevents conditions that violate any reference between the two database tables. A foreign key value refers to another table with the corresponding values of primary key in a database table. Consider the following example where you need to create a database table with a foreign key constraint.

```
CREATE TABLE Class
(Student_id int,
Course_name varchar(10),
Age int,
CONSTRAINT fk_student
FOREIGN KEY (Student_id )
REFERENCES Student (STUD_ID))
```

The `CREATE` statement creates a Class database table with various columns such as `Student_id,Course_name` and `Age`. The `Student_id` defines a foreign key for the Class database table and `STUD_ID` is the primary key for the Student database table. The `Student_id` refers to the `STUD_ID` primary key of Student table.

Entity Integrity

Entity Integrity is a mechanism that allows uniquely identified rows in a table. This is done with either primary keys or unique keys that will prevent duplicate rows. You can apply entity integrity to a table by specifying a `PRIMARY KEY` constraint. A primary key helps retrieve data records uniquely from a database table. Each table must have a primary key constraint to uniquely identify each row in the database. There can be only one primary key constraint for each table. Consider the following example where you need to create a database table with a primary key constraint.

```
CREATE TABLE Student
(STUD_ID int NOT NULL PRIMARY KEY,
STUD_Name varchar (15) NOT NULL,
PhoneNo int NULL)
```

The `CREATE` statement creates a Student database table with various columns such as `STUD_ID, STUD_Name` and `PhoneNo`. The `NOT NULL PRIMARY KEY` constraint is defined for the `STUD_ID` that should not have a null value and should be unique to retrieve the data records from the Student database table.

Check Your Progress

1. What tasks are performed by DBA?
2. Define the term bottlenecks.
3. What are base relations and derived relations?
4. Define the term constraints.

2.5 STRUCTURED QUERY LANGUAGE (SQL)

SQL, which stands for Structured Query Language, is an ANSI (American National Standards Institute) standard and has many different versions. It is also referred to as sequel in common language and pronounced so. SQL is used for interacting with RDBMS (Relational Database Management System). RDBMS is the basis for SQL and also for all other contemporary database systems like MS SQL Server, IBM DB2, Oracle, MySQL and Microsoft Access. The prototype for SQL was originally developed by IBM, based on E.F. Codd's paper 'A Relational Model of Data for Large Shared Data Banks'. In 1979, ORACLE, an SQL product was released. Today, SQL has become a very important relational database management system. Its scope includes data query and update, schema creation and modification, and data access control.

SQL does not depend on the underlying database structure and many different versions of SQL exist. However, it is the current industry standard query language for organizing, managing and retrieving data/information from databases. It is not just a query language. It is not only used for retrieving data but also for managing all the tasks of DataBase Management Systems (DBMS) including:

- Data definitions
- Data manipulation
- Access control
- Data sharing

Despite the name, SQL does not provide constructs of languages like C or Pascal. It lacks branching (IF) and looping (GOTO, DO and FOR) statements. Some database vendors like ORACLE give extensions to SQL to achieve branching, looping, etc. Oracle utilizes a version of SQL known as PL/SQL. However, they do not form part of the SQL standard. SQL can, however, be incorporated into other programming languages known as Embedded SQL.

SQL has many advantages as you will read in the following paragraphs.

2.5.1 Advantages of SQL

The main advantages of SQL are:

- **Vendor portability/independence:** Generally, SQL permits the user to change the given brand of database and DBMS without rewriting the SQL code.

- **Ease of use:** SQL is a 'declarative language (non-procedural). Thus, as compared to other programming languages, SQL is easy to use. High-level programming languages like C, COBOL, PASCAL, etc., require the programmer to convert the program logic into code and use the data structures that hold the data. In SQL, the programmer only indicates *what* data is required but is not required to indicate *how* to get it. The given DBMS examines the SQL and plans the way to get the information needed.
- **Language for all users**: SQL could be applied by all types of users like DBAs, application programmmers, naïve users and others.
- **Standardized:** The SQL standard is established by American National Standards Institute (ANSI—an organization that approves certain standards in many different industries). Many vendors now use the latest SQL 99 version. In 1987, the International Standards Organization (ISO) recognized the ANSI SQL standard as the international standard. The standard was revised again in 1992 and was called SQL 92. The newest standard is now called SQL 99; it is also referred to as SQL 3.
- **Dynamic data definition:** For embedded and multi-level database queries, SQL gives advanced data processing commands.

SQL has become greatly acceptable, with many commercial relational DBMS products, such as ORACLE, DB2, INGRES, SYBASE and others, due to these factors.

2.5.2 Forms of SQL

There are two forms of SQL:

1. Interactive
2. Embedded

Interactive SQL operates on a database to produce output for user demand. In *embedded SQL*, SQL commands can be put inside a program written in some other language (called host language) like C, C++, etc. Data is passed to a program environment through SQL. The combined source code is accepted by a special SQL precompiler and, along with other tools, it is converted into an executable program.

2.5.3 Types of SQL Commands

SQL commands are of varied types to suit different purposes.

The five primary types are as follows:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)
- Data Control Language (DCL)
- Transactional control language (TCL)

(i) **Data Definition Language (DDL)**

DDL is a part of SQL that allows a database user to create and restructure database objects, such as the creation or deletion of a table. Some of the most fundamental DDL commands include the following:

```
CREATE
ALTER
DROP
```

(ii) **Data Manipulation Language (DML)**

DML is that part of SQL which is used to manipulate data within the objects of a relational database. There are three basic DML commands:

- `INSERT`
- `UPDATE`
- `DELETE`

(iii) **Data Query Language (DQL)**

Though comprising only one command, DQL is the most concentrated focus of SQL for modern relational database users. The base command is:

```
SELECT
```

This command, accompanied by many options and clauses, is used to compose queries against a relational database. Queries, from simple to complex, from vague to specific, can be easily created. A *query* is an inquiry to the database for information. A query is usually issued to the database through an application interface or via a command line prompt.

It is to be noted that, according to some authors, the SELECT command may be treated as DML as a query is nothing but a part of DML that is used for retrieval of data from a database.

(iv) **Data Control Language (DCL)**

Data control commands in SQL allow us to control access to data within the database. These DCL commands are normally used to create objects related to user access and also control the distribution of privileges among users. Some data control commands are as follows:

```
GRANT
REVOKE
```

(v) **Transactional Control Languages (TCL)**

In addition to the previously introduced categories of commands, the following commands allow the user to manage database transactions:

```
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION
```

2.5.4 SQL Statements

Directions to write SQL statements

- SQL statements are insensitive to case.
- They can be of one or more lines.
- SQL keywords cannot be split across lines or abbreviated.
- They are generally entered in uppercase; all other words, such as column names and table are entered in lowercase. However, this is not a rule.
- SQL keywords are typically aligned in the first column.
- SQL statements are terminated with a semi-colon.

  Some more information about SQL and ORACLE SQL in particular are as follows:

  A NAME for a view, table, synonym, index, column, or user variable must:

  - Start with a letter
  - Consist of only the characters A–Z, 0–9, _, $, and #
  - Not be the same as an ORACLE reserved word
  - Be 1 to 30 characters long (Database names need not exceed 8 characters)
  - Not consist of a quotation mark

Notations used in the syntax of SQL commands

In this unit, lowercase is used to identify names or conditions entered by the user, and uppercase is used to identify SQL keywords.

- User-given data or object name or expression or condition is enclosed within a "<' and '>"
- Square brackets ("[]") identify optional items. Do not include the brackets when we enter a query.
- A vertical bar ("|") indicates a choice and underlining indicates a default.
- Ellipses ("...") are used to specify items that may repeat.

2.5.5 Data Definition Language (DDL)

DDL is the subset of SQL commands used to modify create or remove Oracle database objects, including tables. These commands may also be used to update information in the Data Dictionary accordingly.

SQL has three main commands for data definition:

- CREATE for creation of database objects
- ALTER for changing database objects
- DROP for deleting database objects

As the creation, alteration and deletion of databases are the tasks of database administration, you will learn only tables in this section. Before discussing the DDL commands, you will learn about data types supported by Oracle.

Different data types in Oracle

There are two types of data types : ANSI standard data types and Oracle-defined data types. Instead of using the ANSI standard data types, the Oracle-defined data types can be used. Table 2.1 gives the Oracle data type alternative for the ANSI standard data types:

Table 2.1 *Oracle Data Type Alternatives*

ANSI Standard Data Type	Oracle Data Type
CHARACTER and CHAR	CHAR
CHARACTER VARYING and CHAR VARYING	VARCHAR2
NUMERIC, DECIMAL, DEC, INTEGER, INT and SMALLINT	NUMBER
FLOAT, REAL and DOUBLE PRECISION	FLOAT

Difference between CHAR and VARCHAR2

In Oracle-based SQL, the CHAR and VARCHAR2 data types store fixed-length and variable-length character strings, respectively. All the string literals have data type CHAR. The representation of CHAR and VARCHAR2 data depends on the database character set which is specified using CHARACTER SET clause of the CREATE DATABASE statement. The following is the syntax for specifying a CHAR or VARCHAR2 data item:

```
[ CHAR | VARCHAR2 ] [( maximum_size [ CHAR | BYTE ] )]
```

For example:

```
CHAR
VARCHAR2
CHAR(10 CHAR)
VARCHAR2(32 BYTE)
```

The *maximum_size* of a CHAR or VARCHAR2 data item is always an integer literal in the range 1 to 32767. The default value is 1.

Following are the basic differences between CHAR and VARCHAR2 data types:

- **Predefined subtype of character data types:** The CHAR data type has one predefined subtype, CHARACTER, whereas the VARCHAR2 data type has two predefined subtypes, VARCHAR and STRING. Both the subtypes have exactly the similar range of values as its base type.
- **Memory allocation for character variables:** For variables CHAR and VARCHAR2 whose maximum size is less than 2,000 bytes, SQL allocates sufficient memory for the maximum size at the compilation time.
- **Blank-padding shorter character values:** In all of the following given conditions, SQL blank-pads the character value:
  - (i) The character value which is assigned to an SQL character variable is shorter than the maximum size of the variable.
  - (ii) The character value which is inserted into a character database column is shorter than the predefined width of the column.
  - (iii) The value which is retrieved from a character database column into a SQL character variable is shorter than the maximum length of the variable.

When the data type of the receiver is CHAR, then SQL blank-pads the value to the maximum size and hence the information is lost about the trailing blanks in the original value. In the following example, the value assigned to last_name has six trailing blanks:

```
   last_name CHAR(10) := 'CHEN '; — //note trailing blank
```

But when the data type of the receiver is VARCHAR2, then SQL neither blank-pads the value nor strips trailing blanks keeping the character values intact and hence no information is lost.

Maximum sizes of values inserted into character database columns: The largest CHAR value which can be inserted into a CHAR database column is 2,000 bytes, whereas for VARCHAR2 it is 4,000 bytes. Any CHAR or VARCHAR2 value can be inserted into a LONG database column, because the maximum width of a LONG column is 2,147,483,648 bytes (2 GB).

There are several built-in data types supported by Oracle (Table 2.2):

Table 2.2 *Built-in Data Types in Oracle*

Built-In Data type	**Description**
CHAR(size)	Fixed-length character data of length 'size' bytes. Default and minimum size is 1 byte and the maximum size is 2000 bytes,.
VARCHAR2(size)	Variable-length character string that has the maximum length 'size' bytes. The maximum size is 4000 and minimum is 1. Here, 'size' for VARCHAR2 must be specified. **Note:** Since the VARCHAR2 data type is the successor of VARCHAR, it is recommended to use VARCHAR2 as a variable-sized array of characters.
NUMBER(p, s)	Specifies integer as well as real numbers, having precision **p** and scale **s**. The precision p has a range from 1 to 38. The scale s has a range from 84 to 127. The default value for p in a number data type is 38. The default scale **s** is 0 for no decimal places. When s is positive, the number of decimal places in scale is increased and when it is negative, the actual data is rounded off to the specified number of places at the left of the decimal point. Example: 7456123.89 $\rightarrow$ Number (7, 2) $\rightarrow$7456100 A NUMBER data type with no parameters is set to its maximum size.
LONG	This denotes character data of variable length up to 2 GB. LONG columns in SQL statements can be referenced in the following places: • SELECT clause • SET clauses in UPDATE statements • VALUES clauses in INSERT statements There are some restrictions to the use of LONG values: • A table should not have more than one LONG column. • LONG columns cannot be in integrity constraints (exception: NULL and NOT NULL constraints). • LONG columns cannot be indexed. • A procedure or stored function should not accept a LONG argument. Also, LONG columns cannot be in certain parts of SQL statements, such as: • WHERE, GROUP BY, ORDER BY, or CONNECT BY clauses or with the DISTINCT operator in SELECT statements.

NOTES

	• SQL functions such as SUBSTR or INSTR. • expressions or conditions. • select lists of queries containing GROUP BY clauses. • select lists of subqueries or queries combined by set operators. • select lists of CREATE TABLE AS SELECT statements.
DATE	Valid date in ORACLE ranges from January 1, 4712 BC to December 31, 9999 AD. The default input date format is DD-MON-YY HH:MM:SS where DD is the Day of the month, MON is a three-letter abbreviation for the Month, YY is the two-digit representation of the Year. HH, MM and SS are two-digit representations of Hour, Minute and Seconds.
RAW(size)	Raw represents binary data of length 'size' bytes. The size for a RAW value must be specified and the maximum size is 2000 bytes.
LONG RAW	Long raw represents binary data of variable length up to 2 gigabytes.
ROWID	Hexadecimal string represents the unique address of a row in a table. This data type is mainly used for the values returned by the ROWID pseudo column. Pseudo column is like a column in a table, but is not actually stored in it. On pseudo column one can only select; delete, insert and update are not allowed.
CLOB	A character large object contains 1-byte characters. Both fixed-width and variable-width character sets are supported and both use the CHAR database character set. The maximum size is 4 GB.
BLOB	This represents a binary large object having a maximum size of 4 GB.
BFILE	This contains a locator to a large binary file that is stored outside the database. This enables byte stream I/O access to external LOBs, which reside on the database server. The maximum size is 4 GB. The size is also limited by the operating system.

CREATE TABLE Command

A table denotes the basic structure that holds user data. The CREATE TABLE command creates a table, defines its columns, integrity constraints and storage allocation.

The CREATE TABLE system privilege must be possessed to create a table in the user's own schema. Likewise, to make a table in the other user's schema, the user must have the CREATE ANY TABLE system privilege. Also, the owner of the schema to contain the table should have either space quota on the tablespace to contain the table or the UNLIMITED TABLESPACE system privilege.

- To make a new table, the following syntax is used:

```
CREATE TABLE <table name>
(<column name> <data type> [(<size>)] <column
constraint>,
   ............................................
   <Table constraints> );
```

where:

- *<table name>* is the name of the table; it must be an alphanumeric identifier.

- *<column name>* is the name of an attribute; it must be an alphanumeric identifier.
- *<data type>* is the type of table.

The syntax is made simple and for simplicity it does not have the optional clauses, such as PCTFREE, PCTINCREASE, STORAGE, etc., which gives instruction to Oracle on how to allocate space dynamically to the table data.

Example:

```
CREATE TABLE emp
      (ecode VARCHAR2(5),
      ename VARCHAR2 (30),
      dno VARCHAR2 (10),
      salary NUMBER(7,2));
```

- ***Note:*** Each table must have at least one column, and column names within one table must be unique.

  A column can be given a default value through the *DEFAULT option*. It assigns a value to be given to the column, if a later INSERT statement omits a value for the column.

```
CREATE TABLE emp
(ecode VARCHAR2(5),
ename VARCHAR2 (30),
dno   VARCHAR2 (10),
salary NUMBER(7,2),
da NUMBER(7,2) DEFAULT 0 );
```

- The default may be literals, an expression having the same data type as the column, but not the name of another column. Functions such as SYSDATE and USER are valid.

```
CREATE TABLE emp
   (ecode VARCHAR2(5),
   ename VARCHAR2 (30),
   dt_jn DATE DEFAULT SYSDATE,
   dno   VARCHAR2 (10),
   salary NUMBER(7,2));
```

A CONSTRAINT *clause* is included in a CREATE TABLE statement for defining an *integrity constraint*.

Integrity constraints in Oracle are of the following types:

- NOT NULL: Specifies a column that cannot contain null values (no value is assigned)
- UNIQUE: Indicates the values of a column (or columns) that must be unique.
- PRIMARY KEY: A set of one or more columns, which acts as the table's primary key.

- FOREIGN KEY: A set of one or more columns, which designates the foreign key in a referential integrity constraint.
- CHECK: Specifies a condition that each row of the table must satisfy.

ORACLE allows integrity constraints to be defined for tables and columns for the following purposes:

- Enforcing rules at table level whenever a row is inserted, updated or deleted from that table, where the constraint must be satisfied for the operation to succeed
- Preventing the deletion of a table if there are dependencies from other tables

There are two basic types of constraints—column constraints and table constraints. Column constraints apply only to individual columns and are provided with the column definition. Table constraints apply to groups of one or more columns and are defined separately from the definitions of the columns in the table.

Column constraints are of the following form:

```
[CONSTRAINT <constraint_name>]
[NOT] NULL |
CHECK (expression ) |
UNIQUE |
PRIMARY KEY |
REFERENCES <table-name> [(<column-name>) ]
[ON DELETE CASCADE]
```

Table constraints are of the following form:

```
[CONSTRAINT <constraint-name>]
UNIQUE (<column-name,..., column-name>) |
PRIMARY KEY(<column-name,..., column-name>) |
FOREIGN KEY (<column-name,..., column-name>)
REFERENCES table-name [(<column-name,..., column-name>)
[ON DELETE CASCADE]
```

[CONSTRAINT <constraint-name>] clause of the CREATE TABLE command is optional. CONSTRAINT denotes the integrity constraint by the name that follows (Pk_snum, Ck_city) it. The definition of the integrity constraint in the data dictionary is stored in this name by ORACLE. Constraints were named with the following prefixes:

- Primary key constraints: pk_
- Foreign key constraints: fk_
- Check constraints: ck_

Naming constraints in this fashion is simply a convenience. Any name may be given to a constraint.

If a name for the constraint is not specified, then Oracle generates a name. Most Oracle's generated constraint names are of the form SYS_C######; e.g., SYS_C000145.

NOT NULL constraints

The requirement of the NOT NULL constraint is that the column contains a value when it is initially inserted into the table or whenever the column is updated.

Specifying PRIMARY KEY constraint

Th most widely used means of enforcing integrity are Column constraints. Of these, the most significant one is the PRIMARY KEY. It ensures that each row in the table is unique. When a column is affirmed as the PRIMARY KEY, an index on this column is automatically created and assigned a unique name by Oracle. The additional constraints UNIQUE and NOT NULL are implied by the PRIMARY KEY constraint.

The following example defines a primary key using column constraint:

```
CREATE TABLE emp
    (ecode   VARCHAR2(5) PRIMARY KEY,
    ename VARCHAR2(30) NOT NULL,
    dno   VARCHAR2(10),
    salary   NUMBER(7, 2));
```

Generally, composite primary key is defined using a table constraint. The following example defines a composite primary key using a table constraint:

```
CREATE TABLE order
    (itemcode   VARCHAR2(10),
    vendorcode  VARCHAR2(10),
    qty   VARCHAR2(10),
    PRIMARY KEY (itemcode, vendorcode));
```

You can give a name to the constraint using the CONSTRAINT clause. The above CREATE TABLE commands can be written as follows:

```
CREATE TABLE emp
    (ecode   VARCHAR2(5) CONSTRAINT pk_emp PRIMARY KEY,
    ename VARCHAR2(30) NOT NULL,
    dno   VARCHAR2(10),
    salary   NUMBER(7, 2));
    CREATE TABLE order
    (itemcode   VARCHAR2(10),
    vendorcode  VARCHAR2(10),
    qty   VARCHAR2(10),
    CONSTRAINT pk_order PRIMARY KEY (itemcode,
    vendorcode));
```

Specifying UNIQUE constraint

The UNIQUE constraint recognizes a column or grouping of columns as a unique key. The unique key, which is made up of a single column, contains NULLS. In case of a composite unique key, any row that contains NULLS in all key columns automatically satisfies the constraint.

Column constraint syntax:

```
[CONSTRAINT <constraint_name>] UNIQUE
```

Table constraint syntax:

```
[CONSTRAINT <constraint_name>] UNIQUE (<column_name> [,
<column_name>] ……)
```

To create a table, Sales, to keep track of the total amount of sales per day per salesperson, ensuring that each day has no more than one row for a given salesperson, enter:

```
CREATE TABLE Sales
    (snum NUMBER (5)NOT NULL,
    odate DATE  NOT NULL,
    totamt   NUMBER (7, 2),
    CONSTRAINT Unq_snum_odate UNIQUE (snum, odate));
```

UNIQUE is a constraint that enforces separate column values on a table column and contains a NULL record also. Similar to UNIQUE is the PRIMARY KEY constraint; however, it cannot have a Null value for the column. There has to be only one primary key in a table. Moreover, a PRIMARY KEY is the parent column for a Parent–Child relationship (Foreign Key).

Specifying CHECK constraint

A condition that each row in the table must evaluate for either true or unknown (due to a NULL) is defined by the CHECK constraint. This condition can only refer to columns in the table to which it belongs. The condition cannot include the following:

- Subqueries
- Functions like SYSDATE, USER
- Pseudocolumns

Syntax: `[CONSTRAINT <constraint_name>] CHECK (<condition>)`

CHECK is useful as a table constraint, when more than one field of a row is involved in a condition. To ensure that city within Kolkata, Chennai, Mumbai or Delhi is permitted for employees, while creating emp table, enter:

```
CREATE TABLE emp
 (ecode   VARCHAR2(5) PRIMARY KEY,
 ename VARCHAR2(30) CONSTRAINT Nn_enm NOT NULL,
 city  VARCHAR2(10) CONSTRAINT Check_city
 CHECK(city IN 'KOLKATA','CHENNAI','MUMBAI','DELHI')),
 salary   NUMBER(7, 2)
 CONSTRAINT Chk_sal CHECK(sal BETWEEN 4000 AND 50000)
 );
```

Another example:

```
CREATE TABLE prod_tbl
(prod_no NUMBER(3) PRIMARY KEY,
 description VARCHAR2(50),
 cost NUMBER(8,2) CHECK (cost > .50));
```

Specifying REFERENTIAL INTEGRITY constraint

To establish a relationship between the foreign key and a specified primary or unique key called the referenced key, a *referential integrity constraint is* assigned a column or combination of columns as *FOREIGN KEY*. In this relationship, the table containing the foreign key is called the **child table** and the table containing the referenced key is called the **parent table**. The referenced unique or primary key constraint on the parent table should already be defined before defining a referential integrity constraint in the child table.

The REFERENCES clause refers to any unique or primary key in the foreign table (foreign key).

Column constraint syntax:

```
[CONSTRAINT <constraint_name>] REFERENCES <table_name>
[(<column>)]
[ON DELETE CASCADE]
```

Table constraint syntax:

```
[CONSTRAINT <constraint_name>] FOREIGN KEY (<column [,
    column] …>)
REFERENCES <table_name> [(<column [, column] ……>)]
[ON DELETE CASCADE]
```

Through ON DELETE CASCADE, the referenced key values in the parent table is deleted, which have dependent rows in the child table. In this case, ORACLE automatically deletes dependent rows from the child table to maintain referential integrity. If this option is omitted, ORACLE does not allow deletion of referenced key values in the parent table, which have dependent rows in the child table.

There is a dept table, which is created using the following CREATE TABLE command:

```
CREATE TABLE dept
(deptno VARCHAR2(3) PRIMARY KEY,
 dname VARCHAR2(20) NOT NULL);
```

To create the emp table, with a referential integrity constraint defined as a table constraint, enter:

```
CREATE TABLE emp
(ecode VARCHAR2(5) PRIMARY KEY,
ename  VARCHAR2 (30) NOT NULL,
city   VARCHAR2 (10),
dno VARCHAR2(3),
salary NUMBER(7, 2)   ,
FOREIGN KEY (dno) REFERENCES dept (deptno)
);
```

Here, the ON DELETE CASCADE option is omitted by the foreign key definitions. So, ORACLE does not allow the deletion of a department if any employee works in that department.

NOTES

The following statement creates the Student table, using the column constraint syntax to define a foreign key on the cnum column that references the primary key of the Course table (ccode) with the constraint name fk_student:

```
CREATE TABLE student
(roll  NUMBER(5) PRIMARY KEY,
sname  VARCHAR2(10)   NOT NULL,
section   VARCHAR2(1),
cnum   NUMBER  (5)  CONSTRAINT  fk_student  REFERENCES
course(ccode) );
```

ALTER TABLE command

The definition of a table is altered by this command in one of the following ways:

- By adding a column
- By adding an integrity constraint
- By redefining a column (data type, size, default value)
- By modifying storage characteristics or other parameters
- By enabling, disabling or dropping an integrity constraint
- By deleting or renaming a column

On the successful execution of the ALTER TABLE command, the system will give a message 'Table altered'.

Adding column(s) to a table

The ADD clause of the ALTER TABLE command defines additional columns and integrity constraints after the creation of a table.

To add a column to an existing table, the ALTER TABLE syntax is as follows:

```
ALTER  TABLE  <table_name>  ADD  (<column_name>
column_definition);
column_ definition includes constraint specification also.
```

For example:

```
ALTER TABLE emp ADD (address VARCHAR2(50));
```

This DDL statement will add a column called address to the emp table.

To add multiple columns to an existing table, the ALTER TABLE syntax is as follows:

```
ALTER TABLE <table_name>
ADD ( <column_name> column-definition,………..);
```

For example:

```
ALTER TABLE emp
ADD (address VARCHAR2(50), tel_no NUMBER(15) );
```

This adds two columns (address and tel_no) to the emp table.

Methods for adding a column to a table

- A column can be added at any time if NOT NULL is not specified. A column defined as NOT NULL cannot be added. If we try to add it, the column will not have anything in it. Every row in the table will have a new empty column defined as NOT NULL. So Oracle will generate an error message.

- A NOT NULL column may be added in the following three steps:
  (i) Add a column without NOT NULL specified
  (ii) Fill every row in that column with data
  (iii) Modify the column to be NOT NULL

Modifying columns in a table

The MODIFY clause can be used to change the some of parts of a column definition for:

- data type
- size
- default value
- NOT NULL column constraint

The MODIFY clause specifies the modified part of the definition and the column name but not the entire definition.

To modify a column in an existing table, the ALTER TABLE syntax is as follows:

ALTER TABLE table_name MODIFY column_name column_ definition;

For example:

```
ALTER TABLE emp MODIFY ename VARCHAR2(35) NOT NULL;
```

This will modify the column called *ename* to be a data type of varchar2 (35) and force the column to not allow null values.

To modify multiple columns in an existing table, the ALTER TABLE syntax is as follows:

```
ALTER TABLE table_name
MODIFY (<column_name> column_definition,
.................);
```

For example:

```
ALTER TABLE emp
MODIFY (ename VARCHAR2(35) NOT NULL, City varchar2(50));
```

This will modify both the *ename* and *city* columns.

Rules for modifying a column

- A column of character type can be increased any time.
- The number of digits in a NUMBER column can be increased any time.
- The number of decimal places in a NUMBER column can be increased or decreased any time.

A column's data type may be changed or its size decreased, only if the column contains NULL in all rows. However, the size of a character column or the precision of a numeric column can always increase.

Any alteration to a column's default value only affects rows that are subsequently inserted into the table and not the values previously inserted.

The only integrity constraint that can be added to an existing column using the MODIFY clause with the column constraint syntax is a NOT NULL constraint. The following ALTER TABLE command adds NOT NULL constraint against dname column:

```
ALTER TABLE dept MODIFY(dname VARCHAR2(14)
CONSTRAINT dept_dname_nn NOT NULL);
```

However, the other integrity constraints like PRIMARY KEY, UNIQUE, CHECK, REFERENTIAL and INTEGRITY is defined on the existing columns using the ADD clause with the table constraint syntax.

A NOT NULL constraint, if it contains no NULLS, is only defined on an existing column.

Dropping columns in a table

With the introduction of Oracle 8i, it is possible to drop a column from a table. Prior to this edition, the only way to do this was to drop the entire table and rebuild it. Using Oracle 8i, a column can be deleted in the following two ways:

(i) Logical delete: marking a column as unused

(ii) Physical delete: delete it completely

Logical delete

The reason for logical delete is that the procedure of physically removing a column from large tables is very resource and time consuming. The syntax for logical delete of column(s) is as follows:

```
ALTER TABLE <table_name> SET UNUSED (<column_name>);
ALTER TABLE <table_name> SET UNUSED (<column_name1>,
<column_name2>, …….);
```

The columns will no longer be visible to the user, once this is done.

```
An example include ALTER TABLE emp SET UNUSED (salary);
```

The columns can be deleted physically later by the following statement:

```
ALTER TABLE <table_name>DROP UNUSED COLUMNS;
e.g. ALTER TABLE emp DROP UNUSED COLUMNS;
```

The DBA_UNUSED_COL_TABS view can be used to view the number of unused columns per table.

Physical delete

To physically drop a column from an existing table, the ALTER TABLE syntax is as follows:

```
ALTER TABLE <table_name >
DROP COLUMN <column_name> [CASCADE CONSTRAINTS];
```

Suppose the dropped columns are part of unique constraint or the primary keys, then it is necessary to use the optional CASCADE CONSTRAINTS clause as a part of the ALTER TABLE command.

For example, the following command will drop the column called *city* from the table called *emp*:

```
ALTER TABLE emp DROP COLUMN city;
```

Multiple columns can be dropped in a single command. The syntax is as follows:

```
ALTER TABLE table_name DROP (column_name1, column_name2,
…….);
```

For example:

```
    ALTER TABLE emp DROP (city, comm);
```

Notice that when dropping multiple columns, the column keyword of the ALTER command should not be used as it causes a syntax error.

```
ALTER TABLE prod_on_ord
DROP COLUMN order_no CASCADE CONSTRAINTS;
```

Notes:

- A column, which is a primary key, cannot be dropped.
- All columns of a table cannot be dropped.

Rename column(s) in a table (only available in Oracle 9i Release 2 onwards)

Starting in Oracle 9i Release 2, we can now rename a column.

To rename a column in an existing table, the ALTER TABLE syntax is as follows:

```
ALTER TABLE table_name
RENAME COLUMN <old_column_name> TO <new_column_name>;
```

For example:

```
ALTER TABLE emp RENAME COLUMN ename TO empname;
```

The column is thus remaned and called *ename* to *empname*.

Adding constraints to a table

The ADD clause is used to add constraint to a table.

```
ALTER TABLE emp ADD ( CONSTRAINT emp_pk PRIMARY KEY
(ecode));
```

Renaming integrity constraints

Apart from allowing to rename tables and columns, Oracle9i Release 2 permits the renaming of constraints on tables.

The syntax for renaming constraint is as follows:

```
ALTER TABLE <table_name>
RENAME CONSTRAINT <old_constraint_name> TO
    <new_constraint_name>;
```

In the following example, we rename the primary key constraint of the EMP table:

```
 SELECT constraint_name
 FROM user_constraints
 WHERE table_name = 'EMP'
 AND constraint_type = 'P';
```

Output:

```
CONSTRAINT_NAME
PK_emp
```

NOTES

Now we enter the following ALTER TABLE command:

```
ALTER TABLE test RENAME CONSTRAINT PK_emp TO emp_PK;
```

Output:

Table altered.

```
SELECT constraint_name
FROM user_constraints
WHERE table_name = 'EMP'
AND constraint_type = 'P';
```

Output:

```
CONSTRAINT_NAME
Emp_ PK
```

Dropping integrity constraints

The DROP clause is used to remove an integrity constraint from a table.

The syntax for the DROP clause is as follows:

```
DROP   {PRIMARY KEY | UNIQUE(<column_name>[,
    column_name]……..>)}
| CONSTRAINT <constraint_name>} [CASCADE]
```

For example, to drop the check_city constraint, the SQL statement is as follows:

```
ALTER TABLE emp DROP CONSTRAINT Check_city;
```

Dropping the primary key constraint of dept table

```
ALTER TABLE dept DROP PRIMARY KEY CASCADE;
```

The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.

Enabling and disabling existing integrity constraints

The ENABLE/DISABLE clause of the ALTER TABLE command allows constraints to be enabled or disabled without dropping or recreating them.

The syntax for the ENABLE/DISABLE clause is as follows:

```
[ENABLE | DISABLE } {UNIQUE (<column_name[, column_name]
……>)      | PRIMARY KEY
 |CONSTRAINT<constraint_name>} [CASCADE]
```

For example:

```
ALTER TABLE emp DISABLE PRIMARY KEY CASCADE;
```

ORACLE enforces a constraint, which is enabled by enforcing it to all data in the table. An enabled constraint has to be satisfied by all table data. ORACLE does not enforce it, if an integrity constraint is disabled. However, along with enabled integrity constraints it appears in the data dictionary.

The alter table command is used to **disable a constraint**. Use the alter table command again to **enable a disabled constraint**. The following example disables and then re-enables the salary check condition:

```
ALTER TABLE emp DISABLE CONSTRAINT CHK_SALARY;
ALTER TABLE emp ENABLE CONSTRAINT CHK_SALARY;
```

DROP TABLE Command

This command removes a table including all its data from the database.

Syntax

```
DROP TABLE <table_name>[CASCADE CONSTRAINTS]
```

CASCADE CONSTRAINTS All referential integrity constraints are dropped that refer to unique and primary keys in the dropped table.

The following statement drops the emp table:

```
DROP TABLE emp:
```

When a table is dropped, the following operations are performed automatically:

- All rows from the table are deleted.
- All the table's indexes regardless of who created or owns them are dropped.
- All data blocks allocated to the table and its indices to the tablespaces containing the table and indices are returned.
- Suppose the table is a base table for views or if it is referenced in stored procedures, functions or packages, these objects are invalidated but not dropped.

Renaming a Table

Renaming a table can be done as

```
RENAME <oldtablename> TO <newtablename>;
```

For example:

```
RENAME emp to empl;
```

Obtaining Table information

To learn the structure of a table, use the DESC command.

For example, the current table columns and type definitions are displayed by the following command:

```
DESC emp;
```

Output:

```
Name Null? Type
ECODE NOT NULL VARCHAR2(5)
ENAME VARCHAR2(30)
SALARY NUMBER(7,2)
DNO VARCHAR2(5)
DESG VARCHAR2(30)
DT_JN DATE
```

To obtain all tables owned by a user, use the following command:

```
      SELECT * FROM TAB;
  Or SELECT * FROM cat;
```

Output:

```
TNAME TABTYPE        CLUSTERID
ASSIGNS          TABLE
DEPT     TABLE
EMP      TABLE
EMP_VIEW VIEW
PROJECT          TABLE
SAL_VIEW VIEW
```

6 rows selected.

2.5.6 Data with Multiple Conditions

DML is used to modify, add, query or delete data.

Inserting rows – INSERT command

The INSERT command is employed to add rows to a table. It is to be noted that to insert rows into a table, the table has to be in the user's own schema or the user must have the INSERT privilege on the table. Users can also insert rows into any table through the INSERT ANY TABLE system privilege.

The two general formats for the INSERT command are as follows:

(*i*) `INSERT INTO <table_name> VALUES (<value>, ………………);`

(*ii*) `INSERT INTO table_name (<column_name>, …………)`
`VALUES (<value>, ……………..);`

In the first format, the columns that will be given values for are not listed. The values must exactly match default and the type order of all the columns in the table. This can be found out with the DESC command. If the column list is omitted, the values clause has to specify values for all columns in the table. Omitted columns will be set to default values, which will be either NULL or an explicitly defined default. If a constraint prevents a NULL from being accepted in a given column, that column must be provided with a value.

For example, to insert a row into the emp table, enter:

```
INSERT INTO emp
VALUES ('E01', 'KOUSHIK GHOSH',5000, 'D01', 'PROGRAMMER',
'10-MAR-93');
```

Oracle will notify the user with the following message for successful insertion:

1 row inserted

It is to be noted that CHARACTER and DATE values should be within single quotes. If we wish to omit a value (and it is valid to do so, i.e., NULL-able column), we can use the keyword NULL, as in the following example:

```
INSERT INTO emp
VALUES ('E01', 'KOUSHIK GHOSH', NULL, 'D01', 'PROGRAMMER',
NULL);
```

In order to insert the same row omitting the designation, salary and dt_jn column, the columns must be specified:

```
INSERT INTO emp(ecode, ename, dno)
VALUES ('E01', 'KOUSHIK GHOSH', 'D01');
```

Here, this INSERT statement has only the ecode, ename and dno columns. The order in which the values appear corresponds to the statement's column list, not the table as listed by the DESCRIBE command.

To interactively prompt users to enter values for the fields at SQL*Plus terminal, use the following syntax:

```
INSERT INTO employee VALUES ('&ecode', '&ename', &salary,
'&dno','&desg','&dt_jn');
```

The SQL*Plus terminal then will prompt users to enter values for the three variables: ecode, ename and salary.

It is possible to insert multiple rows into a table by using the INSERT INTO SELECT statement. The syntax for the INSERT INTO SELECT statement is as follows:

```
INSERT INTO TABLE SELECT statement;
```

To copy all rows from the emp table with the desg = 'PROGRAMMER' into the PROGRAMMER table, enter:

```
INSERT INTO programmer
SELECT * FROM emp WHERE desg = 'PROGRAMMER' ;
```

The choosen list of the subquery must have equal columns as the column list of the INSERT statement.

(*Possible caveats with INSERT*)

When using INSERT, the following integrity errors can occur:

- Attempting to insert NULL values into a column with a NOT NULL constraint.
- Trying to insert non-allowed values into a column with a CHECK constraint.
- Attempting to insert a record with a foreign key value that has no corresponding primary key record in another table.

Updating rows—UPDATE command

The update statement facilitates the change of a column value in a row. The WHERE clause can contain any condition allowed in a SELECT statement including subqueries.

The syntax for the UPDATE command is as follows:

```
UPDATE <table_name>
SET <column_name1> = <value1>, …, <column_nameK> = <valueK>
WHERE <condition>;
```

The SET clause determines which values are updated and what net values are stored in them. The SET clause accepts any number of column assignments, separated by commas. The WHERE clause is optional and can be used to select which row or rows are to be updated. The WHERE clause is followed by a logical expression and every record for which this expression is true is updated.

For example, to increase the salaries of all employees by 5 per cent, enter:

```
      UPDATE emp SET salary = salary * 1.05;
```

The system will notify the user with the following message:

n rows updated

where n is the number of rows affected.

To perform the same change on all employees of department 'D02', enter:

```
  UPDATE emp SET salary = salary * 1.5 WHERE dno = 'D02';
```

To update the record of employee 'E01', enter:

```
  UPDATE emp
  SET ename = 'JAYANTA GANGULY', city = 'KOLKATA',
  salary = salary * 2
  WHERE ecode = 'E04';
```

The above command updates ename, city and comm column.

(*Possible caveats with UPDATE*)

When using UPDATE, the following integrity errors can occur:

- Attempting to update to NULL values a column with a NOT NULL constraint.
- Trying to update to non-allowed values a column with a CHECK constraint.
- Attempting to update a record to a foreign key value that has no corresponding primary key record in another table.
- A flawed or vacuous WHERE clause causing update of all records.

Deleting Records—DELETE command

The DELETE command allows the removal of one or more rows from a table. The table has to be in the user's own schema or the user must have the DELETE privilege on the table to delete rows from a table. The DELETE ANY TABLE system privilege gives freedom to a user to delete rows from any table.

The syntax for the DELETE command is as follows:

```
  DELETE FROM tablename [WHERE condition]
```

Like the UPDATE command, omitting the WHERE clause means the command will be performed on all records. To remove all the rows of emp, enter:

```
  DELETE FROM emp;
```

The system will notify the user with the following message:

```
  n rows deleted
```

Where n is the number of rows deleted.

The following statement deletes from the emp table all employees located in Chennai:

```
  DELETE FROM emp WHERE city = 'Chennai';
```

A condition can reference a table and contain a subquery. To the predicate of a DELETE command subqueries are also used. Use of subquery will be covered while describing subqueries.

(*Possible caveats with DELETE*)

When using DELETE, the following integrity errors can occur:

- Attempting to delete a primary key record with foreign key records of another table. This only works if ON DELETE CASCADE constraint has been set.
- A flawed or vacuous WHERE clause causing the deletions of all records.

TRUNCATE command

This command also allows the removal of all rows from a table, but it flushes a table more efficiently since no rollback information is retained.

For example, to delete all data and index (if any) rows of the emp table and return the freed space to the tablespace containing emp, enter:

```
TRUNCATE TABLE emp;
```

Syntax

`TRUNCATE TABLE`	`<table_name> [{DROP	REUSE} STORAGE]`
`DROP STORAGE`	It performs the deallocation of space from the deleted rows from the table.
`REUSE STORAGE`	It keeps the space from the deleted rows allocated to the table. This space can be used later only by new data in the table resulting from INSERTs or UPDATEs.

(*Possible caveats with TRUNCATE*)

When using TRUNCATE, the following integrity errors can occur:

- An implicit COMMIT will occur. Previous DML statements executed during the same session will be committed.

Difference between DELETE and TRUNCATE commands

At the basic plane, DELETE scans the table and omits any rows that is not similar to the given criteria in the (optional) WHERE clause. It creates rollback information so that if needed the the deletions could be reverted. And from the indexes, the index entries for the deleted rows are changed. The COMMIT data control command must be given to permanently delete them.

Extents are not deallocated while deleting rows from a table; the result is that the extents in the table remain even after the deletion. At the same time, the high watermark is not moved down, thus it also remains at its place as before the deletion.

At the same time, TRUNCATE, just moves the high-watermark on the table to the beginning. This is done very quickly to truncate. Once one table is truncated, there is no going back. Thus, indexes are also truncated as there is no facility to specify which row to 'delete'. In this case, delete command can also be used with WHERE clause.

All the extents of a table are deallocated, when it is truncated, leaving only the extents that were identified when the table was created in the first place. Thus, if the table was in the first place made with MINEXTENTS **3**, 3 extents will remain when the table is truncated.

The extents are **not** deallocated if the REUSE STORAGE clause is specified. It saves time in the recursive SQL department if one intends to reload the table with an exported data, and can lessen the time it takes to do the import as there is no need to vigorously allocate any new extents.

Querying Data—SELECT command

The most important statement in SQL is the SELECT statement. The following table is used for subsequent discussion:

EMP	**DEPT**
ecode VARCHAR2(5) ename VARCHAR2(30) salary NUMBER(7,2) dno VARCHAR2(5) dt_jn DATE desg VARCHAR2(30)	dno VARCHAR2(5) dname VARCHAR2(30) city VARCHAR2(25)

Instance of EMP table:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

Instance of DEPT table:

DNO	DNAME	CITY
D01	PROJECT	KOLKATA
D02	RSEARCH	CHENNAI
D03	PERSONNEL	KOLKATA

The general syntax of this command is as follows:

```
SELECT [ALL | DISTINCT] <column_name1> [,<column_name2>]
FROM table_name1 [,table_name2]
[WHERE <condition>] [ AND|OR <condition>…...]
```

```
[GROUP BY <column-list>]
[HAVING <condition>]
[ORDER BY <column-list> [ASC | DESC] ];
```

where

SELECT	SELECT clause lists the columns to retrieve data from.
FROM	FROM clause lists the tables the columns are located in.
WHERE	WHERE clause specifies criteria return values must match.
GROUP BY	GROUP BY clause specifies groups for summary results.
HAVING	HAVING clause specifies filter conditions for summary results.
ORDER BY	ORDER BY clause specifies sort order.

The sections between the brackets [] are optional.

The following is the basic format of the SELECT statement:

```
SELECT <select_list>
FROM <table_name>
[WHERE <condition>]
```

The SELECT statement returns one or more rows, called the *result set*. If a WHERE clause is not specified, it returns one row for every row in the table. The select_list specifies the columns that are returned in the result set. The simplest form of the column list is an asterisk, which represents all the columns and this statement, therefore, returns all the columns and rows of the table.

Query: Display all rows of emp table.

```
    SELECT * FROM emp;
```

Output:

```
ECODE  ENAME               SALARY  DNO  DESG                 DT_JN
E01    KOUSHIK GHOSH        5000   D01  SYSTEM ANALYST       10-MAR-93
E02    JAYANTA DUTTA        3500   D01  PROGRAMMER           15-JAN-94
E03    HARI NANDAN TUNGA    4000   D02  PROGRAMMER           01-JUL-95
E04    JAYANTA GANGULY      6000   D03  ACCOUNTANT           12-SEP-96
E05    RAJIB HALDER         4000   D03  CLERK                07-OCT-95
E06    JISHNU BANERJEE      6500   D02  SYSTEM MANAGER       19-SEP-96
E07    RANI BOSE            3000   D01  PROJECT ASSISTANT    17-JUN-97
E08    GOUTAM DEY           5000   D01  PROGRAMMER           23-OCT-97
E09    PINAKI BOSE          5500   D02  PROGRAMMER           26-AUG-94
```

9 rows selected.

Query: List all employees' code, name and salary in the EMP table.

```
SELECT ecode, ename, salary FROM emp;
```

NOTES

Output:

ECODE	ENAME	SALARY
E01	KOUSHIK GHOSH	5000
E02	JAYANTA DUTTA	3500
E03	HARI NANDAN TUNGA	4000
E04	JAYANTA GANGULY	6000
E05	RAJIB HALDER	4000
E06	JISHNU BANERJEE	6500
E07	RANI BOSE	3000
E08	GOUTAM DEY	5000
E09	PINAKI BOSE	5500

9 rows selected.

Arithmetic expression in SELECT

It is possible to include arithmetic expressions involving column, literals with +, “, * and / operators. Expressions on NUMBER and DATE data types can be used by means of the arithmetic operators.

Query: Display the employee name, salary and annual salary.

```
SELECT ename, salary, salary * 12 FROM emp;
```

Output:

ENAME	SALARY	SALARY*12
KOUSHIK GHOSH	5000	60000
JAYANTA DUTTA	3500	42000
HARI NANDAN TUNGA	4000	48000
JAYANTA GANGULY	6000	72000
RAJIB HALDER	4000	48000
JISHNU BANERJEE	6500	78000
RANI BOSE	3000	36000
GOUTAM DEY	5000	60000
PINAKI BOSE	5500	66000

9 rows selected.

Operator Precedence

- Precedence over addition and subtraction is taken by multiplication and division.
- The same priority operators are evaluated from left to right.
- Parentheses are used to clarify statements and to force prioritized evaluation.

Column aliases in SELECT

To provide an alternate name to a column or expression, aliases are used that appear as the column heading in the result set. For the above query, the following command can be used:

```
SELECT ename “ Employee Name”, salary “Salary”, salary*12
“Annual salary” FROM emp;
```

Output:

Employee Name	Salary	Annual salary
KOUSHIK GHOSH	5000	60000
JAYANTA DUTTA	3500	42000
HARI NANDAN TUNGA	4000	48000
JAYANTA GANGULY	6000	72000
RAJIB HALDER	4000	48000
JISHNU BANERJEE	6500	78000
RANI BOSE	3000	36000
GOUTAM DEY	5000	60000
PINAKI BOSE	5500	66000

9 rows selected.

Column alias can also be given using 'AS' followed by alias name after column name. The only restriction is that alias should not contain spaces.

```
SELECT ename AS EmployeeName, salary AS Salary, salary*12
AS Annual_salary FROM emp;
```

Output:

EmployeeName	Salary	Annual_salary
KOUSHIK GHOSH	5000	60000
JAYANTA DUTTA	3500	42000
HARI NANDAN TUNGA	4000	48000
JAYANTA GANGULY	6000	72000
RAJIB HALDER	4000	48000
JISHNU BANERJEE	6500	78000
RANI BOSE	3000	36000
GOUTAM DEY	5000	60000
PINAKI BOSE	5500	66000

9 rows selected.

The || operator is used to concatenate character expressions. e.g.,

```
SELECT 'Designation of ' || ename || ' is ' || desg from
emp;
```

Output:

```
'DESIGNATIONOF'||ENAME||'IS'||DESG
Designation of KOUSHIK GHOSH is SYSTEM ANALYST
Designation of JAYANTA DUTTA is PROGRAMMER
Designation of HARI NANDAN TUNGA is PROGRAMMER
Designation of JAYANTA GANGULY is ACCOUNTANT
Designation of RAJIB HALDER is CLERK
Designation of JISHNU BANERJEE is SYSTEM MANAGER
Designation of RANI BOSE is PROJECT ASSISTANT
Designation of GOUTAM DEY is PROGRAMMER
Designation of PINAKI BOSE is PROGRAMMER
9 rows selected.
```

NOTES

Removing duplicates—DISTINCT

Strictly speaking, it is incorrect to refer to SQL tables as relations because SQL queries might result in duplicate tuples. SQL gives a mechanism to do away with duplicates. It is done by specifying the keyword **DISTINCT** after **SELECT**.

The default display of queries is all rows, including duplicate rows.

```
SELECT dno FROM emp;
```

Output:

```
DNO
D01
D01
D02
D03
D03
D02
D01
D01
D02
9 rows selected.
```

This command will display all the dept numbers including duplicate dept numbers.

Instead of this command, if you use:

```
SELECT DISTINCT dno FROM emp;
```

Output:

```
DNO
D01
D02
D03
```

Notice that output contains non-duplicated department numbers. It is to be noted that you can use only one DISTINCT in the select column list.

Filtering Rows by Conditional Selection—WHERE Clause

The WHERE clause specifies the search condition and join criteria on the data that are selected. If a row fulfils the search conditions, it is returned as part of the result set.

The predicate employed in the WHERE clause is a simple comparison that uses the following:

- Relational Operators
- BETWEEN….AND….

- IN
- IS NULL
- IS NOT NULL
- LIKE

WHERE clause and relational operator

These may be as follows:

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

The relational operator condition is fulfilled when the expression on either side of the relational operator satisfies the relation set up by the operator. There are two other differences between the SQL operators and operators used in C/C++. First, the equality comparison and assignment operators, both represented by a single equal sign (=), are the same in SQL. Ambiguity is resolved by context. Second, the standard SQL inequality operator is represented by angle brackets (<>), though Oracle also supports the C/C++-style (!=).

- **Comparisons with numeric column**

  Query: Display all employees getting salary over ₹ 5000.

```
SELECT * FROM emp WHERE salary > 5000;
```

Output:

```
ECODE   ENAME              SALARY   DNO   DESG             DT_JN
E04     Jayanta Ganguly      6000   D03   Accountant       12-SEP-96
E06     Jishnu Banerjee      6500   D02   System Manager   19-SEP-96
E09     Pinaki Bose          5500   D02   Programmer       26-AUG-94
```

- **Comparisons with character and date column**

  String comparisons are performed using the numeric value of the characters, which are determined by the database character set. The character set is generally compatible with ASCII. The decimal values of the numbers and letters, for example, are found in the following table:

Decimal Values of ASCII Numbers and Letters

Character Range	Decimal Value Range
0–9	48–39
A–Z	65–90
a–z	97–122

The most useful simple comparisons for strings can be done with the equality and inequality operators "=" and "<>". They are used to select or omit specific rows. It is to be noted that:

- Date values and character strings are placed within single quotation marks.
- Date values are format-sensitive and character values are case-sensitive.
- Default date format is 'DD-MON-YY'.

Query: List the details of the employee whose name is 'JAYANTA DUTTA'.

```
SELECT * FROM emp WHERE ename = 'JAYANTA DUTTA'
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94

Query: Find the code, name and salary of employees of the department 'D01'.

```
SELECT ecode, ename, salary FROM emp WHERE dno='D01';
```

Output:

ECODE	ENAME	SALARY
E01	KOUSHIK GHOSH	5000
E02	JAYANTA DUTTA	3500
E07	RANI BOSE	3000
E08	GOUTAM DEY	5000

Query: List the information of the employees who are not working in the department 'D01'.

```
SELECT * FROM emp WHERE dno<>'D01';
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

Note: The comparison operators, when used with strings, must match case correctly. There are SQL functions that are employed to convert strings in the database to every uppercase or lowercase to find a case-insensitive match. The functions are as follows:

- LOWER (CHAR) returns CHAR, with each letter in lowercase.
- UPPER (CHAR) returns CHAR, with each letter in uppercase.

Query: Find the code, name and salary of employees of the department 'D01'.

```
SELECT ecode, ename, salary FROM emp WHERE
UPPER(dno)='D01';
```

The equivalent SQL statement is as follows:

```
SELECT ecode, ename, salary
FROM emp WHERE dno='D01' OR dno='d01';
```

Date and time comparisons are similar to comparisons with numbers because every underlying date and time is a number. In Oracle, there is a single data type, DATE, that represents both date and time with a single number. If we want to compare a date column with another date, we can use a string literal in the default date format and Oracle performs the conversion for:

Query: List the employees who joined after 20th September, 1996.

```
SELECT * FROM emp WHERE dt_jn>'20-SEP-96';
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97

The current date and time can be obtained by using the pseudo column, SYSDATE. Oracle automatically stores information, such as row numbers and row descriptions and Oracle is directly accessible, i.e., not through tables. This information is contained within pseudo columns. These pseudo columns can be retrieved in queries. They can be included in queries, which select data from tables.

Available **pseudo columns** in oracle include the following:

- **ROWNUM:** Row number. Order number in which a row value is retrieved.
- **ROWID:** Physical row (memory or disk address) location, i.e. unique row identification.
- **SYSDATE:** System or today's date.
- **UID:** User identification number indicating the current user.
- **USER:** Name of the currently logged in user.
- **Oracle table 'DUAL'**

  Oracle automatically creates DUAL table along with the data dictionary. DUAL is in the schema of the user SYS. However, it is accessible by the name DUAL to all users. It has one column, DUMMY, defined to be VARCHAR2(1), and contains one row with a value 'X'. Selecting from the DUAL table is useful for computing a constant expression with the SELECT statement. Since, DUAL has only one row, the constant is returned only once. Each Oracle account has access to a table called **dual**. We can query against this table to get the current account, system date/time, and execute mathematical functions.

NOTES

The following example shows how to obtain the username used when the user logged into Oracle:

```
SELECT USER FROM DUAL;
```

Output:

```
    USER
-----------
    MANAS
```

```
SELECT SYSDATE FROM DUAL;
```

Output:

```
    SYSDATE
-----------
    15-APR-05
```

```
SELECT POWER(4,3) FROM DUAL;
```

Output:

```
    POWER(4,3)
-----------
          64
```

Query: List all of the names of employees in the EMP table, the date they joined and the current system date.

```
SELECT ename, dt_jn,SYSDATE FROM emp;
```

Output:

ename	dt_jn	SYSDATE
KOUSHIK GHOSH	10-MAR-93	15-APR-05
JAYANTA DUTTA	15-JAN-94	15-APR-05
HARI NANDAN TUNGA	01-JUL-95	15-APR-05
JAYANTA GANGULY	12-SEP-96	15-APR-05
RAJIB HALDER	07-OCT-95	15-APR-05
JISHNU BANERJEE	19-SEP-96	15-APR-05
RANI BOSE	17-JUN-97	15-APR-05
GOUTAM DEY	23-OCT-97	15-APR-05
PINAKI BOSE	26-AUG-94	15-APR-05

9 rows selected.

Suppose there is one of the employees, ASOK BASU, who has joined today. Then SYSDATE and dt_jn are the same. But if we query for the employees joined today using SYSDATE, we will find no results. In the dt_jn, the date is entered using the default date format that does not have a time part. The time is defaulted to 12:00:00 a.m. SYSDATE, which is the current date and time, does have a time part even though only the date part is displayed by default.

```
        SELECT * FROM emp WHERE dt_jn=SYSDATE;
```

The above command will display the message 'no rows selected'.

The TRUNC function is used to remove the time part of an Oracle DATE. This provides us a way to compare the dates in the table with today's date, disregarding any hours, minutes or seconds.

```
      SELECT * FROM emp WHERE dt_jn=TRUNC(SYSDATE);
```

Similarly, suppose the date of join column is filled with SYSDATE as follows:

```
   INSERT INTO emp
   VALUES ('E11', 'KUNTAL GHOSH',5000, 'D03', 'JR ASSISTANT',
   SYSDATE);
```

Say SYSDATE is 20-JUL-05.

Sometimes, to recover data based on something like:

```
   SELECT * FROM emp WHERE dt_jn = '20-JUL-05';
```

The output shows that now rows are selected. But there is a record for that day. What happened is that the records are not set to midnight (which is the default value if time of day is not specified).

One of the solutions is:

```
   SELECT * FROM emp
   WHERE dt_jn >= '20-JUL-05' AND dt_jn < '19-JUL-05';
```

(WHERE clause with logical operator)

AND, OR, NOT are known as logical operators.

Operator	Meaning
AND	Returns TRUE if both component conditions are TRUE
OR	Returns TRUE if either component condition is TRUE
NOT	Returns TRUE if the following condition is FALSE

SQL has three logical values, TRUE, FALSE and NULL. Every condition, simple or compound, evaluates to one of these three values. In a WHERE clause, if this condition evaluates to TRUE, the row is returned if it is part of a SELECT statement. If it is FALSE or NULL, it is not.

NOT

FALSE and NULL are not the same. When FALSE is negated, TRUE is obtained. But when NULL is negated, we still get NULL. The following table is the truth table for NOT:

Truth Table for NOT

	NOT
TRUE	FALSE
FALSE	TRUE
NULL	NULL

NOTES

Query: List the employees who does not work in the department 'D01'.

```
SELECT * FROM emp WHERE dno <> 'D01';
```

The equivalent command using logical NOT is as follows:

```
SELECT * FROM emp WHERE NOT dno ='D01';
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

AND

AND is used to combine two conditions. The following is the truth table for AND:

Truth Table for AND

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	NULL
NULL	NULL	NULL	NULL

Query: List the employees who do not work in the department 'D01'.

```
SELECT * FROM emp WHERE dno <> 'D01';
```

The equivalent command using logical NOT is as follows:

```
SELECT * FROM emp WHERE NOT dno ='D01';
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

The following is a query combining conditions with AND:

Query: List the employees of the department 'D01' who are getting salary over ₹ 4000.

```
SELECT * FROM EMP WHERE dno ='D01' AND salary>4000;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97

Query: List the employees who have joined in the year 1996.

```
SELECT * FROM EMP
WHERE DT_JN >='01-JAN-96' AND DT_JN<='31-DEC-96';
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96

OR

OR is also used to combine two conditions according to the following truth table:

The following is the true table for OR:

Truth Table for OR

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

The following is a sample query with OR:

Query: List all those who are getting salary less than ₹ 5000 or work as Programmer, listed together.

```
SELECT ecode, ename, dno FROM emp WHERE salary < 5000 OR
desg='Programmer';
```

Output:

ECODE	ENAME	DNO
E02	JAYANTA DUTTA	D01
E03	HARI NANDAN TUNGA	D02
E05	RAJIB HALDER	D03
E07	RANI BOSE	D01

Query: List code and designation of the employees whose code is either 'E01' or 'E07'.

```
SELECT ecode, desg FROM emp WHERE ecode ='E01' OR
ecode='E07';
```

Output:

```
ECODE       DESG
E07         PROJECT ASSISTANT
E01         SYSTEM ANALYST
```

Logical operator precedence

Logical operators have an order of precedence. First parenthesized expressions are evaluated, then the NOT operator, followed by the AND operator and finally the OR operator.

Order Evaluated	Operator
1	All comparison operators
2	NOT
3	AND
4	OR

The rules of precedence can be overridden by using parentheses:

Query: Display employee name, designation and salary whose designation is either Programmer or System Manager, and salary is greater than ₹ 5000.

```
SELECT ename, desg, salary FROM emp WHERE desg='PROGRAMMER'
OR desg=' SYSTEM MANAGER' AND salary>5000;
```

Output:

```
ENAME                       DESG              SALARY
JAYANTA DUTTA               PROGRAMMER        3500
HARI NANDAN TUNGA           PROGRAMMER        4000
GOUTAM DEY                  PROGRAMMER        5000
PINAKI BOSE                 PROGRAMMER        5500
```

The query gives wrong output. The output is not the one we want. While employing the combinations of AND and OR, it is vital to use parentheses to make sure the proper selection of rows returned to the results table. The parentheses give for exact criteria in the selection procedure.

If we write the query as follows:

```
SELECT ename, desg, salary FROM emp
WHERE (desg='PROGRAMMER' OR desg=' SYSTEM MANAGER') AND
salary>5000;
```

Then output will be:

```
ENAME                  DESG                 SALARY
JISHNU BANERJEE        SYSTEM MANAGER       6500
PINAKI BOSE            PROGRAMMER           5500
```

This is the right output. First, SQL gets the rows where the designation is either PROGRAMMER or SYSTEM MANAGER; then considering this new list of rows, SQL sees if any of these rows satisfies the condition that the salary column is greater than ₹ 5000.

WHERE clause with IN operator

The IN operator defines a set in which a given value may or may not be included. The IN operator checks for a value to match any value in a list of values. The format of IN comparisons is as follows:

Column_name [NOT] IN (value-1 [, value-2] ...)

Query: List all system managers and programmers.

```
SELECT ecode, ename, dno, desg
FROM emp
WHERE desg IN (‘SYSTEM MANAGER’, ‘PROGRAMMER’);
```

is equivalent to

```
SELECT ecode, ename, dno, desg
FROM emp
WHERE desg =’SYSTEM MANAGER’ OR desg=’PROGRAMMER’;
```

Output:

ECODE	ENAME	DNO	DESG
E02	JAYANTA DUTTA	D01	PROGRAMMER
E03	HARI NANDAN TUNGA	D02	PROGRAMMER
E06	JISHNU BANERJEE	D02	SYSTEM MANAGER
E08	GOUTAM DEY	D01	PROGRAMMER
E09	PINAKI BOSE	D02	PROGRAMMER

WHERE clause with BETWEEN operator

BETWEEN defines a range that value must fall in to make a predicate true.

The BETWEEN operator tests whether a value is *between* two other values. BETWEEN comparisons have the following format:

```
<column_name> [NOT] BETWEEN <value-1> AND <value-2>
```

This comparison tests if *column_name* is greater than or equal to *value-1* **and** less than or equal to *value-2*. It is equivalent to the following predicate:

```
column_name >= value-1 AND column_name <= value-2
```

Or, if NOT is included:

```
NOT (column_name >= value-1 AND column_name <= value-2)
```

For example:

Query: List those employees who are getting salary greater than or equal to ₹ 4000, but less than or equal to ₹ 6000.

```
SELECT * FROM emp WHERE salary BETWEEN 4000 AND 6000;
```

It is equivalent to:

```
SELECT * FROM emp
WHERE salary >= 4000 and salary<= 6000;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

Query: List all those who are not in this range.

```
SELECT * FROM emp WHERE salary NOT BETWEEN 15000 AND
20000;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97

The mystery of Oracle NULL

A column is said to be null, or containing a null if a column in a row has no value. Nulls appear in columns of any data type that is not restricted by NOT NULL or is not PRIMARY KEY or a part of it. Null is used when the actual value is unknown or when a value is not meaningful. Null must not be used to represent a zero value, because they are not equivalent. Though **Oracle currently treats a character value with a length of zero as null,** this may not continue in the future. It is suggested not to treat empty strings as the same as NULLs. An arithmetic expression containing a null always evaluates to null. As for example, null added to 15 is null. All operators (except concatenation) return null when given a null operand.

IS NULL/IS NOT NULL

Oracle implements and checks null values with the help of IS NULL and IS NOT NULL.

Query: List the employees whose designation is NULL.

```
SELECT * FROM emp WHERE desg IS NULL;
```

String Comparisons with Wildcards

Character strings can be compared for equality, using the comparison operators if we want to compare the entire strings for exact matches. SQL provides a special comparison operator, LIKE, which allows us to match portions of strings by using wildcard characters as placeholders in pattern strings to perform the comparison. There are two wildcard characters in SQL, underscore and per cent, as shown in the following table:

Wildcard characters in SQL

_	Matches any one single character
%	Matches any number of occurrences (including zero) of any character

These are similar to the wildcards (?) and (*) used in the DOS and Unix environments, respectively.

To match a pattern string, any one character must appear wherever (_) appears. All literals in the rest of the string (anything except the per cent symbol, in other words) must appear exactly as they appear in the pattern strings. The following table shows some examples:

Examples of the Underscore Wildcard

Pattern	**Matches**
'_'	Any single letter string
'__'	Any string two letters long
'_BC'	Any string three letters long, ending in BC
'A_C'	Any string three letters long, beginning with A and ending with C
'_A_'	Any string three letters long, with middle letter A

The per cent sign (%) can represent any number of any character or characters, including none. However, if there are any literals in the string or (_) in the string, these must be matched as well. The following table shows some examples:

Examples of the Per cent Sign Wildcard

Pattern	**Matches**
'%'	Any string, but not NULL
'%A'	Any string that ends with A
'A%'	Any string that begins with A
'%A%'	Any string that contains the letter A
'_%A%'	Any string that contains the letter A, except as the first letter

The following are a few sample queries:

Query: List the employees whose name begin with 'J'.

```
SELECT * FROM emp WHERE ename LIKE 'J%';
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96

Query: List name and salary of the employees whose title is BOSE.

```
SELECT ename, salary FROM emp WHERE ename LIKE '%BOSE'
```

Output:

ENAME	SALARY
RANI BOSE	3000
PINAKI BOSE	5500

The optional ESCAPE subclause specifies an escape character for the pattern, allowing the pattern to use '%' and '_' (and the escape character) for matching. The ESCAPE value has to be a single character string. In the pattern, the ESCAPE character precedes any character to be escaped.

For example, to match a string ending with '%', use:

x LIKE '%/%' ESCAPE '/'

The following example escapes the escape character:

y LIKE '/%//%' ESCAPE '/'

The optional NOT reverses the result so that:

z NOT LIKE 'abc%'

is equivalent to:

NOT z LIKE 'abc%'

Rearranging rows—ORDER BY clause

The ORDER BY clause may be used to sort the retrieved rows. The general format of the ORDER BY clause is:

```
ORDER BY <column_name> [ASC|DESC],...........
```

Column_name,...... are column names either specified or implied in the select list. A new name is used in the ORDER BY list if a select column is renamed. ASC and DESC request ascending or descending sort for a column, respectively. The default is ASC.

Example:

```
SELECT ecode, ename, salary, desg FROM emp ORDER BY ename
DESC;
```

Output:

ECODE	ENAME	SALARY	DESG
E07	RANI BOSE	3000	PROJECT ASSISTANT
E05	RAJIB HALDER	4000	CLERK
E09	PINAKI BOSE	5500	PROGRAMMER
E01	KOUSHIK GHOSH	5000	SYSTEM ANALYST
E06	JISHNU BANERJEE	6500	SYSTEM MANAGER
E04	JAYANTA GANGULY	6000	ACCOUNTANT
E02	JAYANTA DUTTA	3500	PROGRAMMER
E03	HARI NANDAN TUNGA	4000	PROGRAMMER
E08	GOUTAM DEY	5000	PROGRAMMER

9 rows selected.

Multiple column names can be specified in the ORDER BY clause. ORDER BY sorts rows, scanning the ordering columns in the major-to-minor order and the left-to-right order. On the first column name in the list, the rows are sorted first. If there are any duplicate values for the first column, the duplicates are sorted on the second column (within the first column sort) in the ORDER BY list, and so on. For rows that have duplicate values for all Order By columns, there is no defined inner ordering.

To select the employees from emp table order first by ascending department number name and then by descending salary, issue the following statement:

```
SELECT ename, dno, salary FROM emp ORDER BY dno , salary
DESC;
```

Output:

ENAME	DNO	SALARY
KOUSHIK GHOSH	D01	5000
GOUTAM DEY	D01	5000
JAYANTA DUTTA	D01	3500
RANI BOSE	D01	3000
JISHNU BANERJEE	D02	6500
PINAKI BOSE	D02	5500
HARI NANDAN TUNGA	D02	4000
JAYANTA GANGULY	D03	6000
RAJIB HALDER	D03	4000

9 rows selected.

While employing expressions in the select list, it can be more convenient to specify the select items by number (starting with 1). Both names and numbers can be intermixed.

```
SELECT ename, dno, salary FROM emp ORDER BY 2 ASC, 3 DESC;
```

The positional ORDER BY notation is used above, according to positions in the SELECT list.

Special processing in ORDER By is needed by Database nulls. A null column sorts higher than all regular values; this is reversed for DESC. Nulls are considered duplicates of each other for ORDER BY in sorting.

Selecting rows using ROWNUM pseudo column

The pseudocolumn ROWNUM is available since Oracle versions 7. For each row returned by a query, the ROWNUM pseudo column returns a number indicating the order in which Oracle selects the row from a table or a set of joined rows. The first row selected has a ROWNUM of 1, the second has 2, and so on.

ROWNUM can be used to limit the number of rows returned by a query as in the following example:

```
SELECT * FROM emp WHERE ROWNUM < 6;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95

If an ORDER BY clause follows ROWNUM in the same query, then the rows will be reordered by the ORDER BY clause.

Example:

```
SELECT * FROM emp WHERE ROWNUM < 6 ORDER BY salary;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96

If the ORDER BY clause is embedded in a subquery and the ROWNUM condition is placed in the top-level query, then the ROWNUM condition can be forced to be applied after the ordering of the rows. This is occasionally called ‘top-N query’:

It, in general, refers to getting the top-n rows from a result set. For example, find the top 3 employees ranked by salary:

```
SELECT *
FROM ( SELECT * FROM emp ORDER BY salary DESC )
WHERE ROWNUM <= 3;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

That is a top-n type query. The ROWNUM values are those of the top-level SELECT statement, and they are produced after the rows have already been ordered by salary in the subquery. This is also known as inline view. The latter is a construct in Oracle SQL where we can place a query in the SQL FROM clause, just as if the query was a table name. A general use for inline views in Oracle SQL is to make simple complex queries by removing join operations and condensing several separate queries into a single query.

The above query can also be written as follows:

```
SELECT ename, salary FROM (SELECT ename, salary FROM emp
ORDER BY salary DESC)
WHERE ROWNUM < 4;
```

Output:

ENAME	SALARY
JISHNU BANERJEE	6500
JAYANTA GANGULY`	6000
PINAKI BOSE	5500

Conditions that test for ROWNUM values larger than a positive integer are always false. As for example, the following query returns no rows:

```
SELECT * FROM employees WHERE ROWNUM > 1;
no rows selected
```

The first row that is fetched is assigned a ROWNUM of 1 and this makes the condition false. The second row to be fetched is now the first row and is also assigned a ROWNUM of 1 and this also makes the condition false. All rows subsequently fail to satisfy the condition, so no rows are returned.

Changing date formats using the ALTER SESSION statement

In the earlier examples of SQL statements, the default format of data of type DATE was in the format: DD-MON-YY

The TO_CHAR and TO_DATE functions are used to convert dates to other formats; however, this is not convenient, especially when a large number of rows have to be inserted.

The ALTER SESSION statement alters various characteristics of the current SQL*Plus session including the default date format. This statement is often used to format dates to conform to regional customs. The following is the the syntax for ALTER SESSION for use with changing the default date format:

```
ALTER SESSION
SET NLS_DATE_FORMAT = <date_format>;
```

NOTES

The date_format can include the following codes:

YY	A 2-digit year such as 98
YYYY	A 4-digit year such as 1998
NM	A month number
MONTH	The full name of the month
MON	The abbreviated month (Jan, Feb, Mar)
DDD	The day of the year; for use in Julian dates
DD	The day of the month
D	The day of the week
DAY	The name of the day
HH	The hour of the day (12-hour clock)
HH24	The hour of the day (24-hour clock)
MI	The minutes
SS	The seconds

To change the default date to include a full four-digit year, give the following ALTER SESSION statement:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY'
```

From this point, all INSERT, UPDATE and DELETE statements must format the date accordingly. Also, any SELECT statements will return the date formatted accordingly.

Note that this change only remains in effect for the current session. Logging out of SQL*Plus and logging back in (or reconnecting to the Oracle database using the connect command) will reset the date format back to its default.

Dealing with centuries

The Oracle RDBMS has stores dates using a four-digit year (the 'OraDate' format. To facilitate the year 2000 compliance for applications that use the two-digit year format, Oracle provides a special year format mask **'RR'**. According to Oracle, the date format should be NLS_DATE_FORMAT = 'YY-MON-**RR'** in order to overwrite the default of 'YY-MON-**YY'**. Using the 'RR' format, any two-digit year entered will be converted thus:

Current Year: Last Two Digits	**Two-digit Year Specified**	**Year 'RR' Format Returns**
0–49	0–49	Current Century
50–99	0–49	Current century – 1
0–49	50–99	Current century + 1
50–99	50–99	Current Century

The 'RR' date format is available for inserting and updating DATE data in the database but it is not needed for retrieval/query of data already stored in the database as Oracle always stores the YEAR component of a date in its four-digit form. Therefore, irrespective of the current century, the 'RR' format will ensure that the year stored in the database is as follows:

Current Year	Specified Date	RR Format	YY Format
2005	20-AUG-14	2014	2014
2005	20-AUG-95	1995	2095
2060	20-AUG-14	2114	2014
2060	20-AUG-95	2095	2095

Aggregate functions or group functions

To compute a single summary value from a column of individual values aggregate functions or group functions are used. Aggregate functions supported by oracle are described as follows:

Aggregate Functions	Description
AVG([DISTINCT|ALL]column_name)	Average value for a group of rows.
COUNT([DISTINCT|ALL]column_name)	Counts the number of rows where the expression is not null. If you use * in the select clause, then all rows are counted whether they are null or not.
MIN([DISTINCT|ALL]column_name)	Minimum of all values for a group of rows.
MAX([DISTINCT|ALL]column_name)	Maximum of all values for a group of rows.
SUM([DISTINCT|ALL]column_name)	Sum of all values for a group of rows.

Apart from the above functions, there are two more: STDDEVand VARIANCE. The basic group functions are AVG, COUNT, MIN, MAX and SUM. Their use is uncomplicated. These functions take two options: DISTINCT and ALL. ALL is default.

DISTINCT	This causes a group function to consider only distinct values in the argument expression.
ALL	This causes a group function to consider all values including all duplicates.

All group functions except COUNT(*) ignore nulls. NVL is used in the argument to a group function to substitute a value for a null. If a query with a group function returns no rows or only rows with nulls for the argument to the group function, the group function returns a null.

Query: Compute the average salary of all employees listed in the EMP table.

```
SELECT AVG(salary) FROM emp;
```

Output:

```
AVG(SALARY)
-----------
 4722.22222
```

Query: Find the minimum and maximum salaries listed in the emp table.

```
SELECT MIN(salary), MAX(salary) FROM emp;
```

Output:

```
MIN(SALARY)             MAX(SALARY)
-----------             -----------
 3000                   6500
```

Query: Find the total salary amount paid to employees per month in Department 'D01'.

```
SELECT SUM(salary) FROM emp WHERE dno = 'D01';
```

Output:

```
 SUM(SALARY)
-----------
  16500
```

Query: Find out how many employees are given in the EMP table.

```
SELECT COUNT(*) FROM emp;
```

Output:

```
    COUNT(*)
-----------
      9
```

DISTINCT is used with the COUNT function to count only distinct rows.

Query: Find how many different designations are listed in the EMP table.

```
SELECT COUNT(DISTINCT desg) FROM emp;
```

Output:

```
COUNT(DISTINCTDESG)
-----------
        6
```

Notes:

- Aggregate functions except **COUNT**(*) ignore null values in their input collection. Consequently, the collection of values is empty. The **COUNT** of an empty set is defined to be 0, and all other aggregate operations return a value of null when applied on an empty collection.
- Remember that an aggregate function cannot be used directly with the WHERE clause.

```
    SELECT * FROM emp WHERE salary=MAX(salary);
```

Output:

```
SELECT * FROM emp WHERE salary=MAX(salary)
                  *
ERROR at line 1:
ORA-00934: group function is not allowed here.
SELECT * FROM emp WHERE AVG(salary)>3000;
```

Output:

```
SELECT * FROM emp WHERE AVG(salary)>3000
                              *
ERROR at line 1:
ORA-00934: group function is not allowed here.
```

NOTES

Group functions might not be listed with column names in the SELECT clause (unless the GROUP BY clause is used, which is discussed ahead). For example, to get the name of the employee who is paid the lowest salary, the query,

```
SELECT ename, MIN(salary) FROM emp;
```

will generate an error.

```
SELECT Mename, MIN(salary) FROM emp
       *
```

ERROR at line 1:

ORA-00937: not a single-group group function

A subquery will return the desired information, which is discussed later in this unit.

Interestingly, in the ORDER BY clause, group function may be specified.

For example:

```
SELECT dno, COUNT (*)
FROM emp
GROUP BY dno
ORDER BY COUNT (*) DESC;
```

Output:

DNO	**COUNT(*)**
D01	4
D02	3
D03	2

It should be noted that if a column in the ORDER BY clause is specified that is not a part of group function, it must be in the GROUP BY clause.

DISTINCT IN GROUP FUNCTION

All group value functions have a DISTINCT versus ALL options. COUNT provides good example of how this works.

```
SELECT COUNT (dno) FROM emp;
```

Output:

```
 COUNT(DNO)
-----------
      9
```

SELECT COUNT (DISTINCT dno) FROM emp;

NOTES

Output:

```
COUNT(DISTINCTDNO)
------------------
        3
```

It is clearly seen that DISTINCT forces COUNT to count only the number of unique department member.

Group Query—GROUP BY clause

The GROUP BY clause basically allows us to partition the results of a query into groups with similar characteristics based upon predicate satisfaction. The columns named in the GROUP BY clause are called the *grouping columns.* The ISO standard requires the SELECT clause and the GROUP BY clause to be closely integrated. When GROUP BY is employed, each item in the SELECT list has to be single-valued per group. Further, the SELECT clause may contain only column names, aggregate functions, constants and an expression involving combinations of these.

The GROUP BY clause has the following general format:

```
GROUP BY <column_name1> [, <column_name2>] ............
```

Column_name1 and column_name2 are the grouping columns. They have to be names of columns from tables in the FROM clause, not expressions.

GROUP BY operates on the rows from the FROM clause as filtered by the WHERE clause. GROUP BY assumes a *null* as distinct from every other *null* and so each row, which has a *null* in one of its grouping columns, forms a separate group.

Query: List the total amount paid in salary to each department.

```
SELECT SUM(salary) FROM emp GROUP BY dno;
```

Output:

```
SUM(SALARY)
-----------
  16500
  16000
  10000
```

The GROUP BY clause causes a select statement to produce one summary row for all selected rows that have the identical value as specified in the GROUP BY clause. The column referred to in the GROUP BY clause does not have to be in the select clause. However, it is not known with which group the returned data is associated, if the column referred to in the GROUP BY clause is not listed in the SELECT clause.. Without the DNO column in the SELECT clause, we have data, but it is not known which department is associated with each row of data. To solve this problem, write the query as follows:

```
SELECT dno, SUM(salary) FROM emp GROUP BY dno;
```

DNO	SUM(SALARY)
D01	16500
D02	16000
D03	10000

This results table furnishes information that is helpful.

Query: Find the number of employees in each department from the EMPLOYEE table.

```
SELECT dno, COUNT(*) FROM emp GROUP BY dno;
```

Output:

DNO	COUNT(*)
D01	4
D02	3
D03	2

Multiple aggregate functions may be used in the SELECT clause in association with the GROUP BY clause.

Query: Find the minimum salary, maximum salary and the average salary for each department.

```
SELECT dno, MIN(salary),
MAX(salary), AVG(salary) ← group functions in select clause
FROM emp
GROUP BY dno; ← GROUP BY clause
```

Output:

DNO	MIN(SALARY)	MAX(SALARY)	AVG(SALARY)
D01	3000	5000	4125
D02	4000	6500	5333.33333
D03	4000	6000	5000

Multiple columns may also be used in the GROUP BY clause.

Query: Find the number of designation in each department.

```
SELECT dno, desg, COUNT(*) FROM emp GROUP BY dno, desg;
```

Output:

DNO	DESG	COUNT(*)
D01	PROGRAMMER	2
D01	PROJECT ASSISTANT	1
D01	SYSTEM ANALYST	1
D02	PROGRAMMER	2
D02	SYSTEM MANAGER	1
D03	ACCOUNTANT	1
D03	CLERK	1

7 rows selected.

NOTES

- All the column names in the SELECT list must show in the GROUP BY clause unless the name is used only in an aggregate function; otherwise, SQL generates the following error. The contrary is not true, however, as there may be column names in the GROUP BY clause, which may not appear in the SELECT list. Syntactically it is:
  ```
  SELECT dno, desg, COUNT(*) FROM emp GROUP BY dno;
  ```
  ORA-00979: not a group by expression
- If anyone wants to use aggregate expression in a SELECT list without the GROUP BY clause, then the SELECT list must consist only of aggregate function.

When the WHERE clause is applied in a grouped query, then the groups are formed from the residual rows that satisfy the search condition.

Query: Count the designation wise number of employees of the department 'D01'.

```
SELECT desg, COUNT(*) FROM emp WHERE dno='D01'GROUP BY
desg;
```

Output:

DESG	COUNT(*)
PROGRAMMER	2
PROJECT ASSISTANT	1
SYSTEM ANALYST	1

It should be noted that the WHERE clause must come into view before the GROUP BY clause.

The ISO standard stipulates two nulls to be equal for purposes of the GROUP BY clause. Thus, if two rows have nulls in the same grouping columns and identical values in all the non-null grouping columns, they are combined into the same group.

Combining group functions and arithmetic functions

Suppose we want to find out the total salary paid out per month to all employees listed in the emp table, and also want to find out the total monthly salary to be paid if every employee is given a 3 per cent raise in salary. A single SQL query would give this information.

```
SELECT SUM(salary), SUM(salary * 1.03) FROM emp;
```

Output:

```
        SUM(SALARY)                 SUM(SALARY*1.03)

           42500                        43775
```

We may choose names for columns in the results by including the names in the SELECT clause as illustrated below:

```
SELECT SUM(salary) NOW, SUM(salary * 1.03) WITHRAISE
FROM emp;
```

This query returns the following results table:

```
               NOW                            WITH RAISE

               42500                            43775
```

Be cautious when naming columns in results tables. Reserved words must be avoided, and must not include symbols other than the underscore. Otherwise, Oracle will generates an error.

Query: Get the number of SYSTEM ANALYST, PROGRAMMER, PROJECT ASSISTANT, SYSTEM MANAGER in each department.

```
SELECT
 dno
  , SUM(decode(desg, 'SYSTEM ANALYST', 1, 0)) "SYSTEM
ANALYST"
 , SUM(decode(desg, 'PROGRAMMER', 1, 0)) "PROGRAMMER"
 , SUM(decode(desg, 'PROJECT ASSISTANT', 1, 0)) "PROJECT
ASSISTANT"
  , SUM(decode(desg, 'SYSTEM MANAGER', 1, 0)) "SYSTEM
MANAGER"
 FROM emp
 GROUP BY dno;
```

Output:

DNO	SYSTEM ANALYST	PROGRAMMER	PROJECT ASSISTANT	SYSTEM MANAGER
D01	1	2	1	0
D02	0	2	0	1
D03	1	0	0	0

HAVING Clause with GROUP BY

The HAVING clause is intended for use with the GROUP BY clause to confine the groups that come out in the final result. Although HAVING and WHERE are similar in syntax, they serve different purposes. The WHERE clause filters individual rows going into the result table, whereas the HAVING clause filters groups going into the final result. The WHERE clause eliminates rows before to summarization and the HAVING clause The ISO standard stipulates that column names employed in the HAVING clause must also appear in the GROUP BY clause or be restricted within an aggregate function. In practice, the search condition in the HAVING clause always includes at least one aggregate function; otherwise, the search condition could be moved to the WHERE clause and applied to individual rows. Therefore, **the aggregate functions are used in the HAVING clause. However, they cannot be applied in the WHERE clause.**

The general format of the HAVING clause is as follows:

```
HAVING <search_condition>
```

The search condition applies to each group. It can be formed using predicates such as between, in, like, null, comparison, etc., combined with Boolean (AND, OR, NOT) operators. Since the search condition is a grouped table, the predicates should be:

- On a column by which grouping is done
- A group function (Aggregate function) on other columns

If the Having predicate evaluates to true for a grouped or aggregate row, that row is included in the query result; otherwise, the row is skipped.

As for example, to find which departments have at least three employees, perform the above query with the HAVING clause. We need to return only the rows with more than two employees per department. That query would be as follows:

```
SELECT dno, COUNT(*) FROM emp GROUP BY dno HAVING COUNT(*)
> 2;
```

Output:

```
DNO        COUNT(*)
D01           4
D02           3
```

Note:

- If an expression is used in the HAVING clause that is not in the SELECT list or that is not an aggregate function, then Oracle will generate an error.
- Oracle does not care whether the HAVING clause is before GROUP BY clause or after. GROUP BY clause can be specified before HAVING clause and vice versa.

Identifying duplicate rows using GROUP BY and HAVING

It is feasible to establish duplicate rows using the SELECT with a count of all the rows that have the same values in the 'unique' fields as given here.

```
SELECT a,b,count(*)
from test
group by a,b
having count(*) > 1;
```

Order of Execution of WHERE/GROUP BY/HAVING/ORDER BY

- WHERE clause.
- GROUP BY clause.
- Group functions for each group.
- Elimination of groups based on the HAVING clause.
- ORDER BY clause. The ORDER BY clause has to use either a group function or a column given in the GROUP BY clause.

The order of execution is essential, because it has a direct bearing on the task of the queries. In general, the more the records that can be eliminated via the WHERE clause, the faster the execution of the query due to the decline in the number of rows that should be processed at the time of the GROUP BY operation.

PROJECT and ASSIGN Tables

In addition to emp and DEPT tables, we will discuss two more tables: PROJECT and ASSIGN tables.

PROJECT

PID	VARCHAR2(5)
PNAME	VARCHAR2(30)
LOCATION	VARCHAR2(30)

Instance of the PROJECT table:

PID	PNAME	LOCATION
P01	HOSPITAL MANAGEMENT SYSTEM	KOLKATA
P02	ACCOUNTING SYSTEM	KOLKATA
P03	BANKING INFORMATION SYSTEM	CHENNAI

ASSIGN

ECODE	VARCHAR2(5)
PID	VARCHAR2(5)

Instance of the ASSIGN table:

ECODE	PID
E01	P01
E01	P02
E01	P03
E02	P01
E08	P01
E02	P02
E08	P02
E07	P02
E03	P03
E09	P03
E06	P03

2.5.7 Joins in SQL

A **join** is a query that retrieves rows from more than one table or view. More than one table can be specified in the FROM clause. Sometimes, more than one table may have the same column name. When we use these tables for join, this causes a problem with Oracle; it does not identify the table to which the column pertains. When this occurs, Oracle will terminate the query with the following error message:

'Column ambiguously defined'

To overcome this problem, there are two methods of qualifying a column:

- The column name should be preceded with the table name to qualify all references to these columns throughout the query with table names to avoid vagueness. Two names should be separated by a period, e.g., emp.dno.

- The alternate way is to define the temporary labels in the FROM clause and use them somewhere else in the query. These types of temporary labels are sometimes referred to as table aliases. There are the following two ways to specify a column alias:
  - ▪ Naming the alias after the column specification separated by a space.

    Example:
    ```
    SELECT ecode id, ename name FROM employee
    ```
  - ▪ Use of 'AS' word to specify the alias more clearly

    Example:
    ```
    SELECT ecode AS id, ename AS name FROM employee
    ```

Like before, alias name should be used with a dot before column name, e.g., table_alias_name.column_name.

The advantage of the second method is that it allows the developer to use a shorter custom name for the table. The table alias and the second method is a must for self-join where a table is joined with itself. In that case, both table and column names are the same.

Join conditions

Most of the join queries contain the WHERE clause conditions that compare two columns, each from a different table. Such a condition is referred to as **join condition**. For executing a join, Oracle combines pairs of rows, each containing one row from each table, for which the join condition evaluates to TRUE. The columns in the join conditions need not appear in the select list.

In pre-Oracle 9i, the join condition is specified using the WHERE clause. In Oracle 9i, it is supported by the ANSI SQL-99 syntax. The following join types are available:

- *Cross join*
- *Natural join*
- *Join with USING clause*
- *Join with ON clause*
- *Outer join (right, left and full)*

The order of tables in the FROM clause does not matter. Also, the order of comparisons is insignificant.

(*i*) *Cartesian product cross join*

If two tables in a join query have no join condition, their **Cartesian product is returned**. It is also known as cross-join. Each row of one table is combined with each row of the other. Consider the following query illustrating a Cartesian product:

```
SELECT ecode, ename, salary, dname
FROM emp, dept
```

NOTES

Output:

```
ECODE   ENAME               SALARY   DNAME
E01     KOUSHIK GHOSH       5000     PROJECT
E02     JAYANTA DUTTA       3500     PROJECT
E03     HARI NANDAN TUNGA   4000     PROJECT
E04     JAYANTA GANGULY     6000     PROJECT
E05     RAJIB HALDER        4000     PROJECT
E06     JISHNU BANERJEE     6500     PROJECT
E07     RANI BOSE           3000     PROJECT
E08     GOUTAM DEY          5000     PROJECT
E09     PINAKI BOSE         5500     PROJECT
E01     KOUSHIK GHOSH       5000     RESEARCH
E02     JAYANTA DUTTA       3500     RESEARCH
E03     HARI NANDAN TUNGA   4000     RESEARCH
E04     JAYANTA GANGULY     6000     RESEARCH
E05     RAJIB HALDER        4000     RESEARCH
E06     JISHNU BANERJEE     6500     RESEARCH
E07     RANI BOSE           3000     RESEARCH
E08     GOUTAM DEY          5000     RESEARCH
E09     PINAKI BOSE         5500     RESEARCH
E01     KOUSHIK GHOSH       5000     PERSONNEL
E02     JAYANTA DUTTA       3500     PERSONNEL
E03     HARI NANDAN TUNGA   4000     PERSONNEL
E04     JAYANTA GANGULY     6000     PERSONNEL
E05     RAJIB HALDER        4000     PERSONNEL
E06     JISHNU BANERJEE     6500     PERSONNEL
E07     RANI BOSE           3000     PERSONNEL
E08     GOUTAM DEY          5000     PERSONNEL
E09     PINAKI BOSE         5500     PERSONNEL
```

27 rows selected.

A Cartesian product contains many rows of no practical interest. There are 9 rows in EMP and 3 rows in DEPT; this will result in 9 times 3 (i.e., 27) rows.

In Oracle 9i, the **cross-join** represents the Cartesian product of two or more tables selected without join conditions. The same output as above will be given.

```
SELECT ecode,ename, salary, dname
FROM emp CROSS JOIN dept;
```

It is important to note here that a Cartesian product of two tables is in practice required rarely.

Equijoins

An **equijoin** is a join in which the join condition contains an equality operator. In pre-Oracle 9i, the join condition is specified in the WHERE clause. In Oracle 9i, join predicates can also be defined with ON.

Query: List the employee code, name and department name of each employee.

Using the WHERE clause	Using the ON clause

`SELECT ecode, ename, dname`
`FROM emp, dept`
`WHERE emp.dno=dept.dno;`
	`SELECT ecode, ename, dname`
	`FROM emp JOIN dept`
	`ON emp.dno=dept.dno;`

Output:

ECODE	ENAME	DNAME
E01	KOUSHIK GHOSH	PROJECT
E02	JAYANTA DUTTA	PROJECT
E03	HARI NANDAN TUNGA	RESEARCH
E04	JAYANTA GANGULY	PERSONNEL
E05	RAJIB HALDER	PERSONNEL
E06	JISHNU BANERJEE	RESEARCH
E07	RANI BOSE	PROJECT
E08	GOUTAM DEY	PROJECT
E09	PINAKI BOSE	RESEARCH

9 rows selected.

When we join the table **dept** to the table **emp**, the join condition contains the equality operator. Such joins are known as **equijoins**.

The above query may be written as follows:

```
SELECT ecode, ename, dname
FROM emp e, dept d
WHERE e.dno = d.dno;
```

OR

```
SELECT ecode, ename, dname
FROM emp e JOIN dept d
ON e.dno=d.dno
```

The table **emp** is given an alias e, and **dept** an alias d. Generally, alias or table name may be omitted if the column names are unique.

All other conditions could also be defined in the ON clause. For example:

```
SELECT ename, d.dno, d.dname
 FROM emp e INNER JOIN dept d
 ON (e.dno = d.dno AND desg = 'PROGRAMMER');
```

Output:

ENAME	DNO	DNAME
JAYANTA DUTTA	D01	PROJECT
HARI NANDAN TUNGA	D02	RESEARCH
GOUTAM DEY	D01	PROJECT
PINAKI BOSE	D02	RESEARCH

However, it is better to separate the join condition from the restricting conditions (WHERE). The following statement would, therefore, be better:

```
SELECT e.ename, d.dno, d.dname
FROM emp e INNER JOIN dept d ON (e.dno = d.dno)
WHERE desg = 'PROGRAMMER';
```

USING clause

In the example just discussed, the join column was specified in the ON clause. Specifying the join condition can be simplified by the USING clause if the following conditions are satisfied:

- The join depends on the equality condition between two columns or between sets of two columns to relate the rows from the two tables.
- The columns have the same name and data type in both tables.

The following query

```
SELECT ecode, ename, dname
FROM emp JOIN dept
ON emp.dno=dept.dno;
```

can be rewritten as:

```
SELECT ecode, ename, dname
FROM emp JOIN dept
USING(dno);
```

The USING clause, however, does subtly affect the semantics of the query. It is to be remembered that if a join column is included in the SELECT list, Oracle does not allow qualifying that column with a table name or tabling alias resulting in an error message.

```
SELECT ecode, ename, dept.dno,dname
FROM emp JOIN dept
USING(dno);
```

Output:

```
SELECT ecode, ename, dept.dno,dname
 *
ERROR at line 1:
ORA-25154: column part of USING clause cannot have qualifier
```

Any join that does not use the equality operator(=) is known as a **non-equijoin**. For example:

Using the WHERE clause	**Using the ON clause**

SELECT ecode, ename, salary, dname
FROM emp,dept
WHERE salary > 5000;
	SELECT ecode, ename, salary, dname
	FROM emp JOIN dept
	ON salary > 5000;

NOTES

Output:

ECODE	ENAME	SALARY	DNAME
E04	JAYANTA GANGULY	6000	PROJECT
E06	JISHNU BANERJEE	6500	PROJECT
E09	PINAKI BOSE	5500	PROJECT
E04	JAYANTA GANGULY	6000	RESEARCH
E06	JISHNU BANERJEE	6500	RESEARCH
E09	PINAKI BOSE	5500	RESEARCH
E04	JAYANTA GANGULY	6000	PERSONNEL
E06	JISHNU BANERJEE	6500	PERSONNEL
E09	PINAKI BOSE	5500	PERSONNEL

9 rows selected.

Some more examples of equijoins are as follows:

Query: List the code, name of the employees with their corresponding project ids in which they are assigned.

Using the WHERE clause	Using the ON clause

`SELECT e.ecode, ename, a.pid`
`FROM emp e,assign a`
`WHERE e.ecode=a.ecode;`
	`SELECT e.ecode, ename, a.pid`
	`FROM emp e JOIN assign a`
	`ON e.ecode=a.ecode;`

Output:

ECODE	ENAME	PID
E01	KOUSHIK GHOSH	P01
E01	KOUSHIK GHOSH	P02
E01	KOUSHIK GHOSH	P03
E02	JAYANTA DUTTA	P01
E08	GOUTAM DEY	P01
E02	JAYANTA DUTTA	P02
E08	GOUTAM DEY	P02
E07	RANI BOSE	P02
E03	HARI NANDAN TUNGA	P03
E09	PINAKI BOSE	P03
E06	JISHNU BANERJEE	P03

11 rows selected.

Query: List the names of the projects and department names located in the same city.

Using the WHERE clause	Using the ON clause

`SELECT pname, dname`
`FROM project p, dept d`
`WHERE p.location=d.city;`
	`SELECT pname, dname`
	`FROM project p JOIN dept d`
	`ON p.location=d.city;`

Output:

PNAME	DNAME
BANKING INFORMATION SYSTEM	RESEARCH
HOSPITAL MANAGEMENT SYSTEM	PROJECT
ACCUNTING SYSTEM	PROJECT
HOSPITAL MANAGEMENT SYSTEM	PERSONNEL
ACCUNTING SYSTEM	PERSONNEL

Apart from the join conditions, the WHERE clause or the ON clause can also contain other conditions that refer to columns of only one table.

Query: List the employee code, name and department name of each employee getting salary more than Rs 5000.

Using the WHERE clause	Using the ON clause

`SELECT ecode, ename, dname`
`FROM emp, dept`
`WHERE emp.dno=dept.dno`
`AND salary>5000;`
	`SELECT ecode, ename, dname`
	`FROM emp JOIN dept`
	`ON emp.dno=dept.dno`
	`AND salary>5000;`

Output:

ECODE	ENAME	DNAME
E04	JAYANTA GANGULY	PERSONNEL
E06	JISHNU BANERJEE	RESEARCH
E09	PINAKI BOSE	RESEARCH

Query: List the employee code, name and department name of all PROGRAMMERS.

Using the WHERE clause	Using the ON clause

`SELECT ecode, ename, dname`
`FROM emp, dept`
`WHERE emp.dno=dept.dno`
`AND desg='PROGRAMMER';`
	`SELECT ecode, ename, dname`
	`FROM emp JOIN dept`
	`ON emp.dno=dept.dno`
	`AND salary>5000;`

Output:

ECODE	ENAME	DNAME
E02	JAYANTA DUTTA	PROJECT
E03	HARI NANDAN TUNGA	RESEARCH
E08	GOUTAM DEY	PROJECT
E09	PINAKI BOSE	RESEARCH

Sorting a join

We can use ORDER BY clause also in join.

Query: List the code, name of the employees with their corresponding project ids in which they are assigned in descending order of their employee code.

Using the WHERE clause	Using the ON clause

`SELECT e.ecode, ename, pid`
` FROM  emp e,assign a`
` WHERE e.ecode=a.ecode`
`ORDER BY e.ecode DESC;`
	`SELECT e.ecode, ename, pid`
	` FROM  emp e JOIN assign a`
	` ON e.ecode=a.ecode`
	`ORDER BY e.ecode DESC;`

Output:

```
ECODE           ENAME                      PID
E09             PINAKI BOSE                P03
E08             GOUTAM DEY                 P01
E08             GOUTAM DEY                 P02
E07             RANI BOSE                  P02
E06             JISHNU BANERJEE            P03
E03             HARI NANDAN TUNGA          P03
E02             JAYANTA DUTTA              P01
E02             JAYANTA DUTTA              P02
E01             KOUSHIK GHOSH              P01
E01             KOUSHIK GHOSH              P02
E01             KOUSHIK GHOSH              P03
11 rows selected.
```

Join conditions involving multiple columns

If a join condition consists of multiple columns from each table, it is needed to specify all the predicates in the ON clause or in the WHERE clause. For example, if tables A and B are joined based on columns c1 and c2, the join condition would be as follows:

Using the WHERE clause	Using the ON clause

`SELECT .........`
`FROM A , B`
`WHERE A.c1=B.c1 AND`
`A.c2=B.c2;`
	`SELECT .........`
	`FROM A JOIN B`
	`ON A.c1=B.c1 AND A.c2=B.c2;`

If the column names are identical in the two tables, the USING clause can be used as follows:

```
SELECT .........
FROM A JOIN B
USING(c1,c2);
```

For illustration, say there is a CITY column in the emp table specifying the city in which the employee is staying.

Query: Find the employees who are working in the same city and the dept b.

Using the WHERE clause	Using the ON clause	Using the USING clause
`SELECT ecode, ename, salary FROM emp a, dept b WHERE a.city=b.city AND a.dno=b.dno;`	`SELECT ecode, ename, salary FROM emp a JOIN dept b ON a.city=b.city AND a.dno=b.dno;`	`SELECT ecode, ename, salary FROM emp a JOIN dept b USING(city,dno);`

Joining multiple tables

In a join, more than two tables can participate. This is basically an extension of a two-table join. Three tables, A, B and C can be joined in the following ways:

- A joins B which joins C
- A joins B and the join of A and B joins C
- A joins B and A joins C

Besides the above, several other variations are available. Oracle first joins two of the tables that are based on the join conditions, comparing their columns, and then joins the result to another table based on join conditions containing columns of the joined tables and the new table. Till the point all tables are joined into the result this process continues. The order in which Oracle joins tables based on the join conditions, indexes on the tables, etc. is determined by the optimizer.

Query: Display codes, names of employees with their corresponding assigned project names and locations of the project.

Using the WHERE clause	Using the ON clause
`SELECT e.ecode,ename,pname,location FROM emp e, project p, assign a WHERE e.ecode=a.ecode AND p.pid=a.pid;`	`SELECT e.ecode,ename,pname,location FROM (emp e JOIN assign a ON e.ecode=a.ecode) JOIN project p ON p.pid=a.pid;`

Output:

```
ECODE  ENAME               PNAME       LOCATION
E01    KOUSHIK GHOSH       HOSPITAL    MANAGEMENT SYSTEM KOLKATA
E01    KOUSHIK GHOSH       ACCUNTING   SYSTEM KOLKATA
E01    KOUSHIK GHOSH       BANKING     INFORMATION SYSTEM CHENNAI
E02    JAYANTA DUTTA       HOSPITAL    MANAGEMENT SYSTEM KOLKATA
E08    GOUTAM DEY          HOSPITAL    MANAGEMENT SYSTEM KOLKATA
E02    JAYANTA DUTTA       ACCUNTING   SYSTEM KOLKATA
E08    GOUTAM DEY          ACCUNTING   SYSTEM KOLKATA
E07    RANI BOSE           ACCUNTING   SYSTEM KOLKATA
E03    HARI NANDAN TUNGA   BANKING     INFORMATION SYSTEM CHENNAI
E09    PINAKI BOSE         BANKING     INFORMATION SYSTEM CHENNAI
E06    JISHNU BANERJEE     BANKING     INFORMATION SYSTEM CHENNAI
```

11 rows selected.

NOTES

It is to be noted that when joining more than two tables, parentheses are used to control the join order. In the absence of parentheses, the joins are processed from left to right. The example above uses parentheses for explicitly specifying the default join order.

The following query will give the same output:

```
SELECT e.ecode, ename, pname, location
FROM emp e JOIN assign a ON e.ecode=a.ecode
JOIN project p ON p.pid=a.pid;
```

However, the following one is incorrect:

```
SELECT e.ecode, ename, pname, location
FROM emp e JOIN assign a JOIN project p
ON p.pid=a.pid
AND e.ecode=a.ecode;
```

Output:

```
AND e.ecode=a.ecode
 *
ERROR at line 4:
ORA-00905: missing keyword
```

Query: Retrieve the code, name and department number of the employees who are working in the project in the same city and the.

```
SELECT DISTINCT E.ecode,ename,E.dno
FROM assigns A,dept D,project P,emp E
WHERE A.pid=P.pid
AND e.ecode=A.ecode
AND D.city=P.location;
```

Output:

```
ECODE      ENAME                 DNO
E01        KOUSHIK GHOSH         D01
E02        JAYANTA DUTTA         D01
E03        HARI NANDAN TUNGA     D02
E06        JISHNU BANERJEE       D02
E07        RANI BOSE             D01
E08        GOUTAM DEY            D01
E09        PINAKI BOSE           D02
```

7 rows selected.

More than two tables can be used in the FROM clause. There is no theoretical limit; however, the system might place some limit. If there are N tables in the FROM clause, then normally (N – 1) join conditions are needed.

(*ii*) *Natural join*

The natural join is based on table columns with the same data and name type. This join automatically integrates into the join condition all columns with the similar data and name type. Here, join condition need not be specified. The SELECT * syntax does not, however, return columns doubly. Obviously, it is particularly Oracle 9i feature. For illustration, a previously discussed query is taken.

Query: List the employee code, name and department name of each employee.

```
SELECT ecode,ename, salary, desg
FROM emp NATURAL JOIN dept;
```

Output:

ECODE	ENAME	SALARY	DESG
E01	KOUSHIK GHOSH	5000	SYSTEM ANALYST
E02	JAYANTA DUTTA	3500	PROGRAMMER
E03	HARI NANDAN TUNGA	4000	PROGRAMMER
E04	JAYANTA GANGULY	6000	ACCOUNTANT
E05	RAJIB HALDER	4000	CLERK
E06	JISHNU BANERJEE	6500	SYSTEM MANAGER
E07	RANI BOSE	3000	PROJECT ASSISTANT
E08	GOUTAM DEY	5000	PROGRAMMER
E09	PINAKI BOSE	5500	PROGRAMMER

9 rows selected.

Table aliases are not used for the join columns in the select list. In case of natural join, Oracle recognizes only one version of each join column.

If we write the above query as follows, it will generate an error:

```
SELECT ecode, ename, salary, d.dno, desg
FROM emp NATURAL JOIN dept;
```

Output:

```
SELECT ecode, ename, salary, d.dno, desg
 *
ERROR at line 1:
ORA-00904: invalid column name
```

As usual, one or more WHERE conditions may follow the join clause:

```
SELECT ename, dno, dname
FROM emp NATURAL JOIN dept
WHERE dno = 'D02';
```

Output:

ENAME	DNO	DNAME
HARI NANDAN TUNGA	D02	RESEARCH
JISHNU BANERJEE	D02	RESEARCH
PINAKI BOSE	D02	RESEARCH

(*iii*) *Join with USING clause and ON clause*

The USING clause is another shortcut for performing an equijoin. The difference between USING and NATURAL is that with the former, the join columns are specified explicitly. This case is also true for join with the ON clause where join condition needs to specify explicitly, i.e., the ON clause is mandatory here. Look at the following query:

```
SELECT ecode, ename, salary, desg
FROM emp JOIN dept;
```

Output:

```
FROM emp JOIN dept
 *
ERROR at line 2:
ORA-00905: missing keyword
```

Inner join in SQL

An **inner join** (sometimes called 'simple join') is a join of two or more tables that returns only those rows that satisfy the join condition.

INNER JOIN keywords can be specified in the FROM clause to perform an inner join between the two tables. The ON clause is used to specify the join conditions.

Query: List the names of each employee with his/her department name.

```
SELECT ename, dname
FROM emp INNER JOIN dept
ON emp.dno=dept.dno;
```

Output:

ENAME	DNAME
KOUSHIK GHOSH	PROJECT
JAYANTA DUTTA	PROJECT
HARI NANDAN TUNGA	RESEARCH
JAYANTA GANGULY	PERSONNEL
RAJIB HALDER	PERSONNEL
JISHNU BANERJEE	RESEARCH
RANI BOSE	PROJECT
GOUTAM DEY	PROJECT
PINAKI BOSE	RESEARCH

9 rows selected.

Other clauses like WHERE and ORDER BY can be used, which come after the FROM clause.

```
SELECT ename, dname
FROM emp INNER JOIN dept
ON emp.dno=dept.dno
WHERE dname='PROJECT'
ORDER BY ename;
```

Output:

ENAME	DNAME
GOUTAM DEY	PROJECT
JAYANTA DUTTA	PROJECT
KOUSHIK GHOSH	PROJECT
RANI BOSE	PROJECT

(*iv*) *Outer joins in SQL*

An inner join does not include rows from either of the table that does not have a matching row in the other table. The basic goal of an outer join is to extend the result of a simple join to include rows from one table that may not have matching rows in another table. This means that an outer join will return all rows that satisfy the join condition and in addition will include the rows from one table and from another table which satisfy the join condition. It is also possible that no row has been selected from any table if the join condition is not satisfied. The outer join combines the unmatched row in one of the tables with an artificial row for the other table. This artificial row has all columns set to NULL.

The ANSI syntax identifies three types of outer join: right outer joins, left outer joins and full outer joins. The right and left outer joins are the same thing—all rows from one table are included, along with all matching rows from the other table. The only difference between a right and a left outer join is the order in which the tables are listed by the programmer. The full outer join returns all rows from both the tables. Rows are matched on the join columns wherever possible, and NULLs are used to fill up the empty columns for all rows that do not have a match in the other table.

The pre-Oracle 9i syntax does not provide a full outer join. For this we perform the UNION of left and right outer joins. Oracle 9i allows for ANSI compatible syntax for full outer joins.

Pre-Oracle 9i syntax for outer join

To perform outer join, we have to place a plus sign (+) enclosed by parenthesis after the name of the WHERE clause join condition argument for the table that is deficit in rows (i.e. that has the missing rows). How a join operator is used to perform left outer join, right outer join and full outer join (using UNION) is discussed ahead in details.

Oracle 9i or higher syntax for outer join

The outer join is specified in the FROM clause. It has the following general format:

```
<table_name> { LEFT | RIGHT | FULL } OUTER JOIN <table_name>
ON <predicate>
```

Predicate is a join predicate for the outer join. It can only reference columns from the joined tables. The LEFT, RIGHT or FULL specifiers give the type of join:

- LEFT—only unmatched rows from the left side table are to be retained
- RIGHT—only unmatched rows from the right side table are to be retained
- FULL—unmatched rows from both tables are to be retained

(*a*) ***Left Outer Join***

A left outer join is a join where records from the left table with no matching key in the right table are included in the result set.

Query: List the employee codes, names and project ids assigned to him/her of all employees of the emp table.

In pre-Oracle 9i:

The left table for this discussion is the table on the left side of the join condition and the right table is the table on the right of the join condition. The (+) outer join operator is placed to the right of the right table to signify that all columns returned from the right table corresponding to non-matching records from the left table are returned as NULL values.

```
SELECT E.ecode,ename,pid FROM emp E,assign A
WHERE E.ecode=A.ecode(+);
```

Output:

ECODE	ENAME	PID
E01	KOUSHIK GHOSH	P01
E01	KOUSHIK GHOSH	P02
E01	KOUSHIK GHOSH	P03
E02	JAYANTA DUTTA	P01
E02	JAYANTA DUTTA	P02
E03	HARI NANDAN TUNGA	P03
E04	JAYANTA GANGULY	
E05	RAJIB HALDER	
E06	JISHNU BANERJEE	P03
E07	RANI BOSE	P02
E08	GOUTAM DEY	P01
E08	GOUTAM DEY	P02
E09	PINAKI BOSE	P03

13 rows selected.

In Oracle 9i:

To write a query that performs an outer join of tables A and B and returns all rows from A (a **left outer join**), the LEFT [OUTER] JOIN syntax in the FROM clause can be used. For all rows in A that have no matching rows in B, NULL is returned for any select list expressions containing columns of B.

```
SELECT E.ecode,ename,pid
FROM emp E LEFT OUTER JOIN assign A
ON A.ecode=E.ecode;
```

Output:

ECODE	ENAME	PID
E01	KOUSHIK GHOSH	P01
E01	KOUSHIK GHOSH	P02
E01	KOUSHIK GHOSH	P03
E02	JAYANTA DUTTA	P01
E08	GOUTAM DEY	P01
E02	JAYANTA DUTTA	P02
E08	GOUTAM DEY	P02
E07	RANI BOSE	P02
E03	HARI NANDAN TUNGA	P03
E09	PINAKI BOSE	P03
E06	JISHNU BANERJEE	P03
E05	RAJIB HALDER	
E04	JAYANTA GANGULY	

13 rows selected.

Notice that the PIDs for employees with ecode E04 and E05 are null as they have assigned no projects.

The above query can also be written through the USING clause provided the USING clause cannot have qualifier.

```
SELECT ecode,ename,pid
FROM emp emp LEFT OUTER JOIN assign USING(ecode);
```

(*b*) ***Right Outer Join***

A right outer join is a join where records from the right table with no matching key in the left table are included in the result set. You can rewrite the above query using right outer join.

In pre-Oracle 9i:

The left table for this discussion is the table on the left side of the join condition and the right table is the table on the right of the join condition. The (+) outer join operator is placed to the right of the left table to signify that all columns returned from the left table corresponding to non-matching records in the right table are returned as NULL values.

```
SELECT E.ecode, ename,PID FROM emp E,assign A
WHERE A.ecode(+)=E.ecode;
```

In Oracle 9i:

To write a query that performs an outer join of tables A and B and returns all rows from B (a **right outer join**), the RIGHT [OUTER] JOIN syntax in the FROM clause can be used. For all rows in B that have no matching rows in A, null is returned for any select list expressions containing columns of A.

```
SELECT E.ecode,ename,pid FROM assign A RIGHT OUTER JOIN
emp E
```

```
ON A.ecode=E.ecode;
OR
SELECT ecode,ename,pid FROM assign RIGHT OUTER JOIN emp
USING (ecode);
```

NOTES

Both of the queries give the same output as given in the left outer join because left and right outer joins are similar; the difference is in the ordering of the tables in the FROM clause.

(*c*) ***Full Outer Join***

According to *Wikipedia*, a full outer join combines the results of both the left and right outer joins. The joined table will contain all records from both the tables, and fill in NULLs for missing matches, if any.

To illustrate a full outer join, DEPT table contains a department and if there is no employee in that department, the tuple would not be selected in a query joining the dept and emp tables.

You insert a row in the dept table as follows:

```
INSERT INTO dept VALUES ('D04', 'EDUCATION', 'BANGALORE');
1 row created.
```

Furthermore, in emp table there is an employee to whom no dept has been assigned. For that you insert the following row:

```
INSERT INTO emp VALUES('E10','RAJA SEN',4500,NULL,'JR
PROGRAMMER','12-APR-99');
1 row created.
```

This row also will not appear in the inner join. Now if outer join is performed, result set will contain these newly added rows from both the tables.

Query: List the information of all employees of all departments.

In pre-Oracle 9i:

You can use the outer join on only one side of the join condition. Using the outer join on both sides will cause an error to be issued. Since Oracle will issue an error, the other way to get an outer join is to use UNION.

```
SELECT ecode, ename, D.dno, dname
FROM emp E, dept D
WHERE E.dno(+)=D.dno
UNION
SELECT ecode, ename, E.dno, dname
FROM emp E, dept D
WHERE E.dno=D.dno(+);
```

Output:

ECODE	ENAME	DNO	DNAME
E01	KOUSHIK GHOSH	D01	PROJECT
E02	JAYANTA DUTTA	D01	PROJECT
E03	HARI NANDAN TUNGA	D02	RESEARCH
E04	JAYANTA GANGULY	D03	PERSONNEL

E05	RAJIB HALDER	D03	PERSONNEL
E06	JISHNU BANERJEE	D02	RESEARCH
E07	RANI BOSE	D01	PROJECT
E08	GOUTAM DEY	D01	PROJECT
E09	PINAKI BOSE	D02	RESEARCH
E10	**RAJA SEN**	—	—
E11	—	**D04**	**EDUCATION**

11 rows selected.

Notice the last two rows that are in bold font. The first query gets the employees of all departments including department 'D04' in which no employee has been assigned. The second query gets all the employees including employee 'E10' who has not been assigned to any department. In the case of the first query, any dept without employee would return NULL values for all two columns, ecode and ename. In the case of the second query, any employee without dno would return NULL values for two columns, dno and dname. The UNION puts both together and eliminates any repetitions.

It is to be noted that since 'dno' is a foreign key in the EMP table and a primary key in the DEPT table, the combination of entity integrity and referential integrity tells us that only one of the queries is necessary.

In Oracle 9i:

In Oracle 9*i*, we can use the FULL OUTER JOIN keyword to perform the same thing:

```
SELECT ecode, ename, D.dno, dname
FROM emp E FULL OUTER JOIN dept D
ON E.dno=D.dno;
```

It will give the same output as above.

Self-join in SQL—Joining a Table to itself

A **self-join** is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition.

Query: List the code and name of all the employees who work in the same department as employee code 'E03'.

In this case, the two 'versions' of the emp table must be used, one for employees other than E03, and one for employee E03:

Using the WHERE clause	Using the ON clause

`SELECT x.ecode, x.ename`
`FROM emp x, emp y`
`WHERE x.dno = y.dno`
`AND y.ecode = 'E03'`
`AND x.ecode != 'E03';`
	`SELECT x.ecode, x.ename`
	`FROM emp x JOIN emp y`
	`ON x.dno = y.dno`
	`AND y.ecode = 'E03'`
	`AND x.ecode != 'E03'`

Output:

ECODE	ENAME
E06	JISHNU BANERJEE
E09	PINAKI BOSE

One version of the table **emp** is needed so that we can find the department number of employee E03. In the given example, this table is called **y**. We then look through another version of the table emp, here called **x**, to find people who are in the same department. Finally, we do not want employee code E03 to be displayed, so we should eliminate this case by adding x.empno != 'E03' in the query.

Query: Find out the employees who have exactly the same salary.

Using the WHERE clause	Using the ON clause

`SELECT a.ecode, a.salary`
`FROM emp a, emp b`
`WHERE a.salary=b.salary`
`AND a.ecode<>b.ecode;`
	`SELECT a.ecode, a.salary`
	`FROM emp a JOIN emp b`
	`ON a.salary=b.salary`
	`AND a.ecode<>b.ecode;`

Output:

ECODE	SALARY
E05	4000
E03	4000
E08	5000
E01	5000

The second condition is important. If the above query is written omitting the second condition, it will give the following output:

ECODE	SALARY
E07	3000
E02	3500
E03	4000
E05	4000
E03	4000
E05	4000
E01	5000
E08	5000
E01	5000
E08	5000
E09	5500
E04	6000
E06	6500

13 rows selected.

If the condition is omitted, then it yields duplicate tuples. As the tables are same, obviously salary of any employee must be the same with himself, and then those tuples will appear in the output. The second condition (a.ecode<>b.ecode) will remove those tuples.

Query: Find out the employees in each department getting exactly the same salary.

Using the WHERE clause	Using the ON clause
`SELECT a.ecode, a.ename, a.salary, a.dno FROM emp a , emp b WHERE a.salary=b.salary AND a.dno=b.dno AND a.ecode<>b.ecode;`	`SELECT a.ecode, a.ename, a.salary, a.dno FROM emp a JOIN emp b ON a.salary=b.salary AND a.dno=b.dno AND a.ecode<>b.ecode;`

Output:

ECODE	ENAME	SALARY	DNO
E08	GOUTAM DEY	5000	D01
E01	KOUSHIK GHOSH	5000	D01

Query: Select the pair of employees for each department to whom projects may be assigned.

Using the WHERE clause	Using the ON clause
`SELECT a.ecode, b.ecode, a.dno FROM emp a, emp b WHERE a.dno=b.dno AND a.ecode>b.ecode ORDER BY a.dno;`	`SELECT a.ecode, b.ecode, a.dno FROM emp a JOIN emp b ON a.dno=b.dno AND a.ecode>b.ecode ORDER BY a.dno`

Output:

ECODE	ECODE	DNO
E02	E01	D01
E07	E01	D01
E08	E01	D01
E07	E02	D01
E08	E02	D01
E08	E07	D01
E06	E03	D02
E09	E03	D02
E09	E06	D02
E05	E04	D03

10 rows selected.

The last condition a.ecode>b.ecode may also be written as a.ecode<b.ecode.

Let us illustrate the above query in more details. If the query is written omitting the second condition, it will give the following output:

NOTES

ECODE	ECODE	DNO
E01	E01	D01
E02	E01	D01
E08	E01	D01
E07	E01	D01
E01	E02	D01
E02	E02	D01
E08	E02	D01
E07	E02	D01
E01	E08	D01
E02	E08	D01
E08	E08	D01
E07	E08	D01
E01	E07	D01
E02	E07	D01
E08	E07	D01
E07	E07	D01
E03	E03	D02
E09	E03	D02
E06	E03	D02
E03	E09	D02
E09	E09	D02
E06	E09	D02
E03	E06	D02
E09	E06	D02
E06	E06	D02
E04	E04	D03
E05	E04	D03
E04	E05	D03
E05	E05	D03

29 rows selected.

The output contains some rows implying logically the same combinations, say E01 E02 D01 and E02 E01 D01. Some rows are meaningless as ecodes are combined with themselves like E01 E01 D01.

If the query is written as follows, then the output will be:

Using the WHERE clause	Using the ON clause

`SELECT a.ecode, b.ecode, b.dno`
`FROM emp a, emp b`
`WHERE  a.dno=b.dno`
`AND a.ecode<>b.ecode;`
	`SELECT a.ecode, b.ecode, b.dno`
	`FROM emp a JOIN emp b`
	`ON     a.dno=b.dno`
	`AND a.ecode<>b.ecode;`

ECODE	ECODE	DNO
E02	E01	D01
E08	E01	D01
E07	E01	D01
E01	E02	D01
E08	E02	D01
E07	E02	D01
E01	E08	D01
E02	E08	D01
E07	E08	D01
E01	E07	D01
E02	E07	D01
E08	E07	D01
E09	E03	D02
E06	E03	D02
E03	E09	D02
E06	E09	D02
E03	E06	D02
E09	E06	D02
E05	E04	D03
E04	E05	D03

20 rows selected.

The rows with the same ecode are eliminated. But 10 rows with bold fonts are extraneous because their equivalent rows are already present in the output. To eliminate one of the pair, the following condition has to be specified:

```
a.ecode>b.ecode   or
a.ecode<b.ecode
```

Similarly, you can put the following query.

Query: Get all pairs of employees such that they are working in the same department.

Using the WHERE clause	**Using the ON clause**

`SELECT a.ecode, b.ecode, a.dno`
` FROM  emp a, emp b`
` WHERE a.dno=b.dno`
` AND  a.ecode>b.ecode`
`ORDER BY a.dno`
	`SELECT a.ecode, b.ecode, a.dno`
	` FROM  emp a JOIN emp b`
	` ON a.dno=b.dno`
	` AND  a.ecode>b.ecode`
	`ORDER BY a.dno`

Output:

ECODE	ECODE	DNO
E02	E01	D01
E07	E01	D01
E08	E01	D01
E07	E02	D01
E08	E02	D01
E08	E07	D01
E06	E03	D02
E09	E03	D02
E09	E06	D02
E05	E04	D03

10 rows selected.

Query: Which salary is common to more than one department?

Using the WHERE clause	Using the ON clause

`SELECT a.dno,a.salary`
`FROM  emp a, emp b`
`WHERE a.salary=b.salary`
`AND  a.dno<>b.dno;`
	`SELECT a.dno,a.salary`
	`FROM  emp a JOIN emp b`
	`ON a.salary=b.salary`
	`AND  a.dno<>b.dno;`

Output:

DNO	SALARY
D03	4000
D02	4000

Query: Which salary is common to more than one employee in each department?

Using the WHERE clause	Using the ON clause

`SELECT a.dno,a.salary`
` FROM  emp a ,emp b`
` WHERE a.salary=b.salary`
` AND  a.dno=b.dno`
` AND  a.ecode<>b.ecode;`
	`SELECT a.dno,a.salary`
	` FROM  emp a JOIN emp b`
	` ON a.salary=b.salary`
	` AND  a.dno=b.dno`
	` AND  a.ecode<>b.ecode;`

Output:

DNO	SALARY
D01	5000
D01	5000

2.5.8 Subqueries in SQL

To illustrate the SUBQUERY, you need to consider the instances of the following tables:

EMP table:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

DEPT table:

DNO	DNAME	CITY
D01	PROJECT	KOLKATA
D02	RESEARCH	CHENNAI
D03	PERSONNEL	KOLKATA

PROJECT table:

PID	PNAME	LOCATION
P01	HOSPITAL MANAGEMENT SYSTEM	KOLKATA
P02	ACCOUNTING SYSTEM	KOLKATA
P03	BANKING INFORMATION SYSTEM	CHENNAI

ASSIGN table:

ECODE	PID
E01	P01
E01	P02
E01	P03
E02	P01
E08	P01
E02	P02
E08	P02
E07	P02
E03	P03
E09	P03
E06	P03

So far you have worked with queries that have either a comparison statement or a compound comparison statement in the WHERE clause. You can have a query in the WHERE clause of a SELECT statement. The query in the WHERE clause is called a subquery. The subquery is often referred to as the inner query and the surrounding query is called the outer query.

Subqueries may be written into the following two forms:

- Non-correlated
- Correlated

In a *non-correlated subquery*, the first inner query is executed first. The outer query takes an action based on the results of the inner query. In a *correlated subquery*, the inner query needs values from the outer query and passes results to the outer query. It is important to know that for a simple or correlated subquery, the subquery is executed only once.

Let us illustrate the above forms with an example:

Query: Find the employees who are working in the department located at KOLKATA

NOTES

Using non-correlated subquery, the SQL statement will be as follows:

OUTER QUERY

```
SELECT ecode, ename, dno FROM emp
WHERE dno IN (SELECT dno FROM dept WHERE city='KOLKATA');
```

INNER QUERY

The inner query is independent and gets executed first, passing results to the outer query. Using correlated subquery, the above one can be rewritten as follows:

```
SELECT ecode, ename, dno
FROM emp outer
WHERE dno IN (SELECT dno FROM dept WHERE city = 'KOLKATA'
    AND dno =outer.dno);
```

Output:

ECODE	ENAME	DNO
E01	KOUSHIK GHOSH	D01
E02	JAYANTA DUTTA	D01
E04	JAYANTA GANGULY	D03
E05	RAJIB HALDER	D03
E07	RANI BOSE	D01
E08	GOUTAM DEY	D01

6 rows selected.

Correlated subqueries operate in a repetitive manner. Those familiar with looping control structures of a programming language will see the similarity. In a correlated subquery, the outer query runs producing one test record at a time. For each record that the outer query produces, the inner query is supplied one or more columns from the outer query. The inner query runs when it receives column(s) from the outer query and produces a result set. After the inner query produces a result set, the outer query evaluates the comparison statements containing the inner query. If the comparison statements evaluate favourably, the record produced by the outer query is returned and displayed. This process continues until the outer query tests each test record. Note that a test record is a record that is produced by the outer query by relaxing the conditions containing the inner query.

The full set of test records for this correlated subquery produced is listed below by running the relaxed query. The relaxed query is the original outer query with the condition imposed by the inner query having been removed (remove the inner query and its condition to get the relaxed query).

(Relaxed Query)

```
SELECT ecode, ename, dno FROM emp;
```

NOTES

Test records

ECODE	ENAME	DNO
E01	KOUSHIK GHOSH	D01
E02	JAYANTA DUTTA	D01
E03	HARI NANDAN TUNGA	D02
E04	JAYANTA GANGULY	D03
E05	RAJIB HALDER	D03
E06	JISHNU BANERJEE	D02
E07	RANI BOSE	D01
E08	GOUTAM DEY	D01
E09	PINAKI BOSE	D02

9 rows selected.

How it Works: The steps involved are as follow.

Step 1: First the outer query runs and supplies a column from the first test record to the inner query

The first record is

```
ECODE         NAME                        DNO
---------------------------------------------  ---------------
E01     KOUSHIK GHOSH               D01
```

Column outer.dno = 'D01' is supplied to inner query.

Step 2: Then the outer query shares the outer.dno column value of 'D01' with the inner query and the inner query runs. Let us see the result of the inner query.

```
SELECT dno
FROM dept
WHERE city = 'KOLKATA'
AND dno = 'D01';
```

Column outer.dno is supplied to the inner query. The inner query runs and produces the result set on the right.

```
DNO
-----------
D01
```

Step 3: The inner query can now provide a value to the outer query's WHERE clause. Now the outer query can evaluate the comparison condition in the WHERE clause. Notice that we have to simulate the current test record by adding the first and last name conditions to *freeze* the outer query. Let us run it and see what we get.

```
SELECT ecode, ename, dno
FROM emp outer
WHERE dno = 'D01';
```

Inner query return value replaces the subquery in the outer query evaluation.

This test record satisfies the WHERE clause condition and is returned.

ECODE	ENAME	DNO
E01	KOUSHIK GHOSH	D01

Step 4: The outer query has found a match, so E01's information is returned.

First returned record

ECODE	ENAME	DNO
E01	KOUSHIK GHOSH	D01

Continue Step 1 through Step 4 for each record in the test record set.

As you have seen, a simple subquery is executed once prior to the execution of the outer query. With a correlated subquery, the inner query is executed once for each record that is returned by the outer query. The inner query is driven by a correlation between the outer query and the inner query. The correlation is provided by an exchange of column(s) from the outer query to the inner query. A table alias is used to share the outer query column(s) with the inner query.

The general form of subquery is as follows:

```
SELECT [DISTINCT] <column_list>
FROM   <table_list>
WHERE <column_name> | <expression> {[NOT] IN |
   <comparison_operator >
 [ANY | ALL] | [NOT] EXISTS}
 (SELECT [DISTINCT] <column_list>
 FROM <table_list>
 WHERE <condition>)
GROUP BY <group_by_list>
HAVING   <condition>
ORDER BY <order_by_list>;
```

Types of Subqueries

SQL has the following three basic types of subqueries:

- **Predicate subqueries:** These are extended logical constructs in the WHERE (and HAVING) clause.
- **Scalar subqueries:** These are standalone queries that return a single value; they can be used anywhere a scalar value is used.
- **Table subqueries:** These queries are nested in the FROM clause.

All subqueries should be enclosed in parentheses.

Let us discuss there rule queries in details.

(*i*) **Predicate Subqueries**

Predicate subqueries are used in the WHERE (and HAVING) clause. They may be classified into the following categories:

- Single value subqueries (introduced with relational operator)
- Multiple value subqueries (introduced with IN)
- Quantified subqueries (introduced with comparison operator accompanied by ANY/ALL)
- Subqueries that are an existence test (introduced with EXISTS)

Subqueries returning single value from single column

The following generic SELECT statement depicts the syntax for a simple subquery returning a single value:

```
SELECT <column_list>
FROM <table_list>
WHERE COLUMN <relational_operator> (SELECT statement);
```

(Non-correlated subquery)

Query: List the employees working in the personnel department.

```
SELECT * FROM emp WHERE dno=(SELECT dno FROM dept WHERE
dname='PERSONNEL');
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95

The above query may also be written as follows:

```
SELECT * FROM emp WHERE 'PERSONNEL'=(SELECT dname FROM
dept WHERE dept.dno=emp.dno);
```

Query: Display the information of the employee(s) getting the highest salary.

```
SELECT * FROM emp WHERE salary = (SELECT MAX(salary) FROM
emp);
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96

Query: Retrieve the names and salaries for all employees who earn more than the average salary.

```
SELECT ENAME, SALARY FROM EMP
WHERE SALARY > (SELECT AVG(SALARY) FROM EMP);
```

Output:

ENAME	SALARY
KOUSHIK GHOSH	5000
JAYANTA GANGULY	6000
JISHNU BANERJEE	6500
GOUTAM DEY	5000
PINAKI BOSE	5500

Non-correlated subquery in HAVING—Selecting ONLY the group with the maximum Sum in a group query

Query: Find a designation with the lowest average salary.

```
SELECT desg, AVG (salary)
```

```
    FROM emp
    GROUP BY desg
    HAVING AVG (salary) =
    (SELECT MIN (AVG (salary))
    FROM emp
    GROUP BY desg);
```

Output:

```
            DESG                    AVG(SALARY)
            PROJECT ASSISTANT         3000
```

Query: Display the name of the department and sum of salaries of all employees of that department, which is spending maximum sum of salaries.

```
SELECT a.dname, sum(b.salary)
FROM dept a, emp b
WHERE a.dno = b.dno
GROUP BY a.dname
HAVING SUM(b.salary) = (SELECT MAX(SUM(c.salary))
FROM emp c GROUP BY c.dno);
```

Output:

```
            DNAME                   SUM(B.SALARY)
            PROJECT                   16500
```

Correlated subquery

Query: List the employees who get the same salary as KOUSHIK GHOSH.

```
SELECT * FROM emp A WHERE salary = (SELECT salary FROM
emp B where A.ecode<>B.ecode AND b.ename='KOUSHIK GHOSH');
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97

Query: List the details of the employees getting the highest salary in each department.

```
SELECT * FROM emp A WHERE salary=(SELECT MAX(salary) FROM
emp B
WHERE A.dno=B.dno) ORDER BY dno;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96

NOTES

Query: Retrieve the details of department having more than 2 employees.

```
SELECT * FROM dept
WHERE 2 < (SELECT COUNT(*) from emp WHERE dept.dno=emp.dno);
OR
SELECT * FROM dept
WHERE (SELECT COUNT(*) from emp WHERE dept.dno=emp.dno)> 2;
```

Output:

DNO	DNAME	CITY
D01	PROJECT	KOLKATA
D02	RESEARCH	CHENNAI

Listing a certain number of records with the highest values for a certain column using SQL only

There are times where it is needed to simply return the rows with a certain number of the highest (or lowest) values for a certain column. For example:

Query: Display the code and salary of the first three employees getting the highest salaries.

One solution for this problem is to use a correlated subquery to the same table. The following select will return the correct rows:

```
 SELECT ecode, salary FROM emp e1
 WHERE 3 > (SELECT COUNT(*) FROM emp e2 WHERE e1.salary <
e2.salary)
 ORDER BY salary desc ;
```

Output:

ECODE	SALARY
E06	6500
E04	6000
E09	5500

For every row processed by the main query, the correlated subquery returns a count (*COUNT(*)*) of the number of rows with higher salaries (*WHERE e1.salary < e2.salary)*. Then the main query only returns rows that have fewer than three salaries that are higher (*WHERE 3 > ...)*. For example, for ECODE = E09, the salary is '5500'. There are only 2 rows with a higher salary (ECODE = E06, E04), so the subquery returns '2', which is less than 3, causing the 'WHERE 3 > ...' to evaluate to TRUE, thereby returning the row.

- Retrieving nth maximum salary from emp Table

Query: Retrieve the third highest salary of the emp table.

```
SELECT DISTINCT (e1.salary) FROM emp e1 WHERE 3 = (SELECT
COUNT  (DISTINCT  (e2.salary))  FROM  emp  e2  WHERE
e1.salary<=e2.salary);
```

Output:

```
SALARY
5500
```

Query: Display the details of the employee(s) getting the third highest salary.

```
SELECT * FROM emp WHERE salary =( SELECT DISTINCT
(e1.salary) FROM emp e1 WHERE 3 = (SELECT COUNT (DISTINCT
(e2.salary)) FROM emp e2 WHERE e1.salary<=e2.salary));
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

Subqueries returning single value from each of the multiple columns (Oracle 9i feature)

It is possible for a subquery to return more than one column for each row that it returns. In case a simple subquery returns multiple columns, a comparison can be made by creating a list comparing the return values of the subquery and the columns to be compared. The following is the general syntax for a subquery returning multiple columns:

```
SELECT <column_list>
FROM <table_list>
WHERE ( column_name1, …, column_nameK ) = ( SELECT …..);
```

Query: List the details of the employees getting the highest salary in each department.

```
select * from emp
 where (dno, salary)in(select dno, max(salary) from emp
 group by dno);
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96

Correlated subquery in HAVING

Query: Select the department-wise total salary, eliminating all those departments in which total salary was not at least Rs 11000 more than the maximum salary for the department.

```
SELECT dno, SUM(salary) FROM emp A GROUP BY dno
HAVING SUM(salary) > (SELECT 11000+MAX(salary) FROM emp
B
WHERE A.dno=B.dno);
```

Output:

DNO	SUM(SALARY)
D01	16500

Multiple value subquery (introduced with IN)

A subquery can return multiple values. In that case relational operator cannot be used for comparison. Multi-row subqueries require a multi-value comparison operator, such as IN. The IN subquery is used for testing whether a scalar value matches the single query column value in any subquery result row. The general form of such a query is as follows:

```
SELECT <column list>
FROM <table_list>
WHERE <column_name> [NOT] IN (SELECT statement);
```

Note: The comparison operator can use IN or NOT IN because the inner query can return a list. Single value comparison operators such as >, <, <=, >= and = are not allowed.

A. Subqueries returning multiple values from single column

Non-correlated subquery

Query: List the departments having more than 2 employees working in that department.

```
SELECT * FROM dept WHERE dno IN (SELECT dno FROM emp GROUP
BY dno HAVING COUNT(*)>2);
```

Output:

DNO	DNAME	CITY
D01	PROJECT	KOLKATA
D02	RESEARCH	CHENNAI

Query: Retrieve employee details who are not assigned in any project.

```
SELECT * FROM emp
WHERE ecode NOT IN (SELECT DISTINCT ecode FROM assign);
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95

Query: Retrieve department details in which no PROGRAMMER is working.

```
SELECT * FROM DEPT
WHERE dno NOT IN(SELECT DISTINCT dno FROM emp WHERE
desg='PROGRAMMER');
```

Output:

DNO	DNAME	CITY
D03	PERSONNEL	KOLKATA

Correlated subquery

Query: Which designations are common to more than one department?

```
SELECT DISTINCT   A.desg
FROM   emp A WHERE A.desg IN (SELECT B.desg
FROM emp B WHERE A.dno <> B.dno);
```

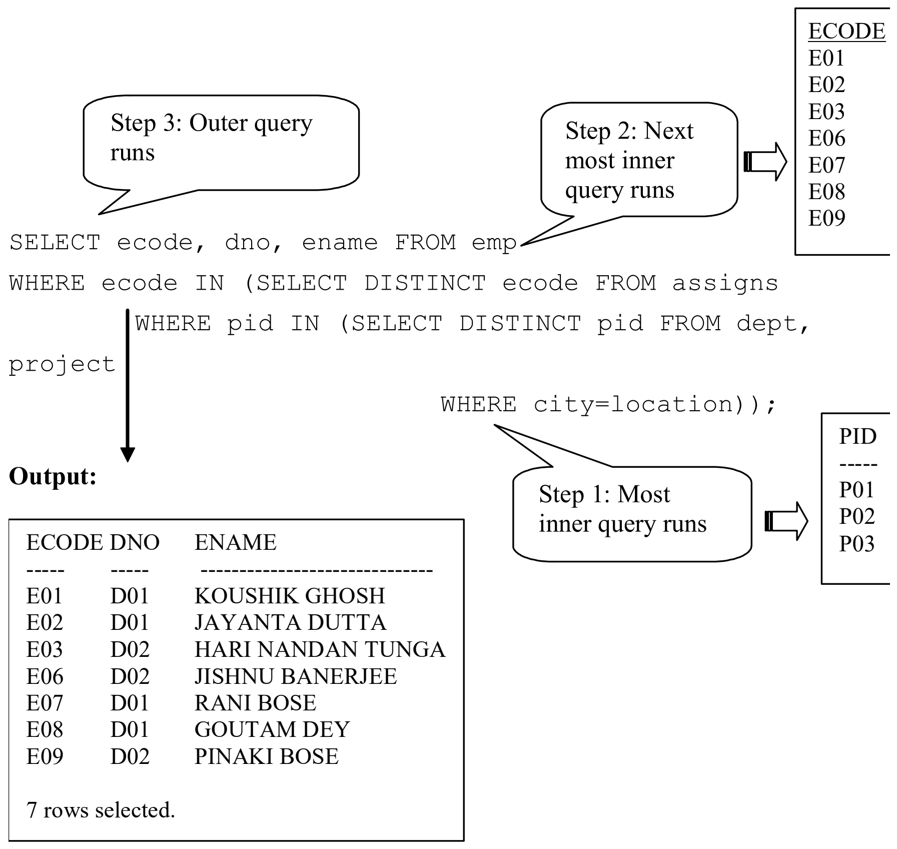
Output:

```
DESG
PROGRAMMER
```

Nested subquery

It is possible to nest subqueries but it is not a practice that makes sense beyond three or four levels deep. The following example demonstrates a nested subquery. Both subqueries are multi-row subqueries.

Query: Retrieve the code, department number and name of each employee who are working in the project located in the same city as their departments.


Step 3: Outer query runs
Step 2: Next most inner query runs
Step 1: Most inner query runs

```
ECODE
E01
E02
E03
E06
E07
E08
E09
```

```
SELECT ecode, dno, ename FROM emp
WHERE ecode IN (SELECT DISTINCT ecode FROM assigns
        WHERE pid IN (SELECT DISTINCT pid FROM dept,
project
                              WHERE city=location));
```

```
PID
-----
P01
P02
P03
```

Output:

```
ECODE DNO    ENAME
-----  -----  ------------------------------
E01    D01    KOUSHIK GHOSH
E02    D01    JAYANTA DUTTA
E03    D02    HARI NANDAN TUNGA
E06    D02    JISHNU BANERJEE
E07    D01    RANI BOSE
E08    D01    GOUTAM DEY
E09    D02    PINAKI BOSE

7 rows selected.
```

Query: List the department details having maximum number of employees.

```
SELECT * FROM dept
WHERE dno IN(SELECT dno FROM emp GROUP BY dno
HAVING COUNT(*) = (SELECT MAX(COUNT(*)) FROM emp
GROUP BY dno));
```

Output:

DNO	DNAME	CITY
D01	PROJECT	KOLKATA

Query: List the employees who are working on all the projects.

```
SELECT * FROM emp WHERE ecode IN( SELECT ecode FROM
assign GROUP BY ecode HAVING COUNT(ecode)=(SELECT
COUNT(pid) FROM project));
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93

B. Subqueries returning multiple values from the multiple columns (Oracle 9i feature)

Multi-row subquery syntax for multiple columns is as follows:

```
SELECT <column_list>
FROM <table_list>
WHERE ( <column_name1>, …, <column_nameK> ) IN ( SELECT
…..);
```

Query: Retrieve the code, name and department number of the employees who are working in the project in the same city as their department.

```
SELECT DISTINCT E.ecode, ename, E.dno
FROM emp E, dept D, assigns A
WHERE E.dno=D.dno
AND E.ecode=A.ecode
AND (pid,city) IN (SELECT pid,location FROM project);
```

Output:

ECODE	ENAME	DNO
E01	KOUSHIK GHOSH	D01
E02	JAYANTA DUTTA	D01
E03	HARI NANDAN TUNGA	D02
E06	JISHNU BANERJEE	D02
E07	RANI BOSE	D01
E08	GOUTAM DEY	D01
E09	PINAKI BOSE	D02

7 rows selected.

NOTES

If we use the equality operator(=) instead of the IN operator in the third predicate, then Oracle generates an error as inner subquery returns multiple rows.

```
SELECT DISTINCT E.ecode, ename, E.dno
FROM emp E, dept D, assigns A
WHERE E.dno=D.dno
AND E.ecode=A.ecode
AND (pid,city) = (SELECT pid, location FROM project)
```

Output:

```
AND (pid,city) = (SELECT pid, location FROM project)
 *
ERROR at line 5:
ORA-01427: single-row subquery returns more than one row
```

- **Quantified subqueries (introduced with comparison operator accompanied by ANY/ALL)**

Subqueries introduced with the comparison operator and ANY/ALL imply that all the values resulting from the inner query must be taken into account. ANY, SOME and ALL are multi-valued comparison operators that allow the use of single-valued comparison operators, such as >, <, <=, >= and = on a list of values. These comparison operators can be used in any multi-row subquery.

Syntax

```
SELECT <select_list>
WHERE <expression> <relational_operator> [ANY | ALL]
(subquery);
ALL
```

ALL works with a comparison operator and a list. The ALL comparison operator is true whenever all the values, produced by the subquery, compare favourably with the column being compared.

>ALL will include all the greater than values which are returned from the subquery, and then it will select–the greatest of the largest value from the values returned from the inner query.

>ALL (1, 2, 3)   implies   >3

Similarly, the less than condition will include all the values and will return the smallest of them. <ALL (1, 2, 3)   implies   <1

=ALL(1, 2, 3) means =1 AND =2 AND =3, which is meaningless. ALL is used basically with inequalities rather than equalities because a value cannot be ‘equal to all’ the values. A value can be equal to all of the results of a subquery if all the said results are infact identical.

Query: Find the employee getting more salary than every employee of the ‘PROJECT’ department.

```
SELECT * FROM emp
WHERE salary>ALL(SELECT salary FROM emp
WHERE dno=(SELECT dno FROM dept WHERE dname = ‘PROJECT’));
```

NOTES

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E04	AYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

Query: Find the details of the employee(s) who is/are getting the highest salary.

It can be written as using MAX( ) function

```
    SELECT * FROM emp WHERE salary = (SELECT MAX(salary)
FROM emp);
```

Using ALL it can be rewritten as:

```
SELECT * FROM emp
WHERE sal>=ALL(SELECT salary FROM emp);
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96

Query: List the department details having the maximum number of employees.

```
SELECT * FROM dept
WHERE dno IN(SELECT dno FROM emp GROUP BY dno
HAVING COUNT(*) >= ALL(SELECT COUNT(*) FROM emp
GROUP BY dno));
```

Output:

DNO	**DNAME CITY**
D01	PROJECT KOLKATA

It is to be noted that if the inner query introduced with ALL and a comparison operator, returns NULL as one of its values, the entire query fails.

ANY (SOME is the same as ANY)

ANY works with a comparison operator and a list. The ANY comparison operator is true whenever one or more of the list elements compares favourably with the column being compared.

>ANY means that greater than at least one value, i.e. greater than the minimum.

>ANY (1, 2, 3) is equivalent to >1

<ANY means that less than the maximum, i.e. <ANY(1,2,3) implies <3

=ANY(1,2,3) means =1 OR =2 OR =3

<> ANY (1, 2, 3) means NOT 1 OR NOT 2 OR NOT 3

The following equivalences are listed:

- >=ALL (LIST) is equivalent to = MAX(LIST)
- <=ALL (LIST)is equivalent to = MIN (LIST)
- <> ANY is equivalent to = NOTIN (LIST)
- =ANY is equivalent to = IN (LIST)

Query: Find the employees getting salary larger than the minimum salary of the PERSONNEL department.

It can be written as follows using MIN() group function:

```
SELECT * FROM emp
WHERE salary> (SELECT MIN(salary) FROM emp
WHERE dno =(SELECT dno FROM dept WHERE dname='PERSONNEL'));
Using ANY query will be
SELECT * FROM emp
   WHERE salary>ANY (SELECT salary FROM emp, dept
   WHERE emp.dno = dept.dno AND dname = 'PERSONNEL');
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

Query: Find the employees getting same salary as the salary of any employee of the PERSONNEL department.

```
 SELECT * FROM emp
 WHERE salary=ANY(SELECT salary FROM emp, dept
 WHERE emp.dno = dept.dno AND dname='PERSONNEL');
OR
 SELECT * FROM emp
 WHERE salary=ANY(SELECT salary FROM emp
 WHERE dno = (SELECT dno FROM dept WHERE dname='PERSONNEL'))
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96

"=ANY" is exactly equivalent to IN. The above query can be replaced as follows:

```
SELECT * FROM emp
WHERE salary IN (SELECT salary FROM emp, dept
WHERE emp.dno = dept.dno AND dname='PERSONNEL');
```

Notice that the records of the PERSONNEL department (with dno='D03') appear in the output. These records should be removed from the result set. Therefore, the query will be as follows:

```
SELECT * FROM emp
WHERE salary=ANY(SELECT salary FROM emp, dept
WHERE emp.dno = dept.dno AND dname='PERSONNEL')
AND dno <>(SELECT dno FROM dept WHERE dname='PERSONNEL');
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95

It is very important to note that "<> ANY" is not equivalent to NOT IN. Consider the following example:

Query: Find the employees who do not work in any project

Look at the ASSIGN table. The employee with codes 'E04' and 'E05' are absent in the ASSIGN table. Obviously, their records will be the output. However, the following query will give all the rows of the EMP table:

```
SELECT * FROM emp
WHERE ecode <>ANY(SELECT DISTINCT ecode FROM assign);
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

Because <> ANY (1, 2, 3) means NOT 1 OR NOT 2 OR NOT 3, and NOT IN (1, 2, 3) implies NOT 1 AND NOT 2 AND NOT 3, the query should be as follows:

```
SELECT * FROM emp
WHERE ecode NOT IN(SELECT DISTINCT ecode FROM assign);
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95

Comparison of ALL & ANY

NOTES

Whenever a legal subquery fails to produce output, ALL is automatically TRUE but ANY is automatically FALSE.

Look at the DEPT table; there is no department at BANGALORE.

Query: Find the employees getting salary larger than the minimum salary of the department located at BANGALORE.

The following query would result in no output:

```
SELECT * FROM emp
WHERE salary>ANY(SELECT salary FROM emp, dept
WHERE emp.dno = dept.dno AND city= 'BANGALORE');
```

Output:

No rows selected

But the following query would produce the entire emp table:

```
SELECT * FROM emp
WHERE salary>ALL(SELECT salary FROM emp, dept
WHERE emp.dno = dept.dno AND city= 'BANGALORE')
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94

- **Subqueries that are an existence test (introduced with EXISTS)**

  The keywords EXISTS and NOT EXISTS are designed for use only with subqueries. They are equivalent to the existential quantifier (and its negated form) that is seen in tuple calculus. Both versions return either true or false only. The EXISTS keyword is a WHERE clause test for the existence or non-existence of data that meets the criteria of the subquery. Existence or non-existence implies presence or absence of the 'empty set' of rows. Even if the subquery returns at least one row, the subquery evaluates to TRUE. The EXISTS operator is used with correlated subqueries. The general format is as follows:

```
SELECT <select_list>
WHERE [NOT] EXISTS (subquery);
```

Any valid EXISTS subquery must contain an *outer reference* (the subquery where clause references a column in the outer query), which means it must be a *correlated subquery*.

Rules with EXISTS

- EXISTS is not preceded by a column name, constant or other expression.
- The SELECT list of the subquery introduced by EXISTS almost always contains an asterisk (*) OR 'X', because we are testing for the existence of rows that meet the subquery conditions.

Query: Find the names of the employees who work in the RESEARCH department.

```
SELECT DISTINCT   ename     FROM emp
WHERE EXISTS
(SELECT * FROM dept
WHERE dept.dno = emp.dno AND    dname = 'RESEARCH');
```

Output:

```
ENAME
HARI NANDAN TUNGA
JISHNU BANERJEE
PINAKI BOSE
```

Query: Which designations are common to more than one department?

```
SELECT DISTINCT desg FROM emp A
WHERE EXISTS( SELECT * FROM emp B
WHERE A.desg=B.desg
AND A.dno<>B.dno);
```

Output:

```
DESG
PROGRAMMER
```

Query: Show all employees names and salaries where there is at least one employee who receives more salary and at least one employee who receives less salary.

```
SELECT ecode, ename, salary
FROM emp
WHERE EXISTS
 (SELECT ename
 FROM emp e2
 WHERE e2.salary > emp.salary)
 AND EXISTS
 (SELECT ename
 FROM emp e3
 WHERE e3.salary < emp.salary);
```

Output:

ECODE	ENAME	SALARY
E01	KOUSHIK GHOSH	5000
E02	JAYANTA DUTTA	3500
E03	HARI NANDAN TUNGA	4000

E04	JAYANTA GANGULY	6000
E05	RAJIB HALDER	4000
E08	GOUTAM DEY	5000
E09	PINAKI BOSE	5500

NOTES

7 rows selected.

NOT EXISTS acts as the reverse of EXISTS.

Query: Retrieve details of employee who are not assigned any project.

```
SELECT * FROM emp
WHERE NOT EXISTS (SELECT * FROM assign WHERE
emp.ecode=assign.ecode);
OR
SELECT * FROM emp
WHERE NOT EXISTS (SELECT 'X' FROM assign WHERE
emp.ecode=assign.ecode);
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95

Query: Show all employees for whom there does not exist an employee who is paid more; in other words, the highest paid employee.

It can be easily written using simple subquery as follows:

```
SELECT *
FROM emp
WHERE salary=(SELECT MAX(salary) from emp);
But using NOT EXISTS it can be written as follows:
SELECT *
FROM emp
WHERE NOT EXISTS
 (SELECT *
 FROM emp e2
 WHERE e2.salary > emp.salary);
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96

(*ii*) Scalar Subqueries

A scalar subquery can appear as a scalar value in the select list and in the WHERE predicate of another query.

For a scalar subquery, the following criteria should be met:

- The subquery should reference just one column in the select list.
- It should also retrieve no more than one row. If the subquery retrieves more than one row, a run-time error is generated and query execution is aborted.

Query: List the code, name of the employees along with the corresponding city of the department in which they are working.

```
SELECT ecode, ename,(SELECT city FROM dept
WHERE emp.dno=dept.dno) " CITY OF DEPT"
FROM emp;
```

Output:

ECODE	ENAME	CITY OF DEPT
E01	KOUSHIK GHOSH	KOLKATA
E02	JAYANTA DUTTA	KOLKATA
E03	HARI NANDAN TUNGA	CHENNAI
E04	JAYANTA GANGULY	KOLKATA
E05	RAJIB HALDER	KOLKATA
E06	JISHNU BANERJEE	CHENNAI
E07	RANI BOSE	KOLKATA
E08	GOUTAM DEY	KOLKATA
E09	PINAKI BOSE	CHENNAI

9 rows selected.

When the subquery does not return a row, a database null is used as the result of the subquery.

Query: Find the employees of the department located at BANGALORE.

```
SELECT ecode, ename,(SELECT city FROM dept
WHERE emp.dno=dept.dno AND city = 'BANGALORE') " CITY OF
DEPT"
FROM emp;
```

Output:

ECODE	ENAME	CITY OF DEPT
E01	KOUSHIK GHOSH	
E02	JAYANTA DUTTA	
E03	HARI NANDAN TUNGA	
E04	JAYANTA GANGULY	
E05	RAJIB HALDER	
E06	JISHNU BANERJEE	
E07	RANI BOSE	
E08	GOUTAM DEY	
E09	PINAKI BOSE	

9 rows selected.

Query: List the employees working in the personnel department.

```
SELECT * FROM emp WHERE 'PERSONNEL'= (SELECT dname FROM
dept WHERE emp.dno=dept.dno);
```

NOTES

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95

If the scalar subquery returns multiple rows, then Oracle generates an error. For example, the listing of the codes and names of the employees along with the corresponding project is, assigned to them will be written as:

```
   SELECT ecode, ename,
   (SELECT pid FROM assign WHERE emp.ecode=assign.ecode)
  "PROJECT ID"
   FROM emp;
```

Output:

```
     (SELECT pid FROM assign WHERE emp.ecode=assign.ecode)
"PROJECT ID"  *
     ERROR at line 2:
     ORA-01427: single-row subquery returns more than one row
```

Here multiple PIDs results from the scalar query. Therefore, the whole query fails.

(*iii*) **Table Subqueries**

Table subqueries are queries used in the FROM clause for replacing a table name. The result set of a table subquery acts like a base table in the FROM list. Table subqueries can have a correlation name in the FROM list. It is also known as *inline view*. They can also be in outer joins.

Query: List the details of the employees along with the corresponding location (city) of the RESEARCH department.

```
   SELECT E.*,city
    FROM emp E, (SELECT dno, city From dept WHERE
  dname='RESEARCH') D
   WHERE E.dno=D.dno;
```

Output:

Ecode	Ename	Salary	DNO	DESG	DT_JN	City of Department
E03	Hari Nandan Tunga	4000	D02	Programer	01-JUL-95	Chennai
E06	Jishnu Banerjee	6500	D02	System Manager	19-SEP-96	Chennai
E09	Pinaki Bose	5500	D02	Programmer	26-AUG-94	Chennai

There is another application of an inline query. If it is tried to use ROWNUM to restrict a query by a range that does not start with 1, it does not work. For example:

```
  SELECT * from emp WHERE ROWNUM BETWEEN 5 AND 9 ;
```

Output:

```
     no rows selected
```

The reason for this is that ROWNUM is a psuedo-column produced AFTER the query returns. Normally, it can only be used to restrict a query to return a rownumber range that starts with 1 (like *rownum < 5*). An ‘Inline View’ can be used to get around this limitation.

```
SELECT rn, ecode,ename,salary
FROM (SELECT ROWNUM rn,ecode,ename,salary
FROM emp)
WHERE rn BETWEEN 5 AND 9;
```

Output:

```
   RN ECODE          ENAME                SALARY
    5 E05            RAJIB HALDER          4000
    6 E06            JISHNU BANERJEE       6500
    7 E07            RANI BOSE             3000
    8 E08            GOUTAM DEY            5000
    9 E09            PINAKI BOSE           5500
```

This SELECT statement in the FROM clause basically does a full query of the table, and then returns the values (along with the pseudo-column *ROWNUM*) to the outer query. The outer query can then operate on the results of the inner query. Since ‘ROWNUM’ is a pseudo-column and therefore, a reserved word, it is needed to alias that column (here it is ‘rn’) in the inner query in order to refer to it in the outer query.

Query: Select EVERY 4th row from EMP table.

```
SELECT*
 FROM (SELECT rownum rn, ecode, ename, salary
 FROM emp ) A
 WHERE MOD(A.ROWNUM,4) = 0;
```

Output:

```
  RN ECODE          ENAME                    SALARY
   4 E04            JAYANTA GANGULY           6000
   8 E08            GOUTAM DEY                5000
```

Update through an Inline View

Through the inline view, updation is possible.

Query: Double the salary of the employees who are working in the department located at CHENNAI.

```
UPDATE (SELECT ecode,emp.dno,salary,dept.dno,dname, city
FROM dept, emp WHERE emp.dno=dept.dno and city='CHENNAI')
SET salary=salary*2;
```

Output:

3 rows updated.

Let us check the updation:

```
SELECT * FROM emp;
```

NOTES

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E03	HARI NANDAN TUNGA	8000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E06	JISHNU BANERJEE	13000	D02	SYSTEM MANAGER	19-SEP-96
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E09	PINAKI BOSE	11000	D02	PROGRAMMER	26-AUG-94

WITH CHECK OPTION in inline view

The WITH CHECK OPTION may be used to create the inline view as a constrained view and prohibit specific insert and update operations through the inline view. The WITH CHECK OPTION is used with the WHERE clause. For example, to curb users from altering data for any employee in the PROJECT department, the SQL statement will be as follows:

```
UPDATE (SELECT ecode, ename, dno,salary FROM emp
WHERE dno != (SELECT dno FROM dept WHERE dname='PROJECT')
WITH CHECK OPTION) empl
SET empl.salary=7000
WHERE empl.ecode='E01';
```

Output:

0 rows updated.

Now the following SQL statement is entered:

```
UPDATE (SELECT ecode, ename, dno,salary FROM emp
WHERE dno != (SELECT dno FROM dept WHERE dname='PROJECT')
WITH CHECK OPTION) empl
SET empl.salary=7000
WHERE empl.ecode='E03';
```

Output:

1 row updated.

Here updation is successful as employee with employee code 'E03' is working in the RESEARCH department (dno is 'D02').

The WITH CHECK OPTION in an inline view also protects against data modification that would not be visible via the inline view. For example, Employee with employee code 'E03' is shifted from the RESEARCH department to the PROJECT department. To do this assignment, the following statement is entered:

```
UPDATE (SELECT ecode, ename, dno,salary FROM emp
WHERE dno != (SELECT dno FROM dept WHERE dname='PROJECT')
WITH CHECK OPTION) empl
SET empl.dno=(SELECT dno FROM dept WHERE dname='PROJECT')
WHERE empl.ecode='E03';
```

Output:

```
WHERE dno != (SELECT dno FROM dept WHERE dname='PROJECT')
      ERROR at line 2:
      ORA-01402: view WITH CHECK OPTION where-clause violation
```

The preceding statement generated an error as the data would no longer be visible via the inline view.

2.6 BASIC RELATION ALGEBRA OPERATIONS

In pure mathematics, relational algebra has an algebraic structure, that has relevance to set theory and mathematical logic.

Relational algebras received attention after relational model of database was published in 1970, by Codd, who proposed this algebra as a foundation for database query languages. Relational algebra, is widely used in, and now, is a part of computer science. It is based on algebra of sets and is an extension of first-order logic. It is concerned with a set of relations, closed under operators, which operate on one or more relations, yielding another relation.

Relational algebra has similar power of expression as relational calculus and first-order logic. To avoid a mismatch between relational calculus and algebra, Codd restricted the operands of this algebra to finite relations only and he put restricted support for NOT and OR operators. Similar restrictions are also found in other computer languages that use logic-based constructs. A new term was defined by Codd as '*relational completeness'* which refers to attributes of a language that is complete with respect to first-order predicate calculus, and also follows restrictions imposed by Codd.

Basic Operations

There are few operators in any algebraic systems which are basic or primitive, while others are defined in terms of these basic ones. Codd made a similar arbitrary choice for his algebra. There are six primitive operators of relational algebra, proposed by Codd. These are:

1. Selection (Unary)
2. Projection (Unary)

NOTES

3. Cartesian product (also called the cross product or cross join) (Binary)
4. Set union (Binary)
5. Set difference (Binary)
6. Rename (Unary)

These six operators are fundamental and omission of any one loses the expressive power of relational expressions. Many other operators have been defined in terms of these six. Among the most important are set operations, set-intersection, division, assignment, and the natural join.

So following operations of relational algebra will be dealt with:

1. Selection
2. Projection
3. Set operations
4. Renaming
5. Joins
6. Division

First four operations are being dealt with here and last two, joins and division are discussed under 'additional operations'. Set operations contain union, intersection, set difference and product.

Notations Used

In relational algebra, symbols are used to denote operations. Such symbols, taken from symbols of greek alphabet, are difficult to use in HTML. Actual name using capital letters are used for this. These symbols are represented in the following table:

Table 2.3 *Symbols in Relational Algebra*

Operation	My HTML	Symbol
Projection	**PROJECT**	π
Selection	**SELECT**	σ
Renaming	**RENAME**	ρ
Union	**UNION**	$\cup$
Intersection	**INTERSECTION**	$\cap$
Assignment	**<-**	$\leftarrow$

Operation	My HTML	Symbol
Cartesian product	**X**	$\times$
Join	**JOIN**	$\bowtie$
Left outer join	**LEFT OUTER JOIN**	⟕
Right outer join	**RIGHT OUTER JOIN**	⟖
Full outer join	**FULL OUTER JOIN**	⟗
Semijoin	**SEMIJOIN**	$\ltimes$

Selection

A **generalized selection** is a unary operation. This is written as a propositional formula consisting of atoms in the normal selection and with logical operator conjunction, disjunction and negation. This selection selects all those tuples in *R* for which holds.

In relational algebra, a selection is written as $\sigma_{a\theta b}(R)$ or $\sigma_{a\theta v}(R)$ where:

- σ stands for selection
- *a* and *b* denote attribute names
- θ shows binary operation
- *v* denotes a value constant
- *R* stands for relation

The selection $\sigma_{a\theta b}(R)$ selects all tuples in *R* for which θ holds between attributes '*a*' and '*b*'.

The selection $\sigma_{a\theta v}(R)$ selects all tuples in *R* for which θ holds between the attribute '*a*'and the value *v*.

For example, consider the following tables in which the first table is about the relation named *Person*, the second table shows the result of σ_{Age} *e* $\geq_{34}$(*Person*) and the third table has the result for $\sigma_{Age = Weight}(Person)$.

Person

Name	Age	Weight
Rohan	34	70
Manju	25	55
Sohan	29	70
Sonia	50	54
Patel	32	60

σ_{Age} ? $_{34}$(*Person*)

Name	Age	Weight
Rohan	34	70
Sonia	50	54
Patel	32	60

$\sigma_{Age = Weight}(Person)$

Name	Age	Weight
Sonia	50	54

Semantics of the selection is shown mathematically as:

$$\sigma_{a\theta b}(R) = \{\ t : t \in R,\ t(a)\ \theta\ t(b)\ \}$$
$$\sigma_{a\theta v}(R) = \{\ t : t \in R,\ t(a)\ \theta\ v\ \}$$

Projection

A **projection** is mathematically written as $\pi_{a1......an}(R)$, where $a_1,...,a_n$ is a set of attribute names. It is a unary operation. The resultant set from such operation is a set obtained when all tuples in *R* contain the set $\{a_1,...,a_n\}$. All other attributes are discarded.

For example, if we use two attributes, name and age, written as (name, age), then projection of the relation {(Raman, 5), (Ratan, 8)} attribute filed list (age), yields {5, 8}and age are discarded. It only gives the value of the field age. If we project (5, 8) on second component will give 8.

Projection in relational algebra is like a counterpart of existential quantification in predicate logic. Existentially quantified variables are attributes, *excluded* corresponding to existentially quantified variables in the predicate and their extensions are represented by its projection. Thus, projection is defined as excluded attributes. In ISBL notations for both have been provided. Other languages have followed ISBL.

Example of this concept exists in category of monoids. Projection of a string, with removal of one or more letters of alphabet in the string is a monoid and is a projection as well.

For example, there are two relations presented in the following two tables. One is the relation *Person* and its projection on the attributes *Age* and *Weight*.

Person

Name	Age	Weight
Hari	30	70
Shila	26	62
Ganesh	27	75
Sonia	50	55
Patel	30	70

$\pi_{Age,Weight}$(***Person***)

Age	Weight
30	70
26	62
27	75
50	55

If name is N, age is A and weight is W, then predicate is, 'N is A years old and weighs W.' Projection of this is the predicate, 'There exists N such that N is A years old and weighs W.'

In this example, Hari and Patel are of the same age and they have same weight, but (name, age) combination appears only once in the result. This is so because it is a relation.

Mathematically, semantics of projection are expressed as follows:

$$\pi_{a_1,\ldots,a_n}(R) = \{\ t[a_1, \ldots, a_n] :\ t \in R\ \}$$

Where $t[a_1,\ldots,a_n]$ indicate the restriction of the tuple t to the set $\{a_1,\ldots,a_n\}$ which is mathematically represented as;

$$t[a_1, \ldots, a_n] = \{\ (a', v) \mid (a', v) \in t,\ a' \in a_1, \ldots, a_n\ \}$$

Such a projection, $\pi_{a_1,\ldots,a_n}(R)$ is defined only if $\{a_1,\ldots,a_n\} \subseteq$ Header (R). The header contains attributes and Header (R) is a set of all attributes. Projection on nil attribute is also possible. This yields a relation of degree zero.

Relational algebra as identical power of expression, like that of domain relational calculus or tuple relational calculus, but it has less expressive power than first-order predicate calculus without function symbols. Expression wise, relational algebra is a subset of first-order logic. These represent Horn clauses with no recursion and no negation.

Set Operators

Three basic operators have been taken from six basic operators of set theory, but with some differences. Some additional constraints are present in their form as adopted in relational algebra. For operations of union and difference, these two relations must have the same set of attributes which is a property known as union-compatibility. The operation of difference, set intersection is used and this too should be union-compatible.

The Cartesian product is also adopted with a difference from that in set theory. Here tuples are taken to be 'shallow' for the purposes of the operation. Unlike set theory, the 2-tuple of Cartesian product in relational algebra has been 'flattened' into an ($n + m$) tuple. Mathematically, $R \times S$ is expressed as follows:

$$R \times S = \{r \cup s \mid r \in R, s \in S\}$$

In relational algebra, the two relations involved in Cartesian product must have disjoint headers (unlike that of union and difference operations). Here R and S should be disjoint and hence, should not have any common attribute.

Rename

A rename operation is mathematically expressed as $\rho_{a/b}(R)$. This too, is a unary operation. Result of $\rho_{a/b}(R)$ is renaming of the '*b*' field, in all tuples by an '*a*' field. This operation is used to rename the attribute of a relation or even the relation, *R*, itself.

Check Your Progress

5. Why is SQL easy to use?
6. What is data definition language?
7. What happens when a column is declared as the PRIMARY KEY?
8. Define the term grouping columns.
9. What is done to avoid a mismatch between relational calculus and algebra?

2.7 ANSWERS TO 'CHECK YOUR PROGRESS'

1. DBA performs the following tasks:
   - It identifies and catalogs the required data for business users
   - It produces conceptual and logical data models to depict accurately the relationship among various business processes of the organization
   - It sets data policies for the organization
   - It identifies data owners, such as administrative staff and local users
   - It sets standards for central use of data
   - It focusses on the DBMS and physical databases
   - It opposes metadata but deals with data
2. Bottleneck is defined as a point where the flow of data is either impaired or stopped completely. Essentially, a bottleneck results when there is not enough data handling capacity for accommodating the current volume of data traffic.
3. Relations storing data are known as base relations. There are other relations too that are derived using operations such as selection, projection, union, intersection, etc., and hence, they are known as 'derived relations'.
4. Constraints define a condition, which needs to be satisfied while storing data in a database. DBMS allows you to define and implement the constraints for a database object.
5. SQL is a 'declarative language' (non-procedural). This makes SQL relatively easy-to-use as compared to other programming languages. In SQL, the programmer only specifies what data is needed, but he is not required to specify how to retrieve it. The underlying DBMS analyses the SQL and formulates the way to retrieve the required information.
6. Data definition language (DDL) is that part of SQL that allows a database user to create and restructure database objects, such as the creation or deletion of a table.
7. When a column is affirmed as the PRIMARY KEY, an index on this column is automatically created and assigned a unique name by Oracle. The additional constraints UNIQUE and NOT NULL are implied by the PRIMARY KEY constraint.
8. The GROUP BY clause basically allows us to partition the results of a query into groups with similar characteristics based upon predicate satisfaction. The columns named in the GROUP BY clause are called the grouping columns.
9. To avoid a mismatch between relational calculus and algebra, E.F. Codd restricted the operands of this algebra to finite relations only and he put restricted support for NOT and OR operators.

2.8 SUMMARY

- DataBase Administration (DBA) installs, configures, troubleshoots and maintains a database system and also separates the function of resource management from database services and technologies, for managing data.
- DBA transforms a logical data model into a physical database design and incorporates the knowledge of DBMS that creates an appropriate and efficient physical database design from a logical model.
- By writing better SQL statements, more than 80 percent of database query performance can be improved. Putting SQL statements inside stored procedures yields good improvement in performance in Microsoft SQL Server and Sybase.
- Bottleneck is defined as a point where the flow of data is either impaired or stopped completely. Essentially, a bottleneck results when there is not enough data handling capacity for accommodating the current volume of data traffic.
- The term 'Bottleneck' in 'SQL Server' refers to the congestion of data traffic on the data routing route, this congestion reduces the flow of data causing the bottleneck at the SQL Server.
- Database tuning describes a group of activities used to optimize and homogenize the performance of a database. It usually overlaps with query tuning, but refers to design of the database files, selection of the DataBase Management System (DBMS) application, and configuration of the database's environment, such as Operating System (OS), CPU, etc.
- Relational model of a database is based on set theory of mathematics. Relation is a central concept in relational model that is borrowed from the concept of set theory. Such concepts are widely applied in designing relational model of a database.
- Relations storing data are known as base relations. There are other relations too that are derived using operations such as selection, projection, union, intersection, etc., and hence, they are known as 'derived relations'.
- The relational model is consistent, which is achieved by using constraints while designing database. This design is also known as logical schema.
- SQL is a 'declarative language' (non-procedural). This makes SQL relatively easy-to-use as compared to other programming languages. In SQL, the programmer only specifies what data is needed, but he is not required to specify how to retrieve it. The underlying DBMS analyses the SQL and formulates the way to retrieve the required information.

- Data definition language (DDL) is that part of SQL that allows a database user to create and restructure database objects, such as the creation or deletion of a table.
- When a column is affirmed as the PRIMARY KEY, an index on this column is automatically created and assigned a unique name by Oracle. The additional constraints UNIQUE and NOT NULL are implied by the PRIMARY KEY constraint.
- The GROUP BY clause basically allows us to partition the results of a query into groups with similar characteristics based upon predicate satisfaction. The columns named in the GROUP BY clause are called the grouping columns.
- Relational algebra is deals with a set of relations, closed under operators, which operate on one or more relations, yielding another relation.
- To avoid a mismatch between relational calculus and algebra, E.F. Codd restricted the operands of this algebra to finite relations only and he put restricted support for NOT and OR operators.
- A projection is mathematically written as Àa1…..,an (R), where a1,...,an is a set of attribute names. It is a unary operation. The resultant set from such operation is a set obtained when all tuples in R contain the set {a1,...,an}.

2.9 KEY TERMS

- **Tuning:** It refers to the use of various techniques for the adjustments and changes made to help the systems work efficiently.
- **Bottleneck:** Bottleneck is defined as a point where the flow of data is either impaired or stopped completely.
- **Relational model:** Relational model of a database is based on set theory of mathematics.
- **Foreign key:** It refers to set of one or more columns that designates the foreign key in a referential integrity constraint.
- **NOT NULL constraint:** It is constraint that requires that the column contain a value when it is initially inserted into the table or whenever the column is updated.
- **UNIQUE constraint:** It is constraint that identifies a column or combination of columns as a unique key.
- **Referential integrity constraint:** It is constraint that designates a column or combination of columns as FOREIGN KEY to establish a relationship between the foreign key and a specified primary or unique key called the referenced key.

- **DROPTABLE command:** It is the command for removing a table and all its data from the database.
- **TRUNCATE command:** It is the command that allows the removal of all rows from a table, but flushes a table more efficiently since no rollback information is retained.
- **Right outer join:** It refers to join where records from the right table that have no matching key in the left table are included in the result set.
- **Full outer join:** It refers to join where records from the first table are included that have no corresponding record in the other table and records from the other table are included that have no records in the first table.

2.10 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is database tuning?
2. What are tunable parameters?
3. Write the features of entended relational model.
4. What is SQL?
5. Write the difference between procedural and non-procedural DML.
6. What do you understand by a literal?
7. What are the SQL operators?
8. What is embedded SQL?
9. Define projection. How is it written mathematically?

Long-Answer Questions

1. How are SQL queries tuned? Explain.
2. Explain the various types of SQL server bottlenecks.
3. Discuss relational model with the help of appropriate examples.
4. Explain various relational constraints.
5. Discuss different Data Query Language SQL commands by giving appropriate examples.
6. Describe Data Definition Language (DDL) giving appropriate examples.
7. Explain the multiple conditions in structure query language.
8. Explain the basic operations of relational algebra with the help of appropriate example.

NOTES

2.11 FURTHER READING

Navathe, S.B. and R. Elmasri. 1997. *Database System Concepts*, Third edition. New York: McGraw-Hill.

Korth, Henry F., Avi Silberschatz and S. Sudarshan. 2010. *Database System Concepts*, 6th edition. New York: McGraw-Hill.

Elmasri, Ramez and Shamkant B. Navathe. 2007. *Fundamentals of Database Systems*, 5th edition. New Jersey: Pearson Education.

Martin, James. 2007. *Principles of Data Base Management*. New Jersey: Pearson Education.

Martin, James. 1975. *Computer Data-Base Organization*. New Jersey: Prentice Hall.

Jackson, Glenn A. 1988. *Relational Database Design With Microcomputer Applications*. New Jersey: Prentice Hall.

Date, C.J. 2000. *An Introduction to Database System*, Seventh edition. Boston: Addison Wesley.

UNIT 3 OBJECT ORIENTED DATABASE SYSTEMS AND DISTRIBUTED DATABASE

Structure

3.0 INTRODUCTION

Computer-based systems use software applications and operating systems to implement flexible services for various industries, for example, banking, telecommunications and other sectors. Object-oriented technology encompasses programming languages, projects, hardware methodologies and object-based approaches. Basically, the objects are 'black boxes' that deal with the object from three distinct interfaces, such as public interface (access to everybody), inheritance interface (accessible only by direct specializations of the object) and parameter interface, which are applied to create an instance of the parameterized class. Object-oriented concepts have become popular in the area of computer programming with interchangeable, updatable and reusable parts. The process of creating new classes is done with the help of inheritance. Inheritance helps in conceptualization and programming that makes ease of distinguishing various class libraries. An object relational database is also called an object relational database management system which is a simplified form of object oriented front end on a relational database.

A distributed database or DDB is a collection of multiple interrelated databases that are spread over a computer network. Each computer contains its own database, which is managed by an individual database management system. A distributed database management system (DDBMS) manages DDBs and makes the distribution transparent to the user so that the user is not aware of the distribution and accesses the data, as it is stored at one place. DDB technology has evolved as a combination of two technologies, one is the database technology and the other is the network and data communication technology.

In this unit, you will study about the object oriented database systems, characteristics of Object-Relation Database Management System (ORDBMS), complex objects, inheritance**,** function overloading, distributed database system, distributed database design, data replication and allocation and distributed database query processing.

3.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basics of object oriented database systems
- Discuss the various characteristics of Object-Relation Database Management System (ORDBMS)
- State the significance of complex objects
- Describe the meaning of inheritance
- Analyse the function overloading
- Explain the distributed database system and design
- Discuss the concept of data replication and allocation
- Describe the distributed database query processing

3.2 OBJECT ORIENTED DATABASE SYSTEMS

Computer-based systems use software applications and operating systems to implement flexible services for various industries, for example, banking, telecommunications and other sectors. Object-oriented technology encompasses programming languages, projects, hardware methodologies and object-based approaches. Basically, the objects are 'black boxes' that deal with the object from three distinct interfaces, such as public interface (access to everybody), inheritance interface (accessible only by direct specializations of the object) and parameter interface, which are applied to create an instance of the parameterized class. Inheritance is defined as the process of one object acquiring (`get()` method and `receive()` method) from one or more objects. In a bank account, checking account and savings account are the examples of single inheritance. Here, public interface of the object contains operations, constants and exceptions. The operation is accomplished by the actual algorithm. Instead of structuring a program as a sequence of instructions, both the data and the associated functions are combined in a single unit called class and the data is made inaccessible to avoid corruption of data. The object-oriented programming (OOP) paradigm was developed to revolutionize the process of software development. It not only includes the best features of structured programming, but also introduces some new and advanced features that the procedural programming lacked. The most important feature is that unlike the procedural approach in which a program is divided into a number of functions, OOP divides the program into a number of objects. An object is a unit of structural and behavioural modularity that contains a set of properties as well as the associated functions. In addition, programmers can create relationships between one object and another. The functions of the object also known as member functions provide the only way to access the object's data. If the user wants to read or manipulate any data item, then it is possible only if the member function is available in the object. Therefore, data is hidden from the outside world and hence, is safe from accidental modifications. Figure 3.1 shows the data and functions in OOP approaches:

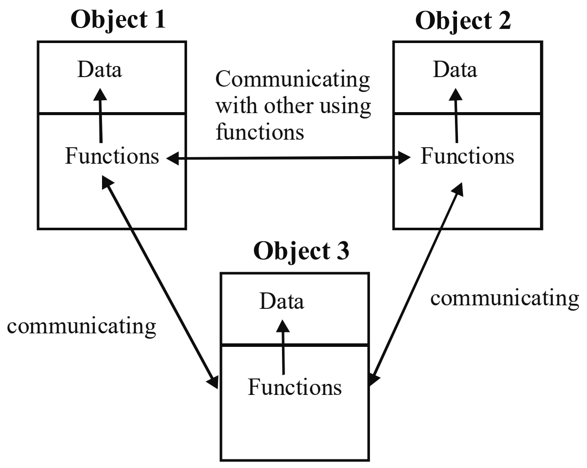


Fig. 3.1 *Data and Functions in OOP*

Features of OOP

The features of OOP approaches are as follows:

- OOP emphasizes on data rather than the functions or the procedures.
- OOP models the real world very well by binding the data and associated functions together under a single unit. Thus, it prevents the free movement of data from one function to another.
- The objects of the entire system can interact with each other by sending messages to each other.
- OOP follows the bottom-up approach for designing the programs. That is, first the objects are designed and then these objects are combined to form the entire program.

To understand the concept of object-oriented programming, it is necessary to know the fundamental terms and concepts of this approach. These include objects, classes, data abstraction, encapsulation, inheritance, polymorphism and message passing. The various concepts used in the object-oriented concept are as follows:

- **Object**

The 'object' is defined as conceptual and physical things. Documents, software concepts and living beings are examples of objects. In an organization, building, documents, employees, benefit packages, etc. are defined as objects. In the field of RDBMS, stacks, queues, cascading style sheets, check boxes, windows are taken as objects. In the banking sector, `customter_id`, `customer_name`, `bank_account`, etc. are considered as objects and the state of the object is current balance. The two prime categories of the objects are classes and instances.

- **Classes**

A class is defined as a user-defined data type, which contains the entire set of similar data and the functions that an object possesses. For example, `Student_Details` is considered as class that serves as template for the objects in RDBMS. The class is determined by the structure and capabilities of an instance. A class is basically a template where a set of items are arranged in a pattern. In fact, a class is known as 'instance factory'. But there is some restriction to the object of classes.

- **Abstraction**

Abstraction is a mechanism to hide irrelevant details and represent only the essential features so that one can focus on important things at a time. It allows managing complex systems by concentrating on the essential features only. Data abstraction ensures security of data by preventing it from accidental changes or manipulations by other parts of the program.

- **Encapsulation**

Encapsulation is the technique of binding or keeping the data and functions operating together in a single unit called class. Encapsulation is the way to implement data abstraction. For example, the function `student_reco_display()` can be encapsulated in a class student.

- **Polymorphism**

Polymorphism is the ability of an entity, such as a function to be processed in more than one form. This concept is widely used in object-oriented database designing. The concept of polymorphism plays an important role in OOP as it allows an entity in RDBMS database to be represented in various forms in the SQL programming approaches.

Database design involves normalizing a database to ensure that relational tables in the database contain only related information and store data only at a single location in the database. The higher normal forms, such as fourth normal form and fifth normal form in normalization are processed with the concept of multi-valued dependency. Partial dependency occurs when a row of a table is uniquely identified by one column that constitutes a primary key without requiring the entire primary key to uniquely identify the row. Functional dependencies are determined from a real world problem that the database models. Multi-valued dependency helps reduce redundancy in the databases. Object-oriented Database Management System (OODBMS) is a type of DBMS that works on the object-oriented data model. Programming languages, such as C++ and the Smalltalk, follow the object-oriented data model. OODBMS stores complex objects and multi-valued attributes. Working with OODBMS is easy because

UML classes are not required to transform into a relational schema which is a type of logical schema. This makes the technology less mature and is usable with niche applications, like CAD. In object-oriented programming, object types are used to reduce the cost and time for building complex applications. Object types create maintainable, reusable and modular software components. The implementation details are hidden by object types so that you can modify data in an application without affecting the client programs. In fact, the relational databases are not worked with certain classes of applications. To overcome this problem, OODBMS came into existence in the mid 1980s. This DBMS works with one item at a time and creates complex data structures. It uses declarative databases that deal with tuples instead of individual data in flat tables. These approaches provide strength in the respective fields. The properties of object-oriented database system are as follows:

- It contains a set of databases.
- It should be an object-oriented system.
- It supports object-oriented programming languages to work as front-end and correlates with back-end.
- It contains persistence, secondary storage management, concurrency, data recovery and ad hoc query facility features.
- It manipulates complex objects, objects identity, types of classes, encapsulation and inheritance features, overriding along with late binding and computational completeness.

Advanced Features of Object-oriented Database

The advanced features of object-oriented database are as follows:

- **Supports complex objects:** OODBMS uses objects to apply constructors. A minimum set of constructors is applied to tuple, i.e., records.
- **Object identity:** The object identity for complex objects is independent of its value associated with the object. There are two types of object identities. They are object sharing and object updates. Object updates ensure the uniqueness of object identity.
- **Encapsulation:** This feature of OODBMS implements the hidden object and provides operations that are visible to the programmer.
- **Supports classes and objects:** A class performs run-time execution in OODBMS. This DBMS supports a set of objects that is used for programming errors during compile-time.
- **Supports inheritance:** This DBMS supports subclass or subtype that inherits methods and attributes from the superclass or supertype.
- **Supports overriding, overloading and late binding:** Objects of different types referred as overloading perform various operations in OODBMS, whereas implementation of operations depends on the type of object to which it is applied and supports overriding. The implementation code is not referenced without run-time. This mechanism supports late binding.

NOTES

- **Supports computational functions:** OODBMS supports data manipulation language (DML) of the database. It allows users to calculate computational functions.
- **Extensibility:** This feature provides new types of classes that can be manipulated in the same way as they were built into the system.
- **Persistence:** This database facilitates database connections, such as creating, retrieving and manipulating data. This feature is applied to integrate objects with transformed data processes.
- **Supports concurrency and recovery management:** This DBMS supports the concurrency feature where various operations access the same objects, whereas recovery provides a mechanism to handle contingencies for databases.
- **Facilitates ad hoc query:** OODBMS facilitates an efficient and high-level query option that generates ad hoc reports.

3.3 CHARACTERISTICS OF AN OBJECT-RELATION DATABASE MANAGEMENT SYSTEM (ORDBMS)

Several major software companies including International Business machines or IBM, Informix, Microsoft, Oracle and Sybase have all released object relational versions of their products. These companies are promoting a new and extended version of relational database technology called object relational database management systems also known as ORDBMSs. A certain group thinks that future applications can only be implemented with pure object oriented systems. Initially, these systems looked promising. However, they have been unable to live up to the expectations. A new technology has evolved in which relational and object oriented concepts have been combined or merged. These systems are called object relational database systems. The main advantages of ORDBMSs are massive scalability and support for object oriented features. The main advantages of extending the relational data model come from reuse and sharing. Reuse comes from the ability to extend the DBMS server to perform standard functionality centrally rather than have it coded in each application. If the functionality in the server is embedded, it saves having to define it in each application that needs it and consequently allows the functionality to be shared by all applications. The relational model was formally introduced by E.F. Codd. The defining standard for relational databases is published by ANSI (the American National Standard Institute) as SQL (ANSI 1986) or SQL1, called SQL-86. A revised standard is called SQL2 also referred to as SQL-92. A relational database is composed of many relations in the form of two-dimensional tables of rows and columns containing related tuples. Organizing data into tables, the form in which data is presented to the user and the programmer, is known as the logical view of the database. The stored data on a computer disk system is called the internal view. The rows (tuples) are called records and the columns (fields in the record) are called attributes. Each column has a data type (i.e., `int`, `float`, `date`). There are various restrictions on the data that can be stored in a relational database. These are called constraints. The constraints are domain constraints, key constraints entity integrity constraints and referential integrity constraints. These constraints ensure that there are no ambiguous tuples in the database. RDBMSs use Structured Query Language (SQL) currently SQL2 as the Data Definition Language or DDL and the Data Manipulation Language or DML. SQL includes statements for data definition, modification, querying and constraint specification. The types of queries vary from simple single-table queries to complicated multi-table queries involving joins, nesting, set union and differences, and others. The standard consists of the object model, the Object Defining Language (ODL), the Object Query Language (OQL), and the bindings to object oriented programming languages. Object Relational Database Management Systems or ORDBMSs allow users to define data types, functions and operators. As a result, the functionality of the ORDBMSs increases along with their performance. The other current ORDBMSs include Oracle 8i from Oracle Corporation and Universal Database (UDB) from IBM. Also, those applications from relational DBMSs (simple data with query) will slowly move towards the object relational DBMSs (complex data with query). The object relational model facilitates an advantage to maintain the relationships between data for specific records in DBMS. For example, in an address book application, an additional table can be added to hold zero or more addresses for each user. RDBMS implements the object type system as an extension to the relational model. The object type interface continues to support standard relational database functionality, such as queries using SELECT, FROM and WHERE clauses, commits, backup and recovery, scalable connectivity and data consistency. The object relational model supports an object interface while using the high concurrency and throughput of a relational database. The object relational model is used to store, manipulate and retrieve and process the data stored in a database. The object relational data model has the database information organized in graphs of objects and it is closer to the real world. An object represents only one individual occurrence of an entity and each object has a number of attributes which may be simple values or complex values and references to other objects and methods. Attributes describe the properties of an object and thus a student who purchases at a college bookshop may be stored as a Student object with attributes of Name, Student Identification Number and Date of Birth. Classes describe a collection of similar objects. The important elements associated with object oriented data models are abstraction, encapsulation, modularity and hierarchy. An object relational data model supports both the relational and the object oriented model in implementing the database. The entities in the data model are represented as objects that are also represented as relational tables. Object Relational Database Management System (ORDBMS) and its standards have been emerged to incorporate the object technology into the relational databases and to work with more complex data and relationships. For object relational databases there is not a consensus in a technique to transform conceptual model into a logical schema. This model uses Data Definition Language or DDL (CREATE, ALTER, DROP) and Data Manipulation Language or DML (INSERT, UPDATE, DELETE) statements. The object relational model has following features with reference to ORDBMS:

- **Object Relational Database Design:** Object relational model design is complex due to its object oriented nature.

- **Query Processing and Optimization:** By extending SQL with functions and rules, object relational model is compounded with query optimization overview.
- **Interaction of Rules with Transaction**: Rule processing as implied in SQL covers more than update rules which are implemented in RDBMSs of triggers. A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database.
- **Extensibility:** You can extend the capability of the database server by defining new data types, accessing methods and functions and using User Defined Routines (UDRs) which collectively allow you to store and manage images, audio, video, large text documents, etc.
- **Complex Data Types**: You can define new data types that combine one or more existing data types. Complex types enable greater flexibility in organizing data at the level of columns and tables, for example BLOB (Binary Large OBject), CLOB (Character Large OBject), BFILE (Binary FILE) and NCLOB (National Character Large OBject). CLOB is used specifically to store character set data whole whereas NCLOB is specifically used to store Unicode national character set data.
- **Inheritance**: You can define objects (types and tables) that acquire the properties of other objects and add new properties that are specific to the object that you define.

Dynamic server allows you to build object relational databases. This server provides object oriented capabilities beyond those of the relational model but represents all data in the form of tables with rows and columns. An object type is a kind of data type. You can use it in the same ways that you use standard data types, such as `NUMBER` or `VARCHAR2`. For example, you can specify an object type as the data type of a column in a relational table and you can declare variables of an object type. The value is a variable or an instance of that type. An object instance is also called an object.

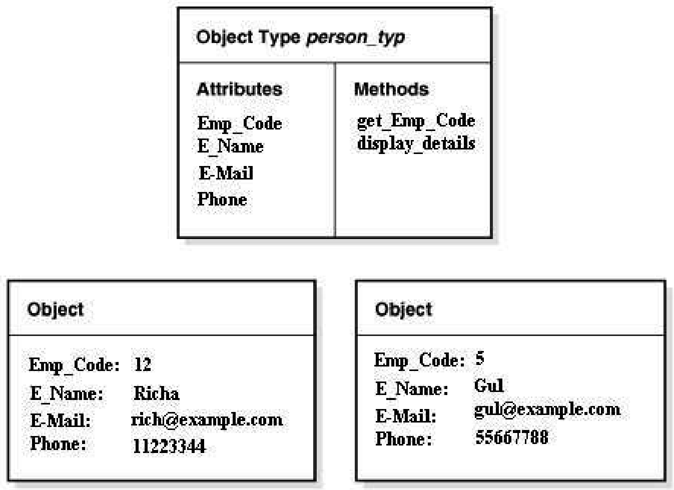


Fig. 3.2 *An Object Type and Object Instances*

Figure 3.2 illustrates the `person_typ` object type and two instances of the object type. The attributes of `person_typ` are `Emp_Code, E_Name, E-Mail` and `Phone` and the methods are `get_Emp_Code` and `display_details`. Instances of the object type are as follows:

Emp_Code	**:**	12	**Emp_Code**	**:**	5
E_Name	**:**	Richa	**E_Name**	**:**	Gul
E-Mail	**:**	rich@example.com	**E-Mail**	**:**	gul@example.com
Phone	**:**	11223344	**Phone**	**:**	55667788

Existing Data Models

The existing data models are described below:

NetCDF-3

The NetCDF-3 data model is generally shown in the Universal Modelling Language or UML. A dataset has dimensions, variables and attributes. Attributes can be global or apply to individual variables. There is a very limited set of low level data types.

OPeNDAP

The OPeNDAP data model has many approaches in common with NetCDF. It has a rich set of low level data types and includes structures, sequences and grids. OPeNDAP data model is a client-server protocol for scientific data access. The data model package includes a C++ client and a server, and Java client and server libraries.

HDF-5

HDF-5 is a machine and operating system independent file format for self describing scientific data. It has evolved from HDF-4 but has some fundamental differences. HDF-5 has a rich set of low level data types and includes the key feature of a group of variables.

Common Data Access Model

At the data access level, the Common Data Model or CDM maintains as much as possible of the elegance of the NetCDF-3 interface and adds important features from OPeNDAP and HDF are more low level data types including string, structures and groups.

3.3.1 Complex Objects

Constructors are used to create complex things from simpler ones. Integers, characters, byte strings of any length, booleans, and floats are the most basic objects (one might add other atomic types). Tuples, sets, bags, lists, and arrays are just a few instances of complex object constructors. The minimal set of constructors that the system should have are set, list and tuple. Sets are important because they provide a natural means of describing collections from real-world.

Tuples are important because they are a natural way to represent an entity's properties. Of course, both sets and tuples are significant because the relational model helped them achieve widespread adoption as object constructors. Lists and arrays are useful because they capture order, which takes place in the real world, and they also arise in many scientific applications, where people need matrices or time series data.

Constructors are used to create complex things from simpler ones. Integers, characters, byte strings of any length, booleans, and floats (to name a few) are the most basic objects. The object constructors must be orthogonal, which means that any constructor can be applied to any object. The relational model's constructors are not orthogonal because the set construct can only be applied to tuples and the tuple constructor can only be applied to atomic values, Non-first normal form relational models, in which the top level component must always be a relation, are another example.

Complex objects are required to provide appropriate operators in order to deal with such objects (whatever their composition). That is, operations on a complex object must propagate transitively to all its components. Examples include the retrieval or deletion of an entire complex object or the production of a "deep" copy (in contrast to a "shallow" copy where components are not replicated, but are instead referenced by the copy of the object root only).

3.4 INHERITANCE

Inheritance is considered as the most powerful feature of object-oriented programming after classes themselves. New classes are created with the help of inheritance. Derived classes can be created from existing or base classes. The derived class inherits all the functions of base classes but it supports embellishments and refinements of its own, i.e., producing properties to the child class form parent class. Inheritance has one of the prime advantages in that once base class is written and debugged it can be revoked as per program design. It permits code reusability. The existing code saves time which increases the program's reliability. Inheritance helps in conceptualization and programming that makes distinguishing various class libraries easy. In a nutshell, inheritance property refers to the classes which are arranged in a hierarchy. A derived class can inherit (automatically) the properties (attributes) and behavior (methods) of its parent or base class. Table 3.1 shows the class, attributes and their methods which are required for inheriting the property.

Table 3.1 *Class, Attributes and their Methods to Inherit the Property*

Class	**Attributes**	**Methods**
`Person`	`Name, Age`	`add(), modify()`
`Student`	`Student_Name, Student_Course`	`info(), record()`
`Employee`	`Emp_Id, Emp_Name`	`empDetail(), ChangeInfo()`

An object-oriented database schema typically requires a large number of classes. Several classes can be similar. For example, bank employees are similar to customers. In order to allow the direct representation of similarities among classes, we need to place classes in a specialization hierarchy. For example, Figure 3.3 is a specialization hierarchy for the ER model.

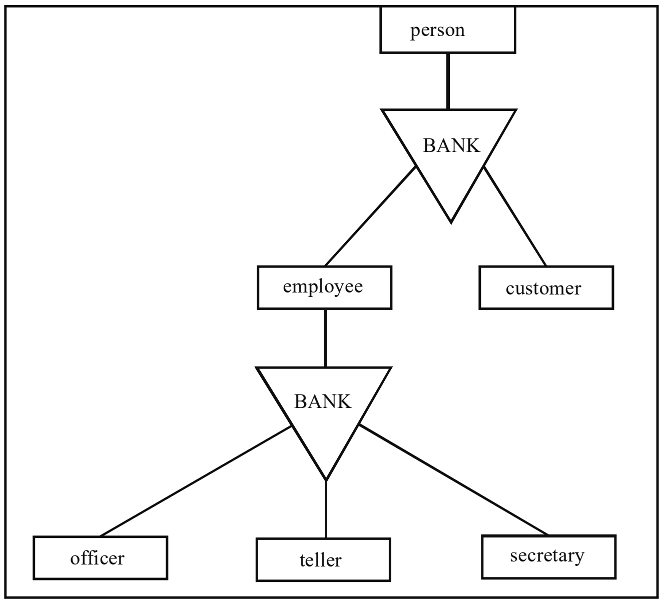


Fig. 3.3 *Specialization Hierarchy in Banking System*

The concept of a class hierarchy is similar to that of specialization in the ER model. The corresponding class hierarchy is shown in Figure 3.4.

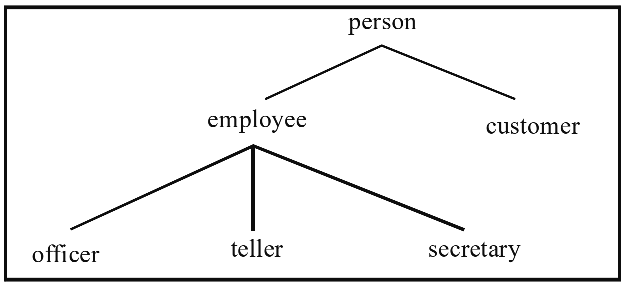


Fig. 3.4 *Class Hierasrchy Corresponding to the Banking System*

The class hierarchy can be defined in pseudocode in which the variables are associated with each class. Following pseudocode is required to define the class hierarchy showing inheritance property as per Figure 3.4:

```
class person{
 string name;
 string address;
};
```

NOTES

```
class customer BANK person{
 int credit-rating;
};
class employee BANK person{
 date start-date;
 int salary;
};
class officer BANK employee{
 int office-number;
 int expense-account-number;
};
class teller BANK employee{
 int hours-per-week;
 int station-number;
};
class secretary BANK employee{
 int hours-per-week;
 int manager;
};
```

The keyword `BANK` is used to indicate that a class is a specialization of another class. The specialization of a class is called subclasses, for example, `employee` is a subclass of `person` and `teller` is a subclass of `employee`. Similarly, `employee` is a superclass of `teller`. An object representing an `officer` contains all the variables of class `officer class, employee` class and `person` class. Methods are inherited in a manner identical to inheritance of variables. An important advantage of inheritance in object-oriented systems is the notion of substitutability. For example, any method of a class A can be equally well to be invoked with an object belonging to any subclass B of A. This characteristic leads to reuse the code. Methods and functions in class A, such as `get-name()` method in class `person` cannot be rewritten for objects of class B. The `employee` class is associated with all objects along with instances of `officer, teller` and `secretary`.

Check Your Progress

1. What are the applications encompassed in object-oriented technology?
2. Define the term relational database management system.
3. What do you mean by inheritance?

3.5 FUNCTION OVERLOADING RULES

When two different functions have the same name but their parameter types or number are different (i.e. function signature is different), then it is a case of function overloading. Thus, you can give the same name to more than one function if they have either a different number of parameters or different types in their parameters.

Example 3.1

```
// Example of function overloading
#include <iostream>
using namespace std;
int operate (int a, int b)
{
  return (a*b);
}
float operate (float a, float b)
{
  return (a/b);
}
int main ()
{
  int x=5,y=2;
  float n=5.0,m=2.0;
  cout << operate (x,y);
  cout << "\n";
  cout << operate (n,m);
  cout << "\n";
  return 0;
}
```

The output that you get is:

```
10
2.5
```

You have defined two functions here with the same name operate. The first one accept two parameters of type `int` and the other one accepts them of type `float`. By examining the types passed as arguments when the function is called the compiler knows which one to call in each case. If it is called with two `ints` as its arguments, it calls to the function that has two `int` parameters in its prototype. Similarly, if it is called with two floats, it will call to the one which has two `float` parameters in its prototype.

The first function returns the result of multiplying both parameters, whereas the second call passes two arguments of type `float`, so the function with the second prototype is called. The second function has a different behaviour. The second function divides one parameter by the other. Therefore, because the function has been overloaded, the behaviour of a call to operate depends on the type of the arguments passed.

It is important to note that only by its return type a function cannot be overloaded. So, the return type is not considered as a part of the function signature and to overload a function, at least one of its parameters must have a different type.

3.6 DISTRIBUTED DATABASE SYSTEM

Distributed Database

A distributed database or DDB is a collection of multiple interrelated databases that are spread over a computer network. Each computer contains its own database, which is managed by an individual database management system. A distributed database management system (DDBMS) manages DDBs and makes the distribution transparent to the user so that the user is not aware of the distribution and accesses the data, as it is stored at one place.

DDB technology has evolved as a combination of two technologies, one is the database technology and the other is the network and data communication technology. DDBs provide the advantages of distributed computing to the field of database management. The components are interconnected by a computer network and work together to perform the assigned tasks.

Advantages of distributed database

DDBs are used for several reasons such as organizational decentralization and cost-effective processing. Some of the advantages of DDBs are as follows:

- Increased reliability and availability
- Improved performance

Increased reliability and availability

Reliability is a measure of the possibility that defines a system which is running at a given point of time. On the other hand, availability is a measure of the possibility that the system is continuously serving the queries made to it during a time interval. When you use DDBs, which are spread over several other sites, even if one site fails, it helps you to connect to the other sites which continue to function normally. However, the data and software that resides on the site which failed cannot be accessed, without affecting the performance of the other sites in the distributed database. This quality improves the reliability and availability.

Improved performance

A distributed DBMS fragments the existing database and keeps the data closer to the related site where it is most required. A database which is large in size is fragmented and distributed over many other sites; and these are referred as smaller databases at each site where these exist. The queries and transactions accessing data at smaller databases have a better performance. In addition, when the database is fragmented into smaller databases, each site has less overhead of transaction in the execution of a query. Figure 3.5 shows the distributed database architecture.

NOTES

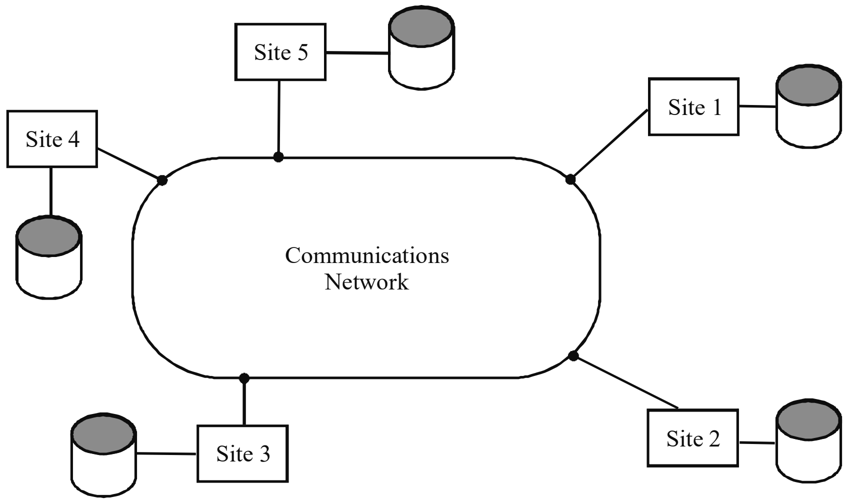


Fig. 3.5 *The Distributed Database Architecture*

Distributed Data Transparency Features

In a DDBMS, distribution is transparent, i.e., it hides the details of where each file is physically stored within the system. Any user on the network can access a file sitting on his terminal disregarding where the file is actually stored. The various features of data transparency are:

- **Distribution or network transparency**: In this type of transparency, the user is not concerned with the network details. It may be location transparency or naming transparency. Location transparency has the property that the command used to perform a task is not dependent on the location of the data and the location of the system from where the command is issued. Naming transparency means that if you specify a name for an object, the named object can be accessed unambiguously without any further specification.
- **Hardware transparency:** This enables the running of the same DBMS on different hardware platforms. The hardware transparency is needed as the real-world computer installations involve a wide range of machines that include IBM machines, ICL machines, HP machines, personal computers and workstations. Hardware transparency helps the DBMS to integrate the data in those machines so that they can be presented to the users as from the single-system machine.
- **Replication transparency**: This enables the storage of copies of data at multiple distributed sites. In other words, replication transparency can allow the creation and destruction of replicas of data in response to user requests. Replication transparency has two advantages. First, it enhances the performance of the database, for example, applications accessing the database can use local copies of data instead of trying to access the data in the remote locations. Secondly, it increases the availability of data of the DBMS, as due to replication transparency, data becomes available on multiple client machines. However, apart from these advantages, replication also has disadvantages. The major disadvantage of replication is encountered

whenever a replicated data is updated. It is because, when you update a replicated data, you also need to update every single copy of that replicated data.

- **Fragmentation transparency**: This type of transparency enables a user to behave in such a manner that he/she is unware of the existing fragments of the data in the database. Fragmentation transparency also implies that the users can view the data as logical combinations by using suitable joins and unions. Here, the system optimizer is responsible for determining the fragments that can be physically accessed by the users. A DBMS is said to support data fragmentation, whenever, a relation variable in the DBMS can be divided into fragments or pieces for the purpose of physical storage. Fragmentation helps enhance the performance of the DBMS by enabling storage of data at the location from which the data is most frequently used. This makes the operations on the data to be performed as local along with reducing the traffic in the network. Basically, there are two types of fragmentations—three-fourth horizontal and vertical. Horizontal fragmentation can divide a relation or a table into sets of rows. Vertical fragmentation can divide a relation or a table into subrelations. Each subrelation is considered as a subset of columns of the original relation. Fragmentation transparency has a special feature that makes the user aware about the existing fragments.
- **Transaction transparency**: It helps in maintaining consistency of data by coordinating the different functions of an object. These functions are used to define various transactions and their dependencies in the database. You need to add different check points at the different states of an object to define these functions of the object.
- **Failure transparency**: It refers to the extent to which the errors and related recoveries of a distributed database are invisible to the users and applications. For example, if a server fails, then the failure transparency helps in automatically redirecting all the users connected to the failed server to a different server in such a manner that the users never notice the server failure. In other words, failure transparency is used to tolerate fault failures and problems of the distributed database. It does so by defining different conditions of the database that can causes problems. For tolerating problems of databases, the failure transparency includes various steps, such as locating an object with its related possible problems, using check points and recovery functions to detect and recover the problems and providing stability of an object using replication function which is used to recover a problem. Failure transparency is one of the most hard-to-achieve transparencies, as it is very difficult to determine whether a server actually has failed or whether it is just responding to the requests very slowly.
- **Performance transparency**: It allows reconfiguration of the distributed database system to match with varying loads on the system for improving the performance of the system. It also helps in executing distributed queries in the distributed system using a distributed query optimizer.

- **Heterogeneity transparency**: It is used to access the database of a single computer of the DDB system.
- **Migration transparency**: It allows movement of data in a distributed database system without affecting the operations performed by the users as well as application programs. Examples of migration transparency include Network File System (NFS) and Web pages available on the World Wide Web (WWW).
- **Access transparency**: It enables users to access local and remote data from a distributed database system using the same operations. Examples of access transparency include:
  - o Various file system operations in NFS
  - o Queries performed on SQL
  - o Navigating though the various Web pages on the Web
- **Location transparency**: It enables the users to behave in such a manner that they do not actually know where the data of the DBMS is physically located. However, they can work on that data as if it were present at their local site. In other words, location transparency helps in accessing data without the knowledge of their actual location. Location transparency is useful in the distributed databases as it simplifies the terminal activities and user programs. Here, data can move from one location to another so that it can respond to the changing performance requirements. Examples of location transparency include:
  - o Operations of the file systems in NFS
  - o Web pages available on the Web
  - o Tables contained in a distributed database

Advantages and Disadvantages of Distributed Databases

Advantages

The following are the advantages of distributed databases:

- Distributed database systems employ a distributed processing architecture. An Oracle database server acts as a client when it requests data that another Oracle database server manages.
- The distributed database system and *database replication* are related, yet distinct. In a *pure,* that is, not replicated distributed database, the system manages a single copy of all data and supporting database objects.
- Typically, distributed database applications use distributed transactions to access both local and remote data and modify the global database in real time. The term *replication* refers to the operation of copying and maintaining database objects in multiple databases belonging to a distributed system. While replication relies on distributed database technology, database replication offers applications benefits that are not possible within a pure distributed database environment. Most commonly, replication is used to improve local database performance and protect the availability of

applications because alternate data access options exist. For example, an application may normally access a local database rather than a remote server to minimize network traffic and to achieve maximum performance. Furthermore, the application can continue to function if the local server experiences a failure, but other servers with replicated data remain accessible.

- A distributed database system is regarded as a kind of partnership among individual local DBMSs at individual sites.
- This system maintains a *database link.* A database link is a connection between two physical database servers that allows a client to access them as one logical database. To access the link, you must be connected to the local database that contains the data dictionary entry. A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, then they must define a link that is stored in the data dictionary of database B. A database link connection allows local users to access data on a remote database. For this connection to occur, each database in the distributed system must have a unique *global database name* in the network domain.
- The global database name uniquely identifies a database server in a distributed system. It supports the user accounts which are necessary to establish server-to-server connections must be available in all databases of the distributed database system.

Disadvantages

In distributed databases, multiple users work on the same database but each user assumes that he is the only user using the database. In such an environment, data consistency and integrity must be maintained properly. To implement the distributed databases successfully, DBA handles the following operations:

- Query processing
- Catalog management
- Update propagation
- Recovery control
- Concurrency control

Each of these operations has various problems associated with it as discussed in the following pages:

Query processing

Minimization of network utilization involves all processes that are related to query processing, such as Query optimization process and Query execution process.

The problems related to query processing are given as follows:

- There is a lot of differences between communication time of each possible way of processing a query.

- The rate of data transfer and delay time is responsible for selecting a way of query execution.
- Computation and I/O (Input/Output) time of poor strategy are negligible as compared to communication time.

Figure 3.6 shows different query processing ways with their communication time.

Query Processing	**Time**
1 Transfer dress database at Site X.	1.11 hours
2 Transfer student and university database at Site Y.	11.22 hours
3 Check process on every tuple of the combination of student and University database according to the dress colour.	55.55 hours
4 Check process on student and university database at Site X.	20 seconds
5 Selected part of the database at Site X is transferred to the Site Y for processing.	1.11 hours
6 Selected part of the database at Site Y is transferred to the site X.	0.5 seconds

Fig. 3.6 *Query Processing Ways and their Communication Time*

Catalogue Management

Distributed system catalogue includes views, authorization and control information. Control information provides information related to the desired location, where the database is required, fragmentation and replication independence. Distributed system catalogue can be stored as centralized, fully replicated, partitioned and as a combination of centralized and partitioned. Centralized means the whole catalogue is stored at a central site of the network. Fully replicated means the complete catalogue is stored at every site of the network. Partitioned catalogue means every site of the network maintains its local catalogue and all local catalogues are combined as the main catalogue. A combination of centralized and partitioned catalogue means that each site maintains a local catalogue and all local catalogues are copied at the central site of the network. There are problems with each storage type of catalogue, which are given as:

- In the centralized and combination of centralized and petitioned system, the whole catalogue is stored at a central location on the network. Thus, there must not be reliance on the central site of the network for some services, because if the central system fails to work, then the whole system would not function properly.
- In the fully replicated system, every site maintains a local catalogue which causes lack of control over catalogues of the whole network.
- In the partitioned system, the central site contains the union of all local catalogues which requires expensive maintenance for the whole local catalogue in the central catalogue.

Update Propagation

The main problem of data replication is that it is not possible to make updation in all copies of the database because of unavailability of data. Primary copy is a scheme which is used for proper updation of database. These are the steps in primary copy scheme:

- Every updated copy of the data is designated as primary copy and the remaining copies are designated as secondary copy.
- Primary copies of a database are distributed at every site of the network.
- Updation of a database is deemed to be complete when the primary copy is updated.

The primary copy scheme also has some problems such as if the primary copy of any database is not available, then it is impossible to update the database.To avoid this problem, a specific time, which is decided by the user, is used for updation propagation. However, this solution is not right for database, because there are possibilities that the information of a database will be changed before that specific time and after creation of the update transaction.

Recovery Control

Recovery control in distributed database is based on the two-phase commit protocol. The two-phase commit protocol is the transaction protocol due to which all nodes and databases agree with each other to commit a transaction. This protocol is required in an environment where a single transaction can interact with multiple independent resource managers as in the case of distributed databases. It supports data integrity by ensuring that modifications made to transactions are either committed by all the databases involved in a distributed system or rolled back by all the databases.

The two-phase commit protocol works in two phases. The first phase is called the prepare phase during which the updates are recorded in a transaction log file, and the resource through a resource manager indicates that it is ready to make the changes. Resources can vote either to commit the updates or to roll back to their previous state. The activities performed in the second phase depend on the vote of resources. If all resources vote to commit then, all the resources participating in the transaction are updated, whereas if one or more of the resources vote to roll back, then, all the resources are rolled back to their previous state.

Consider an example in which an interaction between a coordinator at a local site and a participant at a remote site takes place, and a transaction has requested the commit operation. In the first phase, the coordinator instructs the participants to get ready and sends the get ready message at time T_1. Participants make an entry in the log and send the okay message as acknowledgement to the coordinator. Figure 3.7 shows that the acknowledgement has been sent at time T_3 and received at time T_4. After receiving messages from all the participants, the coordinator takes a decision to either commit or roll back the transaction. The transaction is committed if the messages received by all the participants are okay. It is rolled back if the messages received by all the participants. It is not okay. The first phase of the two-phase commit protocol is complete.

The coordinator then, writes an entry in the log, takes a final decision and sends it to the participants to do it or not to do it at time T_6. It is the beginning of the second phase. The participants receive it at time T_7 and send an acknowledgement to the coordinator at time T_8 that is received by coordinator at time T_9. Figure 3.7 shows the working of two-phase commit protocol.

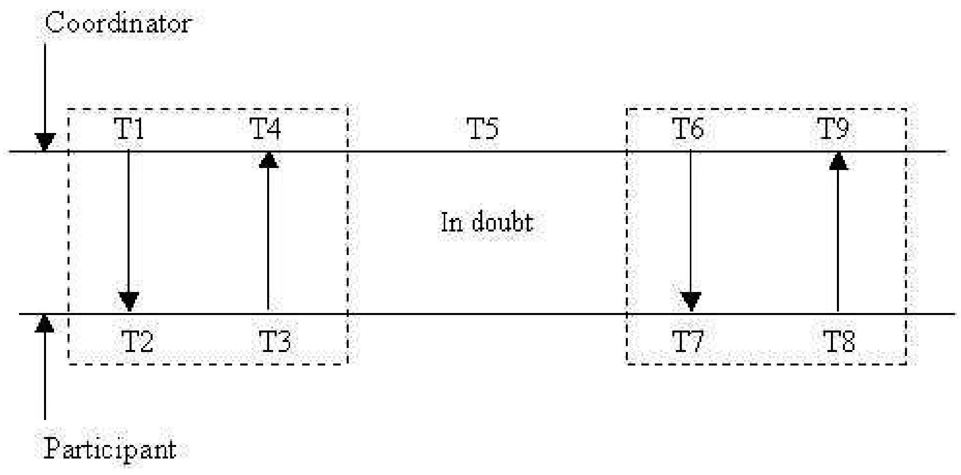


Fig. 3.7 *Two-Phase Commit Protocol*

Concurrency Control

In distributed database system, the problem of concurrency control is more complex than in a non-distributed database system, because in a distributed database system different users access the data stored in different computer systems. The part of a distributed database that is present on one computer system cannot predict the nature and type of transactions executing on other computer systems at that time. It is also difficult to predict the concurrency control mechanism executing on a different computer system. The concurrency control mechanism in a distributed database system can be performed by implementing locking process for which different requests, such as test lock, set lock and release lock are generated. These different requests to lock the part of the database involved in transaction processing are known as messages in the context of distributed database.

Consider a transaction, T_1 that will update an object. The details of the object are present at ten different sites. Each site has to generate the following five messages to lock its copy of the object:

(i) Lock request
(ii) Lock grants
(iii) Update object
(iv) Acknowledgment
(v) Unlock request

Therefore, for ten sites, fifty such messages are required to ensure the success of transaction $T_{1;}$ and the amount of time involved for the completion of T_1 is larger than the time taken by a similar transaction in a non-distributed database system. The time taken to complete the same transaction T_1 can be minimized by using the primary copy scheme, in which the site possessing the primary copy of the object that is to be updated will handle all locking operations.

NOTES

The locking process to implement concurrency mechanism in a distributed database system can lead to a global deadlock, which is a type of deadlock that involves two or more sites. For example, consider the two transactions T_1 and T_2 in Figure 3.8.

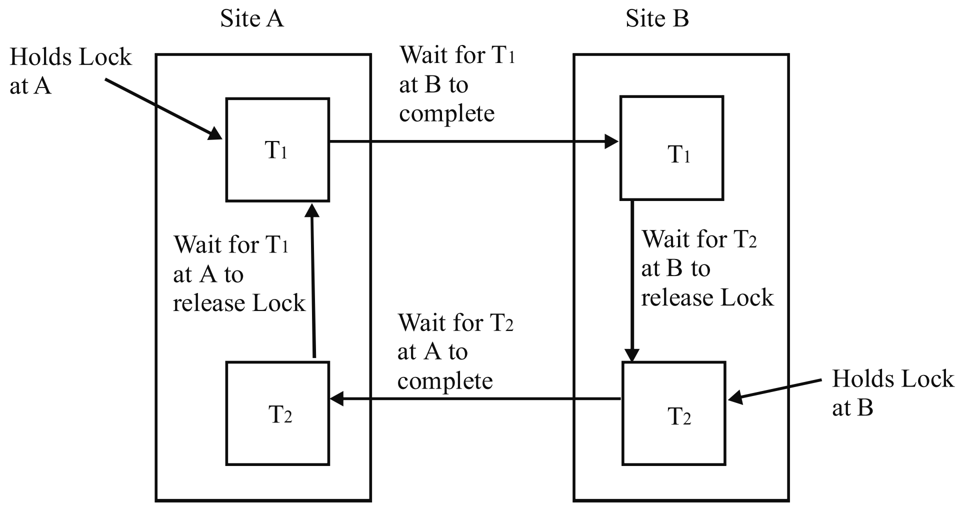

Site A
Site B
Holds Lock at A
Wait for T1 at B to complete
T1
T1
Wait for T1 at A to release Lock
Wait for T2 at B to release Lock
Wait for T2 at A to complete
Holds Lock at B
T2
T2

Fig. 3.8 *Global Deadlock*

From Figure 3.8, the following conclusions are made:

- The user of T_2 at Site A is waiting for the user of T_1 at Site A to release the lock.
- The user of T_1 at Site A is waiting for the user of T_1 at Site B to complete its transaction.
- The user of T_1 at Site B is waiting for the user of T_2 at Site B to release the lock.
- The user of T_2 at Site B is waiting for the user of T_2 at Site A to complete the transaction.

The direction of arrows in the figure shows the existence of a deadlock between the four concurrent transactions execution on two different Sites A and B. Since, the deadlock is spread over two different sites, it is known as global deadlock. In such case, the sites cannot even detect the presence of a deadlock situation. In order, to detect a global deadlock, the sites will have to pass some other information related to the details of the transaction running on each sites that will involve some overheads.

3.7 DISTRIBUTED DATABASE DESIGN

DBMS allows the data used in different applications and software to be processed in different levels. These levels are:

- **Single-Site Processing, Single-Site Data (SPSD)**: In SPSD level, processing is mostly performed on a single CPU or host computer. Here, DBMS is located on local disk of the host computer and users are connected with dumb terminals for accessing the database. The dumb terminals are terminals without having a processing unit. Processing unit of the host computer executes all the processes of these terminals using database of the host computer. DBMS of mainframe, minicomputer and single-user microcomputer uses SPSD level of data processing. Figure 3.9 shows the scenario of SPSD.

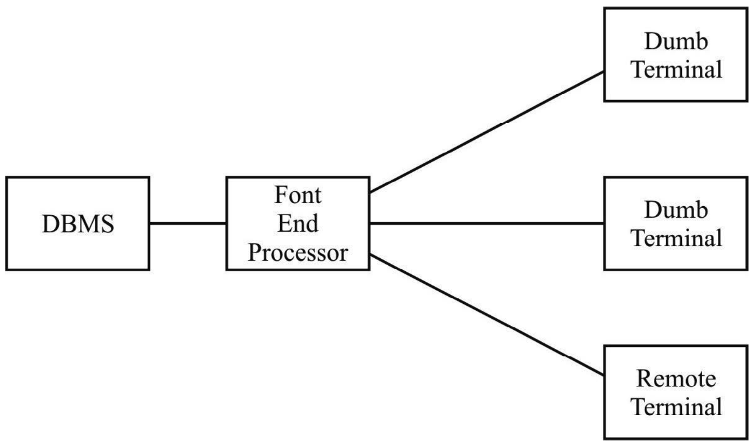


Fig. 3.9 *SPSD Scenario*

- **Multiple-Site Processing, Single Site Data (MPSD)**: In MPSD level of process and data distribution, the processing is done on different computers on the network but the data is stored at a single place. Thus, in MPSD, multiple processes that run on different computers can share a single repository of data. This scenario is also known as client-server architecture. All the applications access and retrieve the data from this single database. A network file server is used in this level, and this is suited for a scenario where many multi-user accounting applications are running under a personal computer network. Figure 3.10 shows the scenario of MPSD.

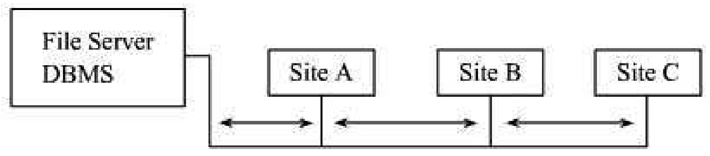


Fig. 3.10 *MPSD Scenario*

- **Multiple-Site Processing, Multiple-Site Data (MPMD)**: At the MPMD level, multiple data processors and transaction processors are connected with a DDBMS. MPMD is classified on the basis of how the DBMS is integrated in the network. There are two types of MPMD, homogeneous and heterogeneous. Homogeneous MPMD supports the integration of a single type of centralized DBMS over a network, while heterogeneous MPMD supports integration of different types of centralized DBMS over the network. Heterogeneous MPMD even supports different DBMSs, that can support varying data models such as relational, hierarchical or network.

These data models run on different computers that may be mainframes or microcomputers. Such types of heterogeneous MPMD are also known as fully heterogeneous MPMD. Figure 3.11 shows the scenario of MPMS.

NOTES

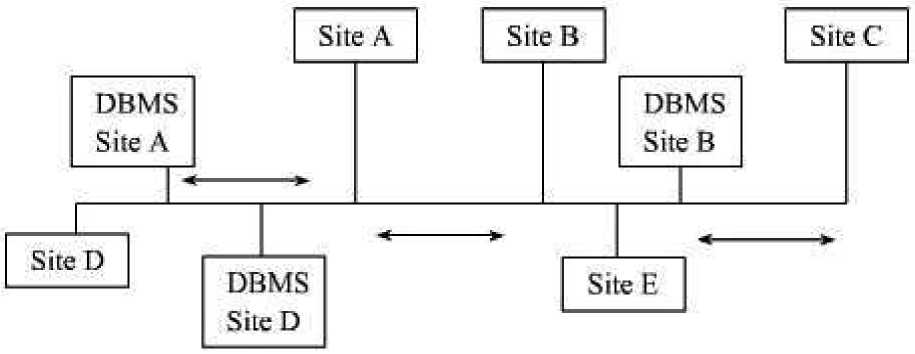


Fig. 3.11 *MPMD Scenario*

Design

The design of DDBs is carried out using the following techniques:

- Data Fragmentation
- Data Replication and Allocation

3.7.1 Data Fragmentation

Data fragmentation can be defined as a process that is used to break up a distributed database into logical units. These logical units are known as fragments that can be assigned to various sites of a Distributed database for storage. The fragments can be defined as relations that are stored a particular site. Distributed database can be is a collection of number of interrelated databases that are spread through a computer network. Distributed database system is used to improve the reliability and the performance of the database. The following are the types of data fragmentation:

- Horizontal Fragmentation
- Vertical Fragmentation

Horizontal Fragmentation: Horizontal fragmentation can be defined as a process that horizontally divides a relation by grouping rows to create subsets of tuples. Each subset has a logical meaning. These subsets of tuples, also called segments, are then assigned to different sites which are part of a distributed database system. A horizontal fragmentation of a relation is a subset of the tuples existing in that relation. These tuples in horizontal fragmentation are specified by a condition on one or more attributes of the relation. Consider, for example, that you have a relation called Employee as shown in the Table 3.2. Table 3.3 and 3.4 show horizontal fragments for the Employee relation. The fragment Employee2 shown in Table 3.3 is obtained by applying the condition (status = 20) and fragment Employee3 as shown in Table 3.4 is obtained by applying the condition (status = 40).

Table 3.2 *Employee*

Serial No.	Employee Name	Status	City
S5	Reeve	20	Paris
S4	John	40	London
S3	Blake	20	Athens
S2	Adams	10	London
S1	Clark	40	Paris

Table 3.3 *Employee 2*

Serial No.	Employee Name	Status	City
S5	Reeve	20	Paris
S3	Blake	20	Athens

Table 3.4 *Employee 3*

Serial No.	Employee Name	Status	City
S4	John	40	London
S1	Clark	40	Paris

Vertical Fragmentation: Vertical fragmentation can be defined as a process used to divide a relation vertically by columns. A vertical fragment of a relation includes only a few attributes of the relation. Consider, for example, you have a relation called Employee as shown in Table 3.2. If you fragment this relation vertically, then you get the two relations as shown in the Table 3.5 and 3.6. The fragment Employee 4 shown in the Table includes attributes Serial number and Employee name. The fragment Employee5 shown in Table 3.6 includes the attributes Status and City.

Table 3.5 *Employee4*

Serial No.	Employee Name
S5	Reeve
S4	John
S3	Blake
S2	Adams
S1	Clark

NOTES

Table 3.6 *Employee 5*

Status	City
20	Paris
40	London
20	Athens
10	London
30	Paris

3.7.2 Data Replication and Allocation

Data replication is a technique of storing certain data at more than one site in a DDB. This improves the availability of the system, because the system can continue to operate as long as at least one site is working properly. It can also improve the performance of retrieval f1or global queries because the result of such a query can be obtained locally from any one site. The most extreme case of replication is to store the whole database at every site in the DDB system. This creates a fully replicated distributed database.

Each fragment or each copy of a fragment must be assigned to a particular site in the DDB system. This process is called data distribution or data allocation. The arrangement of sites and the degree of replication depends on the performance and availability needs of the system, and on the types and frequencies of the transactions submitted at each site. For example, if the requirement of the system is high availability, transactions can be submitted at any site and most of the transactions are retrieval only, then a fully replicated database would be a good choice.

Replication

Database replication is a highly flexible technology for copying updates automatically between databases. The idea is that if you make a change to one database, other database copies update automatically. Replication occurs at the database level and does not require any special actions from client applications. Propagating updates automatically is a simple idea, but it helps solve a surprisingly large number of problems, as shown in Figure 3.12.

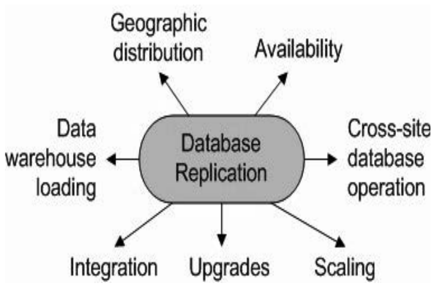


Fig. 3.12 *Replication Benefits*

Database replication is the creation and maintenance of multiple copies of the same database. In most implementations of database replication, one database server maintains the master copy of the database and additional database servers maintain slave copies of the database. Database writes are sent to the master database server and are then replicated by the slave database servers. Database reads are divided among all of the database servers which results in a great performance advantage due to load sharing. In addition, database replication can also improve availability because the slave database servers can be configured to take over the master role if the master database server becomes unavailable. Data replication is very attractive in order to increase system throughput and provide fault-tolerance. However, it is a challenge to keep data copies consistent. Furthermore, in order to fully take advantage of the processing power of all replicas, adaptive load-balancing schemes are needed. One of the very famous methods of replication is Middleware-Mased Replication.

Middle-R is the middleware based replication tool working with DBMs. It provides efficient, fast and consistent database replication for both cluster configurations where all replicas are within a LAN and in WAN environments. Each database replica is an instance of a non-replicated standard database system. It provides a basic approach of fault-tolerance. Replication in DBMS works with following mechanism:

- **Isolation levels:** It allows for different levels of isolation of concurrent transactions. It focusses on snapshot isolation.
- **Wide-area systems:** It provides transparent, efficient and consistent data replication in wide area networks. The usual communication technology used in clusters, e.g., group communication systems does not work well in WAN settings.
- **Partial replication:** While full replication places copies of data items at all replicas, partial replication only assigns copies of an individual data item to some replicas. When there is a high update workload, full replication has too much overhead to keep all copies consistent and the individual replicas have little resources left to execute read operations. In contrast, with partial replication, a replica only has to execute the updates for data items for which it has local copies, and thus, has more potential to execute read operations. It also addresses many challenges associated with partial replication, such as a more complex concurrency control, the challenge of finding a replica with the data copies needed for a request, and finally with the necessity of distributed query execution.
- **Relationship between middleware and database system:** When implementing a replication solution outside the database system, the replication tool does not have access to important components within the database system, such as concurrency control. Thus, functionality has to be re-implemented in the middleware.

Characteristics of DBMS Replication

The following are the characteristics of database replication with reference to concurrency control:

- **Availability.** Keeping multiple copies of data is one of the most effective ways to avoid database availability problems. If one database fails, you

NOTES

can switch to another local copy or even to a copy located on another site.

- **Cross-site database operation.** Applications like credit card processing use multiple open databases on different sites, so that there is always a database available for transactions. Replication can help transfer copies between distributed databases or send updates to a centrally located copy.
- **Scaling.** Replicated copies are live databases, so you can use them to distribute read traffic. For example, you can run backups or reports on a replica without affecting other copies.
- **Upgrades.** Replication allows users to upgrade a replica which can then be switched over to the master copy. This is a classic technique to minimize downtime as well as provide a convenient back-out in the event of problems.
- **Heterogeneous database integration.** It is quite common for data to be entered in one database type, such as Oracle, and used in another, such as MySQL. Replication can copy data between databases and perform transformations necessary to ensure proper conversion.
- **Data warehouse loading.** Replication applies updates in real time. This is very useful as databases become too large to move using batch processes. Data warehouse loading is much easier with capabilities such as transforming data or copying updates to a central location.
- **Geographic distribution.** Replication allows users to place two or more clusters in geographically separated locations to protect against site failure or site unreachability.

The database replication is considered as an essential technology to build and operate a wide variety of business-critical applications. Tungsten Replicator is designed to solve these problems as well as many others.

In a master/slave replication, updates are handled by one database server, known as the *master* and propagated automatically to replicas, which are known as *slaves*. This is a very efficient way to make database copies and keep them up-to-date as they change.

Master/slave replication is based on a simple idea. Let us assume that two databases start with the same initial data, which may be called a *snapshot*. Changes on one database, recording them in order so that they can be replayed with exactly the same effect as the original changes. It is called a *serialized order*. If replayed, you will get the serialized order on the second database, you know how master/ slave replication.

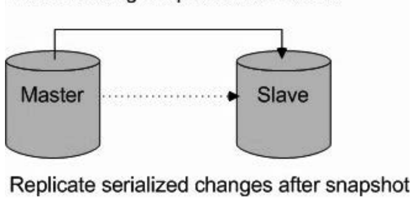

Initialize using snapshot from master
Master
Slave
Replicate serialized changes after snapshot

Fig. 3.13 *Master/Slave Replication*

Figure 3.13 shows that master/slave replication is popular for a number of reasons. First, databases can generate and reload snapshots relatively efficiently using backup tools. Second, databases not only serialize data very quickly but also write it into a file-based log that can be read by external processes. Master/slave application is, therefore, reasonably tractable to implement, even though the effort to do it well is not small.

Fragmentation

Fragmentation consists of breaking a relation into smaller relations or fragments possibly at different sites. Database applications work with views rather than entire relations, therefore, data is stored close to where it is mostly frequently used. It supports parallelism with fragments so that they are the unit of distribution. A transaction can be divided into several subqueries that operate on fragments.

- **Security:** Data not required by local applications is not restored, and consequently not available to unauthorized users.
- **Performance:** The performance of global applications that require data from several fragments located at different sites may be slower.

Types of fragmentation

The types of fragmentation are as follows:

- **Horizontal fragmentation :** A subset of the tuples of a relation, defined as sp(R), where p is a predicate based on one or more attributes of the relation. The union of horizontal fragmentation must be equal to the original relation requiring disjoint.
- **Vertical fragmentation:** A subset of the attributes of a relation, denoted as Pa_1, a_2, .., a_n (R), where a_1, a_2, .., an are attributes of the relation R. Vertical fragmentation has happened in loss-less joins. In this, systems often assign a unique tuple id for each tuple in the original relation and attach it to the vertical fragmentation.
- **Mixed fragmentation:** A horizontal fragment that is subsequently vertically fragmented, or a vertical fragment that is then horizontally fragmented.

Let us consider a very good example of database fragmentation within the skeletal structure of a DBMS such as Turbo Image. While specific internal DBMS fragmentation is outside the scope of all, a DBMS exists on top of the file system. This means that there is still significant impact of defragmenting Turbo Image data sets at the file level. In other words, given the database TRXDB1, you should still defragment the individual files of TRXDB101, TRXDB102, TRXDB103, etc. It is said that the internal fragmentation of data within a DBMS is still a critical performance issue. Fragmentation in a DBMS is done via the DETPACK command, to actually fix DBMS-level fragmentation on Turbo Image databases. Table 3.7 shows fragmentation in DBMS.

Table 3.7 *Fragmentation in DBMS*

Table_Name	Rows	Table_Pages	Text_Pages
Child	25000	2689	10013
Normal	1	1	4917
Parent	1	3	56
tblob	500	46	2

3.8 DISTRIBUTED DATABASE QUERY PROCESSING

Minimization of network utilization involves all processes which are related to query processing, such as query optimization process and query execution process. All optimization processes should contain global optimization step followed by local optimization step. For example, query A implies a union of the relation R1 at site N and relation R2 at site O. R1 and R2 have different number of tuples. Query A is submitted to an optimizer at site M. The optimizer at site M will want to execute query A globally because it is essential to move the query from R1 to O for the desired purpose. Therefore, he will use a global strategy of execution of query A. The union at O site is done by local optimizer of site O.

Consider a query for finding roll numbers of students who are related to ABC university and wear red dress. The query will be applied on three databases: Student, University and Dress from different sites. The student database consists of 10,000 tuples with primary key Student_roll no at site X. The University database consists of 100000 tuples with primary key University code at site X and Dress database consists of 1000000 tuples with primary key Dress color at site Y. Every tuple of the database has a length of 200 bits. The query will give student roll number of ABC University whose dress colour is Red. Query for finding student roll number is given as follows:

```
((Student join University join Dress) where University =
'ABC' AND color = red){Student_rollno #}
```

The result of the above query contains 10 tuples from the Dress database and 100000 from the University database. There are different ways for executing this query, which are as follows:

1. Transfer all required parts of the database to site X and execute the query at site X.
2. Transfer required parts of the Student database and Dress database to site Y and execute the query at site Y.
3. Combine Student and University database using Join operator at site X according to ABC University. After that, apply a check process on every tuple of the combination of Student and University database according to the dress colour. The check process includes two messages: query to check condition and response of the query.
4. Select all tuples from Dress database according to the red colour at site Y. After that, apply the check process on Student and University database at site X.
5. Combine Student and University database using Join operator according to ABC University. Select Student_roll numbers and Universities name columns from the Student and University database using Projection operator. Transfer the result to the site Y and do further processing for finding result of the query at site Y.
6. Select all the tuples having Red dress from Dress database at site Y. Transfer the result to the site X and do further processing for finding result of the query at site X.

There are some assumptions of communication features of the database, such as data rate of the network is equal to 5000 bits per second and access delay for the data transfer is 0.1. Now, you can calculate the total communication time T[n] of above methods for getting the result of the above query. For calculating T[n] , you have to use the following formula:

```
(Total access delay) + (Total data volume/data rate)
```

If the number of messages in a query processing is given, you have to use the following formula for calculating the communication time:

```
(Number of messages/10) + (Total data volume/data rate)
```

Calculation of communication time for different methods is given as follows:

- Communication time calculation for the first method where all required parts of a database are transferred at site X for execution, are given as:

  T[1] = 0.1+(100000*200)/5000
  = 4000.1 sec, i.e., 1.11 hr

  In this calculation, 100000 is the number of Dress database, which will be transferred from site X to site Y.

- Communication time calculation for the second method where Student, Dress and University database are transferred at site Y for execution, is given as:

  T[2] = 0.2+((10000+1000000)*200)/5000
  = 40400.2 sec, i.e., 11.22 hr

  In the above calculation, 0.2 is the access delay time for the two database, i.e., 0.1+0.1 = 0.2. 10000 and 1000000 are the number of tuples of Student and University database that are transferred from site X to site Y.

- Communication time calculation for the third method where check process on every tuple of the combination of student and University database is applied according to the dress colour.

  T[3] = 200000 sec, i.e., 55.55 hr

- Communication time calculation for the fourth method where check process is applied on Student and University database at site X according to dress colour.

  T[4] = 20 sec

- Communication time calculation for the fifth method where Student and University databases are combined and selected and part of the database at site X is transffered to the site Y for finding result, is given as:

  T[5]= 0.1+(100000*200)/5000
  = 4000.1 sec, i.e., 1.11 hr

  In the above calculation, 0.1 is the access time for selected database of site X and 100000 is the number of resultant tuples.

- Communication time calculation for the sixth method where the selected part of the Dress database is transffered to the site X for finding result, is given as:

  T[6] = 0.1+(10*200)/5000
  = 0.5 sec

In this above calculation, 0.1 is the access time for selected database of site X and 10 is the number of resultant tuples of Dress database.

The problems related to query processing are given as follows:

- There is a lot of difference between communication time of each possible way of processing a query.
- The rate of data transfer and delay time is responsible for selecting the way of query execution.
- Computation and I/O time of poor strategy are negligible as compared to communication time.

Check Your Progress

4. What do you understand by distributed databases?
5. Define the term data fragmentation.
6. What are included in minimization of network utilization?

3.9 ANSWERS TO 'CHECK YOUR PROGRESS'

1. Object-oriented technology encompasses programming languages, projects, hardware methodologies and object-based approaches.
2. Several major software companies including International Business machines or IBM, Informix, Microsoft, Oracle and Sybase have all released object relational versions of their products. These companies are promoting a new and extended version of relational database technology called object relational database management systems also known as ORDBMSs.
3. Inheritance is considered as the most powerful feature of object-oriented programming after classes themselves. New classes are created with the help of inheritance. Derived classes can be created from existing or base classes. The derived class inherits all the functions of base classes but it supports embellishments and refinements of its own, i.e., producing properties to the child class form parent class.
4. Distributed database (DDB) is a collection of multiple interrelated databases that are spread over a computer network. Each computer contains its own database that is managed by an individual database management system.
5. Data fragmentation can be defined as a process that is used to break up a DDB into logical units. These logical units are known as fragments that can be assigned to various sites of a Distributed database for storage. The fragments can be defined as relations that are stored a particular site.
6. Minimization of network utilization involves all processes which are related to query processing, such as query optimization process and query execution process.

3.10 SUMMARY

- Object-oriented technology encompasses programming languages, projects, hardware methodologies and object-based approaches.
- The object-oriented programming (OOP) paradigm was developed to revolutionize the process of software development. It not only includes the best features of structured programming, but also introduces some new and advanced features that the procedural programming lacked.
- The ‘object’ is defined as conceptual and physical things. Documents, software concepts and living beings are examples of objects.
- A class is defined as a user-defined data type, which contains the entire set of similar data and the functions that an object possesses.
- Abstraction is a mechanism to hide irrelevant details and represent only the essential features so that one can focus on important things at a time. It allows managing complex systems by concentrating on the essential features only.
- Encapsulation is the technique of binding or keeping the data and functions operating together in a single unit called class. Encapsulation is the way to implement data abstraction.
- Polymorphism is the ability of an entity, such as a function to be processed in more than one form. This concept is widely used in object-oriented database designing.
- Several major software companies including International Business machines or IBM, Informix, Microsoft, Oracle and Sybase have all released object relational versions of their products. These companies are promoting a new and extended version of relational database technology called object relational database management systems also known as ORDBMSs.
- The main advantages of extending the relational data model come from reuse and sharing. Reuse comes from the ability to extend the DBMS server to perform standard functionality centrally rather than have it coded in each application.
- Inheritance is considered as the most powerful feature of object-oriented programming after classes themselves. New classes are created with the help of inheritance. Derived classes can be created from existing or base classes. The derived class inherits all the functions of base classes but it supports embellishments and refinements of its own, i.e., producing properties to the child class form parent class.
- The class hierarchy can be defined in pseudocode in which the variables are associated with each class.
- Distributed database (DDB) is a collection of multiple interrelated databases that are spread over a computer network. Each computer contains its own database that is managed by an individual database management system.
- DDB technology has evolved as a combination of two technologies, one is the database technology and the other is the network and data communication technology.

- Data fragmentation can be defined as a process that is used to break up a DDB into logical units. These logical units are known as fragments that can be assigned to various sites of a Distributed database for storage. The fragments can be defined as relations that are stored a particular site.
- Horizontal fragmentation can be defined as a process that horizontally divides a relation by grouping rows to create subsets of tuples. Each subset has a logical meaning. These subsets of tuples, also called segments, are then assigned to different sites which are part of a distributed database system.
- Vertical fragmentation can be defined as a process used to divide a relation vertically by columns. A vertical fragment of a relation includes only a few attributes of the relation.
- Data replication is a technique of storing certain data at more than one site in a DDB. This improves the availability of the system, because the system can continue to operate as long as at least one site is working properly.
- Fragmentation consists of breaking a relation into smaller relations or fragments possibly at different sites. Database applications work with views rather than entire relations, therefore, data is stored close to where it is mostly frequently used.
- Minimization of network utilization involves all processes which are related to query processing, such as query optimization process and query execution process.

3.11 KEY TERMS

- **Object:** The 'Object' is defined as conceptual and physical things.
- **Abstraction:** Abstraction is a mechanism to hide irrelevant details and represent only the essential features so that one can focus on important things at a time.
- **Inheritance:** The process of one object acquiring from one or more objects.
- **Distributed Database:** A distributed database or DDB is a collection of multiple interrelated databases that are spread over a computer network.
- **Vertical fragmentation**: Vertical fragmentation can be defined as a process used to divide a relation vertically by columns. A vertical fragment of a relation includes only a few attributes of the relation.
- **Data Replication**: Database replication is a highly flexible technology for copying updates automatically between databases.

3.12 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. State three features of object-oriented programming.
2. Write the main advantage of relational database model.

3. What are complex objects?
4. Define the term class hierarchy in inheritance.
5. What are the features of distributed data transparency?
6. What do you understand by fragmentation?
7. State the ways of executing a distributed query.

Long-Answer Questions

1. Describe the concept of object-oriented databases giving appropriate examples.
2. Discuss the characteristics of Object Relational Database Model System (ORDBMS).
3. Explain the meaning of inheritance by giving example programs.
4. Analyse the function overloading rules with the help of appropriate example.
5. Describe the distributed database system by giving example programs.
6. Discuss the process of distributed database design with the help of appropriate examples.
7. Explain the data replication and allocation by giving examples.
8. Discuss distributed database query processing.

3.13 FURTHER READING

Navathe, S.B. and R. Elmasri. 1997. *Database System Concepts*, Third edition. New York: McGraw-Hill.

Korth, Henry F., Avi Silberschatz and S. Sudarshan. 2010. *Database System Concepts*, 6th edition. New York: McGraw-Hill.

Elmasri, Ramez and Shamkant B. Navathe. 2007. *Fundamentals of Database Systems*, 5th edition. New Jersey: Pearson Education.

Martin, James. 2007. *Principles of Data Base Management*. New Jersey:

Pearson Education.

Martin, James. 1975. *Computer Data-Base Organization*. New Jersey: Prentice Hall.

Jackson, Glenn A. 1988. *Relational Database Design With Microcomputer Applications*. New Jersey: Prentice Hall.

UNIT 4 DATABASE SECURITY AND AUTHORIZATION

NOTES

Structure

4.0 INTRODUCTION

Database security comprises policies taken to protect data from being accessed by unauthorized users. This also includes the policies needed to ensure that data items are not modified or deleted from the database by unauthorized users, applications or viruses. Threat refers to any deliberate or accidental occurrence of action that may badly affect the database system. Database security can be violated by many sources. Humans are an important and deliberate source of threat to database. Different types of persons can pose different threats. Most common threat is Users gain unauthorized access through the use of another person's account. A database system is a collection of application programs that interacts with the database along with DBMS (and sometimes the users who use the system). Database systems are designed in a manner that facilitates the management of huge bodies of information.

To prevent this from occurring, the DBMS implements a concurrency control protocol that prevents concurrent accesses from interfering with one another recovery is the process of restoring the database to a correct state following some type of failure. The failure may have occurred in either/both hardware or software. Malicious corruption or destruction of a component of the database system is also a problem associated with the reliability and consistency of the database. While recovery from such an event certainly must be handled by the DBMS, its prevention falls under DBMS security issues.

In this unit, you will study about the security and integrity threats, international or malicious threats, defense mechanisms, security policies, authorization, object and view as objects, granularity and subject, access types, database operating system, features of a database OS, concurrency control model, theory of serializability and concurrency control algorithms.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basics of security and integrity threats
- Discuss the international or malicious threats
- Define the defense mechanisms
- Explain the concepts of security policies
- Describe about the authorization
- Define the object and view as objects
- Discuss the granularity and subject
- Explain the significance of access types
- Understanding the database operating system
- Discuss the features of a database OS
- Analyse the concurrency control model
- Explain the theory of serializability
- Describe the concurrency control algorithms

4.2 DATABASE SECURITY AND AUTHORIZATION

When many users access a database simultaneously, it becomes essential to make sure that there is no effect on the values of the data items and the association among them to disturb the database's integrity.

Hence the data manipulation operations should be performed in such a way that it does not disturb the integrity of the database. Integrity of a database refers to the correctness of the data items. Integrity ensures the nonoccurrence of accidental deletion or alteration.

The data stored in the database has to be protected from unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency. The integrity of a database can be ensured by designing it in such a way that it helps check the integrity constraints. In spite of crashes, failures and potential damage from parallel processing, database integrity can be preserved. Integrity checks can be performed at the level of data entry by checking whether the data values adhere to preset/specific rules. The age of an employee, for instance, will be within a certain range of say 20 to 60 years.

The threats to the database security involve the following:

(a) Unauthorized access can damage a database.

(b) The authorized users can give privileges/benefits, deliberately or accidentally, to those users who lack the time to access databases.

(c) As a result of concurrency, the database items can be read or written simultaneously by two items different transactions. This causes inconsistency in the database.

(d) Insertion of virus and destruction of data in the database or any kind of unauthorized access by unwanted users or application programs could also be considered fatal to the database.

(e) A database user can cause serious damage to the database by bypassing the security mechanisms intentionally or by making unauthorized copies of secret data for malicious purposes.

(f) Secret information can be transmitted forcefully or under pressure by authorized persons; may be even for some personal benefit or gain.

(g) Errors in the DBMS package may be caused due to memory protection failure in warding off attacks by viruses.

(h) It may so happen that a user may gain access to that part of a database which he is otherwise unauthorized to access. This could happen due to system malfunctioning of some kind.

4.2.1 Database Security

Database security comprises policies taken to protect data from being accessed by unauthorized users. This also includes the policies needed to ensure that data items are not modified or deleted from the database by unauthorized users, applications or viruses. Simply put, the following issues are handled by data security:

(a) Maintenance of database integrity

(b) Maintenance of privacy of certain data elements

(c) Security level for the system

(d) Preservation of organizational policies

Privacy of certain data elements

Privacy is concerned with the ethical and legal rights regarding access to personal data items. The medical details of a person, for instance, may be considered critical information. Unauthorized persons should be denied access or kept from modifying such information/records

Preserving organizational policies

Organizations, government and corporate-level, as well as various institutions have certain policies regarding the type of information that needs to be publicized. An individual's credit rating or medical history, for instance, should be kept private.

System-related security level

It is important to know the level at which security should be enforced on the system. It could be enforced at the level of the operating system, or at the level of the hardware or even at the DBMS level.

Maintaining database integrity

Items of data stored in the database should be consistent as well as valid. Database integrity constraints play a significant role in ensuring database integrity. Controlling access to the database items also ensures integrity of the database.

Traditionally, two types of security mechanisms are in use.

(i) Discretionary security mechanisms: Here, each user (or a group of users) is granted privileges and authorities to access certain records, pages or files and denied access to others. The discretion normally lies with the DataBase Administrator (DBA)

(ii) Mandatory security mechanisms: These are standard security mechanisms that are used to enforce multilevel security by classifying the data into different levels and allowing the users (or a group of users) access to certain levels only based on the security policies of the organization. Here the rules apply uniformly across the board and the discretionary powers are limited.

Database Security Requirements

Database security requires the following:

(a) The primary requirement of database security is to forbid an individual from obtaining something by unfair means and entering into an organization's computing facility. Security laws should be so implemented that they make any attempt of unauthorized users to access systems resources illegal.

(b) Use of computers and terminals should be limited to authorized users only. Securing physical storage devices (such as magnetic tapes, disk drives) both inside the organization and while sending them outside is a must.

(c) The database administrator determines whether a user should be given the privilege to access a particular resource and according to that the user should be provided with the username and password and it should be kept confidential from the others.

(d) Along with database the operating system should also be made secure by providing some built-in safety mechanisms such as user authentication and identification. Direct access to the data in primary files should also be avoided.

Dimensions of Security

Different levels of security are provided by database management systems. These are discussed as follows:

(a) **Network-Level Security:** Distributed database systems require security at the network software level. Network Software ensures security by means of some in-built methods that actually provide login security control permitting only the authorized users to log onto the system and gain access over the resources. Network software also provides security by giving rights. The directories, subdirectories and files, accessed by a user or a group of users, are controlled by rights security. Attribute security is also offered by network software. Attribute security allows or disallows a user or a group of users to view, share, rename, modify or delete a particular file (or directory). Therefore, only if the attribute of a file allows a privileged user or a group of users to delete a particular file, can it do so.

(b) **Operating System (OS) Level Security:** Only a strong DBMS software can provide enough security. A weak operating often allows unauthorized access to database files. So, it is equally important to secure the OS.

(c) **Database System Level Security:** Certain users of database are allowed to access some specific portions of the database. In other words, they can generate only certain operations on the database. Suppose a user, Mr. Mukherjee is permitted to view information. That is, he can issue SELECT queries only. He can't issue INSERT, UPDATE or DELETE queries as he doesn't have the permission to make any modification.

(d) **Program-Level Security:** Program-level security ensures that no unauthorized user can get access to particular programs accessing the database. For example, a bank clerk who retrieves only the customer account details has access to the program responsible for that particular job. He does not have access to the program that modifies the accounts.

(e) **Record-Level Security:** It denies unauthorized access to certain kinds of records, for example, the records of managers in the EMPLOYEE table.

(f) **Field-Level Security:** It prohibits unauthorized access in certain fields of a table such as the Salary field.

Implementing Security Features

It is the responsibility of DBA to ensure integrity of database by taking appropriate steps and preventing the database from being accessed by unauthorized users. The DBA permits the use of the database and also stores the profile of each database user. This profile describes what permissions are given and which portions of the database the user can operate on. Through the user profile DBA monitors the access rights given to a particular user and also the time frame within which the permission is valid.

Complex User Management Requirements

Despite its importance, user management is often neglected in maximum database management systems. It is necessary to have an efficient system for addition of users, for management of turnover, for support of connectivity, for issuance of passwords and for directory authentication to be able to implement a good user management system.

These systems should be implemented not only of the existing users and applications but also for the new ones. This is especially true for organizations dealing with business partners, customers, suppliers and other third-party users who make queries, place orders, buy products and communicate online.

To meet the these requirements using current IT tools is almost next to impossible as it takes too long to add and remove users. Addition and removal of privileges and users should be done in real time. The method followed by the current IT tools to approve and register emails and manuals is very slow.

The major issues with the existing methods of user authentication are as follows:

(a) It takes too long to check the validation of user identity. The procedure is costly too.

(b) Accounts of the users who leave without any notification are often left non-deleted which increase security risks.

(c) In some cases, corruption of authentication directory or disturbances due to system failure prevents users from accessing applications.

Solutions to user management problems are as follows:

(a) One way to solve the problem related to cost can be solved by following a centralized administration system. The user's account and administration system are combined at one place and hence reduces the infrastructure cost. The responsibility of adding, deleting and approving users are handed over to the local administrators.

(b) The user management methodology being followed should provide provision for enrollment, administration and support for external user communities (such as vendors, customers etc.)

(c) The methodology should provide provision for generation reports on user performance in daily, weekly and monthly basis. Accepted or rejected authentication attempts should also be listed in the report.

(d) Apart from making sure that the users can authenticate and access applications round the clock the user management methodology being employed should offer scalability, performance and availability backed up by a service level agreement.

(e) Most of the time invalid or old accounts are mistakenly left active. Now this causes risks. These risks can be mitigated by revoking user's access privileges in the least possible intervals.

4.2.2 Defense Mechanism

Protecting data within the database

The primary requirement of protecting data stored in the database is that it should be kept confidential to the unauthorized users and application programs. We adopt numerous security measures to prevent the database from unauthorized accesses. These are discussed in the following sections.

Database Audit

Database system lists the required information regarding all the users and the operations performed by them during a login session. As soon as a user logs-in the system records the account number and the terminal from which it has logged in.

Until the user logs off all the operations performed by the user is recorded and associated with the user's account number.

Any suspicion of tampering the database leads to a database audit. The operations or accesses performed on a certain period of time by a particular user account is then reviewed. If an illegal operation is found the DBA immediately determines the account number performing the operation.

A database log that is used mainly for security purposes is called an audit trail.

Authenticating Users to the Database

Identification and authentication of a user can be done through any of the following methods:

a) **What you know:** Easiest way to identify an authorized user is making use of its username and password for authentication purpose. Along with the previous technique some database systems ask some questions for authenticate valid user. Only the authorized users can come up with correct answers.

b) **What you have:** Identification of valid user can be carried out by giving each of them a badge, card or key. Providing with a password or holding a question answer session is also possible for identification.

c) **What you are:** Sometimes it is also done by some software or hardware. These make use of some physical or psychological characteristics of the user.

Statistical Databases

Statistical databases store confidential details about organizations or users, for example, a database containing the income and medical records of various individuals.

Statistical database aims at maximizing information-sharing and preserving the privacy of individuals. It lets users have full access to the data in a database with an aim of increasing information sharing. Privacy of individual information is preserved by processing only those queries that produce large records. For example, a statistical database is not going to give any response to the query that wants to retrieve the salary of a particular employee from the database.

A little knowledge about the state of the database can help a malicious user know the salary of a particular employee. For example, suppose Mr Banerjee and Mr Thakur are the only employees in a particular department of a company. Mr Banerjee wants to know the salary of Mr Thakur. But the statistical database denies retrieving the information. Now, if Mr Banerjee knows that Mr Thakur is the only employee except him in the department then he can easily find out the salary of Mr Thakur by posing the following query as he knows his own salary:

```
SELECT SUM (EMP_SALARY) FROM EMP WHERE DEPT_CODE = 10;
```

Suppose, the answer to the above query comes to be 50,000. Then, Mr Banerjee's salary being ₹ 20,000, Mr Thakur's salary can be computed as ₹ 50,000 – ₹ 20,000 = ₹ 30,000.

To protect private information stored in the database from being accessed by such smart users, the following methods can be used:

a) If a query involves a small number of records, it must be rejected.
b) If a user makes a query that involves a result whose intersection with the records retrieved in the previous queries is very high, the former query should be rejected. This is to prevent such users as Mr Banerjee. This can be implemented by maintaining a history of records that were used in queries asked by a particular user.
c) A third method is random falsification. In this approach, a small amount of data is randomly made wrong. The result of statistical computation of such data remains correct so that normal users do not suffer.
d) Another solution is to choose a random data sample representing the database in total. This sample data is then used to give response to any query. This method relieves the normal user from any sufferings and also protects the database from any danger.
e) The users can be also discouraged from involving in any malicious activities by performing audit trials at regular intervals and recording the identity of the user and their interaction with the database.

Data Encryption

In a database, it is possible to store confidential data in either cipher form or in an encrypted (coded) form. To read the content of the data stored one needs to decrypt (or decipher) it first. The mechanism needs an encryption device which applies an encryption algorithm to convert it to cipher text and also a decryption device that applies reverse method to get the original text back. The method of encryption is accomplished by specialized software.

The algorithm that converts the meaningful plain text to meaningless cipher text is known as encryption algorithm. A good encryption algorithm should possess the following features.

a) The algorithm should be so simple that it can be easily used by authorized users for efficient encryption and decryption.
b) It is very difficult to keep the algorithm secret. Hence, instead of keeping the algorithm secret the parameters that the algorithm takes are kept secret. So, security of data depends on the secrecy of the parameters generally known as key(s). Key should be known to the authorized users only.
c) The algorithm being used should be so secured that the intruders cannot easily determine the key(s) used.

Public-Key Encryption

One of the many encryption techniques is the public key encryption. This technique uses two separate keys for encryption and decryption. Say, the key used for encryption is A and for decryption is B. The encryption key is placed in a public register and is known as the public key. The private key, also known as the decryption key is kept secret.

A public key, A is given to every user, U along with a private key, B1. The public key, as the name suggests is made public or is known to everybody. The private key, on the other hand, is only known to its owner. User U1 can store encrypted data by using A, the public key, to encrypt them. The private key, B, is used to decrypt as it is only known to the user U1. However, if user U1 wishes to share this data with user U2, he will encrypt the data using A2. But, if the user U1 wants to share this data with user U2, he will use A2 to encrypt and user U2 can safely decrypt the data with his private key B2.

Granting and Revoking Privileges and   Roles

Access control mechanisms are implemented to restrict the database access only to the authorized users. Each user is given a username and a password. The database administrator grants access permissions and user accounts to various users of the database.

Following are the responsibilities of DBAs:

(a) Account Creation	DBA creates user accounts and each user or group of users is given a password
(b) Privilege Granting	A user or a group of users is granted privilege to access certain parts of a database
(c) Privilege Revocation	DBA can also revoke permission (that was previously given) to access certain parts of a database from a user or group of users.
(d) Security Level Assignment	Assigning user accounts to the appropriate security level.

Any user or group of users interested in accessing the database needs to apply for a user account first. It is up to the DBA to grant or not to grant permission. If granted, the user or group of users can log in to the system using the username and password. Then, the validity of the username and password is checked. A valid username and password pair permits the user to access certain portions of the database.

Note: Application programs are also considered as users by the DBMS and are required to supply user account and password.

4.3 SECURITY AND INTEGRITY THREATS

Threat refers to anydeliberate or accidental occurrenceof action that may badly affect the database system. Database security can be violated by many sources. Humans are animportant and deliberate source of threat to database. Different types of persons can pose different threats. Most common threat is Users gain unauthorized access through the use of another person's account. Some of the threats: People use another person's log in credentials to access their data, unauthorizedreading and stealing data, Program/Data modification, Illegitimate entry by hacker, Sending Virusesto block others data Etc.

- **Different Security Threats in Database**

1. **Excessive Privilege Abuse:** When a user is granted with database accessrights that go beyond their job functionrequirements, then thoserightsare sometimes misused for malicious purpose.

   For example: ACollege officeadministrator, whose job isto change only student contact information, may take advantage of unduedatabase privileges to modify grades of the Students.

2. **Privilege Abuse:** WhenUsers misuseappropriate database privileges granted to them for unauthorized purposes, then it leads to the abuse of Privilege.

   For example: Consider an internal healthcare application which allows viewing of individual patient records via a custom Web interface. The Web application normally limits users to view an individual patient's healthcare history; multiple patient records cannot be viewed simultaneously and Soft copies of the records are not allowed to be saved by the system. However, a malicious user might be able to bypass these restrictions by connecting to database using an alternative client such as MS-Excel. Using Excel and their authentic login identifications, the user could retrieve and save all patient records to their laptop. Once patient records reach a client machine, the data then becomes vulnerable to a wide variety of possible breach situations.

3. **SQL Injection:** A successful SQL injection attack can give someone limitlessand clear access to an entire database. SQL injection involves inserting (Or "injecting") illegal or malicious database statements into a at risk SQL data channel such as a stored procedure or Web application. If these injected statements are executed by the database, critical data stored in the database can be look at, copied, and changed.

4. **Malware:** Advanced attacks such as phishing emails and malware penetrate organizations database and steal sensitive data. An authorized user being unaware of infected device by malware becomes a channel for these groups to access your networks and sensitive data. In corporate office, if the malware is able to get installed on one computer in a network, then it can possibly spread to the other computers and bring down the entire network.

5. **Denial of Service:** Denial of Service (DoS) is an attack which denies access to network applications or data by intended users. There are many techniques that can allowDoSconditions; the most common technique used in database environments is to overload server resources such as memory and CPU by flooding the network with database queries that ultimately cause the server to crash. Extortion scams are often the motivation behind DoS attacks in which a remote attacker will continually crash servers until the victim meets their demands. Thus, DoS represents a serious threat for many organizations.

4.4 INTERNATIONAL OR MALICIOUS THREATS

There are several viruses or malicious code, some of which are explained here.

- **Viruses:** These have the capability to duplicate and multiply into different files. A large number of viruses also distribute or contain different types of viruses, such as file infecting viruses and script viruses.
- **Worms:** These are specially planned to spread from one computer to another computer.
- **Trojan Horse:** It appears to be benign in the beginning, but then infects unexpectedly.
- **Bots:** These can be secretly installed on computers and they react to external commands which are sent by the intruder.

Hacking

Hacking, cracking and cyber vandalism is also done by unauthorized persons. These terms are explained as follows:

- **Hackers:** These are persons who plan to expand unauthorized access to computer systems.
- **Crackers:** They hack computer systems with illegal intention.
- **Cyber Vandalism:** It is deliberate trouble making, spoiling or destroying a Website.

Computer Virus

A computer virus is a set of executable code that attaches itself to other programs to replicate itself without the awareness of a system user. These computer viruses can damage the system of a computer. The various types of computer viruses are boot sector virus, parasitic virus, multi partite virus, companion virus, link virus and macro virus. Every virus first occupies disk space in the main memory and then effects CPU processing time. Viruses are frequently transmitted through e-mail attachments, peer- to- peer downloads, phishing (a fraudulent process to get user's credentials) and instant messages. Among these, e-mail attachments carry and spread virus fast in an address book or a random combination of address book. If these viruses are not controlled quickly, the servers can disrupt the e-mail services for all systems.

According to their location, viruses can be of two types, namely non-resident virus and resident virus. The non-resident virus looks for other sources to infect and transfer control to the host program whereas the resident virus looks for new host if the infected files are accessed by other programs.

Steps to Control and Check Viruses

The following steps are required to control viruses:

- Keep virus definitions update.
- Get a sound knowledge of antivirus program subscription.

- Restrict to open or execute the unexpected attachments.
- Turn off the preview features in the computer program for added protection.
- Turn off .vbs (Visual Basic Script) function.
- Check out sincerely the extension of a file attachment.
- Do not use pirated system and application software.
- Secure your system with latest antivirus.
- Scan your system regularly.
- Update your antivirus software.

Computer viruses hoax the whole system if the particular system is defectively infected with viruses. They generate unnecessary network traffic and can cause damage by instructing the users to delete system files.

E- Mail Viruses

E-mail viruses are spread through sending email. For example, the **Mellisa** virus was being spread in Microsoft Word document via e-mail. Anyone who downloads and opens the document can get infected with this virus. This virus contains a friendly note including person's name and other details so that users would think that it asks really for his/her personal details. After entering the required information, the virus creates 50 new mails from the recipient's device. It is the fastest spreading virus that forces many of big organizations to close their e-mail systems. The **'HELLO'** virus, that appeared on May, 2000, contains a part of code as an e-mail attachment. After clicking the attachment, it first launches the code and then sends the copy to other's address book and starts corrupting files on other's systems. It is more dangerous than Trojan horse virus distributed via e-mail.

Macro Viruses

A **Macro** virus is written in a macro language. Macro viruses are spread by various applications which use and run created macros, for example, a Word processor allows macro programs to be embedded in documents so that the program can run automatically when the Word document is opened. Along with the embedded macro program the macro virus also get initialized and damages the system files. The macro virus is specific to Word 6.0, WordBasic and Excel. A **'CAP'** macro virus infects macros in the Word application attached to Word 6.0 for Windows, Word 6.0.1 for Windows Macintosh and Word for 95 Windows. A **'Concept'** virus only spreads after opening a document containing the virus. Modern antivirus software detects macro viruses as well as other types. A macro virus can be spread through e-mail attachments, disks, networks, modems and the Internet, and is very difficult to detect. Most malicious macros start automatically when a document is opened or closed. The macro virus replaces the regular commands with the same name and runs when the command is selected.

Trojan-Horse Virus

A **Trojan horse** is a virus program which internally works something else then what is specified by the user. The Trojan Horses are generally enclosed so that they appear attractive, for example a saxophone.wav file is a virus file which draws the attention of a system user interested in collecting sound samples of musical instruments. A Trojan horse is a different from other destructive virus because it does not reproduce. It steals the passwords and sends an e-mail to the hacker and then the hacker has all your description in his hands.

Signs of Virus

Any odd behaviour of computer system cannot be directly related to computer virus, because many operating systems and programs sometimes behave in strange manner. When you run antivirus program in such cases, it will not detect any virus.

The indication of virus sometimes can be seen as unusual screen displays or messages. A virus can slow down the function of the computer. Even longer disk activity or strange hardware behaviour can be caused by legitimate software, harmless 'prank' programs or by hardware faults. A virus may cause a drive to be accessed surprisingly and the drive light to go on. The basic dependable indicator of a virus infection is an alteration in the length of executable (*.com/*.exe) files, a modification in their content or a change in their file date/time in the Directory listing. Some viruses can hide the changes they have made to files, especially if they are active in RAM memory. Another important indication of a virus infection is modification and replacement of system resources.

Viruses are frequently transmitted through e-mail attachments, peer-to-peer downloads, phishing and instant messages. Phishing refers to a fraudulent process in which user's credentials are easily grabbed. Among these, e-mail attachments carry and spread virus fast in an address book or a random combination of address book. If these viruses are not controlled quickly, the servers can disrupt the e-mail services for all systems. The functions of computer viruses are as follows:

- It deletes registry attaches, system files and log files. It also destroys the OS of system units.
- Many viruses slow down the computer performance.
- Viruses decline suddenly in speed and in loss of files or data.
- They cause to display unknown and uncommon messages or even playing a tune.
- They are responsible for the loss of hard disk partition.
- They release of files normally via e-mail.

To know the working and spreading of viruses on PC or even on the Internet, you must know the lifecycle of computer viruses. The lifecycle of computer virus is as follows:

- **Coding** $\rightarrow$ In this phase, the virus program is coded and developed.
- **Releasing** $\rightarrow$ Once the code is ready, it is spread to the system and network.
- **Spreading** $\rightarrow$ It is out forwarded through a simple e-mail.
- **Quarantining** $\rightarrow$ In this phase, the virus gets quarantined. This phase often happens when it validates the signature of the virus and develops an antivirus update.

Viruses spread from machine to machine and across network in a number of ways. The viruses are always trying to trigger and execute the malicious programs that intently spread the computer system. For example, a macro virus is booted with infected disk and spread the viruses to boot sector. Then it started to share the network drive or other media that exchanges infected files across Internet by downloading files and attachments form Internet. The transmission of viruses is possible by following ways:

- If system unit is booted from infected files.
- If programs are executed with an infected programs.
- The virus infiltration includes common routes, floppy disks and e-mail attachments.
- If pirated software and shareware are used in the system files.

Viruses are malicious and smart. Many a times, users are not able to realize that their system unit has got infected with viruses. The property of viruses is that they hide themselves among regular system files or as attachments and camouflage themselves as standard files. The following steps are usually taken if the system gets infected with the viruses:

- The Internet facility and LAN utilities must be disconnected for the some time.
- The OS must be installed within the system unit if the system has not been booted properly. If the system does not recognize the hard drive it means that it is infected with virus. It is better to boot from Windows to rescue the disk. The partition table of scandisk must be recovered using scandisk for a standard Windows program.
- A back-up of all important and critical data must be taken at regular intervals by using external devices, such as CD, USB, floppy disks or even flash memory etc.
- Antivirus software must be installed in the system and should be made to rweekend or at the end of the month. A good quality antivirus disinfects infected objects, quarantines infected objects and is able to delete Trojans and worms.
- The latest updates must be taken to remove the antivirus database. The infected computer is not included to download these updates.
- Scans disinfect the mails of a client's database and ensure the maliciousograms are not reactivated if messages are sent from one address to another across the network.
- Firewall security features must be installed in the system that prevent attacks from malicious and foreign programs.
- Delete and clean the corrupted applications and files. Try to reinstall the required applications in your system but be sure that the corresponding software is not pirated.

Antivirus Software(s)

There are three different classes of antivirus packages, namely, activity monitors, authentication or change detection software and scanners. Each type has its own advantages and disadvantages or weaknesses. There are many good antivirus programs available. The most popular are Data Fellows F-Prot, EliaShim, Virusafe, ESaSS ThunderBYTE, IBM Antivirus, McAfee Scan, Microsoft Antivirus, Symantec, Norton AntiVirus and S&S Dr Solomon's AVTK. Scanning of hard drives and disks should be performed on a regular basis.

Now you know the concept of worms, spyware and viruses. You need to install and run the antivirus programs to clean the virus and provides the security for Windows. It helps in protecting Windows from crashing. The antivirus software available in the market to deal with virus-related issues are as follows:

- Symantec Antivirus that is used to check the security of foreign programs and applications.
- Windows Vista AntivirusSpyware, AntivirusNorton Antivirus that is used to catch worms, rootkits, spywares, viruses, etc.
- Avast Antivirus.
- Kaspersky Antivirus that is used for HTTP traffic-checking and for providing a security wizard.

These antiviruses are useful for those types of viruses that are downloaded from the net or from email attachments. The most popular antivirus programs are Data Fellows F-Prot, EliaShim ViruSafe, ESaSS ThunderBYTE, IBM Antivirus, McAfee Scan, Microsoft Anti Virus, Symantec Norton Antivirus and S&S Dr Solomon's AVTK. The hard disks and drives must be scanned on a daily basis. Every week, hackers and malicious programmers release their virus programs across the Internet so it is better to keep the system updated with the latest antivirus software and programs. The updated user manual and help files must be provided to the users during the installation of expensive applications and the operating system. In fact, automatic updates to the list of antivirus and multithread detection are the standard features of an antivirus program.

Spam

The term spam is derived from the 1970 "Spam" sketch of the BBC television comedy series Monty Python's Flying Circus. The sketch, set in a cafe, has a waitress reading out a menu where every item but one includes Spam canned luncheon meat. As the waitress recites the Spam-filled menu, a chorus of Viking patrons drown out all conversations with a song, repeating "Spam, Spam, Spam, Spam,… Lovely Spam! Wonderful Spam!" The excessive amount of Spam mentioned is a reference to the ubiquity of it and other imported canned meat products in the UK after World War II (a period of rationing in the UK) as the country struggled to rebuild its agricultural base. In the 1980s the term was adopted to describe certain abusive users who frequented BBSs and MUDs, who would repeat "Spam" a huge number of times to scroll other users' text off the screen. In early chat-room services like People Link and the early days of Online America (later known as America Online or AOL), they flooded the screen with quotes from the Monty Python Spam sketch. This was used as a tactic by insiders of a group that wanted to drive newcomers out of the room so the usual conversation could continue. It was also used to prevent members of rival groups from chatting—for instance, Star Wars fans often invaded Star Trek chat rooms, filling the space with blocks of text until the Star Trek fans left. It later came to be used on Usenet to mean excessive multiple posting—the repeated posting of the same message. The unwanted message would appear in many, if not all newsgroups, just as Spam appeared in all the menu items in the Monty Python sketch. The first usage of this sense was by Joel Furr, this use had also become established—to "spam" Usenet was to flood newsgroups with junk messages. The word was also attributed to the flood of "Make Money Fast" messages that clogged many newsgroups during the 1990s. In 1998, the New Oxford Dictionary of English, which had previously only defined "spam" in relation to the trademarked food product, added a second definition to its entry for "spam": "Irrelevant or inappropriate messages sent on the Internet to a large number of newsgroups or users." There was also an effort to differentiate between types of newsgroup spam. Messages that were cross posted to too many newsgroups at once, as opposed to those that were posted too frequently, were called "Velveeta" (after a cheese product), but this term did not persist.

Spamming is the use of messaging systems to send an unsolicited message (spam) to large numbers of recipients for the purpose of commercial advertising, for the purpose of non-commercial proselytizing, or for any prohibited purpose (especially the fraudulent purpose of phishing). While the most widely recognized form of spam is email spam, the term is applied to similar abuses in other media: instant messaging spam, Usenet newsgroup spam, Web search engine spam, spam in blogs, wiki spam, online classified ads spam, mobile phone messaging spam, Internet forum spam, junk fax transmissions, social spam, spam mobile apps, television advertising and file sharing spam. It is named after Spam, a luncheon meat, by way of a Monty Python sketch about a restaurant that has Spam in almost every dish and where Vikings annoyingly sing "Spam" repeatedly. Spamming remains economically viable because advertisers have no operating costs beyond the management of their mailing lists, servers, infrastructures, IP ranges, and domain names, and it is difficult to hold senders accountable for their mass mailings. The costs, such as lost productivity and fraud, are borne by the public and by Internet service providers, which have added extra capacity to cope with the volume. Spamming has been the subject of legislation in many jurisdictions. A person who creates spam is called a spammer.

Email spam, also known as Unsolicited Bulk Email (UBE), or junk mail, is the practice of sending unwanted email messages, frequently with commercial content, in large quantities. Spam in email started to become a problem when the Internet was opened for commercial use in the mid-1990s. It grew exponentially over the following years, and by 2007 it constituted about 80% to 85% of all e-mail, by a conservative estimate. Pressure to make email spam illegal has resulted in legislation in some jurisdictions, but less so in others. The efforts taken by governing bodies, security systems and email service providers seem to be helping to reduce the volume of email spam. According to "2014 Internet Security Threat

Report, Volume 19" published by Symantec Corporation, spam volume dropped to 66% of all email traffic.

Firewalls

The Internet provides a two-way flow of traffic that may be undesirable in many organizations where some information is needed exclusively for the organization or for the Intranet. The Intranet is a TCP/IP network that is modelled after the Internet that only works within the organization. In order to delineate information meant only for the benefit of the organization or its Intranet and the other open to all or meant for the Internet, some sort of security measures are needed to control the two-way flow of traffic. A measure known as firewall is used for this purpose.

A firewall is a combination of software and hardware components that controls the traffic between a secure network (usually an office LAN) and an insecure network (usually the Internet), using rules defined by the system administrator. The firewall sits at the gateway of a network or sits at a connection between the two networks, and the entire traffic between two or more networks has to traverse the firewall. The firewall stops or allows the traffic based on the security policy as defined in rules' table.

The secure trusted network is said to be 'inside' the firewall. The insecure untrusted network is said to be 'outside' the firewall. The firewall's architecture has to be such that it would permit a certain amount of traffic to get through, otherwise it would be more of a 'stonewall', preventing access to the Internet, or sending of e-mails or any other information from either side of the firewall, thus turning into a self-defeating exercise.

The fact that it allows some traffic through provides a channel that could potentially be exploited, and could carry viruses.

However, principally, the philosophy behind firewall is as follows:

- It exists to block traffic.
- It exists to permit traffic.

In brief, the basic aim of firewall is to provide only one entrance and exit to the network. There are two firewalls. One blocks the undesirable traffic, while the other allows traffic.

Network Architecture of a Firewall

The most important aspect of a firewall is that it is at the entry point of the networked system it protects. This means that the firewall is the first program that receives and handles incoming network traffic, and it is the last to handle outgoing traffic.

The logic is simple—a firewall must be positioned in a network to control all incoming and outgoing traffic. The internal network also needs to be structured and configured in such a way as to implement security policy of firewall to protect specific services running on the systems. The following are some examples of network structure to protect it from external threats using a firewall:

1. A router on dedicated connections to the Internet can be plugged into the firewall system as shown in Figure 4.1. This can also be provided

with the help of a hub for full access servers outside the firewall as shown in Figure 4.2.

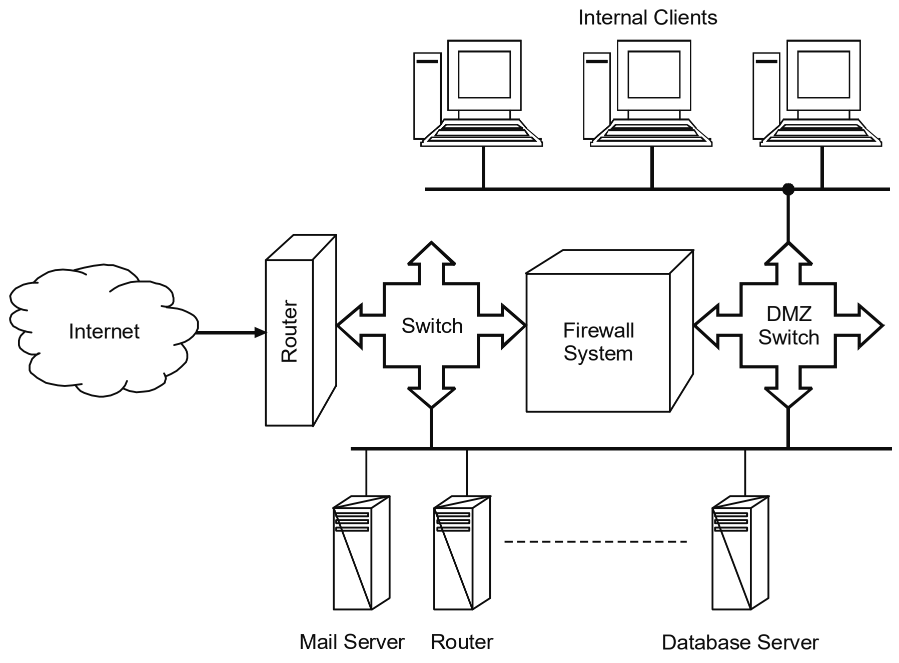


Fig. 4.1 *Router Connected to Firewalls on a Dedicated Connection*

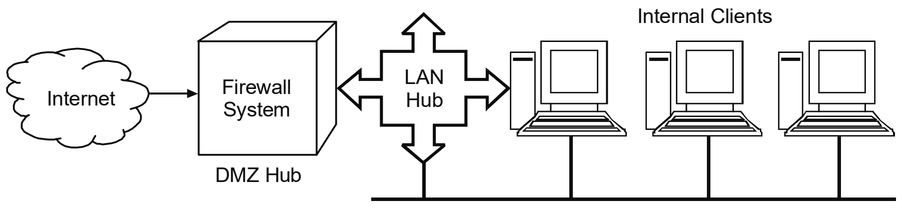


Fig. 4.2 *LAN Hub Connected to Firewalls*

2. The router can be configured with some filtering rules. However, this router may be owned by (Internet Service Provider) ISP, therefore, ISP may be asked to put all desired control.
3. In a dial-up service like an ISDN (Integrated Services Digital Network) line, a third network card is used to provide a filtered DMZ. This gives full control over the Internet services and still separates them from the regular network.
4. A proxy server can be used to monitor the traffic on the network and allow the users a limited number of services or some unwanted services may be blocked. This can be integrated with the firewall as shown in Figure 5.3

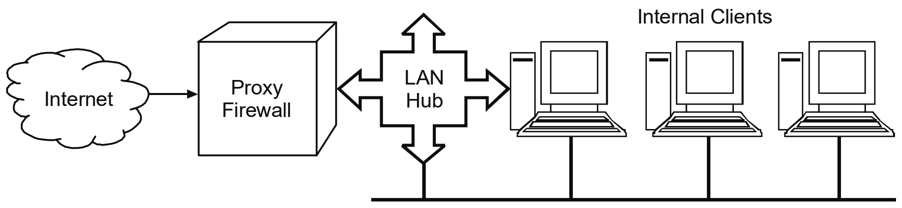


Fig. 4.3 *Proxy Server Connected with Firewall*

5. A proxy server on an organization's LAN connected with the firewall should have rules to only allow the proxy server to connect to the Internet for the services it is providing. This way the users can get to the Internet only through the proxy as shown in Figure 4.4.

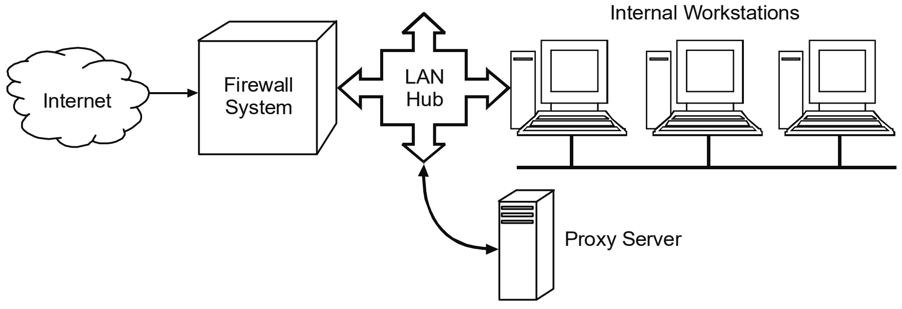


Fig. 4.4 *LAN Connected to the Internet via Proxy*

Data Encryption

Encryption hides your data from curious eyes. This is a method of encoding data to prevent unauthorized persons from viewing or modifying it. The main features of data encryption are as follows:

- Prevents unwanted access to documents and e-mail messages.
- Even the strongest levels of encryption are very difficult to break.

Processes and Types of Encryption

The process of data encryption consists of certain steps. The data passes through a mathematical formula called an algorithm, which converts it into encrypted data called ciphertext. These algorithms create a key and then encapsulate the message with this key.

There are two types of encryptions: asymmetric and symmetric.

Asymmetric Encryption

In public key (asymmetric) encryption, two mathematically-related keys are used – one to encrypt the message and the other to decrypt it. These two keys combine to form a key pair. Asymmetric encryption provides both data encryption and validation of the communicating parties' identities and is considered more secure than symmetric encryption, but is computationally slower.

A public key encryption scheme has following six major parts:

(i) Plaintext: This is the text message to which an algorithm is applied.

(ii) Encryption Algorithm: It performs mathematical operations to conduct substitutions and transformations to the plaintext.

(iii) Public and Private Keys: These are a pair of keys where one is used for encryption and the other for decryption.

(iv) Ciphertext: This is the encrypted or scrambled message produced by applying the algorithm to the plaintext message using keys.

(v) Decryption Algorithm: This algorithm generates the ciphertext and the matching key to produce the plaintext.

Encryption Process

The asymmetric data encryption process has the following steps:

- The process of encryption begins by converting the text to a pre-hash code. This code is generated using a mathematical formula.
- This pre-hash code is encrypted by the software using the sender's private key.
- The private key is generated using the algorithm used by the software.
- The encrypted pre-hash code and the message are encrypted again using the sender's private key.
- The next step is for the sender of the message to retrieve the public key of the person for whom this information is intended.
- The sender encrypts the secret key with the recipient's public key, so that only the recipient can decrypt it with his/her private key, thus concluding the encryption process.

Decryption Process

The asymmetric data decryption process has the following steps:

- The recipient uses his/her private key to decrypt the secret key.
- The recipient uses his/her private key along with the secret key to decipher the encrypted pre-hash code and the encrypted message.
- The recipient then retrieves the sender's public key. This public key is used to decrypt the pre-hash code and to verify the sender's identity.
- The recipient generates a post-hash code from the message. If the post-hash code equals the pre-hash code, then this verifies that the message has not been changed enroute.

Symmetric Encryption

Private key encryption (symmetric)—also known as conventional or single-key encryption —is founded on a secret key shared by two communicating parties. It requires all parties that are communicating to share a common key. The secret key is used by the sending party to convert simple text to encrypted text, i.e., text that is enciphered using the secret key as the security component of the mathematical process. The receiving party then proceeds to decipher the encrypted material, using the same secret key that it shares. Examples of symmetric encryption systems would include the RSA RC4 algorithm (that furnishes the basis for Microsoft Point-to-Point Encryption (MPPE), Data Encryption Standard (DES), International Data Encryption Algorithm (IDEA), and the procedure now put forward by the US Government called 'Skipjack' Encryption Technology already utilized in the clipper chip.

An encryption scheme has five major parts. These are as follows:

(i) Plaintext: This is the text message to which an algorithm is applied.

(ii) Encryption Algorithm: It performs mathematical operations to conduct substitutions and transformations to the plaintext.

(iii) Secret Key: This is the input for the algorithm as the key dictates the encrypted outcome.

(iv) Ciphertext: This is the encrypted or scrambled message produced by applying the algorithm to the plaintext message using the secret key.

(v) Decryption Algorithm: This is the encryption algorithm in reverse. It uses the ciphertext and the secret key to derive the plaintext message.

When using this form of encryption, it is essential that the sender and the receiver have a way to exchange secret keys in a secure manner. If someone knows the secret key and can figure out the algorithm, communications will be insecure. There is also the need for a strong encryption algorithm. What this means is that if someone were to have a ciphertext and a corresponding plaintext message, they would be unable to determine the encryption algorithm. There are two methods of attacking conventional encryption— brute force and cryptanalysis. Brute force is just as it sounds; using a method (computer) to find all possible combinations and eventually determining the plaintext message. Cryptanalysis is a form of attack that strikes the characteristics of the algorithm to deduce a specific plaintext or the key used. One would then be able to figure out the plaintext for all past and future messages that continue to use this compromised setup.

Check Your Progress

1. What is meant by integrity of a database?
2. What is database security?
3. Differentiate between discretionary security mechanisms and mandatory security mechanisms.
4. What is Trojan-horse virus?

4.5 OBJECTS AND SUBJECTS

A database object is any defined object in a database that is used to store or reference data. Anything which we make from create command is known as Database Object. It can be used to hold and manipulate the data. Some of the examples of database objects are: view, sequence, indexes, etc.

Subject

In a secure system, access to any resource is controlled by two entities. The subject of the access is the user or process that makes a request to access a resource. Access can mean reading from or writing to a resource. The object of an access is the resource a user or process wants to access. Remember that the subject and object refer to a specific access request, therefore the same resource can be used as both a subject and an object in different access requests. For example, process A may ask for data from process B. To satisfy process A's request, process B must ask for data from process C. In this example, process B is the object of the first request and the subject of the second request

4.5.1 View as a Object

A view is a basic schema object in an Oracle database, that is, an access path to the data in one or more tables. A view may be defined as a 'stored query' or a 'virtual table'. It is called virtual table because, they do not exist as independent entities in the database as do 'real' tables. A view is a specification consisting of a SELECT statement that tests Oracle what records and attributes to retrieve, When view is called, the system associates the appropriate data with it. It retrieves derived data only when they are called.

NOTES

To create a view the user should have the CREATE VIEW privilege.

Why Use Views?

There are several advantages of using views:

- Simplicity: A view masks the complexity of retrieving the data. Even if a view is defined as multiple table queries, the view still looks like a single table query. The complexities are placed in the view by the database administrators and are hidden from the user.
- Security: Views enhance the confidentiality of data. Access permissions can be granted to views quite separately from tables, and so a user may have permission to query a view without having any permission to query its underlying tables. A user has access to all of the columns and rows in a table after the user has been granted the SELECT privilege. A view can limit the availability of the table's column. The view was a SELECT statement; therefore, the SELECT and WHERE clauses can filter the expressions and rows available to the user. User is given access only the rows or columns of the tables those concern them.
- User reports: End-users may not be able to understand the relationships present among the tables. Views can be used to perform all of the complex join and filtering in the background, producing a simple view that the users can query either directly or through reporting tools.
- Data Integrity: The WITH CHECK OPTION clause is used to enforce the query conditions on any updates to the view.

There are three types of views:

- **Inline view:** Consists of SELECT statement that is embedded in the FROM clause of another SELECT statement.
- **Database view:** Performs same basic function retrieves data to a calling object.
- **Materialized view:** It has some physical properties and consists of data that is moved into.

Inline view replaces a physical table in the FROM clause. It is executed at run-time and furnishes its result set to the outer statement. The main difference between an inline view and a database view is that the inline view resides within another SELECT statement and can be called by only that statement. A database view resides in the database and can be called by many other objects. Another difference is that a database view can be used in the DML statement.

1. INLINE View

The inline view is a construct in Oracle SQL where a query is **placed in the FROM clause of the SELECT statement**, just as if the query was a table name. We have already used the inline view when we discussed table subquery. Notice the following example:

Query: List the details of the employees along with the corresponding location (city) of the RESEARCH department

```
 SELECT E.*,city
  FROM emp E, (SELECT dno, city From dept WHERE
dname='RESEARCH') D
 WHERE E.dno=D.dno;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN	CITY
E03	HARI NANDAN TUNGA	4000	D02	PROGRAMMER	01-JUL-95	CHENNAI
E06	JISHNU BANERJEE	6500	D02	SYSTEM MANAGER	19-SEP-96	CHENNAI
E09	PINAKI BOSE	5500	D02	PROGRAMMER	26-AUG-94	CHENNAI

There is another application of an inline query. If it is tried to use ROWNUM to restrict a query by a range that does not start with 1, it is found that it does not work. For example:

```
SELECT * from emp WHERE ROWNUM BETWEEN 5 AND 9 ;
```

Output:

no rows selected

The reason for this is that ROWNUM is a psuedo-column produced AFTER the query returns. Normally, it can only be used to restrict a query to return a rownumber range that starts with 1 (like *rownum* < *5*). An 'Inline View' can be used to get around this limitation.

```
SELECT rn, ecode,ename,salary
FROM (SELECT ROWNUM rn,ecode,ename,salary
FROM emp)
WHERE rn BETWEEN 5 AND 9;
```

Output:

RN	ECODE	ENAME	SALARY
5	E05	RAJIB HALDER	4000
6	E06	JISHNU BANERJEE	6500
7	E07	RANI BOSE	3000
8	E08	GOUTAM DEY	5000
9	E09	PINAKI BOSE	5500

NOTES

This SELECT statement in the FROM clause basically does a full query of the table, and then returns the values (along with the pseudo-column *ROWNUM*) to the outer query. The outer query can then operate on the results of the inner query. Since 'ROWNUM' is a pseudo-column and therefore, a reserved word, it is needed to alias that column (here it is 'rn') in the inner query in order to refer to it in the outer query.

Query: select EVERY 4th row from EMP table

```
SELECT *
 FROM (SELECT rownum rn, ecode, ename, salary
 FROM emp ) A
 WHERE MOD(A.ROWNUM,4) = 0;
```

Output:

RN	ECODE	ENAME	SALARY
4	E04	JAYANTA GANGULY	6000
8	E08	GOUTAM DEY	5000

UPDATE Through an Inline View

Through the inline view, updation is possible.

Query: Double the salary of the employees who are working in the department located at CHENNAI

```
UPDATE (SELECT ecode,emp.dno,salary,dept.dno,dname, city
FROM dept, emp WHERE emp.dno=dept.dno and city='CHENNAI')
SET salary=salary*2;
```

Output:

3 rows updated.

Let's check the updation:

SELECT * FROM emp;

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E03	HARI NANDAN TUNGA	8000	D02	PROGRAMMER	01-JUL-95
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95
E06	JISHNU BANERJEE	13000	D02	SYSTEM MANAGER	19-SEP-96
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97
E09	PINAKI BOSE	11000	D02	PROGRAMMER	26-AUG-94

9 rows selected.

WITH CHECK OPTION in Inline View

The WITH CHECK OPTION may be used to create the inline view as a constrained view and prohibit specific insert and update operations through the inline view. The WITH CHECK OPTION is used with the WHERE clause. For example, to restrict the users from modifying data for any employee in the PROJECT department, the SQL statement will be as follows:

```
UPDATE (SELECT ecode, ename, dno,salary FROM emp
WHERE dno != (SELECT dno FROM dept WHERE dname='PROJECT')
WITH CHECK OPTION) empl
SET empl.salary=7000
WHERE empl.ecode='E01';
```

Output:

0 rows updated.

Now the following SQL statement is entered:

```
UPDATE (SELECT ecode, ename, dno,salary FROM emp
WHERE dno != (SELECT dno FROM dept WHERE dname='PROJECT')
WITH CHECK OPTION) empl
SET empl.salary=7000
WHERE empl.ecode='E03';
```

Output:

1 row updated.

Here updation is successful as employee with employee code 'E03' is working in the RESEARCH department (dno is 'D02').

The WITH CHECK OPTION in an inline view also protects against data modification that would not be visible via the inline view. For example, Employee with employee code 'E03' is shifted from the RESEARCH department to the PROJECT department. To do this assignment, the following statement is entered:

```
UPDATE (SELECT ecode, ename, dno,salary FROM emp
WHERE dno != (SELECT dno FROM dept WHERE dname='PROJECT')
WITH CHECK OPTION) empl
SET empl.dno=(SELECT dno FROM dept WHERE dname='PROJECT')
WHERE empl.ecode='E03';
```

Output:

```
WHERE dno != (SELECT dno FROM dept WHERE dname='PROJECT')
 *
ERROR at line 2:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

The above statement generated an error as the data would no longer be visible via the inline view.

2. DATABASE View

The CREATE VIEW command is used to add the views to the database. The syntax is as follows:

NOTES

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW <view_name>
AS SELECT statement
[WITH CHECK OPTION [CONSTRAINT <constraint_name>]]
[WITH READ ONLY];
```

The CRATE VIEW command has several available options:

View Option	Description
OR REPLACE	Replaces the existing view with the new one. If this option is omitted, Oracle will not allow to overwrite the existing view. It must first be dropped.
FORCE	This allows a view to be created even if the tables do not exist and the owner of the schema containing the view has privileges on them. Both of these conditions must be true before any SELECT, INSERT, UPDATE or DELETE statements can be issued against the view.
NOFORCE	Allows the view to be created only if the tables exist and the components are valid. This is the default case.
WITH CHECK OPTION	Restricts DML operations (mainly inserts and updates**)** to only the rows that are accessible to the view. The CHECK OPTION cannot guarantee if there is a subquery in the query of this view or any view on which this view is based.
CONSTRAINT	It is the name assigned to the CHECK OPTION constraint. If this identifier is omitted, Oracle automatically assigns the constraint a name of this form: **SYS_Cn :** where **n**is an integer that makes the constraint name unique within the database.
WITH READ ONLY	Ensure that no DML operations can be performed using the view.

The following command will crate a simple database view:

```
CREATE VIEW emp_view
AS
SELECT * FROM emp
WHERE dno = ‘D01’;
```

Output:

View created.

It can be queried, updated, inserted into, deleted from and joined with other tables and views. The SELECT statement can be used to list the rows just like a real table.

```
SELECT * FROM emp_view;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E01	KOUSHIK GHOSH	5000	D01	SYSTEM ANALYST	10-MAR-93
E02	JAYANTA DUTTA	3500	D01	PROGRAMMER	15-JAN-94
E07	RANI BOSE	3000	D01	PROJECT ASSISTANT	17-JUN-97
E08	GOUTAM DEY	5000	D01	PROGRAMMER	23-OCT-97

It is also possible to issue a DESCRIBE command against a view.

```
DESC emp_view;
```

Output:

NAME	NULL?	TYPE
ECODE	NOT NULL	VARCHAR2(5)
ENAME		VARCHAR2(30)
SALARY		NUMBER(7,2)
DNO		VARCHAR2(5)
DESG		VARCHAR2(30)
DT_JN		DATE

Sometimes it is needed to give names to view columns. Circumstances for naming view columns are as follows:

- One or more of the view's columns is derived from an arithmetic expression, a built-in function or a constant.

Example:

```
CREATE VIEW empl_view (deptno, total_salary, emp_count)
AS
SELECT dno, SUM (salary), COUNT (ecode)
FROM emp GROUP BY dno;
```

- The view would wind up with more than one column of the same name.

Example:

```
CREATE VIEW city_view (empname, ecity, dname, dcity)
AS
SELECT ename, emp.city, dname, dept.city
FROM emp, dept
WHERE emp.city = dept.city;
```

Here, there is a join condition in the SELECT statement and joining columns have the same name.

If a view is based on a single underlying table, INSERT or UPDATE or DELETE rows can be performed. This will actually insert, update or delete rows in the underlying tables unless a read-only view is explicitly created. It is to be noted that there are some restrictions to be considered about a view's defining query if the view would have to be inherently updatable.

Attempting to query a view where the underlying table has seen dropped will produce an error message. Views do not contain data. Tables contain data. As of Oracle 8i, 'materialized view' can be created that contain data but they are truly tables.

WITH CHECK OPTION in Database View

The WITH CHECK OPTION is used to create a view as a constrained view and prohibit specific insert and update operations with the view. For example, let us consider a view that store all employee information getting salary more than ₹ 5000. What happens if we update salary of one of those employees through the view, changing his salary to ₹ 4000. Oracle allows this updation. Placing the WITH CHECK OPTION clause in the view will prevent this updation. The WITH CHECK OPTION clause tells Oracle to reject any attempt to modify a view in a way that makes one or more of its row ineligible for the view.

A view that includes all employees whose salary is more than ₹ 5000 is as follows:

```
CREATE VIEW sal_view
AS
SELECT ecode, ename, salary
FROM emp
WHERE salary>5000
WITH CHECK OPTION;
```

Output:

View created.

```
SELECT * FROM sal_view;
```

Output:

ECODE	ENAME	SALARY
E04	JAYANTA GANGULY	6000
E06	JISHNU BANERJEE	6500
E09	PINAKI BOSE	5500

The above view can also be created using the following statement:

```
CREATE VIEW sal_view
    AS
    SELECT ecode, ename, salary
    FROM emp
    WHERE salary>5000
    WITH CHECK OPTION CONSTRAINT sal_view_cnst;
```

Here the CHECK OPTION creates the view with the constraint (named sal_view_cnst) that INSERT and UPDATE statements issued against the view cannot result in rows that the view cannot select.

Now we want to update the salary of one of the employees (whose salary is ₹ 6500 and ecode is 'E04') through the following command:

```
UPDATE sal_view
SET salary = 4000
WHERE ecode = 'E04';
```

The following is the output of the above command:

```
UPDATE sal_view
 *
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

This is an unsuccessful attempt to update the view. The WITH CHECK OPTION clause prevented the modification. If we rewrite the above UPDATE command as follows, then there is no problem:

```
UPDATE sal_view
SET salary = 7500
WHERE ecode = 'E04';
```

Output:

1 row updated.

```
SELECT * FROM sal_view;
```

Output:

ECODE	ENAME	SALARY
E04	JAYANTA GANGULY	7500
E06	JISHNU BANERJEE	6500
E09	PINAKI BOSE	5500

Obviously, this updating will affect the underlying table also.

```
SELECT ecode,ename,salary FROM emp;
```

Output:

ECODE	ENAME	SALARY
E01	KOUSHIK GHOSH	5000
E02	JAYANTA DUTTA	3500
E03	HARI NANDAN TUNGA	4000
E04	JAYANTA GANGULY	7500
E05	RAJIB HALDER	4000
E06	JISHNU BANERJEE	6500
E07	RANI BOSE	3000
E08	GOUTAM DEY	5000
E09	PINAKI BOSE	5500

9 rows selected.

It is to be noted that only salary column is not updatable. The following query would work as it did not refer the non-constrained column:

```
UPDATE sal_view
    SET ename = 'TANMOY PAL'
    WHERE ecode = 'E04';
```

Output:

1 row updated.

Let's see the updation.

```
SELECT ecode,ename,salary FROM emp;
```

Output:

ECODE	ENAME	SALARY
E01	KOUSHIK GHOSH	5000
E02	JAYANTA DUTTA	3500
E03	HARI NANDAN TUNGA	4000
E04	TANMOY PAL	7500
E05	RAJIB HALDER	4000
E06	JISHNU BANERJEE	6500
E07	RANI BOSE	3000
E08	GOUTAM DEY	5000
E09	PINAKI BOSE	5500

9 rows selected.

WITH READ ONLY Option in Database View

The WITH READ ONLY option prevents any DML operation from occurring. The following is a CREATE VIEW command using the WITH READ ONLY option:

```
    CREATE VIEW employee_view
    AS
    SELECT ecode, ename, salary, dno
    FROM emp
    WHERE dno = 'D01'
    WITH READ ONLY;
```

Output:

View created.

Now we want to update salary of the employee 'E01' with ₹ 7000 using the following UPDATE statement:

```
    UPDATE employee_view
    SET salary = 7000
    WHERE ecode = 'E01';
```

Output:

```
SET salary = 7000
  *
ERROR at line 2:
ORA-01733: virtual column not allowed here
```

The UPDATE command fails due to the READ ONLY clause.

Note:

Oracle returns an error when the following statement is tried to execute:

```
CREATE VIEW prgmr_view AS
SELECT *
FROM emp
WHERE desg = 'PROGRAMMER'
WITH READ ONLY
WITH CHECK OPTION;
```

Output:

```
WITH CHECK OPTION
 *
ERROR at line 6:
ORA-00933: SQL command not properly ended
```

A read-only view cannot be created with a constraint using WITH CHECK OPTION because WITH CHECK OPTION is used for updatable views only.

Join Views

Views can also specify more than one base table or view in the FROM clause. These are called **join views**. The following statement creates the empl_view view that joins data from the emp and dept tables:

```
CREATE VIEW emp_view
    AS
    SELECT ecode, ename, dname, dt_jn, salary
    FROM emp, dept
    WHERE emp.dno = dept.dno;
```

An **updatable join view** (also referred to as a **modifiable join view**) is a view that contains more than one table in the top-level FROM clause of the SELECT statement and is not restricted by the WITH READ ONLY clause. It is a join view where UPDATE, INSERT and DELETE operations are allowed. Though there are some restrictions and conditions, which can affect whether a join view is updatable. These restrictions are based on the concept of **key-preserved table and** the view does *not* contain any of the following:

- DISTINCT operator

- Aggregate functions: AVG, COUNT, GLB, MAX, MIN, STDDEV, SUM, or VARIANCE
- Set operations: UNION, UNION ALL, INTERSECT, MINUS
- GROUP BY or HAVING clauses
- START WITH or CONNECT BY clauses
- ROWNUM pseudocolumn

A base table of a view is considered a *key-preserved* table if every primary key or unique key value in the base table is also unique in the result set of the join view; in other words, if the entity integrity of the base table is preserved by the join view, so, a key-preserved table has its keys preserved through a join.

The key preserving property of a table does not depend on the actual data in the table. It is, rather, a property of its schema. For example, if in the emp table there was at most one employee in each department, then dno would be unique in the result of a join of emp and dept, but dept would still not be a key preserved table.

The following are the appropriately constrained table definitions for emp and dept:

```
CREATE TABLE dept (
    deptno    VARCHAR2(5) PRIMARY KEY,
    dname     VARCHAR2(30),
    city      VARCHAR2(25));
CREATE TABLE emp (
    ecode     VARCHAR(5) PRIMARY KEY,
    ename     VARCHAR2(30),
    desg      VARCHAR2(30),
    salary    NUMBER(7,2),
    dno       VARCHAR2(5),
    dt_jn     DATE
    CONSTRAINT fk_emp FOREIGN KEY (dno) REFERENCES DEPT(dno));
```

Now the following join view is created:

In pre-Oracle 9i

```
CREATE VIEW emp_dept_view AS
SELECT  emp.ecode,  emp.ename,  emp.dno,  emp.salary,
dept.dname, dept.city
 FROM emp, dept
 WHERE emp.dno = dept.dno
 AND dept.city='KOLKATA';
OR
```

In Oracle 9i

```
CREATE VIEW emp_dept_view AS
SELECT  emp.ecode,  emp.ename,  emp.dno,  emp.salary,
dept.dname, dept.city
 FROM emp JOIN dept
 ON emp.dno = dept.dno
 AND dept.city='KOLKATA';
```

Output:

View created.

Now all rows of the emp_dept_view are listed below.

```
SELECT * FROM emp_dept_view;
```

Output:

ECODE	ENAME	DNO	SALARY	DNAME	CITY
E01	KOUSHIK GHOSH	D01	5000	PROJECT	KOLKATA
E02	JAYANTA DUTTA	D01	3500	PROJECT	KOLKATA
E04	JAYANTA GANGULY	D03	7500	PERSONNEL	KOLKATA
E05	RAJIB HALDER	D03	4000	PERSONNEL	KOLKATA
E07	RANI BOSE	D01	3000	PROJECT	KOLKATA
E08	GOUTAM DEY	D01	5000	PROJECT	KOLKATA

6 rows selected.

Notice that in this view, emp is a key preserved table, because empno is a key of the emp table, and also a key of the result of the join. dept is *not* a key preserved table, because although dno is a key of the dept table, it is not a key of the join.

NOTES

DML Statements and Join Views

The **general rule** is that any UPDATE, DELETE or INSERT statement on a join view can modify only one underlying base table.

The rules for updatable join views are as follows:

- **UPDATE Rule:** The rows can be updated in a join view if an UPDATE statement targets only columns from a single key preserved base table of the view; additionally, if the join view is a constrained view with 'WITH CHECK OPTION', an UPDATE statement cannot update columns used in a join condition, or columns referenced more than once in the view.
- **DELETE Rule:** The rows can be deleted from a join view if the view is based on only one key preserved base table. If the view is defined with the WITH CHECK OPTION clause and the key preserved table is repeated, then the rows cannot be deleted from the view.
- **INSERT Rule:** An INSERT statement must not explicitly or implicitly refer to the columns of a **non-key preserved table**. If the join view is defined with the WITH CHECK OPTION clause, INSERT statements are not permitted.

Let's apply these rules to the view emp_dept_view

UPDATE on join view

The following example shows an UPDATE statement that successfully modifies the emp_dept_view view:

```
UPDATE emp_dept_view
 SET salary = salary * 1.10
 WHERE dno = 'D01';
```

Output:

4 rows updated.

Now all rows of the emp_dept_view are listed below.

NOTES

```
SELECT * FROM emp_dept_view;
```

Output:

ECODE	ENAME	DNO	SALARY	DNAME	CITY
E01	KOUSHIK GHOSH	D01	5500	PROJECT	KOLKATA
E02	JAYANTA DUTTA	D01	3850	PROJECT	KOLKATA
E04	JAYANTA GANGULY	D03	7500	PERSONNEL	KOLKATA
E05	RAJIB HALDER	D03	4000	PERSONNEL	KOLKATA
E07	RANI BOSE	D01	3300	PROJECT	KOLKATA
E08	GOUTAM DEY	D01	5500	PROJECT	KOLKATA

6 rows selected.

The following UPDATE statement would be disallowed on the emp_dept_view view:

```
UPDATE emp_dept_view
 SET city = 'CHENNAI'
 WHERE ename = 'KOUSHIK GHOSH';
```

Output:

```
SET city = 'CHENNAI'
 *
ERROR at line 2:
ORA-01779: cannot modify a column which maps to a non-key preserved table
```

This statement fails with an error. Because it attempts to modify the base dept table, and the dept table is not key preserved in the emp_dept_view view.

In general, all updatable columns of a join view must map to columns of a key preserved table. If the view is defined using the WITH CHECK OPTION clause, then all join columns and all columns taken from tables that are referenced more than once in the view are not modifiable.

So, for example, if the emp_dept_view view was defined using WITH CHECK OPTION, the following UPDATE statement would fail:

```
CREATE VIEW emp_dept_view_chk AS
SELECT emp.ecode, emp.ename, emp.dno, emp.salary,
dept.dname, dept.city
 FROM emp, dept
 WHERE emp.dno = dept.dno
 AND dept.city='KOLKATA'
WITH CHECK OPTION;
```

Output:

View created.

```
UPDATE emp_dept_view_chk
 SET dno = 'D03'
 WHERE ename = 'RANI BOSE';
```

Output:

```
SET dno = 'D03'
 *
ERROR at line 2:
ORA-01733: virtual column not allowed here
```

The above statement fails because it is trying to update a join column.

DELETE on join view

We can delete from a join view provided there is *one and only one* key preserved table in the join.

The following DELETE statement works on the emp_dept_view view:

```
DELETE FROM emp_dept_view
WHERE ename = 'KOUSHIK GHOSH';
```

Output:

1 row deleted.

This DELETE statement on the emp_dept_view view is legal because it can be translated to a DELETE operation on the base emp table, and because the emp table is the only key preserved table in the join.

Obviously, this deletion will affect the underlying table (emp)also.

```
SELECT ecode,ename,salary FROM emp;
```

Output:

ECODE	ENAME	SALARY
E02	JAYANTA DUTTA	3500
E03	HARI NANDAN TUNGA	4000
E04	JAYANTA GANGULY	7500
E05	RAJIB HALDER	4000
E06	JISHNU BANERJEE	6500
E07	RANI BOSE	3000
E08	GOUTAM DEY	5000
E09	PINAKI BOSE	5500

8 rows selected.

A DELETE operation could not be performed on the view, which is created using self-join because both the tables are key preserved tables.

Consider the following view:

```
CREATE VIEW emp_emp AS
 SELECT DISTINCT e1.ename, e1.ecode, e1.dno
 FROM emp e1, emp e2
 WHERE e1.desg = e2.desg
 AND e1.ecode<>e2.ecode;
select * from emp_emp;
```

ENAME	ECODE	DNO
GOUTAM DEY	E08	D01
HARI NANDAN TUNGA	E03	D02
JAYANTA DUTTA	E02	D01
PINAKI BOSE	E09	D02

Now we try to delete the record of GOUTAM DEY using the following command:

```
DELETE FROM emp_emp WHERE ename='GOUTAM DEY';
```

Output:

```
DELETE FROM emp_emp
 *
ERROR at line 1:
ORA-01732: data manipulation operation not legal on this view
```

No deletion can be performed on this view because the view involves a self-join of the table that is key preserved.

WITH CHECK OPTION as such does not prevent deletion of rows from a join view as follows:

```
DELETE FROM emp_dept_view_chk
WHERE ecode='E07';
```

Output:

```
1 row deleted.
```

If the key-preserved table is repeated, then rows cannot be deleted from such a view.

INSERT on join view

The following INSERT statement on the emp_dept_view view succeeds:

```
INSERT INTO emp_dept_view (ename, ecode, dno)
 VALUES ('MUKUL ANAND', 'E10', 'D03');
```

Output:

1 row created.

This statement works because only one key preserved base table is being modified (emp), and 'D03' is a valid dno in the dept table (thus satisfying the FOREIGN KEY integrity constraint on the emp table).

Obviously, this deletion will affect the underlying table (emp)also.

```
SELECT ecode,ename,salary FROM emp;
```

Output:

```
ECODE  ENAME                SALARY
E02    JAYANTA DUTTA          3500
E03    HARI NANDAN TUNGA      4000
E04    JAYANTA GANGULY        7500
E05    RAJIB HALDER           4000
E06    JISHNU BANERJEE        6500
E07    RANI BOSE              3000
E08    GOUTAM DEY             5000
E09    PINAKI BOSE            5500
E10    MUKUL ANAND
```

8 rows selected.

Now the following INSERT statement is entered:

```
INSERT INTO emp_dept_view (ename, ecode, dno)
 VALUES ('SUNIL GHOSH', 'E11', 'D04');
```

Output:

```
INSERT INTO emp_dept_view (ename, ecode, dno)
*
ERROR at line 1:
ORA-02291: integrity constraint (MAN.SYS_C002723) violated - parent key not found
```

The above INSERT command fails as the FOREIGN KEY integrity constraint on the emp table is violated because there is no department with dno 'D04'.

The following INSERT statement would also fail:

```
INSERT INTO emp_dept_view (ecode, ename, city)
 VALUES ('E11', 'SALIL MITRA', 'KOLKATA');
```

Output:

```
INSERT INTO emp_dept_view (ecode, ename, city)
 *
ERROR at line 1:
ORA-01776: cannot modify more than one base table through a join view
```

An INSERT cannot implicitly or explicitly refer to columns of a non-key preserved table.

If the join view is defined using the WITH CHECK OPTION clause, then the INSERT command cannot be performed, even if attempt is made to insert into key preserved table only. For example,

```
INSERT INTO emp_dept_view_chk (ecode,ename,dno,salary)
VALUES('E11','SANJOY SAHA','D01',9000);
```

Output:

```
(ecode,ename,dno,salary)
 *
ERROR at line 2:
ORA-01733: virtual column not allowed here
```

HOW to know updatable columns in join view?

USER_UPDATABLE_COLUMNS, data dictionary view, is used to show all columns in all tables and views in the user's schema that are modifiable. The following query shows the updatable columns in view emp_dept_view:

```
SELECT COLUMN_NAME, UPDATABLE
FROM USER_UPDATABLE_COLUMNS
WHERE TABLE_NAME = 'EMP_DEPT_VIEW';
```

Output:

COLUMN_NAME	UPD
ECODE	YES
ENAME	YES
DNO	YES
SALARY	YES
DNAME	NO
CITY	NO

6 rows selected.

Creating Database Views with Errors

If there are no syntax errors in a CREATE VIEW statement, Oracle can create the view even if the defining query of the view cannot be executed due to a non-existent table or an invalid column of an existing table, or when the view owner does not have the required privileges.

When it is tried to create such a view, Oracle returns a message indicating that the view was created with errors.

To create a view with errors, we must include the FORCE option of the CREATE VIEW statement.

```
        CREATE FORCE VIEW AS ...;
```

When the conditions later change, the view can be recompiled and be made executable.

Dropping Database Views

In common with every other SQL CREATE... command, CREATE VIEW has a corresponding DROP... command. The syntax is as follows:

```
    DROP VIEW <view_name>;
```

For example:

DROP VIEW emp_dept_view;

Output:

View dropped.

The only thing to remember when using the DROP VIEW command is that all other views that reference that view are now invalid.

Replacing Database Views

Views can be replaced in either of the following ways:

- Drop and recreate the view.
- The view can be redefined with a CREATE VIEW statement that contains the OR REPLACE option. The OR REPLACE option replaces the current definition of a view and preserves the current security authorization. To replace a view, we must have all the privileges required to drop and create a view.

As an illustration, a view empl_v is created using the following CREATE VIEW command:

```
CREATE VIEW empl_v
    AS
    SELECT * FROM emp
    WHERE dno = 'D01'
WITH CHECK OPTION;
```

Now it is needed to redefine the empl_v view to change the department number specified in the WHERE clause. Let's try it.

```
CREATE VIEW empl_v
    AS
    SELECT * FROM emp
    WHERE dno = 'D03'
WITH CHECK OPTION;
```

Output:

```
CREATE VIEW empl_v
 *
ERROR at line 1:
ORA-00955: name is already used by an existing object
```

NOTES

To do this, the first option may be applied, i.e. drop and recreate the view. Using the second option, the current version of the empl_v view can be replaced with the following statement:

```
CREATE OR REPLACE VIEW empl_v
    AS
    SELECT * FROM emp
    WHERE dno = 'D01'
WITH CHECK OPTION;
```

Output:

```
View created
```

Now let's try to redefine the view for changing the department number specified in the WHERE clause.

```
CREATE OR REPLACE VIEW empl_v
    AS
    SELECT * FROM emp
    WHERE dno = 'D03'
WITH CHECK OPTION;
```

Output:

```
View created.
```

To see the changes, the following command is entered:

```
SELECT * FROM empl_v;
```

Output:

ECODE	ENAME	SALARY	DNO	DESG	DT_JN
E04	JAYANTA GANGULY	6000	D03	ACCOUNTANT	12-SEP-96
E05	RAJIB HALDER	4000	D03	CLERK	07-OCT-95

Altering Database Views

Views are based on the SELECT statement and base tables queried by the SELECT statement. When base tables are changed, the view must be recompiled. For using the ALTER VIEW statement, the view has to be in user's schema, or the user should have the ALTER ANY TABLE system privilege.

An example of when a view needs to be recompiled is rendered. For example, create a table and a view based on that table. Alter the table by adding a column. Check the view and recompile or recreate as necessary.

```
CREATE TABLE TEST (COL1 NUMBER );
```

Output:

Table created.

```
CREATE VIEW TEST_VIEW AS
SELECT *
FROM TEST;
```

Output:

View created.

```
ALTER TABLE TEST
ADD ( COL2 VARCHAR( 10 ) );
```

Output:

Table altered.

```
DESC TEST_VIEW
```

Output:

ERROR:

ORA-24372: invalid object for describe

```
SELECT * FROM TEST_VIEW;
```

Output:

no rows selected

```
DESC TEST_VIEW
```

Output:

NAME	NULL?	TYPE
COL1		NUMBER

Changing the base tables of a view requires that the view be recompiled. A view can be recompiled by accessing the view using SQL or by issuing an ALTER VIEW COMPILE statement.

```
ALTER VIEW TEST_VIEW COMPILE;
```

Output:

View altered.

To include the new column, the view will need to be recreated.

DML and Database Views

Updating the data through a view may or may not be possible. The five conditions that view must meet in order to allow updates are as follows:

- The view must not have a DISTINCT clause.
- The view must not contain the pseudocolumn ROWNUM
- All columns must be real columns, no expression, calculated columns or group function.

- The WHERE clause must not contain a subquery.
- There must be no GROUP BY or HAVING clauses.

The DELETE command cannot be used against a complex view if the view has any of the following:

- Contains a group function.
- Contains a GROUP BY clause.
- Contains the DISTINCT keyword.
- Contains the pseudo-column ROWNUM.

A view cannot be used to add records to the database if the view contains any of the above features. The view also cannot be used to insert records if the base tables have the following property:

Contains NOT NULL constrained columns that do not exist in the view.

Drawbacks of Database View

In spite of several advantages, database views have the following disadvantages:

- *Performance*: If a view is complex, then queries may take a long time. Because a view may look like a simple table but underneath the DBMS is usually still running multi-table queries. Because view queries that involve sorting, grouping, etc. can lead to a high performance overhead, it might be better to write some reports with a procedural component that fills up a temporary table and then does a number of queries from it.
- *Update restriction*: The following list details the general restrictions to consider about a view's defining query if it is wanted the view to be inherently updatable. An inherently updatable view's defining query cannot include the following:
  - A set operator (UNION, UNION ALL, INTERSECT, or MINUS)
  - The DISTINCT operator
  - A group (aggregate) function such as AVG, COUNT, MAX, MIN, etc.
  - A GROUP BY, ORDER BY, CONNECT BY, or START WITH clause
  - A reference to a collection expression in a SELECT list
  - A subquery in a SELECT list
  - A JOIN query

Additionally, when the SELECT list in the defining query of a view contains virtual columns (expressions or references to pseudocolumns), we can only update base table rows using the view when an UPDATE statement does not refer to any of the view's virtual columns.

3. Materialized View

In Oracle 8, the materialized view, was introduced. A materialized view is like a view in that it represents data that is contained in other database tables and views; yet it is unlike a view in that it contains the actual data. A materialized view is like an index in that the data it contains is derived from the data in database tables and views; yet unlike an index in that its data must be explicitly refreshed.

Like a snapshot, a materialized view creates a local table to store the data and a view that accesses that data and an administrator can specify when the data is to be refreshed; but it is unlike a snapshot in that a materialized view should include either summary data or data from many different joined tables. Depending on the complexity of the materialized view, ORACLE may also create an index on the materialized view's local table.

At this point, it may be difficult to understand the importance of a materialized view. For a large database, a materialized view may offer several performance advantages.

4.6 GRANULARITY

Data granularity transforms the data into structure. Loading of data involves transfer of data from source system, data preparation area or tables to the database. These operations create heavy load on the system and you need to complete the loading when systems are free. Basically, granularity means data precision. It includes space granularity that gives geographical clusters for data. The maximum granularity, also called atomic data gives perfect data precision.

For this, you need to ensure that each column in a table that is to be normalized is atomic. For example, the **student** table can be divided into two columns, the first name and the last name to make the table atomic.

Table 4.1 *The Student Table*

Student_id	First_name	Last_name
S001	Natasha	Vij
S002	Richa	Gulati
S003	Anu	Lamba

Table 4.2 *Types of Data Granularity*

Types of Data Granularity	Function
Multiplication Granularity	This type of data granularity facilitates intervals if various indexed fields are updated.
Division Granularity	This type of data granularity prefers a choice between groups within same database domain. For example, tuple is bind to a protection attribute meanwhile other choices bind a protection level to the attributes of the fields.

The algorithms used for data granularity have the following characteristics:

- **Scalability:** The algorithm used for data granularity is enough scalable so that these can be used for real-world data such as employee and student details.

- **Work with real-world data problems:** Data granularity algorithms should perform correctly with problems related to real-world. Examples of such problem include missing attribute values related to real world data.
- **Update:** This algorithm has the ability to update data stored in databases.
- **Ease of use:** Data granularity algorithms should be easy to understand and learn so that they can be used by users having less knowledge.

4.6.1 Access Types

There are four types of access that are applicable to a database.

1. **Discretionary Access Control (DAC):** In DAC model, the data owner allows access to the user based on some user-specified protocols. They are defined by user identification during authentication, for example user-id and password. The aim is to grant and revoke privileges to users.
2. **Mandatory Access Control (MAC):** This access model restricts the ability of owner to grant or deny access to a database. Here, the criteria is defined by the administrator. It enables multilevel security by classifying the users and data into security classes.
3. **Role Based Access Control (RBAC):** It is a most common method today. This mechanism allows the user to access the database based on their role and implements the security principles accordingly. Hence, it allows to access that much information that is necessary for their role.
4. **Attribute Based Access Control (ABAC):** In this model, every user is assigned with a series of attributes. The decision to access a resource is based on assessment based on the attributes like position, location and time of day.

4.7 DATABASE OPERATING SYSTEMS

DBMS, which is the short form for database management systems, is a software system that allows users to not only define and create a database, but also maintain it and control its access. It can be called a collection of interrelated data (usually called database) and a collection or set of programs that helps in accessing, updating and managing that data (which forms part of a database management system).

Database

Database means a place where data can be stored in a structured manner. It is a shared collection or batch of data that is logically related, along with their descriptions designed to meet the information requirements of an organization. A database is a complex data structure. It is stored in a system of mutually dependent files containing the following information:

- The set of data available to the user, the so-called ‘end-user data’. This is the real data, which can be read, connected and modified by the user.

- The so-called 'metadata' (the data describing the end-user data). Here, the properties (e.g., their type) and the relative relations of the end-user data are described.

The primary benefit of using DBMS is to organize data logically. DBMS provides simple mechanisms for processing huge volumes of data because it is optimized for operations of this type. The two basic operations performed by any DBMS are as follows:

(i) Management of data in the database

(ii) Management of users associated with the database

(*i*) ***Management of data:*** This means specifying how data will be stored, structured and accessed in the database. This includes the following:

- *Defining:* Specifying data types and structures and constraints for data to be stored
- *Constructing:* Storing data in a storage medium
- *Manipulating:* Querying, updating and generating reports
- *Sharing:* Allowing multiple users and programs to access data simultaneously

Further, the DBMS must offer safety and security of the information stored, in case unauthorised access is attempted or the system crashes. If data is required to be shared among many users, the system must ensure that possible anomalous results are avoided.

(*ii*) ***Management of database users:*** This means managing the users in such a way that they are able to perform any desired operation on the database. DBMS also ensures that a user cannot perform any operation for which he is not authorised.

In short, DBMS is a collection of programs performing all necessary actions associated with a database. There are many DBMS available in the market, such as Access, dBase, FileMaker Pro, FoxPro, ORACLE, DB2, Ingress, Informix and Sybase.

A database application is a program or a set of programs that interacts with the database at some point in its execution. It is used for performing certain operations on data stored in the database. These operations include inserting data into a database, extracting data from a database based on certain conditions, updating data in a database and producing data as output on any device such as a screen, a disk or a printer.

A database system is a collection of application programs that interacts with the database along with DBMS (and sometimes the users who use the system). Database systems are designed in a manner that facilitates the management of huge bodies of information.

A database clearly separates the physical storage of data from its use by an application program to attain program–data interdependence. While using a database system, the user or programmer is unaware of the details of how the data is stored. Data can be changed or updated without making any difference to the other components of the system.

4.7.1 Features of Database OS (Operating System)

The database approach involves storing of data in individual files which can be shared by different programs rather than relating it to specific individual programs. Usually, the data is represented in a way which is normal for the applications that require using it, not in a favourable format that is beneficial to the systems storing it. There should be no need for duplicating it and everyone should be able to access it. Another advantage of database is the ease in the application of standards, and the central maintenance of the security and reliability by entering data just once.

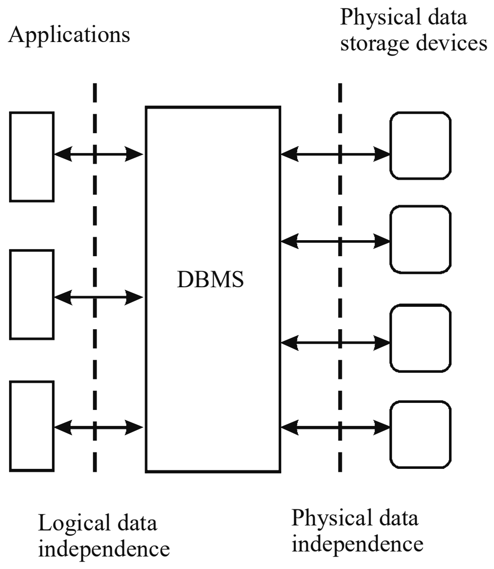
There are two levels of data independence in a database—logical data independence and physical data independence. Logical data independence refers to the ability to modify the logical structuring of the data without modifying the applications. In physical data independence, modifications can be done to the layout of the data without involving application programs.

DBMS is an application software controlling the arrangement of data and the ways to access it in a database. There is no direct interaction between the application programs and the database and coordinates with the DBMS for retrieving, adding or deleting data in the database.

The following are the basic objectives of DBMS:

- Data independence in which the executable code is seperated from the data for the purpose of modifying and maintaining them independently
- Union of data which is favourable to the applications requiring to access it
- Data integrity, reliability, preciseness and timeliness
- Protection from unauthorised access
- Reduction of redundancy to improve consistency
- Facilities regarding more than one search policy or accessing techniques through the structure so that applications are not necessarily restricted in the way where they are needed to access the data
- Centralized control, generally in the type of a database administrator
- Concurrent access by many programs
- Ad hoc query, with no requirement for specific anticipating programs

The central feature in the database philosophy is the idea of data independence, which is the separation of the programmer's logical view of the database from the actual physical storage of the data. This separation permits various programs to have entirely distinct views of the same physical data. This can be described more clearly with the help of an illustration of data independence as shown in Figure 4.5.


Applications
Physical data storage devices
DBMS
Logical data independence
Physical data independence

NOTES

Fig. 4.5 *A Simplified View of Data Independence provided by DBMS*

- The physical database report is basically a map of the physical layout of the data on a number of storage devices within an association.
- The viewpoint of the overall logical database explanation can be defined as a map of the complete logical database, also known as a schema.
- Every application that shows the parts of the data is known as sub-schema which is used by each application.

The database philosophy helps to analyse the essential events that take place when the application program calls on a DBMS for accessing a particular record. With the help of the database approach, intricate relationship within the data can be represented.

4.8 THEORY OF SERIALIZABILITY

One obvious way to prevent transactions from interfering with one another is to execute them serially. That is, a given transaction must be committed before the next transaction can begin execution. While this will certainly guarantee a consistent and correct database, it goes against one of the primary goals of a DBMS. In other words, multi-user access to the database should be maximized. In order to accomplish this goal, concurrent access to the database is clearly required. However, many of the concurrent operations on a database would be accessing different items in the database and would clearly not cause any conflicts and could thus be scheduled concurrently. Some of these operations however, will potentially conflict and cause consistency problems if left unresolved. It is the responsibility of the scheduler, in conjunction with the recovery subsystem, to determine if a concurrent schedule of transactions will leave the database in a correct state or if some intervention is required to ensure that it does. In order to do this, the scheduler needs to be able to determine if the concurrent schedule of transactions is equivalent to some serial ordering of the same transactions.

A schedule is a sequence of operations performed by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions. As before, we are only interested in the set of operations performed by the transactions which affect the database, so the schedule consists only of a sequence of read and/or write operations followed by either a commit or abort operation.

Serial Schedule

In a serial schedule, the transactions are performed in serial order. For example, two transactions T1 and T2 could be performed in serial order in two ways: T1 followed by T2 or T2 followed by T1. Similarly, for a set of three transactions T1, T2, and T3, the following are the possible ways in which transactions can be executed in serial order:

T1 followed by T2 followed by T3	T1 followed by T3 followed by T2
T2 followed by T1 followed by T3	T2 followed by T3 followed by T1
T3 followed by T1 followed by T2	T3 followed by T2 followed by T1

In a serial schedule, there is no possibility of interference between transactions since they are executed in isolation. However, there is no guarantee that the results of all serial executions of a given set of transactions will be identical. For example, in banking, whether interest is calculated on an account before a large deposit is made or after it is made is of great significance.

Serializability

Suppose two transactions, T_1 and T_2 are to be scheduled. They can be scheduled in a number of ways. To schedule them serially without being bothered about interleaving would be the most common way. In a serial schedule transaction, all operations of transaction T2 should follow all the operations of T1 or vice versa.

	T_1	T_2
	Read(X)	
	X=X+N	
	Write(X)	
	Read(Y)	
	Y=Y+N	
	Write(Y)	
Time		Read(X)
		X=X+P
		Write(X)

NOTES

Non-interleaved Serial Schedule A

T_1	T_2		T_1	T_2
Read(X)	read(X)			Read(X)
		$\rightarrow$		
X=X+N	X=X+P			X=X+P
Write(X)	Write(X)			Write(X)
Read(Y)			Read(X)	
Y=Y+N			|	
Write(Y)			|	

Non-interleaved Serial Schedule B

Now, these can be termed as serial schedules, since all the operations of one transaction are being followed by the entire sequence of operations of the other transaction.

In the interleaved mode, the operations of T_1 are mixed with the operations of T_2. This can be done in a number of ways. Two such sequences are given below:

T_1	T_2
Read(X)	
X=X+N	
	Read(X)
	X=X+P
Write(X)	
Read(Y)	
	Write(X)
Y=Y+N	
Write(Y)	

Interleaved non-serial schedule C

T_1	T_2
Read(X)	
X=X+N	
Write(X)	
	Read(X)
	X=X+P
	Write(X)
Read(Y)	
Y=Y+N	
Write(Y)	

Interleaved (Non-serial) Schedule D

If for every transaction T in a schedule S all the operations are executed consecutively, we call it a serial schedule; otherwise, we call it a non-serial schedule. A non-interleaved schedule of independent transactions always results in a consistent database state, as the transactions commit or abort before the beginning of the next transaction. A non-interleaved schedule is guaranteed to produce a correct result as long as the individual transactions are free of errors. However, non-interleaved schedules suffer from low utilization and wastage of resources. Often, a transaction waiting for an I/O makes the subsequent transactions wait causing wastage of resources or reduction in resource utilization. If the former transaction takes too long to execute, the latter one keeps waiting till its completion.

The problem with serial schedules is resource wastage. In case of a serial schedule, if a transaction is waiting for an I/O, the subsequent transactions will also wait causing wastage of resources. If a transaction T is extremely long, the other transactions will have to keep waiting till T is completed. In such a schedule, no concurrency is supported. Therefore, in general, the serial scheduling concept is unacceptable in practice.

Serial schedules are unacceptable in practice. To solve this problem, the operations need to be interleaved. But the interleaving sequence should be well planned. A wrong interleaving sequence may leave the schedule incorrect and the database inconsistent. Therefore, a methodology should be to determine which schedules produce correct results and which ones do not.

A schedule S of N transactions is said to be a serializable schedule if it is equivalent to a serial schedule which comprises the same N transactions. N transactions can produce N! serial schedules. If you carry on interleaving them, the number of possible combinations becomes very large. To solve this problem, all the non-serial schedules are divided into two disjoint groups. One comprises those schedules that are equivalent to one or more serial schedules and the other consists of those schedules that are not. The first category of schedules is called 'serializable schedules' and the latter is called 'non-serializable schedules'.

Conflicting Actions

Conflict between two or more actions takes place if:

1. The read or write action is performed on the same data object.
2. The actions are performed by different transactions.
3. At least one of the actions performed is a write operation.

The following set of actions is conflicting:

- T1: Read(X), T2:Write(X), T3:Write(X)

While the following sets of actions are not:

- T1: Read(X), T2:Read(X), T3:Read(X)
- T1: Read(X), T2:Write(Y), T3:Read(X)

NOTES

Conflict Equivalence

Two schedules can be said to be conflict equivalent, if any two conflicting operations in both the schedules are executed in the same order. If somehow the order of conflicting operations in both the schedules is not the same then the schedules produce different database states at the end and hence they cannot be equivalent to each other.

Conflict-Serializable Schedule

A conflict-serializable schedule is one which is conflict-equivalent to other serial schedule(s).

You can say that a schedule is conflict-serializable only if an acyclic precedence graph or a serializability graph exists for the schedule.

G =

T1	T2
Read(A)	
	Read(A)
Write(B)	
COMMIT	
	Write(A)
	COMMIT

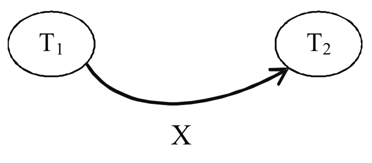
Which is conflict-equivalent to the serial schedule <T1,T2>

Testing for Conflict Serializability of a Schedule

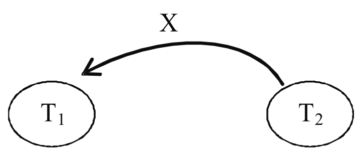
To test a schedule for conflict serializability, an algorithm may be suggested:

1. For each transaction T_i, participating in the schedule S, create a node labelled T_1 in the precedence graph.
2. For each case where T_j executes a Read(X) after T_i executes write(X), create an edge from T_i to T_j in the precedence graph.
3. For each case where T_j executes Write(X) after T_i executes a Read(X), create an edge from T_i to T_j in the graph.
4. For each case where T_j executes a Write(X) after T_i executes a Write(X), create an edge from T_i to T_j in the graph.
5. The schedule S is serializable if and only if there are no cycles in the graph.

If we apply these methods to write the precedence graphs for the above four cases, we get the following figure. (Refer Figure 4.6)



Schedule A



Schedule B

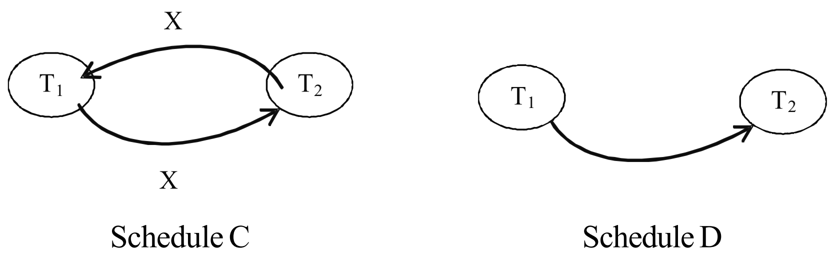


Fig. 4.6 *Testing Serializability of a Schedule*

We may conclude that schedule D is equivalent to schedule A.

View Equivalence

Two schedules S1 and S2 are said to be view-equivalent when the following conditions are satisfied:

1. For every data item A if transaction T_i reads the initial value of A in S1, then in S2 also transaction T_i must read the initial value of A.
2. For each data item A if transaction Ti executes a Read(A) on data item A and if the value was produced by a Write(A) operation performed by another transaction T_i in S1, then in S2 also the same transaction T_i should perform Read(A) on the value produced by Write(A) operation by T_i.
3. For each data item A in S if the transaction Ti performs the final Write (A) operation, then in S2 also the final Write (A) operation must be performed by T_i.

View-Serializable Schedule

A view-serializable schedule is view-equivalent to some serial schedule. All conflict-serializable schedules are view-serializable.

G =

T1	T2
Read(A)	
	Read(A)
Write(B)	
COMMIT	
	Write(A)
	COMMIT

Above is an example of a schedule which is both view-serializable as well as conflict-serializable. There are view serializable schedules which are not conflict serializable. Such a schedule contains 'blind writes'.

H =

T1	T2	T3
Read(A)		
	Write(A)	
	COMMIT	
Write(A)		
COMMIT		
		Write(A)
		COMMIT

The above example is not conflict-serializable, but it is view-serializable since it has a view-equivalent serial schedule <T1, T2, T3>.

Let us take another example,

S1 = {[T3:read(A)], [T4:write(A)], [T3:write(A)], [T6:write(A)]}

S2 = {[T3:read(A)], [T3:write(A)], [T4:write(A)], [T6:write(A)]}

Note that S1 is a concurrent schedule and S2 is a serial schedule. S1 is view equivalent to S2 since the one read(A) instruction reads the initial value of A in both schedules and T6 performs the last write(A) operation in both schedules. Therefore, S1 is a view serializable schedule.

The only difference between conflict serializability and view serializability is that the former satisfies 'constrained write' while the latter does not. This condition states that any write operation Wi(A) in Ti is preceded by an Ri(A) is Tj and that the value written by Wi(A) in Tj depends only on the value of A read by Ri(A). The assumption behind the concept of constrained write is that the new value of a data item A is a function based on the old value of the same data item in the database. So view serializability is less stringent than conflict serializability. View serializability applies the concept of 'unconstrained write assumption' where the value of a data item produced by a write operation performed by any transaction does not depend upon the old value of the data item.

The only problem with view serializability is its complexity with regard to computation. No efficient algorithm exists to do the same.

Uses of Serializability

If you say that S is correct, it implies or is equivalent to proving the serializability of schedule S.

Therefore, it is guaranteed that the schedule will provide correct results. However, being serial and being serializable are different things. A serial scheduling may be inefficient due to reasons explained earlier, which causes underutilization of the CPU, I/O devices and in some cases such as mass reservation system, becomes untenable. A serializable schedule, on the other hand, not only has the advantages of concurrent execution (ability to cater to numerous simultaneous users, efficient system utilization) but also guarantees correctness.

The scheduling process is done by the operating system routines after considering various factors such as the priority of the process in comparison to other processes, time of submission of transaction, system load, and many other factors. Also, since it is possible to have numerous interleaving combinations, it is very difficult to determine, in advance, the way in which the transactions are interleaved. That is, getting the various schedules itself is tough, leave alone testing them for serializability.

Therefore, most DBMS protocols employ a more practical technique. Instead of generating the schedules and checking for serializability before using them, they apply restrictions or controls on the transactions themselves. These restrictions are followed by each participating transaction, automatically ensuring serializability in all the schedules that the participating schedules create.

In addition, it is not easy to determine the start of a schedule and its finish, since transactions get submitted at different times.

Therefore, the theory of serializability can be used to tackle this problem by Considering only the Committed Projection C(CS) of the schedule. Therefore, as an approximation, a schedule S can be defined as serializable if its committed C(CS) is equivalent to some serial schedule.

4.9 CONCURRENCY CONTROL MODEL AND ALGORITHMS

As we have seen from our discussions regarding serializable schedules, concurrent access to the database will guarantee a consistent and correct database only if the concurrent schedule is equivalent to some serial schedule. What we now need is a technique that will ensure that the concurrent schedule is serializable. In other words, as transactions execute concurrently on the database, we will allow them to do so only with respect to some protocol that will guarantee that their schedule is serializable.

Serializablity can be achieved by several different means. *Locking* and *time-stamping* protocols are two very common techniques. Locking is a more pessimistic approach to concurrency control while time-stamping is a more optimistic approach. *Versioning* is a more recent optimistic approach to concurrency control which is rapidly gaining popularity amongst DBMS developers. Locking protocols can be quite restrictive and may lead to unnecessary blocking, live lock or starvation, and in extreme situations deadlock. Time-stamping protocols, while not suffering from these problems, are more complicated to implement and require a higher-level of sophistication from the DBMS. We'll focus primarily on locking protocols which are the most common type of concurrency control mechanisms in standard DBMS.

Locking Protocol for Concurrency

Locking is the most popular concurrency control mechanism implemented by many important techniques. Lock is a variable associated with every data item to indicate whether the data item is available for an operation to be performed on it. Lock variable synchronizes the operations performed on these data items by the concurrent transactions. A planned and proper implementation of lock solves many problems due to concurrency listed earlier. However, there are some problems created by lock itself. We shall be discussing these in the subsequent sections.

Lock granularity

Locking ensures conflict prevention by holding a lock on various parts of a database. Sometimes, these locks are held by the DBMS; sometimes the DBA or a user himself holds the lock explicitly. Applying the highest level of concurrency in databases with a high level of concurrency degrades the performance. Typical levels of locking are given as follows:

(i) **Database locking:** The entire database is locked in this method during updating by any user. This is the easiest method to be implemented but not a very accepted one. This method applied only when the DBA is performing some maintenance task on the database like restoring, backing up etc.

(ii) **Table locking:** Some operating systems consider tables to be equivalent to files. If it is so, then those systems can apply file locking systems to lock the tables of the database. This method is efficient but can lead to congestion in busy databases.

(iii) **Page locking:** It is better to lock a separate page instead of locking the entire page. In fact, a page is a disk concept. A file stored on the disk is divided and written into physically separate sections called sectors. So, it is efficient to lock access to a disk page or sector permitting the users to access the rest of the table.

(iv) **Row locking:** This locks a database row denying any request of modification by any transaction on the database. This technique involves high implementation overhead.

(v) **Column locking:** Column locking is supported by very few databases. This method is generally considered unacceptable because of the high overhead involved in it.

Types of Locks

- **Binary locks:** In this method, the lock variable can have only two states. This method is too simple, but too stringent. This is not a very good method to be accepted.
- **Shared/exclusive locks:** This method is more practical and accepted in general lock-based systems. This scheme locks the data items in two modes. Shared or read mode, and exclusive or write mode.
- **Certify lock:** This is used to improve the performance of locking protocols.

(i) *Binary locks*

As we said previously, a binary locking scheme has two states. The states are designated by '0' and '1'. These two digits are used to indicate whether a data item is locked or unlocked. For example if '1' is presumed to indicate that a lock is held then '0' indicates that the lock is open. This means that if the value of the lock associated with the data item X is '1' then no operation can be performed on the data item. It indicates that the item is locked.

The concept is that a data item cannot be modified by transactions if it is locked. If a transaction is modifying the value of a data item it cannot be accessed until the modification is completed. So, if a transaction wants to modify a data item it requests a lock on it. Now, the lock request is granted only if the data item is not already locked by some other transaction. Otherwise, it has to wait for the lock. After the lock is acquired, the transaction performs the modification and finally releases the lock.

The two operations lockitem(X) and unlockitem(X) are required to be performed to implement this method. Lockitem(X) locks a data item and prevents it from being accessed by other transactions and unlockitem(X) releases the lock on a data item and makes it available to the other transactions. Therefore, from the point of view of the lock variable we can describe the procedure as follows:

Any transaction wanting to hold a lock on a data item first checks the value of the lock variable associated with it. If it finds the lock status of the variable to be '1', then it is already locked. The transaction then waits. Once the lock status of the data item becomes '0' the waiting transaction can hold a lock on it to perform the modifications. It then issues a lockitem(X) command. Once it completes its operation with the data item, it issues unlockitem(X) and resets the value of lock variable so that some other transaction waiting for acquiring a lock can proceed.

Notice that the binary lock essentially produces a 'mutually exclusive' type of situation for the data item, so that only one transaction can access it. These operations can be easily written as an algorithm as follows:

The locking algorithm

```
      Lockitem(X):
      Start: if Lock(X)=0 /* item is unlocked*/
             Lock(X)=1 /*lock it*/


   Else
       {
             wait(until Lock(X)=0) and the lock manager wakes up the
  transaction)
          go to start;
   }
```

The unlocking algorithm

```
     Unlock item(X):
     Lock(X) ← 0;
     {If any transactions are waiting,
     Wakeup one of the waiting transactions}
```

The only restriction on the use of the binary locks is that they should be implemented as indivisible units (also called 'critical sections' in operating systems terminology). That means, no interleaving operations should be allowed, once a lock or unlock operation is started, until the operation is completed. Otherwise, if a transaction locks a unit and gets interleaved with many other transactions, the locked unit may become unavailable for long times to come with catastrophic results.

To make use of the binary lock schemes, every transaction should follow certain rules:

1. The lockitem(X) command is to be issued before issuing a read(X) or write(X).

2. The unlockitem(X) command is to be issued after completion of all read(X) and write(X) operations on X.
3. If a transaction holds a lock on a data item X, it should not issue another lockitem(X) command.
4. If a transaction is not currently holding a lock on a data item X, it should not issue an unlockitem(X) command.

No other transaction Tj would be permitted to operate on a data item X between a lockitem(X) and an unlockitem(X) issued by Ti. Therefore, between these intervals only Ti holds the value of the data item X. Thus, many of the above mentioned problems are solved by this scheme.

(ii) *Shared/Exclusive locks*

A binary lock is easy to implement, it also looks satisfactory enough but suffers from serious difficulties. It strictly prohibits more than one transaction to access a data item simultaneously. This scheme does not even permit more than one transaction to perform a read operation on the same data item simultaneously. This is where the problem lies. While one transaction is performing a write operation, the other transactions should not be permitted to perform a write or read operation on the same data item. But where is the harm in allowing two read operations to take place simultaneously? In fact, allowing simultaneous read operations on the same data item would increase the system's performance without causing any harm to the database.

This concept gave rise to the idea of shared/exclusive locks. The notion of exclusive lock is too rigid and has performance side effects. There is a need to make the locking mechanism less stringent by introducing shared or read locks.

There are two types of shared locks:

- Exclusive locks/write locks (X locks)
- Shared locks/read locks (S locks)

(a) Write lock

A write lock set up on a data item, allows a transaction to read and | or modify its value, exclusively. No other transaction can read or write to that data item while the write lock is in effect.

If transaction A holds an X lock on record p, then transaction B requesting a lock on the same record will be denied.

(b) Read lock

A read lock is set up by a transaction. It is non-exclusive, i.e., it can be shared among many readers, allowing parallel reads to happen. Nobody can change the data item while the read lock is on.

NOTES

Figure 4.7 shows the ***lock compatibility matrix***.

Current State of Lock of Data Items

Requested Lock		Exclusive	Shared	Unlocked
	Exclusive	N	N	Y
	Shared	N	Y	Y
	Unlock	Y	Y	–

Fig. 4.7 *Lock Compatibility Matrix*

Normally, locks are implicit. A FETCH request is an implicit request for a shared lock whereas an UPDATE request is an implicit request for an exclusive lock. Explicit lock requests need to be issued if a different kind of lock is required during an operation. For example, if an X lock is to acquired before a FETCH it has to be explicitly requested for.

We need to think of three operations, a read lock, a write lock and unlock. The algorithms can be as follows:

Read lock(X):

```
Start:  If Lock (X) = 'unlocked'
        {
                Lock(X) ← 'readlocked';
                no_of_reads(X) ← 1;
        }
        else if Lock(X) = 'read locked'
                no_of_reads(X) = no_of_reads(X)+1;
        else {
                wait until Lock(X) = "unlocked" and the lock manager
                wakes up the transaction);
         }
                go to start;
```

Write lock(X):

```
        Start : If lock(X) = 'unlocked'
                Lock(X) ← 'locked';
        else {
                Wait until Lock(X) = 'unlocked' and the lock manager wakes up
the transaction;
             }
      Go to start;
```

Unlock(X):

```
      If lock(X) = 'write locked'
            {
                Lock(X) ← 'unlocked';
                     Wakeup one of the waiting transaction, if any;
```

```
}
else if Lock(X) = 'read locked'
   {
         No_of_reads(X) ← No_of_reads –1;
         if no of reads(X)=0
         {
                  Lock(X) = 'unlocked';
                  Wakeup one of the waiting transactions, if any;
          }
}
```

The algorithms are fairly straightforward, except that during the unlocking operation, if a number of read locks are there, then all of them are to be unlocked before the unit itself becomes unlocked.

To ensure smooth operation of the shared / exclusive locking system, the system must enforce the following rules:

1. Readlock(X) or writelock(X) commands must be issued before any read or write operations are performed.
2. Writelock(X) command must be issued before performing any writetr(X) operation on the same data item.
3. An unlock (X) command must be issued after completion of all readtr(X).
4. A readlock(X) command must not be issued while holding readlock or writelock on X.
5. A writelock(X) command must not be issued while holding readlock or writelock on X.

Conversion of Locks

In some cases, it is desirable to allow lock conversion by relaxing the conditions (4) and (5) of the shared/ exclusive lock mechanism. That is, there should be provision for a transaction holding one type of lock on a data item to be converted to some other type of lock. For example, if a transaction is holding a read lock on a data item X, it may be permitted to upgrade it to writelock. If no other transaction is holding a lock on a data item X, it can upgrade its read lock to write lock on issuing a writelock(X) command. Otherwise the transaction waits for the others to release their readlocks on X. Similarly, any transaction holding a write lock on X is allowed to downgrade its write lock to read lock. The above algorithm can be amended to accommodate lock conversion.

It is important to note that use of binary locks does not guarantee serializability. The reason behind this is that in some situations or combinations of them, a key holding transaction may unlock the unit prematurely due to many reasons. One situation could be wherein a transaction does not need a certain data unit and therefore unlocks it but may be indirectly writing into it via some other unit at a later date. This would give rise to ineffective locking performance and loss of seralizability. Such serializability can be guaranteed by implementing two-phase locking.

(iii) *Optimistic locking and pessimistic locking*

Optimistic Locking: The data is locked only when data is being saved. The lock is released after data is saved.

Pessimistic Locking: The data is locked when editing begins. The lock is released after the data is saved or discarded.

Optimistic locking is only for solving physical I/O conflict. It cannot be used to handle the concurrency mentioned above. In other words, it cannot provide a solution the update loss problem and uncommitted dependency problem.

Other Concurrency Control Protocols

Concurrency control protocols are distinguished into *conservative* (or *pessimistic)* and *aggressive* (or *optimistic*).

(i) ***Conservative/Pessimistic protocols*** are based on the assumption that conflicts are frequent, so a transaction has to get permission before performing an operation on a data item.

A permission is denied if there would be a conflict that can potentially lead to a non-serialisable execution. Depending on the scheme, a requesting transaction is either made to wait (*blocked*) until the permission can be granted, as in the case of *locking* methods, or aborted and restarted, as in the case of ***timestamp*** methods.

(ii) ***Aggressive/Optimistic protocols*** are based on the assumption that conflicts are rare, therefore, it is possible for transactions to perform conflicting operations without having to wait. When they try to commit, serializability is ensured by validating or certifying the transactions. A transaction that is validated implies that no conflicting operations have been executed and commit can be done.

Each proposed concurrency control protocol has been shown to exhibit superior performance under certain situations, but no single protocol can perform best in all situations.

However, commercial DBMS, have adopted the *strict two-phase locking* protocol, because of its simplicity and ease of implementation compared to other alternatives.

Two-Phase Locking Protocols

In 2PL, a transaction requests an appropriate lock just before performing an operation. If a transaction requests a conflicting lock, it is *blocked*, awaiting the release of the conflicting lock. Otherwise, the appropriate lock is granted to the transaction.

Each transaction is executed in two phases:

Phase 1: The *growing phase* during which the transaction acquires all the required locks, but cannot release any lock.

Phase 2: The *shrinking phase* during which the transaction releases its locks, but cannot request additional locks on any data item.

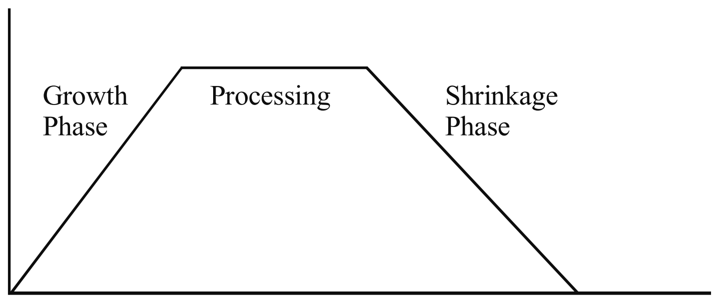


Fig. 4.8 *Two-Phase Locking Strategy*

readlock(Y)	
read(Y)	Growth Phase I
writelock(X)	
unlock(Y)	
read(X)	Shrinkage Phase II
X=X+Y	
write(X)	
unlock(X)	

No downgrading lock

According to these rules, a transaction cannot downgrade a write lock into a read lock, because downgrading is equivalent to releasing the write lock and subsequently requesting and acquiring a new read lock.

Upgrading a lock is accepted

However, it is possible for a transaction to upgrade a read lock into the stronger write lock during the growing phase.

The order in which transactions are granted locks forces an execution ordering on the transactions in connection with their conflicting operations because transactions are forced to wait on conflicting locks.

By preventing transactions from acquiring any lock after the release of a lock, the 2PL protocol ensures serializability.

This can be illustrated by showing how the *non-repeatable read* problem is solved using 2PL.

Recall our example below, of inconsistent retrieval involving two transactions T and S and two columns A and B. We depict below their non-2PL and 2PL executions with time diagrams. We denote local variables with lower case letters and with * values produced by committed transactions. In the 2PL execution, we specify the required lock requests.

A Non-2PL Execution

Time	T	S	A	B	Sum
t1	v=read(A)		3*	5*	8*
t2		x=read(A)	3*	5*	8*
t3	v= v+3		3*	5*	8*
t4	write(A,v)		6	5*	11
t5	v= read(B)		6	5*	11
t6	v = v-3		6	5*	11
t7	write(B,v)		6	2	8
t8	commit		6*	2*	8*
t9		y=read(b)	6*	2*	8*
t10		output(x+y) →	5	<>	8*
t11		commit			

In the non-2PL execution, there is a cyclic ordering captured by the conflicting of read(A) by S with the write(A) of T, and the conflicting of write(B) by T with the read(B) of S.

In the 2PL execution, by requesting and acquiring a write lock on A at the beginning, T blocks S's read lock request, forcing S to wait until T commits and reads the consistent values of A and B that are produced by T. In the non-2PL, a non-serializable execution, S reads inconsistent values for A and B. It reads B from T, and A from another previously committed transaction.

A 2PL Execution

Time	T	S	A	B	Sum
t1	v=read(A)		3*	5*	8*
t2		x=read(A)	3*	5*	8*
t1	writelock(A)				
t2	v=read(A)		3*	5*	8*
t3		readlock(A)	3*	5*	8*
t4	v= v+3	WAIT	3*	5*	8*
t5	write(A,v)	WAIT	6	5*	11
t6	writelock(B)	WAIT	6	5*	11
t7	v= read(B)	WAIT	6	5*	11
t8	v = v-3	WAIT	6	5*	11
t9	write(B,v)	WAIT	6	2	8
t10	release-locks	WAIT	6	2	8
t11	commit	WAIT	6*	2*	8*
t12		x=read(A)	6*	2*	8*
t13		readlock(B)	6*	2*	8*
t14		y=read(b)	6*	2*	8*
t15		output(x+y) →	8	==	8*
t16		release-locks			
t17		commit			

Advantages of 2PL

- It is easy to enforce. DBMS has to keep a track of only what phase a transaction is in.
- 2PL has been a major factor in the success of databases specially while handling concurrent transactions.
- It does not involve the real-time testing for serializability and the generation of precedence graphs.

Variations of 2PL

- Basic 2PL: This involves the growth and the release phase.
- Conservative 2PL: This involves acquiring of all locks at the beginning and releasing when done. It is inefficient, because you will request more than and longer than what is really need.
- Strict 2PL: There is a distinct growth phase, but the shrinkage phase is abrupt. Used by Oracle.

There are a number of variations of Two-Phase Locking (2PL). The technique just described is known as **basic 2PL.** A variation known as **conservative 2PL** (or **static 2PL**) requires a transaction to lock all the items it accesses *before the transaction begins execution,* by **predeclaring** its *read-set* and *write-set.* As mentioned earlier, the **write-set** is the set of all items that a transaction it writes and the **read-set** of a transaction is the set of all items that are read by the transaction. If any of the predeclared items required cannot be locked, the transaction does not lock any item. Instead, it waits until all the items are available for locking. Conservative 2PL is a deadlock-free protocol. However, it is difficult to use in practice because of the need to predeclare the read-set and write-set, which is not possible in most situations.

In practice, the most popular variation of 2PL is **strict 2PL,** which guarantees strict schedules. Here, none of the exclusive (write) locks are released by a transaction T till it has committed or aborted. Therefore, no item written by T can be read or written in by any other item until T commits. This leads to a strict recoverability schedule. Strict 2PL is not deadlock-free. A more restrictive variation of strict 2PL is **rigorous 2PL,** which also guarantees strict schedules. In this variation, a transaction T does not release any of its locks (exclusive or shared) until after it commits or aborts, and so it is easier to implement than strict 2PL. Notice the difference between conservative and rigorous 2PL; the former must lock all its items *before it starts* so once the transaction starts, it is in the shrinking phase, whereas the latter does not unlock any of its items until *after it terminates* (by committing or aborting) so the transaction is in its expanding phase until it ends.

In many cases, the **concurrency control subsystem** itself is responsible for generating the readlock and writelock requests. For example, if the system is to enforce the strict 2PL protocol, then, whenever transaction T issues a read(X), the system calls the readlock (X) operation on behalf of T. If the state of LOCK (X) is write-locked by some other transaction T', the system places T on the waiting queue for item X; otherwise, it grants the readlock(X) request and permits the read(X) operation of T to execute. On the other hand, if transaction T issues a writeitem (X), the system calls the writelock (X) operation on behalf of T. If the state of LOCK (X) is writelocked or readlocked by some transaction T', the system places T on the waiting queue for item X; if the state of LOCK (X) is readlocked and T itself is the only transaction holding the read lock on X, the system upgrades the lock to write locked and permits the write_item (X) operation by T; Finally, if the state of LOCK (X) is unlocked, the system grants the writelock (X) request and permits the write(X) operation to execute. After each action, the system must update its lock table appropriately.

Although the two-phase locking protocol guarantees serializability (that is, every schedule that is permitted is serializable), it does not permit *all possible* serializable schedules (that is, some serializable schedules will be prohibited by the protocol). In addition, the use of locks can cause two additional problems: deadlock and starvation.

Hierarchical locking

We can read/write lock data at various granularities.

- Entire database
- Relation
- tuple
- Disk page/block
- B-tree node
- Attribute value

Locking the entire database would reduce concurrency considerably. It may seem that maximal concurrency would be attained by just locking each attribute value, however, disk read/writes are usually done in pages or blocks, so there is no value to locking just a single attribute value, an entire block must be locked. But locking at this level requires more overhead on figuring out what to lock and where it is located on the disk. In some cases, we want to lock an entire relation, such as when we are reading an item in a B-tree. In such cases, we can't change the B-tree in the middle of looking for the item!

So let us modify our locking to look at a *tree* of items rather than just a single item. In this case, we can view the tree as either a tree where each node is independent (e.g., a B-tree) or a tree where the children are subsets of the parent (e.g., a tree of the database down to tuple level).

A simple protocol for independent node trees is the following:

- Can only lock if the lock is currently held by parent (except for first item locked).
- A single transaction cannot lock the same item twice.

Schedules that obey this simple protocol are serializable. (General idea of proof given in reading.)

This won't work for database-relation-tuple trees since locking a parent implies that you have exclusive use of all the children. We need to also warn others that they can't lock children. We have three types of locks:

(i) LOCK - Lock item and all descendants

(ii) WARN - No other transaction may lock (but may warn)

(iii) UNLOCK - Removes lock/warning from item

The warning protocol is as follows:

1. First locks or warns root
2. Only locks/warns if it holds a warning on the parent (by the transaction)
3. Only removes locks/warns if no child is locked/warned (by the transaction)
4. Obeys two-phase protocol - All locks/warns first, then all unlocks
5. Scheduler only allows a lock if no other transaction has a lock or warning on A, and allows a warning only if no other transaction has a lock on A.

Concurrency Control Protocols based on Timestamp

Timestamp ordering

Another protocol that we can use to generate serial schedules is **timestamp ordering**. The general idea is to order the transactions by time, and to use this ordering to give priority to read and writes in earlier transactions. This will result in a serializable schedule that is equivalent to the serial schedule of executing each transaction in the order in which it arrived.

Step 1: Assign a **timestamp** to each transaction. This is either:

- A unique integer (increased for each transaction), or
- A system clock time (only one transaction permitted per tick).

Step 2: The next step is to associate.

- Associate a read-time RT with each item which is the latest transaction time at which an item has been read.
- Associate a write-time WT with each item which is the latest transaction time at which an item has been written.

We want a serial schedule which only reads/writes in the following conditions:

- We can only read an item if it was written by some transaction earlier than us.
- We can only write an item if it has yet to be read by some transaction later than us.

So, when we read an item, we have to check that the WT is less than our transaction timestamp. When we write an item, we have to check that the RT is less than our transaction timestamp. In both cases, we update the WT and RT if the operation is permitted. If we have violated one of the conditions then the transaction must *abort*

TS(T): Timestamp of a transaction T

Read_TS(X): Read timestamp of an item X. This is the largest timestamp among all the timestamps of transactions which have successfully read X

Write_TS(X): Write timestamp of an item X. The largest timestamp among all the timestamps of

transactions which have successfully written X.

Protocol

Case 1: T requests READ(X).

Case 1.1: If TS(T) >= Write_TS(X), execute READ(X) and update Read_TS(X).

Case 1.2: If TS(T) < Write_TS(X), abort T.

Case 2: T requests WRITE(X).

Case 2.1: If TS(T) >= Write_TS(X) and TS(T) >= Read_TS(X),

execute WRITE(X) and update Write_TS(X).

Case 2.2: If Write_TS(X) > TS(T) >= Read_TS(X), ignore WRITE(X) and continue T.

Case 2.3: If TS(T) < Read_TS(X), abort T.

Abortion of a transaction may causes abortion of another transaction which has read a data item written by the aborted transaction called Cascading Rollback

Strict timestamp ordering

A variation of basic timestamp ordering called **strict** timestamp ordering ensures that the schedules are both **strict** (for easy recoverability) and (conflict) serializable.. In this variation, a transaction T that issues a read_item (x) or write_item(x) such that TS(T) > write_TS(X) has its read or write operation *delayed* until the transaction T that *wrote* the value of X (hence TS(T) =write_TS(X)) has committed or aborted. To implement this algorithm, it is necessary to simulate the locking of an item X that has been written by transaction T until T either commits or aborts. This algorithm does not cause deadlock, since T waits for T' only if TS (T)>TS(T').

Thomas's Write Rule. A modification of the basic TO algorithm, known as **Thomas's write rule,** does not enforce conflict serializability; but it rejects fewer write operations, by modifying the checks for the **write(X)** operation as follows:

1. If read_TS(X)>TS(T), then abort and roll back T and reject the operation.
2. If write_TS(X)>TS(T), then do not execute the write operation but continue processing. This is because some transaction with timestamp greater than TS(T) and hence after T in the timestamp ordering-has already written the value of X. Hence, we must ignore the **write(X)** operation of T because it is already outdated and obsolete. Notice that any conflict arising from this situation would be detected by case (1).
3. If neither the condition in part (1) nor the condition in part (2) occurs, then execute the **write(X)** operation T and write_TS(X) to TS(T).

Concurrency Control using Multi-Versioning

We can observe that concurrency control is required when two transactions are reading and writing to the same location. Problems arise because the correct *version* of the data at that location may have been corrupted by the other transaction, e.g., a transaction could read an *obsolete* value, that is, one that will be written in future by an earlier transaction in the serialization order.

An obvious solution then is to keep around *all* versions of a data item, either indefinitely (a transaction-time database) or just during the lifetime of a set of concurrent transactions.

Keep not only the latest value X1 of each data item X but also one or more old values X2,X3,... of the data item.

Reduce the probability of Cases 1.2 and 2.3 by letting transactions to access previous values of data items so that they preserve the order of their timestamps.

Read_TS(Xi): The largest timestamp among all the timestamps of transactions which have successfully read the version Xi of item X

Write_TS(Xi): The largest timestamp among all the timestamps of transactions which have successfully written the version Xi of item X

A new version of item X is created whenever WRITE(X) is executed.

Its Write_TS and Read_TS are initialized to the timestamp of the transaction which has created it.

A version Xk is removed when Write_TS(Xk) becomes smaller than max { Write_TS(Xj) | Write_TS(Xj) <= min{TS(T)} }.

Protocol

Case 1: T requests READ(X).

Find the version Xi such that Write_TS(Xi) =max { Write_TS(Xj) | Write_TS(Xj) <= TS(T) },

Execute READ(Xi), and update Read_TS(Xi).

Case 2: T requests WRITE(X).

Case 2.1: If there exists a version Xi with Write_TS(Xi) = max { Write_TS(Xj) | Write_TS(Xj) <= TS(T) } and TS(T) < Read_TS(Xi), abort T.

Case 2.2: Otherwise, create a new version of X.

Advantages

- Has access to previous versions of data
- Aborts less frequently in timestamp ordering protocol

Disadvantages

- Space blow-out (can mitigate with write-once storage media)
- Overhead on retrieving desired version
- Overhead on maintaining versions
- Doesn't solve concurrency control

Multiversion two-phase locking using certify locks

In this multiple-mode locking scheme, there are three locking modes for an item: read, write, and certify, instead of just the two modes (read, write) discussed previously. Hence, the state of LOCK (X) for an item X can be one of read-locked, certify-locked, or unlocked. In the standard locking scheme, with only read and write locks, a write lock is an exclusive lock. We can describe the relationship between read and write locks in the standard scheme by means of the lock compatibility table shown in Figure 4.9 (a). An entry of 'yes' means that, if a transaction T holds the type of lock specified in the column header on item X and if transaction T' can obtain the lock because the locking modes are compatible. On the other hand; an entry of 'no' in the table indicates that the locks are not compatible, so T' must wait until T releases the lock.

(a)

	Read	Write
Read	Yes	No
Write	No	No

(b)

	Read	Write	Certify
Read	Yes	Yes	No
Write	Yes	No	No
Certify	No	No	No

Fig. 4.9 *Lock Compatibility Tables: (a) A Compatibility Table for Read/Write Locking Scheme. (b) A Compatibility Table for Read/Write/Certify Locking Scheme*

In the standard locking scheme, once a transaction obtains a write lock on an item, no other transactions can access that item. The idea behind multiversion 2PL is to allow other transactions T' to read an item X while a single transaction T holds a write lock on X. This is accomplished by allowing *two versions* for each item X_i one version must always have been written by some committed transaction. The second version X' is created when a transaction T acquires a write lock on the item. Other transactions can continue to read the *committed version* of X while T holds the write lock. Transaction T can write the value of X' as needed, without affecting the values of the committed version X. However, once T is ready to commit, it must obtain a **certify lock** on all items that it currently holds write locks on before it can commit. The certify lock is not compatible with read locks, so the transaction may have to delay its commit until all its write-locked items are released by any reading transactions in order to obtain the certify locks. Once the certify locks – which are exclusive locks – are acquired, the committed version X of the data item is set to the value of version X'. Version X' is discarded, and the certify locks are then released. The lock compatibility table for this scheme is shown in Figure 4.9 (b).

In this multiversion 2PL scheme, reads can proceed concurrently with a single write operation – an arrangement not permitted under the standard 2PL schemes. The cost is that a transaction may have to delay its commit until it obtains exclusive certify locks on *all the items* it has updated. It can be shown that this scheme avoids cascading aborts, since transactions are only allowed to read the version X that was written by a committed transaction. However, deadlocks may occur if upgrading of a read lock to a write lock is allowed, and these must be handled by variations of the techniques discussed later on.

Validation (Optimistic) Concurrency Control Techniques

Optimistic concurrency control techniques are also known as validation or certification techniques. In such techniques no checking is done while the transaction is being executed. In many of the concurrency control methods, validation-based technique is used. Until the transaction reaches its end, no updates in the transaction are directly applied to the items in the database. The updates are applied to the local copies of transactions while the transaction is being executed.

After completion of the transaction a **validation phase** checks whether or not the serializability order is maintained. The system must keep certain information that the validation phase requires. The transaction is allowed to commit if it does not violate serializability. The local copies are used to update the database. However, it serializability is violated, the transaction aborts and restarts later.

There are three phases for this concurrency control protocol:

1. **Read phase:** In this phase, a transaction reads the values of committed data items from the database and applies updates on the local copies of data items. These local copies are kept in a workspace reserved for the transaction.
2. **Validation phase:** This phase checks whether the updates made to the database violate the serializability order.
3. **Write phase:** If the validation phase finds that application of updates on the database items does not disturb the serializability order, then the updates are made permanent on the database. Otherwise, the transaction is aborted discarding the updates. It is restarted later.

The aim of a validation-based protocol is to do all the checks at once. This also reduces the overhead of transaction until the validation phase. In systems with little interference between transactions, most of the schedules will be found to be serializable. However, systems including high interference between transactions will fail in the validation phase. As a consequence, many transactions will have to abort after their completion. The updates committed by them will have to be discarded. They will have to start later. Under these circumstances, this approach does not perform well. This technique assumes that there will not be much interference in the system. Hence, the schedule will be validated in the validation phase. Therefore, this approach is named as '**optimistic approach'**.

The optimistic protocol uses transaction timestamps and maintains write_sets and read_sets of the transactions in the system. In addition, for each transaction, the start and end times for some of the three phases need to be kept.

The set of items that a transaction writes is referred to as the **write_set** of that transaction. Similarly, the set of items that a transaction reads is called the **read_set.**

For transaction T_i, the protocol checks to ensure that T_i, does not interfere with any committed transactions or with any other transactions currently in their validation stage. This checking takes place in transaction T_i,'s validation phase. This phase also checks that for each similar transaction T_j which is committed or which is in the validation phase, any one of the following conditions is met:

1. Write phase of transaction T_j gets completed before the read phase of transaction T_i begins.
2. T_i begins the write phase after the write phase of T_j is completed; and the read_set of T_i does not have any items in common with the write_set of T_j.
3. The read_set and write_set of T_i do not have any times in common with the write_set of T_j and T_j completes its read phase before the read phase of $T_{i.}$

While validating transaction T_i, the condition that is checked first is the first condition for each transaction $T_{j.}$

This is because of the following reasons:

(i) It is the most simple condition to check

(ii) The second condition will be checked only if the first condition is false.

(iii) The third condition which is the most complicated, is checked only if the second one is false.

There will be no interference and T_i is validated successfully if any one of these three conditions is met. However, if not even one of these three conditions holds, the validation of transaction T_i fails. AS interference may have taken place, it is aborted and restarted later.

4.10 SECURITY POLICIES

Each database has one or more administrators who are responsible for maintaining all aspects of the security policy: the security administrators. If the database system is small, the database administrator may have the responsibilities of the security administrator. However, if the database system is large, a special person or group of people may have responsibilities limited to those of a security administrator.

Database User Management

Database users are the access paths to the information in an Oracle database. Therefore, tight security should be maintained for the management of database users. Depending on the size of a database system and the amount of work required to manage database users, the security administrator may be the only user with the privileges required to create, alter, or drop database users. On the other hand, there may be a number of administrators with privileges to manage database users. Regardless, only trusted individuals should have the powerful privileges to administer database users.

User Authentication

Database users can be *authenticated* (verified as the correct person) by Oracle using the host operating system, network services, or the database. Generally, user authentication via the host operating system is preferred for the following reasons:

- Users can connect to Oracle faster and more conveniently without specifying a username or password.
- Centralized control over user authorization in the operating system: Oracle need not store or manage user passwords and usernames if the operating system and database correspond.
- User entries in the database and operating system audit trails correspond.

User authentication by the database is normally used when the host operating system cannot support user authentication.

Operating System Security

If applicable, the following security issues must also be considered for the operating system environment executing Oracle and any database applications:

- Database administrators must have the operating system privileges to create and delete files.
- Typical database users should not have the operating system privileges to create or delete files related to the database.
- If the operating system identifies database roles for users, the security administrators must have the operating system privileges to modify the security domain of operating system accounts.

Data Security Policy

Data security includes the mechanisms that control the access and use of the database at the object level. A data security policy determines which users have access to a specific schema object, and the specific types of actions allowed for each user on the object. The policy should also define the actions, if any, that are audited for each schema object.

A data security policy will be determined primarily by the level of security one wish to establish for the data in one's database. For example, it may be acceptable to have little data security in a database when a firm wishes to allow any user to create any schema object, or grant access privileges for their objects to any other user of the system. Alternatively, it might be necessary for data security to be very controlled when the firm wishes to make a database or security administrator the only person with the privileges to create objects and grant access privileges for objects to roles and users.

Overall data security should be based on the sensitivity of data. If information is not sensitive, then the data security policy can be more lax. However, if data is sensitive, a security policy should be developed to maintain tight control over access to objects.

User Security Policy

User security policy includes the following points:

- General User Security
- End-User Security

- Administrator Security
- Application Developer Security
- Application Administrator Security

1. General User Security

For all types of database users, password security and privilege management come under general user security.

Password Security: If user authentication is managed by the database, security administrators should develop a password security policy to maintain database access security. For example, database users should be required to change their passwords at regular intervals, and of course, when their passwords are revealed to others. By forcing a user to modify passwords in such situations, unauthorized database access can be reduced.

Privilege Management: Security administrators should consider issues related to privilege management for all types of users. For example, in a database with many usernames, it may be beneficial to use roles (which are named groups of related privileges that you grant to users or other roles) to manage the privileges available to users. Alternatively, in a database with a handful of usernames, it may be easier to grant privileges explicitly to users and avoid the use of roles.

Security administrators managing a database with many users, applications, or objects should take advantage of the benefits offered by roles. Roles greatly simplify the task of privilege management in complicated environments.

2. End-User Security

Security administrators must also define a policy for end-user security. If a database is large with many users, the security administrator can decide what groups of users can be categorized, create user roles for these user groups, grant the necessary privileges or application roles to each user role, and assign the user roles to the users. To account for exceptions, the security administrator must also decide what privileges must be explicitly granted to individual users. Roles are the easiest way to grant and manage the common privileges needed by different groups of database users.

3. Administrator Security

Security administrators should have a policy addressing administrator security. For example, when the database is large and there are several types of database administrators, the security administrator may decide to group related administrative privileges into several administrative roles. The administrative roles can then be granted to appropriate administrator users. Alternatively, when the database is small and has only a few administrators, it may be more convenient to create one administrative role and grant it to all administrators.

Protection for Administrator Connections: Only database administrators should have the capability to connect to a database with administrator privileges. Connecting as SYSDBA gives a user unrestricted privileges to do anything to a database (such as startup, shutdown, and recover) or the objects within a database (such as create, drop, and delete from). Roles are the easiest way to restrict the powerful system privileges and roles required by personnel administrating of the database.

4. Application Developer Security

Security administrators must define a special security policy for the application developers using a database. A security administrator may grant the privileges to create necessary objects to application developers. Alternatively, the privileges to create objects may only be granted to a database administrator, who receives requests for object creation from developers.

Application Developers and Their Privileges: Database application developers are unique database users who require special groups of privileges to accomplish their jobs. Unlike end users, developers need system privileges, such as CREATE TABLE, CREATE PROCEDURE, and so on. However, only specific system privileges should be granted to developers to restrict their overall capabilities in the database.

5. Application Administrator Security

In large database systems with many database applications (for example, precompiler and Forms applications), you might want to have application administrators. An application administrator is responsible for the following types of tasks:

- creating roles for an application and managing the privileges of each application role
- creating and managing the objects used by a database application
- maintaining and updating the application code and Oracle procedures and packages, as necessary

Often, an application administrator is also the application developer that designed the application. However, these jobs might not be the responsibility of the developer and can be assigned to another individual familiar with the database application.

Password Management Policy

Database security systems depend on passwords being kept secret at all times. Still, passwords are vulnerable to theft, forgery, and misuse. To allow for greater control over database security, Oracle's password management policy is controlled by DBAs.

Let us discuss the following aspects of password management.

1. **Account Locking:** When a particular user exceeds a designated number of failed login attempts, the server automatically locks that user's account. DBAs specify the permissible number of failed login attempts using the CREATE PROFILE statement. DBAs also specify the amount of time accounts remain locked.
2. **Password Aging and Expiration:** DBAs use the CREATE PROFILE statement to specify a maximum lifetime for passwords. When the specified amount of time passes and the password expires, the user or DBA must

change the password. The following statement indicates that ASHWINI can use the same password for 90 days before it expires:

```
CREATE PROFILE prof LIMIT
 FAILED_LOGIN_ATTEMPTS 4
 ACCOUNT_LOCK_TIME 30
 PASSWORD_LIFE_TIME 90;
ALTER USER ashwini PROFILE prof;
```

DBAs can also specify a grace period using the CREATE PROFILE statement. Users enter the grace period upon the first attempt to login to a database account after their password has expired. During the grace period, a warning message appears each time users try to log in to their accounts, and continues to appear until the grace period expires. Users must change the password within the grace period. If the password is not changed within the grace period, the account expires and no further log ins to that account are allowed until the password is changed.

3. **Password History:** DBAs use the CREATE PROFILE statement to specify a time interval during which users cannot reuse a password.

   In the following statement, the DBA indicates that the user cannot reuse her password for 60 days.

```
CREATE PROFILE prof LIMIT
 PASSWORD_REUSE_TIME 60
 PASSWORD_REUSE_MAX UNLIMITED;
```

   The next statement shows that the number of password changes the user must make before her current password can be used again is 3.

```
CREATE PROFILE prof LIMIT
PASSWORD_REUSE_MAX 3
PASSWORD_REUSE_TIME UNLIMITED;
```

4. **Password Complexity Verification:** Oracle's password complexity verification routine can be specified using a PL/SQL script (utlpwdmg.sql), which sets the default profile parameters.

   The password complexity verification routine performs the following checks:

   - The password has a minimum length of 4.
   - The password is not the same as the userid.
   - The password has at least one alpha, one numeric and one punctuation mark.
   - The password does not match simple words like welcome, account, database, or user.
   - The password differs from the previous password by at least 3 letters.

Auditing Policy

Security administrators should define a policy for the auditing procedures of each database. You may, for example, decide to have database auditing disabled unless questionable activities are suspected. When auditing is required, the security administrator must decide what level of detail to audit the database; usually, general system auditing is followed by more specific types of auditing after the origins of suspicious activity are determined.

Check Your Progress

5. Define the term view.
6. What is materialized view?
7. What is database system?
8. What is the basic problem with serial schedules?
9. Define the term conflict-serializable schedule.
10. State the ways in which serializability can be achieved?
11. Differentiate between optimistic locking and pessimistic locking.
12. What is timestamp ordering?

4.11 ANSWERS TO 'CHECK YOUR PROGRESS'

1. Integrity of a database refers to the correctness of the data items. Integrity ensures the non-occurrence of accidental deletion or alteration.
2. Database security comprises policies taken to protect data from being accessed by unauthorized users. This also includes the policies needed to ensure that data items are not modified or deleted from the database by unauthorized users, applications or viruses.
3. In the former each user is granted privileges and authorities to access certain records, pages or files and denied access to others. In the latter, there are standard security mechanisms that are used to enforce multilevel security by classifying the data into different levels and allowing the users access to only certain levels based on the security policies of the organization.
4. A Trojan horse is a virus program which internally works something else then what is specified by the user. The Trojan Horses are generally enclosed so that they appear attractive, for example a saxophone.wav file is a virus file which draws the attention of a system user interested in collecting sound samples of musical instruments. A Trojan horse is a different from other destructive virus because it does not reproduce. It steals the passwords and sends an e-mail to the hacker and then the hacker has all your description in his hands.
5. A view may be defined as a 'stored query' or a 'virtual table'. It is called virtual table because, they do not exist as independent entities in the database as do 'real' tables. A view is a specification consisting of a SELECT statement that tests Oracle what records and attributes to retrieve, When view is called, the system associates the appropriate data with it. It retrieves derived data only when they are called.

6. A materialized view is like a view in that it represents data that is contained in other database tables and views; yet it is unlike a view in that it contains the actual data. A materialized view is like an index in that the data it contains is derived from the data in database tables and views; yet unlike an index in that its data must be explicitly refreshed.
7. A database system is a collection of application programs that interacts with the database along with DBMS (and sometimes the users who use the system). Database systems are designed in a manner that facilitates the management of huge bodies of information.
8. The basic problem with serial schedules is of resource wastage.
9. A conflict-serializable schedule is one which is conflict-equivalent to other serial schedule(s).
10. Serializablity can be achieved by several different means. Locking and time-stamping protocols are two very common techniques.
11. In optimistic locking, the data is locked only when it is being saved. The lock is released after the data is saved. In pessimistic locking, the data is locked when editing begins. The lock is released after the data is saved or discarded.
12. Timestamp is a protocol that can be used to generate serial schedules. The general idea is to order the transactions by time, and to use this ordering to give priority to read and writes in earlier transactions. This will result in a serializable schedule that is equivalent to the serial schedule of executing each transaction in the order in which it arrived.

4.12 SUMMARY

- Integrity of a database refers to the correctness of the data items. Integrity ensures the non-occurrence of accidental deletion or alteration.
- The data stored in the database has to be protected from unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency.
- Database security comprises policies taken to protect data from being accessed by unauthorized users. This also includes the policies needed to ensure that data items are not modified or deleted from the database by unauthorized users, applications or viruses.
- Privacy is concerned with the ethical and legal rights regarding access to personal data items. The medical details of a person, for instance, may be considered critical information. Unauthorized persons should be denied access or kept from modifying such information/records.
- System-related security is important to know the level at which security should be enforced on the system. It could be enforced at the level of the operating system, or at the level of the hardware or even at the DBMS level.

- The primary requirement of database security is to forbid an individual from obtaining something by unfair means and entering into an organization's computing facility. Security laws should be so implemented that they make any attempt of unauthorized users to access systems resources illegal.
- Database system lists the required information regarding all the users and the operations performed by them during a login session.
- Statistical databases store confidential details about organizations or users, for example, a database containing the income and medical records of various individuals.
- Statistical database aims at maximizing information-sharing and preserving the privacy of individuals.
- Threat refers to any deliberate or accidental occurrence of action that may badly affect the database system. Database security can be violated by many sources.
- A successful SQL injection attack can give someone limitless and clear access to an entire database. SQL injection involves inserting (Or 'Injecting') illegal or malicious database statements into a at risk SQL data channel, such as a stored procedure or Web application.
- Hacking, cracking and cyber vandalism is also done by unauthorized persons.
- A computer virus is a set of executable code that attaches itself to other programs to replicate itself without the awareness of a system user. These computer viruses can damage the system of a computer.
- A Macro virus is written in a macro language. Macro viruses are spread by various applications which use and run created macros.
- There are three different classes of antivirus packages, namely, activity monitors, authentication or change detection software and scanners.
- The Internet provides a two-way flow of traffic that may be undesirable in many organizations where some information is needed exclusively for the organization or for the Intranet.
- A view may be defined as a 'stored query' or a 'virtual table'. It is called virtual table because, they do not exist as independent entities in the database as do 'real' tables. A view is a specification consisting of a SELECT statement that tests Oracle what records and attributes to retrieve, When view is called, the system associates the appropriate data with it. It retrieves derived data only when they are called.
- A materialized view is like a view in that it represents data that is contained in other database tables and views; yet it is unlike a view in that it contains the actual data. A materialized view is like an index in that the data it contains is derived from the data in database tables and views; yet unlike an index in that its data must be explicitly refreshed.

- Database means a place where data can be stored in a structured manner. It is a shared collection or batch of data that is logically related, along with their descriptions designed to meet the information requirements of an organization.
- A database system is a collection of application programs that interacts with the database along with DBMS (and sometimes the users who use the system). Database systems are designed in a manner that facilitates the management of huge bodies of information.
- The basic problem with serial schedules is of resource wastage.
- A conflict-serializable schedule is one which is conflict-equivalent to other serial schedule(s).
- Serializablity can be achieved by several different means. Locking and time-stamping protocols are two very common techniques.
- Locking and time-stamping protocols are two very common techniques. Locking is a more pessimistic approach to concurrency control while time-stamping is a more optimistic approach.
- In 2PL, a transaction requests an appropriate lock just before performing an operation. If a transaction requests a conflicting lock, it is blocked, awaiting the release of the conflicting lock. Otherwise, the appropriate lock is granted to the transaction.
- Timestamp is a protocol that can be used to generate serial schedules. The general idea is to order the transactions by time, and to use this ordering to give priority to read and writes in earlier transactions. This will result in a serializable schedule that is equivalent to the serial schedule of executing each transaction in the order in which it arrived.

4.13 KEY TERMS

- **Database security:** Policies taken to protect data from being accessed by unauthorized users.
- **Explicit lock requests:** Requests issued if a different kind of lock is required during an operation.
- **Optimistic locking:** Locking wherein data is locked only when it is being saved.
- **Strict timestamp ordering:** A variation of basic timestamp ordering which ensures that the schedules are both strict and serializable.
- **Trojan horse:** It appears to be benign in the beginning, but then infects unexpectedly.
- **Granularity:** The degree of detail or precision contained in data. Granularity has several dimensions, including time granularity and space, such as the size of the geographical clusters into which customers may be classified.

- **Database system:** A collection of application programs that interacts with the database along with DBMS (and sometimes the users who use the system).
- **Locking:** The most popular concurrency control mechanism implemented by many important techniques.
- **Lock:** A variable associated with every data item to indicate whether the data item is available for an operation to be performed on it.

4.14 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is data security risks?
2. What is database audit?
3. Define the term security threats.
4. What are macro viruses?
5. Write the uses of views.
6. State the advantages of views.
7. How are views deleted?
8. Mention the types of access.
9. Write the operations of database management.
10. Why is concurrency control needed?
11. Name the three basic concurrency problems encountered in a DBMS.

Long-Answer Questions

1. Explain the database security giving appropriate examples.
2. Describe the different security threats in database.
3. Discuss firewalls, and its architecture along with appropriate diagram of the network layers.
4. Explain the data encryption and its types by giving appropriate example.
5. Explain the views and how are views different from base tables.
6. How are views updated? Explain the restrictions in updating views.
7. Discuss the purpose of GROUP BY clause in the SELECT statement.
8. Explain the data granularity by giving appropriate example.
9. Describe the features of database operating system with the help of example.
10. Discuss the theory of serializability by giving appropriate examples.
11. Describe the different types of locks—binary, shared and exclusive.
12. Explain the timestamp-based concurrency control protocol.
13. Describe the security policies of a database.

4.15 FURTHER READING

Navathe, S.B. and R. Elmasri. 1997. *Database System Concepts*, Third edition. New York: McGraw-Hill.

Korth, Henry F., Avi Silberschatz and S. Sudarshan. 2010. *Database System Concepts*, 6th edition. New York: McGraw-Hill.

Elmasri, Ramez and Shamkant B. Navathe. 2007. *Fundamentals of Database Systems*, 5th edition. New Jersey: Pearson Education.

Martin, James. 2007. *Principles of Data Base Management*. New Jersey:

Pearson Education.

Martin, James. 1975. *Computer Data-Base Organization*. New Jersey: Prentice Hall.

Jackson, Glenn A. 1988. *Relational Database Design With Microcomputer Applications***.** New Jersey: Prentice Hall.

Date, C.J. 2000. *An Introduction to Database System*, Seventh edition. Boston: Addison Wesley.

UNIT 5 MULTIMEDIA AND DEDUCTIVE DATABASE

NOTES

Structure

5.0 INTRODUCTION

Multimedia Database Management System (MMDBMS) supports all multimedia data types. In addition, MMDBMS also supports traditional DBMS features, such as data modeling, retrieval and organisation. Today, the usage of multimedia content is increasing tremendously. Various applications of multimedia fall into fields that include digital libraries, manufacturing and retailing, journalism, art and entertainment, to name a few. All these multimedia data types are large so their storage, retrieval and transmission are different from general data types such as integers and real numbers. So, multimedia data structures must support content-based search to retrieve multimedia data types easily and quickly. Multimedia data, such as text, audios, images, and videos are rapidly gaining popularity as important forms of creating, exchanging, and storing information.

DDBs (Deductive Databases) is a field that combines databases, artificial intelligence, and logic. A DDB is a database system with the ability to establish deductive rules, which allow us to derive or infer additional information from database facts. Mathematical, logic is the theoretical foundation for DDBs. Other types of systems (for example, expert database systems or knowledge-based systems) include reasoning and inference capabilities as well as common feature is usage of artificial intelligence techniques (production systems).

In this unit, you will study about the multimedia database, impact of IT on libraries, applying database to the internet, deductive database system, deductive database notations, and interpretation of rules, inference mechanism, and deductive object oriented database.

5.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basics of multimedia database
- Discuss the impact of IT on Libraries
- Analyse the database to the internet
- Define the deductive database system
- Explain the concept of deductive database notations
- Describe about the interpretation of rules
- Define the inference mechanism
- Discuss the deductive object oriented database

5.2 MULTIMEDIA DATABASE

Multimedia Database Management System (MMDBMS) supports all multimedia data types. In addition, MMDBMS also supports traditional DBMS features such as data modeling, retrieval and organisation. Today, the usage of multimedia content is increasing tremendously. Various applications of multimedia fall into fields that include digital libraries, manufacturing and retailing, journalism, art and entertainment, to name a few. All these multimedia data types are large so their storage, retrieval and transmission are different from general data types such as integers and real numbers. So, multimedia data structures must support content-based search to retrieve multimedia data types easily and quickly. These types of multimedia objects are handled by using multimedia databases. Different data types that form part of multimedia databases include:

- Text (arbitrary long text fields)
- Still images
- Graphic objects like CAD/CAM objects
- Video data
- Audio data

Concepts of Multimedia Databases

Today, multimedia data such as text, audios, images, and videos are rapidly gaining popularity as important forms of creating, exchanging, and storing information. This rapid need can be noticed in the following three fields:

1. **VLSI technology** which is responsible for increasing the processing power of computers.

2. **Broadband networks,** such as ISDN and ATM, which are responsible for providing high bandwidth for many practical applications including online banking and online counseling.
3. **Multimedia compression standards** such as JPEG, MP3, and MPEG which are responsible for providing efficient storage and communication.

You can get an idea about the popularity of this trend-setting usage of multimedia from the statistics given by the following two major search engines.

- **Google:** 'In December 2001, the search engine offers access to over 3 billion web documents and its Image search comprises more that 330 million images.'
- **AltaVista:** 'AltaVista has been serving around 25 million search queries per day in more than 25 languages, with its multimedia search featuring over 45 million images, videos and audio clips.'

These two statements prove that there is an explosive growth in the request for multimedia data. This explosive requirement for multimedia poses a number of challenges related to its storage and retrieval. Some of the issues, such as transactional updates, querying facilities and indexing, cause a major problem. Thus, a multimedia database must address the following concerns:

1. **Support for large objects:** An MMDB must provide storage for large objects because multimedia data occupy up to a few gigabytes. A database can store a large object either by dividing the object into smaller fragments and then store it in the database or store the whole object outside the database as done in a file system. The latter method would require a pointer to the object.
2. **Guarantee steady data delivery rate:** An MMDB must provide steady delivery of all multimedia data. If multimedia is delivered faster or slower than the required rate then there are chances of loosing data. For example, suppose you are downloading an audio of your favorite song. If the rate of download of the song is low, then you will hear the song intermittently. In case, the download rate is high, then the buffer could overflow and there are chances of loosing some data. The data that downloads at a steady-rate is known as isochronous data or continuous-media data.
3. **Support similarity-based data retrieval:** An MMDB must support similarity-based data retrieval. This means that it must retrieve from a database all the multimedia data that looks similar. Suppose you have submitted a query to retrieve a fingerprint image. Then the MMDB must display all similar fingerprints available in the database.

Addressing Multimedia Databases Concerns

To address all the three concerns discussed in the previous section, you must know about some aspects such as data formats and requirements of continuous-media data.

5.2.1 Multimedia Database Data Formats

As MMDB stores multimedia data of a large number of bytes; hence, it is essential for an MMDB to store and transmit multimedia data in the compressed from.

NOTES

There are various types of compression formats used by an MMDB to compress data. For example, to compress an image data the Joint Pictures Experts Group (JPEG) format is used. To compress video and audio the Moving Picture Experts Group (MPEG) format is used. These compression techniques enable an MMDB to store large objects.

5.2.2 Continuous Media Data

A continuous media data requires steady download of the media data. The most important continuous media data stored in an MMDB includes video and audio data. There are various requirements for storing and retrieving a continuous media data. The following are some of the requirements:

- Delivery rate of data must be sufficiently fast that no gaps in the audio or video occur.
- Delivery rate of data must not cause overflow of system buffers.
- Synchronisation among distinct data streams must be maintained. This must be adhered to when a video is playing in sync with an audio.

The concept of continuous media data can be understood with the help of the following example.

Video on Demand

In the video-on-demand application, a viewer requests for a particular video online and then the MDBS server searches for, locates and plays the requested video. Currently, the video-on-demand servers employ file systems because the existing database systems do not provide the kind of real-time response that the video on demand requires. The basic architecture of a video-on-demand system comprises the following:

(i) ***Video server:*** All the required videos are stored in the server in several disks. In case a server contains large amounts of video, then the server can use tertiary storage to store less frequently accessed video data.

(ii) ***Terminals:*** These are used by viewers to view the demanded video. These could be any of various terminals, for example, a personal computer and a television attached to a small, inexpensive computer known as set-top box.

(iii) ***Network:*** The transmission media used to send a multimedia data from a server to various terminals is known as a network.

5.2.3 Similarity-Based Retrieval

An MDBS must support similarity-based retrieval which means the database should retrieve all images that look similar when a query for multimedia data is submitted. As discussed in the fingerprint example, there are few more such examples for other multimedia data types which are as follows:

(i) ***Example for audio data:*** An example of similarity-based retrieval for audio data is the voice recognition technique. In the voice recognition technique, a user can give a command or identify a data item by speaking.

This technique is used by various multimedia phones. These phones dial the number of a person whose name a caller speaks out.

(ii) ***Handwritten data:*** The similarity-based retrieval technique for handwritten data enables a user to identify handwritten data items or commands stored in a database.

(iii) ***Pictorial data:*** The similarity-based retrieval technique for pictorial data retrieves from an MMDB all pictures that look similar.

Components of Multimedia Databases

An MMDB should be a blend of functionalities and features of traditional databases and a framework for the storage, retrieval, transmission and presentation of multimedia data types. An MMDB must contain the following two components to describe all the different types of multimedia data types stored:

1. **Media data:** This represents the actual data that can include images, audios and videos that are captured, digitized, processed and stored.
2. **Meta data:** This consists of information related to different aspects such as media format, media features and media keywords of multimedia data. The information stored in meta data can be divided into the following three categories:

   (i) ***Media format data*****:** This comprises information regarding the format of the media data after it goes though three phases of acquisition, processing and encoding. It consists of information such as the sampling rate, resolution, frame rate and encoding scheme.

   (ii) ***Media keyword data:*** This consists of information regarding the keywords of a data type. The keywords for multimedia data type are specific and are different for different multimedia data types. For example, for a video, the media keyword data could contain information about the keywords such as date, time and place of recording and person who recorded.

   (iii) ***Media feature data:*** This contains information regarding the features of a multimedia type. This set of information totally depends on the features of the multimedia data type and consists of information pertaining to color distribution, texture used and shapes of images.

How is multimedia data type different?

Multimedia data type is different from other data types in that:

- It is bigger in size.
- It has different data capture methods.
- Has crucial time constraints like for streaming.
- Follows different querying methods to retrieve data.

To retrieve data, a multimedia database system can employ conventional data or information retrieval methods such as logging and indexing, graph or tree pattern matching methods and content-based retrieval methods. You have learned about indexing and tree pattern matching methods in earlier units. You will learn about content-based retrieval methods in Section 7.4.

Multimedia Database Management System Architecture

The software that manages an MMDB is known as MMDBMS. MMDBMS architecture enables a database to store, access and manage multimedia data and carryout different types of transactions. MMDBMS must have:

- Minimal response time
- Reliability and availability
- Ability to sustain a guaranteed number of streams
- Real-time delivery of data
- Exploit user access patterns

The architecture of an MMDBS is complex and can follow any of these three systems Centralised DBMS, Client–Server DBMS and Distributed DBMS. All these systems have a three-layered architecture. The three layers—external level, conceptual level and internal level are shown in Figure 5.1.

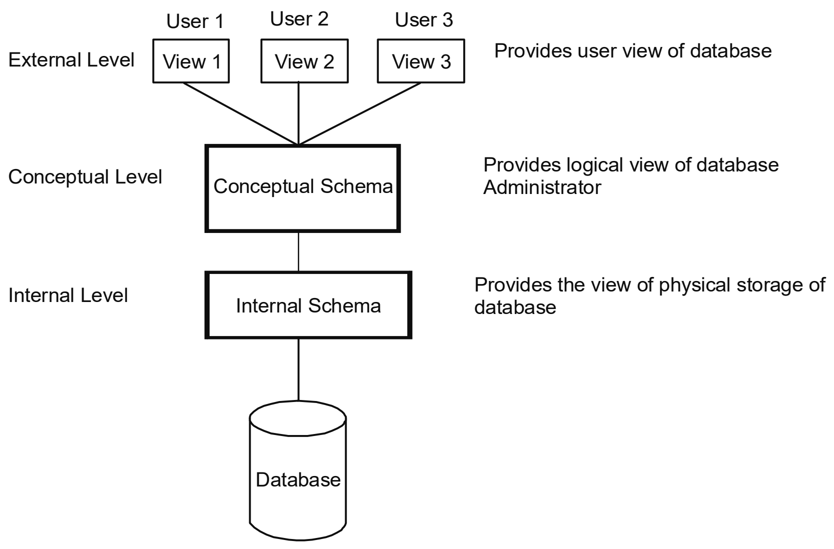

User 1
User 2
User 3
External Level
View 1
View 2
View 3
Provides user view of database
Conceptual Level
Conceptual Schema
Provides logical view of database Administrator
Internal Level
Internal Schema
Provides the view of physical storage of database
Database

Fig. 5.1 *Three-Layered DBMS Architecture*

MMDBMS architecture must:

- Adhere to ACID properties, which means that it must possess:
  - o ***Atomicity:*** Update either all the information or nothing for a transaction.
  - o ***Consistency:*** Transform a database in one consistent state into another consistent sate.
  - o ***Independence:*** Execute all the transactions independently.
  - o ***Durability:*** Permanently record the effects of committed transactions in the database.

- Adhere to server requirements: Server requirements include:
  - o Support for large-scale applications
  - o Adapting to user access behavior
  - o Consider the bandwidth
  - o Provide storage for complex multimedia formats

Storage Hierarchy in MMDBMS Architecture

To understand the concept of storage hierarchy in MMDBMS architecture, consider the example of a Video-on-Demand application. In this application, a user submits an online request for a particular video. While processing and fulfilling the user demand, the challenge for the server is to retrieve and deliver the requested video in real-time. In addition, the user should be able to reverse or forward the video as per requirement.

While storing the videos, the most popular videos are stored in a media with the highest bandwidth so that many user requests can be fulfilled simultaneously. Figure 5.2 shows MMDBMS storage hierarchy.

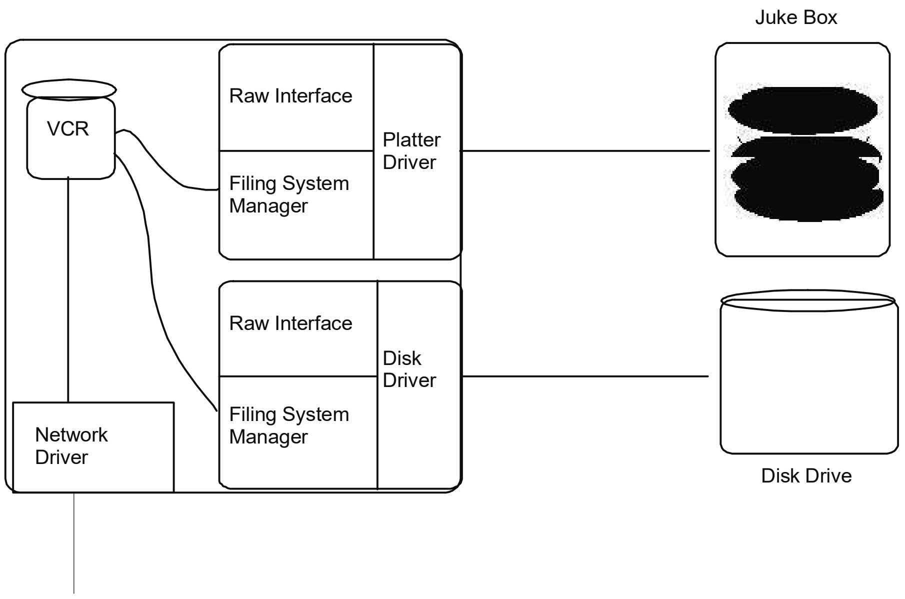


Fig. 5.2 *MMDBMS Storage Hierarchy*

5.3 IMPACT OF IT ON LIBRARIES

The library plays a vital role for any organization, be it school, college, university or any business organization. With the advancement in computer and telecommunication technologies in the past few years the availability of information has grown tremendously. These technologies have a great impact on traditional academic libraries with traditional information preservation, organization, provision, access and retrieval. The activities performed in libraries are no longer confined to the traditional ones. There is a profound impact of ICT on libraries and most of the libraries today are networked electronically and Internet (computer and telecommunication technologies) is playing an important role in providing information services and has added a great value to the library and information services. Thus, networking among libraries and information center has become inevitable. The prime objective of Internet based library and information services is pooling of information resources and their related infrastructure to make them sharable.

The transformation in library system has changed the view of the library resources and library services where web based library services are attending the users round the clock by providing links to various library sites, which are specialized in the topic of interest and can be accessed directly in every corner of the world. The use of ICT in library operations saves considerable amount of time, resources and labour. It brings quality of service and speed up the processing of information services. With the help of Internet, a student at any university or college in India can browse the electronic documents any where across the globe through computers, and hence, gets an instant access to billions of resources in the form of books, reports, videos, journals, films and variety of other resources.

The academic libraries in India have set up themselves to provide an ICT based information services platform. The Internet has proved to be a boon for both the libraries as well as the users. It has provided an easy to use and inexpensive teaching tool to the information seekers. The Internet has bridged the information gap between the libraries and information professionals by defining new and different service operations. Some of these are as follows:

- By creating a well organized, well published and easily accessible library web sites that has extended the use of information technology in traditional librarianship.
- By initiating a bulletin board of library citing complete information about the services provided and products available with them and the various events organized by them.
- By using e-mail services to deliver the information to the users and to communicate with the fellow information professionals.
- By providing access to the various database and OPAC of other libraries located at remote areas.

Scope of Internet based Library and Information Services

With the involvement of Internet in library activities, resource sharing and cooperative functioning has also become vital which has eliminated the barrier of distance and size among the users. It has also made the acquisition related services such as ordering and purchase of information resources/ documents (books, journals and electronic publication) more speedy and simple. These activities can be carried out through e-mail. Also, most of the booksellers and publishers place their catalogues and leaflets of new publications on their websites which can be easily accessed through Internet. All the publishers of primary journals are providing their journals online. The Internet facilitates the library and information professionals to browse the various sites for all the current publications available with the price and allow them to place the order online. The communication regarding any query or discrepancy can be done through e-mail which saves time, reduces paper work and efforts.

Internet has also made it possible to prepare standard catalogues without much effort. Centralized and online public access cataloguing services are provided by Internet. Union cataloguing has also made it possible to avoid the duplications in holding to a greater extend. The library professionals are allowed to access the Internet resources for verification and downloading the bibliographic information from other organization. OPAC has also become a popular source of bibliographic information which can be accessed via Internet. It is useful to get information about the organization of knowledge by other institutions.

The circulation of in house documents is also become easy through Internet. The new books document can be placed in OPAC on the same day of acquisition itself after certain technical processing and the readers can browse and reserve the material from their homes or offices that too within seconds after the arrival of the material. Also, subscribed journals by the libraries are accessible from anywhere across the world, the users can get access to the electronic form of journal from their offices or departments without visiting the libraries.

Since the information is increasing day by day on Internet, the information is used by librarians for reference services by the librarians to answer the questions they are asked. These are known as ready reference collection. The availability of various primary and secondary sources of information online made it possible to provide short-range and long-range reference services through Internet. The Internet has proved as an alternative to the traditional face-to-face reference service where there is a provision of chat based e-mail service for virtual reference and web tools such as FAQ are provided on the libraries websites. Real-time reference service is also provided using Instant Messaging (IM) which is a type of virtual communication between two people. Some IMs such as Trillion, Library H3IP, Meebo are providing access to all e-mail IDs while logged into any such platform eliminating the need to login to different e-mail address.

The Internet has made possible the availability of major libraries online which are accessible directly from any part of the world. It has provided access to the catalogues of various libraries that are attached to universities and colleges and allow them to place a request for their users. To avoid financial crunches, libraries have agreed for resource sharing which is not at all possible without the Internet. Publishers are also providing their journals electronically that has facilitated libraries to subscribe a large collection of journals from different publishers that also support cross journal searching and extensive browsing. Online collection of publications enables the users to search and browse the articles directly from journals subscribed by their libraries. Resource sharing using Internet has remarkably reduced the cost of collections by the libraries.

Resource sharing is also done through Inter-Library Loan (IIL) and traditional IIL operations are time consuming and labour intensive. The Internet has facilitated libraries to share their resources through IIL effectively and efficiently. IIL through Internet offers the following benefits:

- Single solution to manage the activities of IIL.
- The paper work and record-keeping in browsing and lending a material can be effectively managed with reduced paper work.
- Easy to track the status of the request at all stages in IIL process.
- To integrate bibliographic information with online union catalogue using Internet.
- Request and messages through electronic transmission using Internet.

In a way, Internet has become a great help in almost all the activities of library operations. Access to all types of material has become easy and speedy with the help of electronic documents.

Services Available on Internet

There is a huge impact of Internet on the library services and has now become an integral part of LIS. Internet is able to reach the library services to the user's desktop. Some of the services that are available through Internet are as follows:

- **Online Information Retrieval:** One of the important roles of library is to provide access to the information to the society. To improve the learning activity and teaching, it is important to have access to the comprehensive and current information. Through Internet, libraries are able to access online information resources at some nominal fees. This has become the most utilized service of the libraries.
- **Free browsing:** The availability of large number of information on the Web, some libraries may provide the facility to access the same by providing free browsing to the users through Internet.
- **Broad Band Internet Center:** Libraries are providing interconnection through networking with other libraries and information center to provide access to e-resources. Internet is also used for e-mailing, accessing e-journals, database, web OPAC with this facility.
- **Library homepage for Information dissemination:** Libraries are able to provide regular display of information of the latest editions and other information related to academic, research through their websites. This service facilitates the researchers and other users to make effective use of their academic and research interests.
- **Dynamic library websites:** Libraries are developing their websites with a link to OPAC to familiarize users about the library activities and to allow users to access the library catalogue remotely through Internet. Users are allowed to perform some library functions such as renewal of books, access of the content page of materials and ask for a copy to be delivered at their home/work place. Websites also provide the link to other resources that are created by the library.
- **Bulletin board service:** Through this service people are provided an area for discussion called bulletin board by posting messages without

sending them to anyone's e-mail. The post is seen by everyone who enters into the area. In campuses, these bulletins are called forum. The latest information about daily news, job opportunities, fellowship and so forth can be posted on these boards. These bulletins are available via Internet to specific category of user discipline.

- **OPAC:** The Online Public Access Catalogue (OPAC) provides facilities to browse and locate information and is known as the gateways to the information in the libraries. The purpose of OPAC was to provide access to the housekeeping activities of the libraries and to provide direct access to the machine readable bibliographic records.

E-Books

The introduction and growth of e-books has changed the relationships between the libraries, publishers and distributors and also the relationship between the libraries and their users. There are several libraries who are adopting e-book collections on a very large scale. Armstrong and Lonsdale have defined the term e-book as 'any content that is recognizably book like, regardless of size, origin or composition, but excluding serial publications, made available electronically for reference or reading on any device that includes a screen'. Algenio and Thompson-Young have considered that these e-books can be accessed by multiple persons at a time. Moreover, a patron need not go to library in person to borrow a book and the e-books can be downloaded from the website and can be read later offline. Although, there can be a restriction of time limit for the usage of the e-books and may get terminated after a certain span of time. These e-books can be accessed apart from computers, from other handy devices such as laptops, smart phones, i-pads, notebooks and so on.

Also, it seems to believe that the cost of e-books is far less as compared to the cost of producing the copies of printed books. This is due to the reduced cost of printing and distribution by the publishers. Although, the basic tasks involved in producing e-books are same as that of print books such as acquisition, production, sales and marketing, delivery and finance of books. However, some publishers have stated in their blogs that expense, incurred to manufacturing and distribution is 12 per cent and does not reduce the total publishing cost greatly. In fact, the publishers have highlighted the three new costs associated with e-books. These are digitization that is, producing the books in different formats, quality assurance that is, digital distribution to different retailers or distributers with digital asset management system and different upload protocols. But the production cost of an additional copy of an e-book is lower as compared to the production cost of a printed copy.

Types of E-Books

There are different types of e-books that can be recognized. These are as follows:

- Issue or re-issue of a print book in e-format. These types are called e-hybrid book.
- A book created by e-revision or e-feedback and the formally-issued version of which is in the print format.

- E-books enhance text with audio, video or images.
- A book with text, images, audio and video, and are regularly updated as warranted. These are known as e-reference books. A series that combines the latest information is also considered in this type.
- An e-book is published in pdf format and is similar to the printed version. The chapters can be downloaded in this type.
- A web page having an e-book that are readable online in HTML format only.
- E-books in the form of an app where the text and other content are published in the form of an app with features such as sound, movement and other special features.
- An e-book that is available under communication common license.

Technology Involved

The development of e-books required technical aspects such as hardware, software and a screen for displaying content.

- **Hardware aspect:** There is a need of some portable e-book devices for reading purpose. The hardware may include desktops, laptops, and multipurpose devices or even dedicated reading devices. These devices are provided with a screen such as a monitor, LCD or a touch screen.
- **Software aspect:** Some software are required for e-book reading that support special functions such as search, colors/grayscale display, user defined text size, hyperlinks within the books, and so on. Some of the common examples of such e-book software reader are Adobe Acrobat, Microsoft Reader, Palm Reader, Franklin Reader, and so forth. Most of the software are available for free downloads and support several operating system.
- **E-book creator software:** Special software are needed for the creation of e-books and there are certain software tools available for the purpose such as Adobe Page Maker, Adobe Acrobat Capture, Adobe Frame Maker, Adobe Design, and so on.

E-Book Standardization

It is necessary to standardize the formats of e-books as there are far many variables such as operating system, executable software, memory space, and so on. that are associated with e-books. Attempts have been made to standardize the e-books. The Open eBook Forum (OEBF), which is an association of software and hardware companies, publishers, authors, users and other related organizations of e-books, has established some common specifications for e-book system, products and applications that are beneficial for the content creators, makers of reading systems and of course, the consumers that help them in adopting e-books and to increase the awareness and acceptance of the emerging e-publishing industry. They have attempted in providing common e-book format. The common formats of e-books are Adobe PDF, DAISY Digital Talking Books and Microsoft Reader and are equipped with Digital Rights Management (DRM) technology (an access technology to protect copyright material and also limits the usage of digital media and devices).

E-Book Licensing

The e-books are leased rather than purchased. There are three types of e-book leases as offered by the publishers and vendors:

- **Annual Access:** An annual fee for one year's access is paid by the libraries and the lease gets renewed every year.
- **Permanent Access:** A one-time fee is paid by the libraries.
- **Pay per use:** Based on the number of uses (pages viewed, titles viewed and so on), the library is billed or a prepaid account is debited. This may be an annual fee platform also.

The e-book licensing can be modeled into three broad categories as: Print, database and open access licensing arrangements. The first model allows the access to an e-book with one user at a time. The restriction has been implemented on printing, copying, saving and sharing of e-books on the reading devices by DRM. This restriction of viewing and printing limits has affected the access of e-books. The second model, database model, has been developed to overcome the DRM restriction by enabling simultaneous access to e-book contents. The third model, Open Access (OA) has been developed to allow the access of the content of e-book freely with few restrictions.

Distribution of E-Books

The e-books are sold either directly to the consumer or through different suppliers that includes retailers such as Amazon, Barnes and Noble and Apple and through different suppliers such as Ingram. Earlier till 2010, the *wholesale model* was used by the publishers to sell e-books where a retail price was fixed by publisher and used to sell the book to the intermediary at a heavy discount, which was usually 50 per cent. The retail prices of e-books were usually fixed to the lowest price incurred for the print book. After the availability of paperback edition the publishers would lower the price of e-book. The trend is still follow by some small publishers.

In 2010, an *agency model* was introduced for the sale of e-books where the publishers made deals with Apple, who was introducing its new Apple Tablet. The agency which would usually be a retailer would get a commission from the publishers in doing so. This method gained a greater degree of control over e-book pricing as compared to the pricing over print book and the publishers were gaining permanent edge over the e-book margins.

Advantages of E-Books

There are many advantages of using e-books. They are discussed as follows:

- One of the most important advantages of e-books is that there is a provision to store thousands of books into your electronic device without any worries of managing them on the multiple bookshelves taking a huge space in your house. Portability is also associated with e-books.

- The accessibility and availability of the e-books are more convenient and speedy. Moreover, there is a possibility to customize the display brightness, font size and style, links and annotations. There is a provision of seamless integration of multimedia in e-books.
- The accessibility of e-books from remote place make it possible to purchase single copy and make it available in multiple locations at any time round the clock.
- The instant delivery is another feature of e-books with no loses or damage to the titles.
- The process of publishing an e-book is also quicker as compared to print book. The content can be changed easily at any point of time and new edition can be distributed through Internet instantly. The cost of printing, inventory, binding and so on. can also be eliminated and the storage space in the warehouses can also be saved.
- The material can be accessed equally by on-campus students as well as by distance learning students.
- E-books can be updated on daily, weekly or monthly basis providing latest information on current affairs.

Disadvantages of E-Books

There are some cognitive disadvantages of e-books. They are as follows:

- Researchers have studied and found that reading text onscreen takes 20-30 per cent more time as compared to reading on paper.
- Researchers have also found increased workload reading an e-book which is a combination of factors such as feeling of exhaustion, increased stress and concluded that more mental efforts are required for reading on screen than paper.
- Also, print readers digest information more quickly and more deeply and remember things by associating with texts as compared to screen reading.
- Some e-books are costly for downloading.
- Selection, acquisition and management of e-books can become complex and expensive if necessary procedures are not made a routine.

Barriers of E-Books

There are certain barriers in e-books which hinder the users to opt for e-books.

- User's reliability on printed text is more, and hence, neglects the benefits of e-books.

- The policies for purchase/subscription of e-books are different form print book and are difficult to understand.
- The cost of e-books from foreign publishers is high for Indian readers.
- Internet access also contributes to the barrier in India.
- Suspect for online products.
- Lack of awareness in information literacy program for e-books.
- Non-standardization of hardware/software required for various types of e-books.
- Lack of common platform for e-books.
- Building of a strong e-book collection is difficult as many of the titles are not available in e-format.
- Licensing issues may affect the availability of e-books across the countries.

E-Journals

The term journal as described by the Encyclopedia Dictionary of Library and Information Science is 'the record of proceeding of transactions of a learned society'. 'It is a publication in any medium issued in successive parts bearing numerical or chronological designations and intended to be continued indefinitely' – as defined by AACR2. It is considered as a information shell in a subject. Journals have important role in information management for information creation and dissemination. It consists of research papers, review articles and scholarly communication.

With the emergence of ICT in the recent years, there are ways of providing information to the society at lower cost with easy and reliable mechanism. Publishing industry has brought about the development where information is also created and offered digitally and is referred to as electronic documents. One such sophisticated form of this type of information is e-journals. The term e-journal or electronic journal denotes a broader category of e-publication that include journals, e-zine, web-zine, magazines, newsletter or any type of electronic serial publication and is available on Internet which is accessible through different technologies like www, ftp, gopher, telnet or e-mail.

Although there is no universally accepted definition of e-journals, some researchers have defined e-journals as:

'Any serials produced, published and distributed nationally and internationally via electronic networks such as Binet and Internet' – As defined by **GailMacmillan.**

'E-journals are available electronically via a computer or a computer network, that they may or may not be published in some other physical medium, but that are not CD-ROM's' – As defined by **Jones.**

The above definitions state that e-journals are periodicals which can be made available over Internet as individual titles. The term e-journal is interchangeably used with paperless journal or virtual journal. E-journals can be accessed online by more than one user simultaneously with timely access. Similar to other electronic documents, e-journals save physical storage.

Characteristics of E-Journals

The few characteristics of e-journals are as follows:

- Serial publication that is available in digital format.
- Located on Web and accesses over World Wide Web (WWW).
- Can either be available for free or by subscription.
- Available formats for e-journals are: ASCII text, HTML pages or PDF (Portable Document Format).
- Accessibility from any place.
- Available in downloadable format.

Types of E-Journals

- Also called classic e-journals, these were originally distributed through e-mail and are now available on Internet. These journals can be accessed free of cost.
- The journals are available in print as well as electronic form. There is sometimes a difference in the content of the two versions in a way that either electronic version contains some supplement issue or the electronic version is released before printed. The electronic version may be a collection of full text or it may consist of only table of content or some selected article from the printed version.
- Some e-journals are called database and software models where the articles are stored in publisher maintained centralized database and the access permission is given to the subscriber to locate the database and download the article. There is a expiration date associated with the software.
- Some commercial publishers have full text of the journals available on CD-ROMs. Libraries are often needed to get subscribed for both the CD-ROM and print form to such journals.
- There are full text e-journals where the complete articles are available online instead of summaries and abstract.
- Some e-journals are available only online with no counterparts such as CD-ROMs or printed version.

Access to E-Journals

The e-journals can be accessed mainly through Internet, although there is mechanism to access e-journals through CD-ROMs as well. The following are the different ways to access e-journals provided by the publishers:

- *Free Access* to the e-journal with the subscription of print journal.
- Libraries can completely access all the e-journals by *exclusive subscription* for electronic form without having to subscribe to the counterpart print version.
- *Selective Access* to the few chosen e-journals by subscribing to them from the publishers as per agreed terms and conditions.

- *Consortium Access* by forming a consortium of institutions with common requirements and interests. This provides the access to the expensive and international e-journals which otherwise is not possible by many libraries to afford in India.

There are also three types of access modes available for e-journals. They are as follows:

- **Remote Access:** This type of model allows publishers/vendors to host the journals through their websites and provide the right of access to the patrons including individuals or institutions who are subscribed with the publishers for the e-journals. The rights to access can be provided through user-id-password, IP enabled intranet or both.
- **On-Site Access:** The e-journals are delivered by the publishers through CD-ROMs or by their website, or through FTP option to the subscribed libraries. The e-journal in turn is hosted by the library within the campus. This way, library can host the journal through LAN with wider and better bandwidth within the campus as compared to the access through Internet.
- **Access through Database:** The publishers are creating the content in the electronic format and maintain a bibliographic database over the years. The users can access the articles of their interests through these databases.

Creation of E-Journals

The basic steps involved in creating e-journals are as follows:

- **Planning:** The first step is planning. In this case, the hierarchical structure, navigation and logical composition of e-journal is planned.
- **Content Creation:** Content of any literature is an essential part and has to be created carefully.
- **Realization:** It involves the designing of the structure of HTML, deciding links between pages, sending to the server and testing.

Advantages of E-Journals

E-journals are published in an electronic format and have various advantages over printed journals. They facilitate a new relationship between information and knowledge and offer new form of scholarly practice. They offer many advantages to users as well as publishers. The following are some advantages of e-journals:

- The e-journals offer the most attractive features of navigation and searching and provide better retrieval capabilities as compared to paper format. The article can be retrieved through any word of the article.
- Instead of hours and days, it takes some minutes to access e-journals than printed journals as it reduces the printing and mailing time.
- They are readily available at your desktop and can be read by more than one person at a time.
- They can be made attractive by including multimedia and graphics including audio-visual materials.

- They provide the user with the facility to link directly to references cited in the articles through the creation of hyperlink both internally and to other publications.
- Since there is no space restrictions, the publishers can print any number of articles in e-journals and the length of the articles have no restrictions.
- E-journals provide tremendous searching capabilities based on titles, authors, keywords, subjects, full text, abstract and so on. The publishers allow viewing the abstract of an article to decide the worth of an article.
- They are available 24x7 and can be accesses remotely by the users. It reduces the efforts of patrons to visit the library to obtain a copy of the article. It also omits the geographical barrier for the user.

Disadvantages of E-Journals

Following are some disadvantages associated with e-journals:

- The e-journals involve reading from the device screen and long reading may cause eyestrain.
- There is an additional cost investment initially as special equipment (electronic devices) such as computer or other handheld devices are required to read the journals.
- Also, access to e-journals need electricity, telephone lines, Internet, appropriate hardware and software which might be a problem to people living in India.
- Since the technology is not far distant, there is a small percentage of electronic articles available and so it might be possible to have no access to some of the e-journals.
- Maintenance of e-journals requires more facilities and an expert and trained staff.
- Threat of duplication as it is very easy to make copies of e-journals.

Adversities of E-Journals

There are certain issues and apprehensions which are associated with e-journals in spite of so many advantages of e-journals. They are as follows:

- The peer review of the articles in e-journals for the authenticity and quality of the information being reviewed.
- The comprehensiveness of e-journal.
- The easy downloading of e-journals.
- The limit to the number of views and the flow of information with heavy traffic.
- Online help for users.

Examples of E-Journals on Web

There are thousands of e-journals available on web. They are presented in many languages and are of different themes and interests.

Check Your Progress

1. List the various applications of media types.
2. What is a multimedia database?
3. Define the term e-books.

NOTES

5.4 APPLYING AND SERVING DATABASE ON THE INTERNET

The use of the Internet is considered as a prime tool to decide the role and scope of database market. In fact, the impact of the Internet on database market has generated a large area for new database strategies in which two prime tools, such as Object-oriented Data Model (OODM) and Extended Relational Data Model-Object/Relational Database Model (ERDM-O/RDM) are used as a platform for Internet- based database development. Various database vendors focus on OODM and ERDM-O/RDM data modeling to create and access an effective Internet interfacing to provide suitable tools for data mining and warehousing. With the dominance of World Wide Web (WWW) there is growing need of managing unstructured data as well as databases too. The unstructured data refers to data found in web pages. Let us take an example of extended relational databases. Oracle 10g and IBM's DB2 support XML data types to store and manage database across the NET. This server is a set of components that work together to meet the data storage and analysis needs of the largest web sites and enterprise data processing systems. SQL Server 2005 is available in 32-bit and 64-bit editions. The 64-bit editions of SQL Server 2005 are installed using the same methods as the 32-bit editions. The role of search engine is very important in archiving databases on the Internet. Most search engines are aware of this and Google recently started the pre-empting task followed by user that suggests the combination of keyword for fetching the best result. The fundamental concept behind search engine is that how best to divine the user's intent and provide the exact result which the user requires. Google search is based on the mechanism of algorithm. No matter, it is complex mathematical formula that crawls via unlimited sites. These sites are on the Net and shown by the ranks according to pages which are relevant and well informed information. Google search engine blends listings from the news, well informed documents and images, local book search engines that deliver the exact and comprehensive results. Bing, Microsoft's search portal, was launched in 2009. It works as 'decision engine'. One more example of search engine is 'Wolfram Alpha', which answers rather than each page which contains the keyword searched for. In the mean time, popular search engines contend with the rapid verticalization of the Web, especially in the field of tour-travel focuses and delivers the best possible results. The effects are visible in Search Engine Optimization (SEO). In this, the mechanism is used with the help of Web masters to ensure that the websites get higher ranking in the searching the information. Sometimes, Web masters spam their code with keywords that impacts on getting recognition from the algorithm of social engines. Web master is integrated with artificial intelligence. The searching domain becomes more user oriented in the network era. Figure 5.3 shows the accessing flow of database by clients. The application User Interface (UI) is switched to data switch with server as well as local layer too. The Local Data Layer (LDL) archives the required data via search engine. Then the requested data is sent to the Internet. In these days, switching to various databases and Internet is being maintained by Oracle Internet Directory (OID). An OID is an LDAP server which uses an Oracle database as a data store or container.

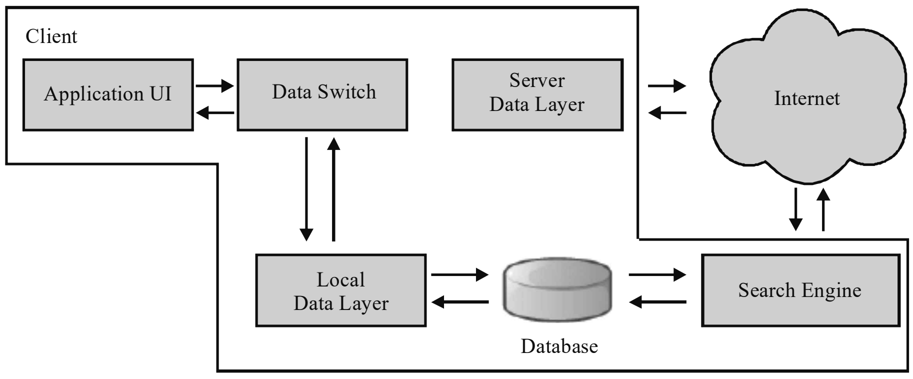


Fig. 5.3 *Internet Connectivity of Databases*

Search engine improves the user's browsing experience. Figure 5.3 explains the process of Internet connectivity of databases. All users who use Google search engine can get quick and fast information. Database servers that are built to serve Internet applications are designed to handle millions of concurrent connections and complex SQL queries. There are many open source programming languages, such as PHP, Python, Ruby and powerful open source libraries, for example, tools and plug-ins specifically built to interact with latest database servers too. Following factors are supported by the Internet to correspond with database and hence required to:

- **Explain the importance of attaching a database to a web page:** Web pages are considered as documents on WWW. The Web page is recognized by unique Uniform Resource Locator (URL). If you want to search the information as 'computer graphics basics' in Google search, then you have to type the text that has to be searched. After pressing the [ENTER] key or clicking on 'Google Search' button you can get the various Web links that refers basically to the web pages. You can get the information after clicking the specific web page link. You can also create web page for your website. For this, you need to have FrontPage, Macromedia Dreamweaver that are useful to design an effective web page. The database attached to a web page requires updates and feeding information for synchronizing the data. It also handles various online products, news feed reports, and members based subscription and other functions as well. It not only supports accessing information but also automating most of the daily tasks. For example, using a library mass requires efforts in compiling and updating. If you have a database to play around you can connect your book online. Web pages created with database data can be either static or dynamic. Statically created database-based web pages are made by converting the results of a query or a database report into HTML. With dynamically created database based web pages, each page is linked to the database and created automatically by a query entered by the user.

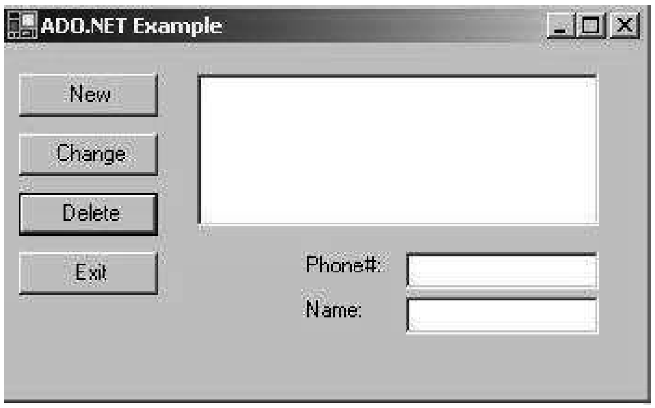
- **Explain the purpose of WWW consortium and Web standards:** The Web standards term is used for the formal standards and other technical specifications that define and describe the various aspects of the WWW. Various standardized specifications are set with Internet markup languages, such as HTML, CSS, XML, XHTML, XSLT, that works in almost any browser or Internet-enabled devices instead of being specific to certain versions of the Internet. The coding written in these specifications are executed and shown on the web pages with the help of Web browsers. The Internet Explorer (IE) is a frequently used browser by the users. IE is a program that Microsoft has been running with Windows since 1995. HTML5 and a new JavaScript engine called 'Chakra' is launched which according to Microsoft makes new browsers the fastest. HTML5 add with Web standards as a boon particularly for streaming video playback without plug-ins. The XHTML conform to the Web standards. Tags like <HTML> are not valid in XHTML.
- **Describe security with Web services:** The concept of security features forms the basis and foundation of Internet technology. Authentication and authorization are the two important factors to decide security features. In authentication process, the identity of user is determined. It confirms that the user is the same person who has already registered. The two prime credentials, such as password and user name are required to log in the specific web page. The credentials are first authenticated by the list of users which is maintained by admin side in a backend database server. When the user is authenticated by the name and password the process goes to the authorization approach. In this approach, the user is able to click on the events that are performed, as per actions. Events are items that transpire based on an action, for example clicking the button on the form. Actions involve the operations of viewing files and support the mechanism to retrieve information from the specific database. Web services contains Web methods and returns a dataset after querying a database given the connection string, password, database, table, user ID and the SQL commands. It also contains methods that receive a dataset object and uses them to update the database. Each of the datasets either returned by some methods or passed in as a parameter to other methods to update the database are saved as XML. Each dataset is saved as XML has a file name which contains a digit which is incremented, i.e., the index variable, each time therefore giving each dataset saved a unique filename. In this way, data security is maintained through Web services.
- **Describe Web server interfaces:** Programming languages are used for the Internet or for creating a website using the server and client-side scripting. This document is created by the server whenever the browser requests the document. When a request arrives, the Web server runs an

application program or a script that creates the dynamic document. The server returns the output of the program or script in response to the browser that requested the document. A fresh document is created for each request and the contents of the dynamic document can vary from one request to another, for example, retrieval of date and time from a server. CGI is considered as the ways in which dynamic documents can be created and connected to database server. This interface is created to handle dynamic documents. It follows a set of instructions that tell us how dynamic data is created to get the input for program and produce result. A CGI script is a server-side program that is launched by a Web server to generate a dynamic document. It receives encoded information from the remote client user's browser via STDIN and environment variables, and it must produce a valid HTTP header and body on STDOUT. CGI is a protocol, a formal agreement between a Web server and a separate program. The server encodes the client's form input data, and the CGI program decodes the form and generates output. The protocol supports the programs and scripts written in C, Shell, Rexx, C++, VMS DCL, Smalltalk, TCL, Python and Perl. Perl language is also known as CGI and Python, UNIX shell script, AppleScript, Visual Basic, C, C++ etc. This interface allows reading the standard input and writing the standard output for the Web programming part. This interface reads the data, processes the data and sends output as an HTTP response header to generate the document. It sends a blank line to separate the HTTP header and reads data via GET and POST methods. One difference between GET and POST is that in the GET method, the searched information appears with the URL whereas POST hides the searched query. The Web surfer fills out a form and clicks on 'Submit'. The information in the form is then sent over the Internet to the Web server. The Web server 'grabs' the information from the form and passes it over to the CGI software. The CGI software then executes the prepared database statement, which is then passed to the database driver. The database driver sends the information from the database to the CGI software. The CGI software takes the information from the database and manipulates it into the format that is desired. The CGI software then sends the result it wants to the Web surfer's browser which is then sent back to the Web server. The Web server sends the result it got from the CGI software back to the Web surfer's browser. If any static HTML pages needs to be created, the CGI program accesses the Web server computer's file system and reads, writes and edits the files.

- **Describe Web load balancing methods:** Load balancing is the process by which inbound Internet protocol traffic is distributed across multiple servers. Load balancing enhances the performance of the servers and leads to their optimal utilization. It is important for busy networks where it is difficult to predict the number of requests that will be issued to a server. It brings down the service time by allowing multiple servers to handle the requests. This service time is reduced by using a load balancer to identify which server has the appropriate availability to receive the traffic. Usually, two or more Web servers are employed in a load balancing scheme. In case one of the servers begins to get overloaded the requests are forwarded to another server. Load balancing brings down the service time by allowing multiple servers to handle the requests. This service time is reduced by using a load balancer to identify which server has the appropriate availability to receive the traffic.

- **Provide plug-ins services:** A plug-in is a small program which extends the capabilities of another program. It is a small application that is used to plug-in other programs. It makes the host program work differently. In case of Web browsers, plug-ins frequently allow the browser to play different types of multimedia or to run small Web-based programs. To view a list of currently-installed plug-ins in the Firefox Web Browser, type 'about: plug-in**s'** in the address bar and press Return. A plug-in is referred to as a player for extending the animations, graphics and music capabilities of the Web browser. If the Web page is encountered, you need a special plug-in for processing it smoothly. Plug-ins are small programs and are easy to download. The browser could be IE, Mozilla Firefox, Netscape Navigator, Safari or some other browser. You can download and install the plug-in by selecting the 'Temp files'. Inside that, you have to create a sub folder. Select the Browse button to download the programs. Authentication in DB2 is processed using security plug-ins. A security plug-in refers to dynamically loadable library which provides authentication security services. DB2 provides group retrieval plug-in which retrieves group membership information for a given user, client authentication plug-in which manages authentication on a DB2 client and server authentication plug-in which manages authentication on a DB2 server. DB2 supports two mechanisms for plug-in authentication and they are user name and password authentication with authentication plug-ins, such as CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT and DATA_ ENCRYPT _CMP.
- **Database connectivity and the Internet:** The database connectivity is required for Internet programming. It is maintained by admin side as backend services. The Active Data Object (ADO.NET) is an extended version database connectivity option which provides three tier supports. The first tier supports SQL server and Oracle, which supports OLEDB providers. Object Linking and Embedding database OLEDB providers manage the extensions. The unmanaged OLEDB providers supports second tier, whereas ODBC providers support third tier. The two prime companies, such as Oracle and Microsoft, optimize the providers after making a considerable effort. Microsoft manages the SQL Server 2000 based native OLEDB provider with the help of benchmark. The OLEDB benchmark is, in fact, upgrade service from SQL server 7.0 to SQL Server 2000. The prime vendors, for example Microsoft, Sun, Oracle frequently provide the benchmarks for database connectivity options. The ActiveX Data Object (ADO.NET) provides a group of libraries

and is helpful to create complex databases with the help of, Microsoft SQL Server, Oracle, XML, Microsoft Access, etc. You must know the concept of ADO.NET because of making the connection of databases with code. ADO.NET is supported by .NET framework and privileged by various classes for processing the requests and performing the transactions between users and database systems. The DataSet class handles the various operations while creating and managing the database design and manipulation. The Dataset class supports intermediary service between user interface and database engine. The user interface is known as Windows controls in which users interact with the system. To access database system via C# programs, you need to interact with ADO.NET based API. The .NET platform contains a good variety of classes and namespaces that have to be used to access the databases. Generally, software developer keeps the back end database as MySQL due to its flexible features. Therefore, you must have a good knowledge about Oracle, MS Access or even MYSQL. It is to be suggested that C# forms based applications are associated with ADO.NET interface. The basic operations, such as select, insert, change and delete rows or records which are based on query can be performed once you get connected with database. The following screen shows ADO.NET tab which provides the layout of 'Phone#:' and 'Name:' bars populating with various buttons to perform operations of add, change and delete the records in the specified database.



Making and establishing a connection with databases are possible if backend is set with SQL Server or system unit is installed with MySQL.

- **Explain the purpose of XML as a standard:** This stands for Extensible Markup Language. It was designed to transport and store data, with focus on what data is. In most Web applications, XML is used to transport data. It is the most common tool for data transmission between all types of applications, and is gaining popularity in the field of storing and describing information. Generally, Internet applications keep the following syntax for XML. For example, to check the bank account XML coding would be written as follows:

```
<Account type="checking"></Account>
```

```
And this coding used with reference to SOAP is as
   follows:
<soap:Envelope
Xmlns:soap=
http://schemas.xmlsoap.org/soap/envelope/>
<soap:Body>
-
-
-
</soap:Body>
</soap:Envelope>
```

- **To play the role of database server:** The database server is installed with MySQL packages and hence configured for Linux or Windows operating system as database server. The MySQL database software is treated as popular back end server. The advantages of database server are to manage the centralized data management, data independence and data integration. In database server all database files are arranged and integrated into one system thus making data management more efficient by providing centralized control on the operational data. By providing centralized control of data, DBMS brings a number of advantages including to reduce the redundancy that avoids inconsistencies, to share the data that gives better service to the end-users and use standards that can be enforced. Minimal data redundancy means improved data consistency that will improve data quality. Data independence can be defined as immunity of applications to change the physical representation and access technique. The provision of data independence is a major objective for database systems. If the data is well designed, the user can access different combinations of the same data for query and report purpose. Since related data is stored in one single database, enforcing data integrity is much easier. In distributed DBMS, the integrity rules can be enforced with minimum programming in the application programs. Related data can be shared across programs since the data is stored in a centralized manner. This provides improved data sharing, improved data accessibility and responsiveness. Even new applications can be developed to operate against the same data. The distributed DBMS should provide facilities for protecting the security and privacy of the data from unauthorised users. The distributed DBMS provides increased productivity of application development. There are various types of database servers in which RDBMS is used widely and popularly. Various types of database servers are flat file database servers, relational database servers, object database servers and object relational database servers. The application programmer need not build the functions for handling issues like concurrent access. The programmer only needs to implement the application business rules. This brings in application development ease. Adding additional functional modules is also easier than in file-based systems. Though data in database

NOTES

can be shared by many applications and systems, maintenance is less and easy, again, due to the centralized nature of the system. The benefits of database server are as follows:

- End-users can work with databases while the database server is running.
- Automated close is needed for real time database file which is used to minimize the network traffic.
- It controls the least resources for communicating with various databases within or even beyond domain.

The database servers that are built to serve desktop applications usually can handle only a limited number of connections and are not able to deal with complex SQL queries. *The Oracle Collaboration Suite (OCS) database is an Oracle10g Relational Database Management System (RDBMS). It refers to the repository for the OCS schema information. It uses Oracle Application Server Release 10.1.2.0.2 metadata repository. The default version is 10.1.0.5 if RDBMS is installed to the system. For this, Oracle10g database is created to install the Oracle Collaboration Suite component schema. Oracle Internet Directory (OID) is used to create a general purpose directory service to store the security and management information. It provides authentication and centralized database which creates and manages users on database Enterprise.* The following figure provides an overview of the Oracle Collaboration Suite architecture:

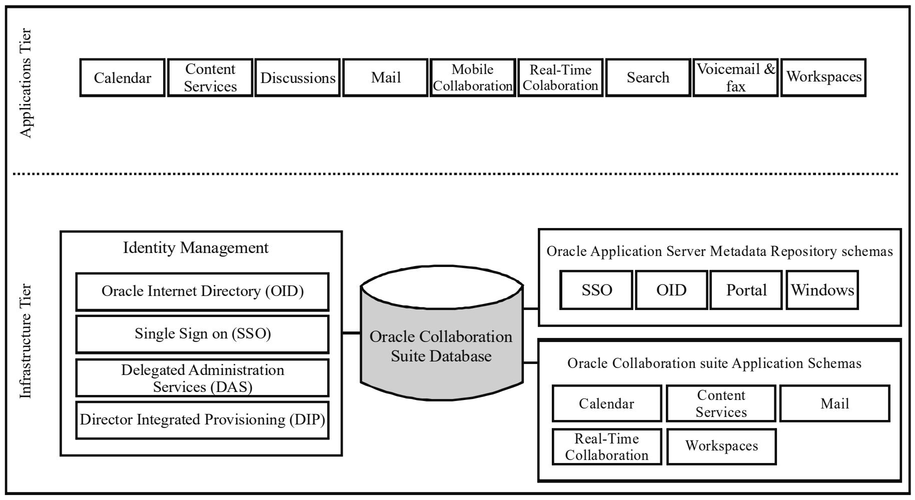


Fig. 5.4 *Oracle Collaboration Suite Architecture*

Figure 5.4 shows the Oracle Collaboration Suite architecture which is comprised of Infrastructure Tier and Applications Tier. The Infrastructure Tier contains the Oracle Collaboration Suite Database that includes Identity Management, the Oracle Application Server metadata repository schemas and the Oracle Collaboration Suite application schemas. Applications Tier contains all of the Oracle Collaboration Suite applications. The Infrastructure Tier consists of the components that provide services for the Applications tier, such as identity management and metadata storage. Infrastructure Tier includes Oracle Calendar, Oracle Content Services, Oracle Discussions, Oracle Mail, Oracle Mobile Collaboration, Oracle Real-Time Collaboration, Oracle Collaboration Suite Search, Oracle Voicemail and Fax and Oracle Workspaces. Users can access these services using a variety of methods including the Web, fax, voice, phones and Personal Digital Assistant (PDA) over wireless networks.

E-mail protocols use Simple Mail Transfer Protocol (SMTP), Post Office Protocol version 3 (POP3) and Internet Message Access Protocol, version 4 (IMAP4). SMTP is used to transmit e-mail across the Internet. When you use POP3, all e-mail messages are downloaded onto your computer from the server and can be subsequently accessed offline. Once the e-mail messages are downloaded they can only be accessed from the computer on which they are stored. IMAP4 protocol is used to access e-mails from the mail server. By using IMAP, you can retrieve e-mails as well as manipulate e-mails on the server itself. Because the e-mails are on the server, they can be accessed from any computer. File protocols use File Transfer Protocol (FTP) which is used for transferring files from one computer to another. FTP is the most widely used protocol to upload and download the files to and from the Internet. It also uses FTP Over SSL (FTPS). You can access Oracle Content Services using either implicit or explicit FTPS. Because FTPS does not send unencrypted passwords over the network; therefore, an FTP password is not necessary. It uses Hypertext Transfer Protocol (HTTP) to get Web browser based access. The file protocol also uses Web based Distributed Authoring and Versioning (WebDAV) which comprises a set of extensions to Hypertext Transfer Protocol (HTTP). This protocol defines a standard for all authoring operations, such as editing and managing files on a remote server. The Calendar services use SyncML/HTTP. SyncML is the standard language that is used to enable synchronization of remote data and personal information between various devices and networks. These services also use Oracle Calendar Access Protocol (OCAP) which is in fact the access protocol through which Oracle Calendar Application System (OCAS) sessions communicate with the Oracle Calendar (OCAL) server. It also communicates with desktop clients communicate with OCAL. Using wireless services, employees can access their e-mail and voicemail, manage their appointments, search the corporate directory, browse and fax shared online files from any mobile device with a browser, through speech from any telephone or instantly by SMS. Oracle 10*g* Application Server Portal enables companies to quickly build, administer, and deploy enterprise portals that are standards driven, scalable, secure and dynamic. Web Cache monitors requests from a client and stores information that it retrieves from the server. The Web cache delivers the content from its memory rather than passing on the request to the server, improving access time and efficiency and reducing traffic. Components involved in Oracle Collaboration Suite Identity Management are explained as follows:

- **Oracle Collaboration Suite Database (OCSD):** With the help of OID, an Oracle 10g, database instance processes and the database listener can be created.
- **Oracle Internet Directory (OID):** This Internet Directory is a general purpose directory service that stores security and management information for Oracle Application Server and Oracle Collaboration Suite instances, components and Infrastructure. It provides authentication and a centralized

user model that is used to create and manage the users on an enterprise scale. It also enables fast retrieval and centralized management of information about dispersed users and network resources. This directory is Lightweight Directory Access Protocol Version 3 (LDAP v3) and Oracle 10*g* Database technology. It communicates with the database using Oracle Net Services which is an operating system independent database connectivity solution offered by Oracle.

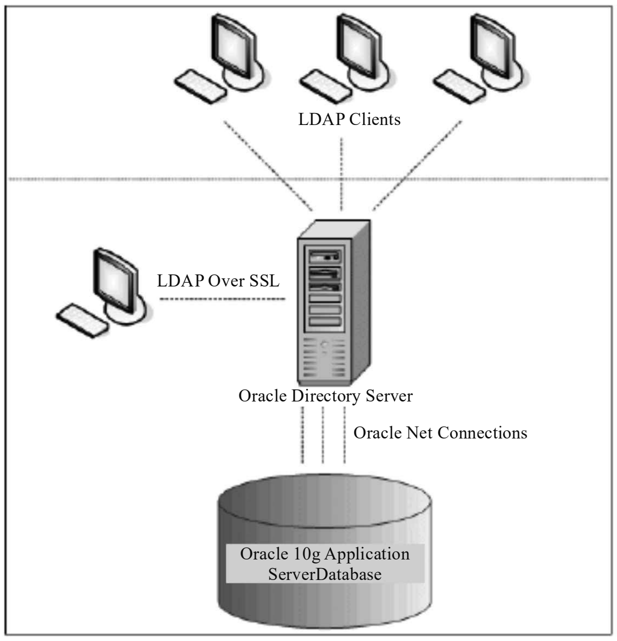


Fig. 5.5 *Oracle Internet Directory Architecture*

Figure 5.5 shows that various LDAP clients are linked with Oracle directory Server. LDAP over Secure Socket Layer (SSL) is also connected with this server. Oracle 10g application server database provides suitable platform for Internet connectivity. For this, Oracle net connections are required.

5.5 DEDUCTIVE DATABASE

DDBs (Deductive Databases) is a field that combines databases, artificial intelligence, and logic. A DDB is a database system with the ability to establish deductive rules, which allow us to derive or infer additional information from database facts. Mathematical, logic is the theoretical foundation for DDBs. Other types of systems (for example, expert database systems or knowledge-based systems) include reasoning and inference capabilities as well as common feature is usage of artificial intelligence techniques (production systems) difference: Expert database systems presume that the data needed is in main memory, whereas DDBs employ data from secondary storage.

5.5.1 Deductive Database Notations

A declarative language is used to express rules in a DDB (what, not how, e.g., SQL). By evaluating these two principles, a DDB's inference engine or deduction mechanism can derive new facts from the database. Model used for DDBs is closely related to:

1. Relational Data Model
2. Logic Programming, Prolog Language

Datalog is a declarative version of Prolog that is used to define rules. In the language, the current collection of relations is regarded as a set of literals.

Datalog syntax is similar to Prolog syntax

Datalog semantics[a] ≠ Prolog semantics

A DDB uses two main types of specifications: facts and rules-

1. Facts are specified in a similar way to relations, but without the need for attribute names. In a DDB, an attribute's meaning is defined by its position in a tuple.
2. Relational views are analogous to rules. They define virtual relationships that aren't actually recorded, but can be inferred from facts using inference methods based on rule descriptions.

   The primary distinction between rules and views is that rules can include recursion-

   Backward chaining incorporates a top-down examination of goals which is used to evaluate Prolog programs. Bottom-up assessment is roughly used to evaluate Datalog programs. Because the order of specification of facts and rules, as well as the order of literals, is important in Prolog, an attempt has been made to avoid these issues in Datalog evaluation.

Prolog/Datalog Notation

Principle: Give predicates names that are distinct from one another, Predicate:

a. Implicit meaning
b. Fixed number of arguments

The predicate asserts that a specific fact is true if all of the inputs are constant values. The predicate is treated as a query or as part of a rule or constraint if the arguments are variables. Here are the Prolog conventions that have been adopted:

1. In a predicate, constant values are either numeric or character strings, and they are represented by identifiers that begin with a lowercase letter only.
2. Only uppercase letters are used to begin variable names in this example based on the corporate database.
3. Supervise, superior, and subordinate are some of the predicate names here.

(a) **The supervise predicate:** Characterised by a set of facts, each of which is supported by two arguments: (name of supervisor, name of supervisee), supervise is a word that conveys the concept of direct supervision, the facts relate to the database's actual data, tuples in a relation with two characteristics and a schema correspond to facts:

   SUPERVISE (SUPERVISOR, SUPERVISEE)

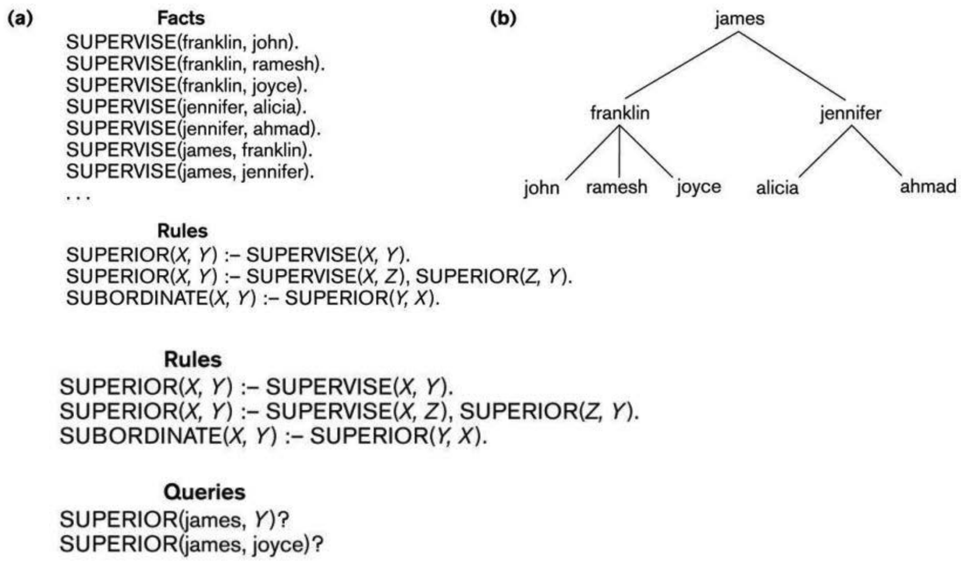
In facts, attribute names are absent; instead, attributes are represented by position. The supervisor is the first argument, while the supervisee is the second. Supervise (X,Y) indicates that X is in charge of Y.

NOTES

(b) **The superior, subordinate predicates:** Superior allows us to represent the idea of non-direct supervision through a set of rules.

MAIN IDEA OF DDBs

Specify rules and a framework to infer (deduce) new information based on these rules.



(a) **Facts**

```
SUPERVISE(franklin, john).
SUPERVISE(franklin, ramesh).
SUPERVISE(franklin, joyce).
SUPERVISE(jennifer, alicia).
SUPERVISE(jennifer, ahmad).
SUPERVISE(james, franklin).
SUPERVISE(james, jennifer).
. . .
```

Rules

```
SUPERIOR(X, Y) :– SUPERVISE(X, Y).
SUPERIOR(X, Y) :– SUPERVISE(X, Z), SUPERIOR(Z, Y).
SUBORDINATE(X, Y) :– SUPERIOR(Y, X).
```

Rules

```
SUPERIOR(X, Y) :– SUPERVISE(X, Y).
SUPERIOR(X, Y) :– SUPERVISE(X, Z), SUPERIOR(Z, Y).
SUBORDINATE(X, Y) :– SUPERIOR(Y, X).
```

Queries

```
SUPERIOR(james, Y)?
SUPERIOR(james, joyce)?
```

Fig. 5.6 (a) *Prolog Notation* ***(b)*** *Supervisory Tree*

Rule Syntax:

Head :- body

The symbol: Interpreted as IFF (iF and only iF) the head is also known as the conclusion or LHS. It consists of a single predicate (in most cases). RHS or premise is another name for the body (s). It consists of one or more predicates that are separated by commas (implicitly connected by AND) Ground predicates or instantiated predicates are predicates with constant parameters variables are (usually) the parameters of a predicate in a rule.

Query Syntax

A query usually consists of a predicate symbol and a set of variable arguments. The response to a query is to derive all possible combinations of constant values that make the predicate true when assigned to the variables.

Datalog Notation

- On combining atomic formulas, a Datalog programme is created.
- Atomic formulas are literals of the form p (a_1, . . . , a_n) where p is the predicate name and n is the # of its arguments.

- n is called the arity or the degree of p.
- Constant values (lowercase) or variable names can be used as arguments (uppercase).
- Built-in predicates: (syntax: less(X, 3), X< 3)

  a. binary comparison, ≥ over ordered domains

  b. binary comparison, ≥ over ordered domains
- Attention to infinite ranges: greater(X, 3).
- An atomic formula (positive literal) or an atomic formula preceded by not is referred to as a literal (negative literal).
- In Datalog, only formulas specified in a restricted clausal form known as Horn clauses can be used.

Clausal Form, Horn Clauses

A formula can contain predicates called atoms & quantifiers (universal $\forall$, existential $\exists$). In clausal form, a formula must be transformed into another formula with the following characteristics:

- All variables in the formula are universally quantified. (Since there is no need to include $\forall$ explicitly, it is removed and all variables are implicitly quantified by $\forall$).
- There are several clauses in the formula, each of which is a disjunction of literals (the literals are connected by OR).
- A formula is a collection of clauses put together in a specific order (the clauses are connected by AND).

Any formula can be transformed to clausal form, as demonstrated. Rules in Datalog are written as Horn clauses, which are a special type of clause that can only include one positive literal:

- NOT(P_1) OR NOT(P_2) OR $\cdots$ OR NOT(P_N) OR Q
- NOT(P_1) OR NOT(P_2) OR $\cdots$ OR NOT(P_N)

  (1) NOT(P_1) OR NOT(P_2) OR $\cdots$ OR NOT(P_N) OR Q
  Is equivalent to P_1 AND P_2 AND $\cdots$ AND $P_N \Rightarrow$ Q
  which is written in Datalog as the rule: Q: –P_1, P_2, . . . , P_n
  this rule says that if the predicates P_i are all true, for a particular binding of their variables, then Q is also true and can be deduced.

  (2) NOT(P_1) OR NOT(P_2) OR $\cdots$ OR NOT(P_N)
  Is equivalent to P_1 AND P_2 AND $\cdots$ AND $P_N \Rightarrow$
  which is written in Datalog as the rule: P_1, P_2, . . . , P_n
  this rule can be considered as an integrity constraint.

Rule Semantics

If a binding of constant values to variables in a rule's body (RHS) makes all of the rule's predicates TRUE, then the same binding makes the head (LHS) TRUE. A rule allows you to generate new facts, which are instantiations of the rule's head.

These new facts are built on previously known facts, and correspond to predicate bindings in the rule's body. It's worth noting that include several comma-separated predicates in the body of a rule implies that they're linked by AND.

For example: The corporate database's superior predicate (direct/indirect supervision).

Superior (X, Y): supervise (X, Y)

Superior (X, Y): supervise(X, Z), superior (Z, Y)

Note 1: A rule may be defined recursively.

Note 2: The head predicate is true if the first body is TRUE OR if the second body is TRUE when two (or more) rules have the same (LHS) head predicate.

5.5.2 Interpretation of Rules and Inference Mechanism

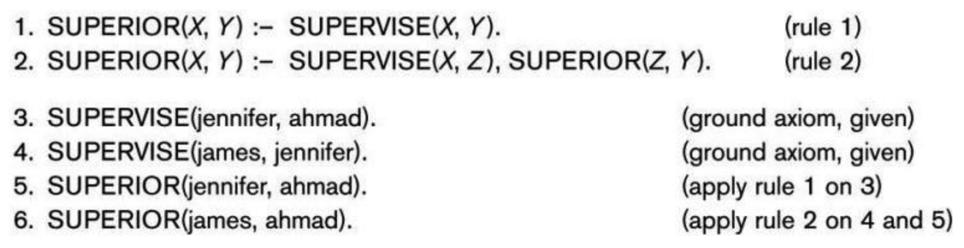
There are two main ways of interpreting the theoretical meaning of the rules:

1. Proof-Theoretic Interpretation
2. Model-Theoretic Interpretation

In practice, the inference mechanism in individual systems establishes the exact interpretation and offers a computational understanding of the rules' meaning.

Proof-Theoretic Interpretation

True statements or axioms are facts and laws. There are no variables in ground axioms. Facts are axioms that are assumed to be correct. Deductive axioms are rules that can be utilised to infer new information. Deductive axioms are used to create new facts from old ones. Provides a procedural method for calculating the result of a Datalog query.



```
1. SUPERIOR(X, Y) :-  SUPERVISE(X, Y).                          (rule 1)
2. SUPERIOR(X, Y) :-  SUPERVISE(X, Z), SUPERIOR(Z, Y).          (rule 2)

3. SUPERVISE(jennifer, ahmad).                      (ground axiom, given)
4. SUPERVISE(james, jennifer).                      (ground axiom, given)
5. SUPERIOR(jennifer, ahmad).                       (apply rule 1 on 3)
6. SUPERIOR(james, ahmad).                          (apply rule 2 on 4 and 5)
```

Fig. 5.7 *Proving a New Fact*

Model-Theoretic Interpretation

Specify a constant-valued domain that is (typically) finite. Assign every conceivable combination of values as arguments to a predicate. Determine whether or not the predicate is true.

In practice, we provide the argument value combinations that make the predicate true while stating that all other combinations make the predicate false. This is referred to as an interpretation of this particular set of predicates when done for all predicates. A meaning for the two predicates supervise and superior For a set of predicates, an interpretation assigns a truth value (T, F) to every feasible combination of argument values (chosen from a finite domain).

Rules

SUPERIOR(*X*, *Y*) :– SUPERVISE(*X*, *Y*).
SUPERIOR(*X*, *Y*) :– SUPERVISE(*X*, *Z*), SUPERIOR(*Z*, *Y*).

Interpretation

Known Facts:

SUPERVISE(franklin, john) is **true**.
SUPERVISE(franklin, ramesh) is **true**.
SUPERVISE(franklin, joyce) is **true**.
SUPERVISE(jennifer, alicia) is **true**.
SUPERVISE(jennifer, ahmad) is **true**.
SUPERVISE(james, franklin) is **true**.
SUPERVISE(james, jennifer) is **true**.
SUPERVISE(*X*, *Y*) is **false** for all other possible (*X*, *Y*) combinations

Derived Facts:

SUPERIOR(franklin, john) is **true**.
SUPERIOR(franklin, ramesh) is **true**.
SUPERIOR(franklin, joyce) is **true**.
SUPERIOR(jennifer, alicia) is **true**.
SUPERIOR(jennifer, ahmad) is **true**.
SUPERIOR(james, franklin) is **true**.
SUPERIOR(james, jennifer) is **true**.
SUPERIOR(james, john) is **true**.
SUPERIOR(james, ramesh) is **true**.
SUPERIOR(james, joyce) is **true**.
SUPERIOR(james, alicia) is **true**.
SUPERIOR(james, ahmad) is **true**.
SUPERIOR(*X*, *Y*) is **false** for all other possible (*X*, *Y*) combinations

Fig. 5.8 *An Interpretation that is a Minimal Model*

When the rules of a certain set of rules are always true in this interpretation, it is termed a model. (For example, the head is true for any values supplied to the variables in the rules; when the body is true, the rules are not broken) This interpretation is a model for superior Q. When does a rule get broken? The rule body predicate is true, but the rule head predicate is false, due to a specific binding of the arguments of the predicates in the rule body.

Example: I denotes a certain interpretation

Supervise (a, b) is TRUE in I

Superior (b, c) is TRUE in I

Superior (a, c) is FALSE in I

Then, I cannot be a model for the (recursive) rule

superior(X, Y): supervise(X, Z), superior(Z, Y)

The meaning of the rules is established in the model-theoretic approach by giving a model for these rules. If we can't change any fact from T to F and still have a model for these rules, it's termed a minimal model for a set of rules. We still have a model if we add superior (james, bob) to the true predicates. Because altering superior (james, bob) to false still gives us a model, this is a non-minimal model.

NOTES

Comparison of Proof-Theoretic & Model-Theoretic Interpretations

The minimal model of the model-theoretic interpretation is the same as the facts generated by the proof-theoretic interpretation for rules with simple structure (i.e., no negation involved). The use of negations destroys the uniqueness property of minimal models, and thus the correspondence between the two types of interpretations.

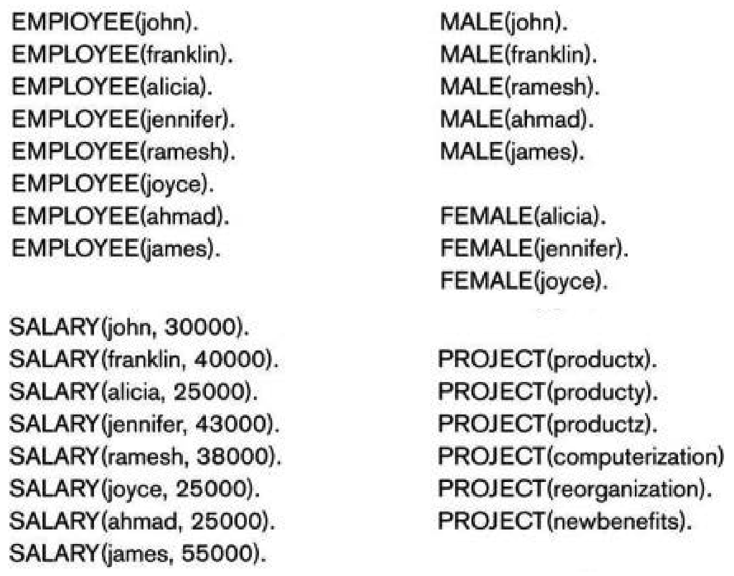
Datalog Programs

In Datalog programmes, there are two major approaches for defining the truth values of predicates:

1. Predicates (or relations) defined by facts correlate to RDBMS relations formed by stating all the combinations of values that make the predicate true.

   EMPLOYEE, MALE, FEMALE, DEPARTMENT, SUPERVISE, PROJECT, WORKS ON
2. LHS of one or more Datalog rules defines rule-defined predicates (or views).

   If a programme or rule generates a finite set of facts, it is said to be safe. The task of deciding whether a set of rules is safe is theoretically unsolvable.



```
EMPIOYEE(john).                MALE(john).
EMPLOYEE(franklin).            MALE(franklin).
EMPLOYEE(alicia).              MALE(ramesh).
EMPLOYEE(jennifer).            MALE(ahmad).
EMPLOYEE(ramesh).              MALE(james).
EMPLOYEE(joyce).
EMPLOYEE(ahmad).               FEMALE(alicia).
EMPLOYEE(james).               FEMALE(jennifer).
                               FEMALE(joyce).
SALARY(john, 30000).
SALARY(franklin, 40000).       PROJECT(productx).
SALARY(alicia, 25000).         PROJECT(producty).
SALARY(jennifer, 43000).       PROJECT(productz).
SALARY(ramesh, 38000).         PROJECT(computerization)
SALARY(joyce, 25000).          PROJECT(reorganization).
SALARY(ahmad, 25000).          PROJECT(newbenefits).
SALARY(james, 55000).
```

NOTES

```
                                     WORKS_ON(john, productx, 32).
DEPARTMENT(john, research).          WORKS_ON(john, producty, 8).
DEPARTMENT(franklin, research).      WORKS_ON(ramesh, productz, 40).
DEPARTMENT(alicia, administration).  WORKS_ON(joyce, productx, 20).
DEPARTMENT(jennifer, administration). WORKS_ON(joyce, producty, 20).
DEPARTMENT(ramesh, research).        WORKS_ON(franklin, producty, 10).
DEPARTMENT(joyce, research).         WORKS_ON(franklin, productz, 10).
DEPARTMENT(ahmad, administration).   WORKS_ON(franklin, computerization, 10).
DEPARTMENT(james, headquarters).     WORKS_ON(franklin, reorganization, 10).
                                     WORKS_ON(alicia, newbenefits, 30).
SUPERVISE(franklln, john).           WORKS_ON(alicia, computerization, 10).
SUPERVISE(franklln, ramesh)          WORKS_ON(ahmad, computerization, 35).
SUPERVISE(frankin , joyce).          WORKS_ON(ahmad, newbenefits, 5).
SUPERVISE(jennifer, alicia).         WORKS_ON(jennifer, newbenefits, 20).
SUPERVISE(jennifer, ahmad).          WORKS_ON(jennifer, reorganization, 15).
SUPERVISE(james, franklin).          WORKS_ON(james, reorganization, 10).
SUPERVISE(james, jennifer).
```

Fig. 5.9 *Fact Predicates for Part of the Database*

```
SUPERIOR(X, Y) :– SUPERVISE(X, Y).
SUPERIOR(X, Y) :– SUPERVISE(X, Z), SUPERIOR(Z, Y).

SUBORDINATE(X, Y) :– SUPERIOR(Y, X).

SUPERVISOR(X) :– EMPLOYEE(X), SUPERVISE(X, Y).

OVER_40K_EMP(X) :– EMPLOYEE(X), SALARY(X, Y), Y >= 40000.
UNDER_40K_SUPERVISOR(X) :– SUPERVISOR(X), NOT(OVER_40_K_EMP(X)).
MAIN_PRODUCTX_EMP(X) :– EMPLOYEE(X), WORKS_ON(X, productx, Y), Y >=20.
PRESIDENT(X) :– EMPLOYEE(X), NOT(SUPERVISE(Y, X) ).
```

Fig. 5.10 *Rule-Defined Predicates*

Forward Chaining, Bottom-Up Inference Techniques

Starting with the facts, the inference engine applies the rules to generate new facts. For a match, the generated facts are compared to the query predicate goal. The word forward chaining refers to the inference moving forward from the facts to the goal.

For example: query superior (james,Y).

- Does any of the existing facts match the query? NO
- Apply the first (non-recursive) rule to existing facts to generate new facts Ò! facts for the superior predicate are generated.
- Each generated fact is tested for a match against the query. The first match is superior (james, franklin) Ò! Y=franklin. The next match is superior(james, jennifer) Ò! Y=jennifer.
- Apply the second (recursive) rule to existing (initial & generated) facts matches each supervise fact with each superior fact, to satisfy both the RHS predicates supervise(X,Z) & superior(Z,Y) Ò! additional answers: Y=john, Y=ramesh, Y=joyce, Y=alicia, Y=ahmad.

NOTES

Top-Down Inference Mechanisms, Chaining Backwards

The inference engine starts with the query predicate goal and looks for matches to the variables that lead to valid facts in the database. Backward chaining means that the inference moves backward from the goal to identify facts that fulfil the goal. Facts are not intentionally created (as in forward chaining).

For example: query superior (james,Y).

- Search for any facts of the superior predicate whose first argument matched james, no such facts found.
- Locate the first rule whose head has the same predicate name as the query Ò! superior(X,Y):-supervise(X,Y).
- Bind the variable X to james, leading to the rule

  superior(james,Y):-supervise(james,Y).
- Search (in the order of listing in the program) for facts that match supervise(james,Y) Ò! Y=franklin, Y=Jennifer.
- Locate the next rule whose head has the same predicate name as the query Ò! superior(X,Y):-supervise(X,Z),superior(Z,Y).
- Bind the variable X to james, leading to the rule

  superior(james,Y):-supervise(james,Z),superior(Z,Y).

  Search for facts that match both subgoals (depth-first) supervises (james,franklin) bind Z to franklin.

5.5.3 Deductive Object Oriented Database

Deductive object-oriented databases are two important extensions of the traditional database technology. Deductive databases extend the expressive power of traditional databases by means of recursion and declarative querying while object-oriented databases extend their data modeling power by means of object identity, complex objects, classes, class hierarchy and inheritance. However both extensions have their shortcomings deductive databases lack powerful data modeling mechanisms while object-oriented databases lack logical semantics and declarative query language. Therefore, by combining the two approaches to gather we can eliminate their shortcomings and reap their advantage.

5.6 MOBILITY AND PERSONAL DATABASE

Mobile users will be able to access information from anywhere and at any time thanks to the fast evolving technology of mobile communication. Continuous connectivity in a mobile environment is now possible due to wireless technology.

Mobile databases are independent of the primary database and may be simply relocated to different locations. They can communicate with the database to share and exchange data even if they are not connected to the main database.

The mobile database includes the following components –

- Main system database: It stores all the data and is linked to the mobile database.
- Mobile database: It allows users to view information even while on the move. It shares information with the main database.
- Device: It uses the mobile database to access data. This device can be a mobile phone, laptop, etc.
- Communication link: It allows the transfer of data between the mobile database and the main database.

Advantages of Mobile Databases

Given below are the advantages of mobile databases:

- Mobile databases require very little support and maintenance.
- The data in a database can be accessed from anywhere using a mobile database. It provides wireless database access.
- The mobile database can be synchronized with multiple devices such as mobiles, computer devices, laptops etc.
- The database systems are synchronized using mobile databases and multiple users can access the data with seamless delivery process.

Disadvantages of Mobile Databases

Given below are the disadvantages of mobile databases.

- The mobile unit that houses a mobile database may frequently lose power because of limited battery. This should not lead to loss of data in database.
- The mobile data is less secure than data that is stored in a conventional stationary database. This presents a security hazard.

5.6.1 Database Technologies

In the last two decades, database technology has advanced at a breakneck pace. At the start of the twenty-first century, Online Analytical Processing (OLAP), which rose to prominence in the 1990s, began to lose ground to in-memory databases.

Modern business intelligence requirements, on the other hand, have posed a challenge that in-memory databases will struggle to meet. As a result, the next generation of databases and querying has emerged: In-Chip® analytics. This newly developed technology makes novel use of the CPU, RAM, and disc storage to address the complexity and size of data sets that existing BI software is compelled to deal with in order to give meaningful insights to end users in a fair amount of time.

Database technologies gather information and store, organise, and process it in a way that allows users to go back and discover details they are looking for quickly and intuitively. Database technologies are available in a variety of sizes and shapes, ranging from complicated to basic and big to small. It is crucial to consider how your database technology will scale as your data expands in size, as well as how it will interact with any applications you use to query your data.

In-Memory Databases

In-memory technology – that is, loading the entire database into RAM and then transferring it to the CPU to perform calculations – has become a popular business intelligence solution because it allows users to get quick answers to their queries without having to wait for long builds and pre-calculations; however, the size and complexity of modern data is forcing in-memory databases to reach their limits.

A computer generally has two types of data storage mechanisms – disk (often called a hard disk) and RAM (random access memory). Most modern computers have 15-100 times more available disk storage than they do RAM.

However, reading data from disk is much slower than reading the same data from RAM. This is one of the reasons why 1GB of RAM costs approximately 320 times that of 1GB of disk space. In-memory technology aims to address performance issues by pre-loading the entire database into RAM, and loading data from RAM to the CPU to perform calculations and data retrieval.

ElastiCubes and In-Chip® Analytics

The latest advancement in database technology is In-Chip® Technology. It combines the flexibility of in-memory querying with the speed and robustness of OLAP cubes, all without the high hardware costs and time-consuming implementation that traditional solutions require. In-Chip® is quickly gaining popularity because to its increased performance and ability to handle complicated and big data sets, despite the fact that it was only recently designed and released.

ElastiCube is a one-of-a-kind database created by Sisense as a result of a thorough examination of the advantages and disadvantages of both OLAP and in-memory technologies. Its name refers to the database's unique ability to go beyond the constraints imposed by older technologies. In-Chip® is the most recent version of in-memory technology for business analytics, and it distinguishes itself by being both quick and scalable. It uses a disk-based columnar database for storage to provide fast disk reads and is able to load data from disk to RAM (and vice versa) when needed. The queries themselves are processed entirely in-memory without any disk-reads throughout.

Check Your Progress

4. What mechanism is Google search based on?
5. What is deductive database system?
6. What is deductive object oriented database?

5.7 ANSWERS TO 'CHECK YOUR PROGRESS'

1. Various applications of multimedia types include digital libraries, manufacturing and retailing, journalism, art and entertainment.

NOTES

2. Multimedia data such as text, audios, images, and videos are rapidly gaining popularity as important forms of creating, exchanging, and storing information.
3. The introduction and growth of e-books has changed the relationships between the libraries, publishers and distributors and also the relationship between the libraries and their users. There are several libraries who are adopting e-book collections on a very large scale. Armstrong and Lonsdale have defined the term e-book as 'any content that is recognizably book like, regardless of size, origin or composition, but excluding serial publications, made available electronically for reference or reading on any device that includes a screen'.
4. Google search is based on the mechanism of algorithm.
5. A DDB is a database system with the ability to establish deductive rules, which allow us to derive or infer additional information from database facts. Mathematical, logic is the theoretical foundation for DDBs.
6. Deductive databases extend the expressive power of traditional databases by means of recursion and declarative querying while object-oriented databases extend their data modeling power by means of object identity, complex objects, classes, class hierarchy and inheritance.

5.8 SUMMARY

- Multimedia Database Management System (MMDBMS) supports all multimedia data types. In addition, MMDBMS also supports traditional DBMS features such as data modeling, retrieval and organisation.
- Various applications of multimedia types include digital libraries, manufacturing and retailing, journalism, art and entertainment.
- Multimedia data such as text, audios, images, and videos are rapidly gaining popularity as important forms of creating, exchanging, and storing information.
- A continuous media data requires steady download of the media data. The most important continuous media data stored in an MMDB includes video and audio data.
- In the video-on-demand application, a viewer requests for a particular video online and then the MDBS server searches for, locates and plays the requested video.
- An MMDB should be a blend of functionalities and features of traditional databases and a framework for the storage, retrieval, transmission and presentation of multimedia data types.
- To retrieve data, a multimedia database system can employ conventional data or information retrieval methods such as logging and indexing, graph or tree pattern matching methods and content-based retrieval methods.
- The software that manages an MMDB is known as MMDBMS. MMDBMS architecture enables a database to store, access and manage multimedia data and carryout different types of transactions.

- The library plays a vital role for any organization, be it school, college, university or any business organization. With the advancement in computer and telecommunication technologies in the past few years the availability of information has grown tremendously.
- The introduction and growth of e-books has changed the relationships between the libraries, publishers and distributors and also the relationship between the libraries and their users. There are several libraries who are adopting e-book collections on a very large scale.
- Armstrong and Lonsdale have defined the term e-book as 'any content that is recognizably book like, regardless of size, origin or composition, but excluding serial publications, made available electronically for reference or reading on any device that includes a screen'.
- The use of the Internet is considered as a prime tool to decide the role and scope of database market. In fact, the impact of the Internet on database market has generated a large area for new database strategies in which two prime tools, such as Object oriented Data Model (OODM) and Extended Relational Data Model-Object/ Relational Database Model (ERDM-O/ RDM) are used as a platform for Internet based database development.
- DDBs (Deductive Databases) is a field that combines databases, artificial intelligence, and logic.
- A DDB is a database system with the ability to establish deductive rules, which allow us to derive or infer additional information from database facts. Mathematical, logic is the theoretical foundation for DDBs.
- Facts are specified in a similar way to relations, but without the need for attribute names. In a DDB, an attribute's meaning is defined by its position in a tuple.
- Relational views are analogous to rules. They define virtual relationships that aren't actually recorded, but can be inferred from facts using inference methods based on rule descriptions.
- A query usually consists of a predicate symbol and a set of variable arguments. The response to a query is to derive all possible combinations of constant values that make the predicate true when assigned to the variables.
- Deductive object-oriented databases are two important extensions of the traditional database technology.
- Deductive databases extend the expressive power of traditional databases by means of recursion and declarative querying while object-oriented databases extend their data modeling power by means of object identity, complex objects, classes, class hierarchy and inheritance.

5.9 KEY TERMS

- **Multimedia database:** Multimedia data such as text, audios, images, and videos are rapidly gaining popularity as important forms of creating, exchanging, and storing information.

- **Continuous media data:** The data that downloads at a steady-rate is known as isochronous data or continuous-media data.
- **Internet:** The use of the Internet is considered as a prime tool to decide the role and scope of database market.
- **Deductive database system:** A DDB is a database system with the ability to establish deductive rules, which allow us to derive or infer additional information from database facts. Mathematical, logic is the theoretical foundation for DDBs.

5.10 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What do you mean by video on demand?
2. What is similarity-based retrieval?
3. Write the scope of Internet based library and information services.
4. Mention the benefits of database server.
5. What is proof-model interpretation?
6. What is deductive object oriented database?
7. What do you understand by mobile database?

Long-Answer Questions

1. Discuss the various components of a multimedia database by giving appropriate examples.
2. Explain MMDBMS architecture and storage hierarchy in the MMDBMS architecture with relevant diagrams.
3. Describe the e-journals and its advantages and disadvantages with the help of appropriate examples.
4. Explain the various factors that are supported by the Internet to correspond with database.
5. Explain the deductive database notations by giving appropriate examples.
6. Discuss rules of interpretation and inference mechanism.
7. Discuss the database technologies.

5.11 FURTHER READING

Navathe, S.B. and R. Elmasri. 1997. *Database System Concepts*, Third edition. New York: McGraw-Hill.

Korth, Henry F., Avi Silberschatz and S. Sudarshan. 2010. *Database System Concepts*, 6th edition. New York: McGraw-Hill.

Elmasri, Ramez and Shamkant B. Navathe. 2007. *Fundamentals of Database Systems*, 5th edition. New Jersey: Pearson Education.

Martin, James. 2007. *Principles of Data Base Management*. New Jersey:

Pearson Education.

Martin, James. 1975. *Computer Data-Base Organization*. New Jersey: Prentice Hall.

Jackson, Glenn A. 1988. *Relational Database Design With Microcomputer Applications***.** New Jersey: Prentice Hall.

Date, C.J. 2000. *An Introduction to Database System*, Seventh edition. Boston: Addison Wesley.