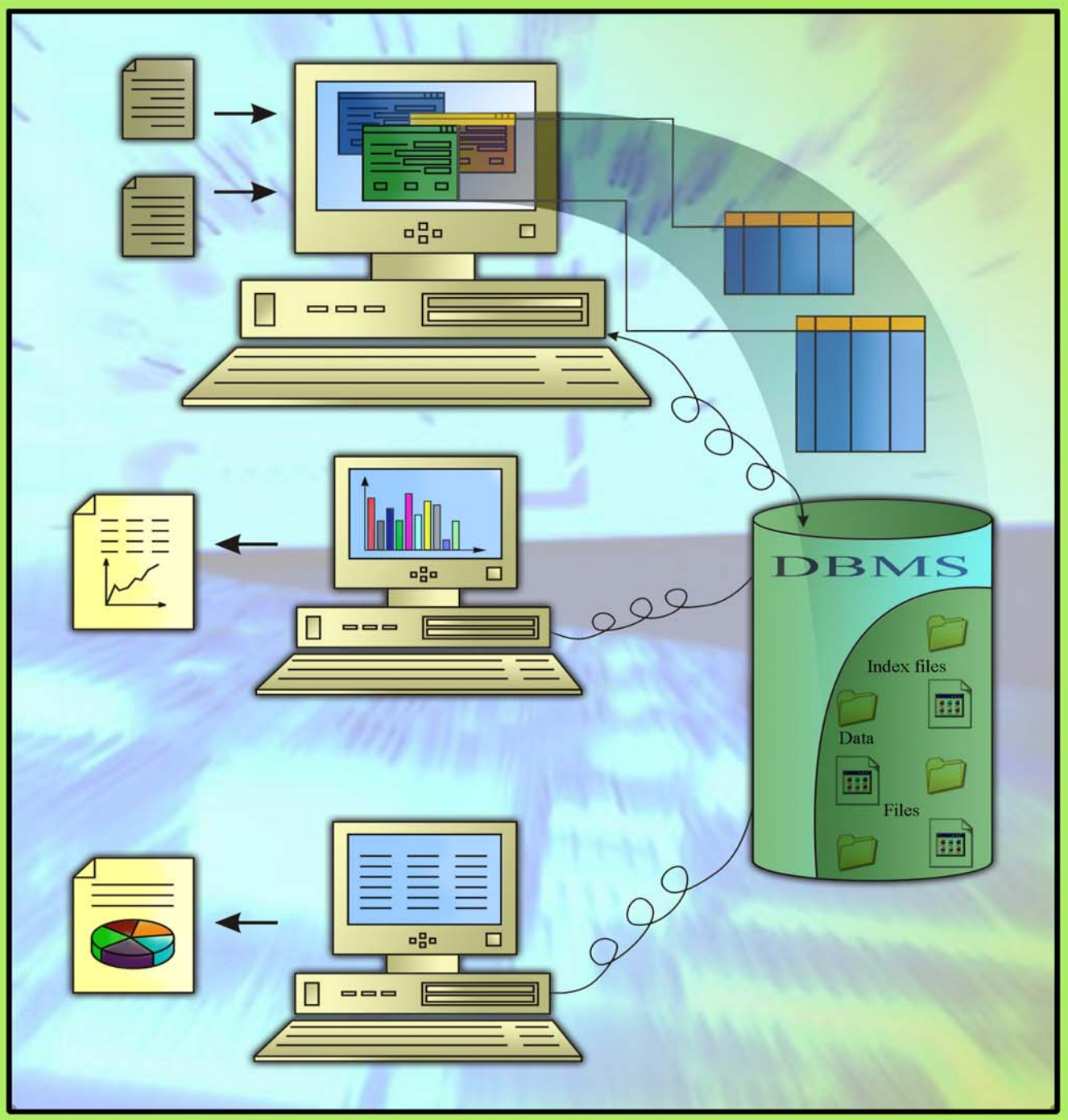
Indira Gandhi National Open University
School of Computer and Information Sciences

MCS - 023
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS



The Database Management System

2

Indira Gandhi
National Open University
School of Computer &
Information Sciences

MCS-024
INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

Block

2

STRUCTURED QUERY LANGUAGE AND TRANSACTION MANAGEMENT

UNIT 1

The Structured query Language **5**

UNIT 2

Transactions and Concurrency Management **35**

UNIT 3

Database Recovery and Security **57**

UNIT 4

Distribution and Client-Server Databases **73**

Programme / Course Design Committee

Prof. Sanjeev K. Aggarwal, IIT, Kanpur
Prof. M. Balakrishnan, IIT, Delhi
Prof. Harish Karnick, IIT, Kanpur
Prof. C. Pandurangan, IIT, Madras
Dr. Om Vikas, Sr. Director, Ministry of ICT, Delhi
Prof. P. S. Grover, Sr. Consultant, SOCIS, IGNOU

Faculty of School of Computer and Information Sciences

Shri Shashi Bhushan
Shri Akshay Kumar
Prof. Manohar Lal
Shri V.V. Subrahmanyam
Shri P.Venkata Suresh

Block Preparation Team

Dr. D.K. Lobiyal (Content Editor)
School of Computer & System Sciences
Jawaharlal Nehru University
New Delhi

Prof. Adarsh Kumar Verma
(Language Editor)

Shri M.P. Mishra
SOCIS, IGNOU

Course Coordinator: Shri M.P. Mishra

Block Production Team

Shri T.R. Manoj, Section Officer (Pub.) and Shri H.K. Som, Consultant

Acknowledgements

To Shri Akshay Kumar, SOCIS for his valuable comments and suggestions in Unit 1 and Unit 2.

April, 2005

©Indira Gandhi National Open University, 2005

ISBN—81-266-1747-0

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Indira Gandhi National Open University.

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110068.

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by the Director, SOCIS.

BLOCK INTRODUCTION

In the previous block, you have gone through the basic concepts of database and database management system. Some of the key concepts introduced in Block 1 were three level architecture, data independence, integrity constraints, normalisation, ER model and file organization. However, information about database management system would be incomplete if we do not discuss about how to get information into and out of them in a structured way, and transaction management and security. This block deals with theses issues of database Management system.

Unit 1 of this block is devoted to a standard query Language called Structured Query Language (SQL), which is the most acceptable standard across all commercial database management systems. This unit covers almost all the important features required for database creation, updating and retrieval. The unit also discusses about the complex queries.

Unit 2 discusses the concept of transaction and their management. Transactions are everywhere in the Industry, so how do we deal with transactions in database specially when concurrent transactions are going on? This unit attempts to provide the answer to this question.

Unit 3 provides information with respect to database security and recovery. These two topics are very important for you as they are mainly the responsibility of database administrator, a job where an MCA profile with some experience fits very well.

Unit 4 provides information about two important database approaches, the distributed database system and the client server system. The distributed database system is relational in nature so they are conceptually the logical extensions of relational system. Clint server model is one of the most acceptable implementation Model in the industry. Almost all commercial DBMSs are available in client server mode.

UNIT 1 THE STRUCTURED QUERY LANGUAGE

Structure **Page Nos.**

1.0	Introduction	5
1.1	Objectives	5
1.2	What is SQL?	6
1.3	Data Definition Language	7
1.4	Data Manipulation Language	9
1.5	Data Control	15
1.6	Database Objects: Views, Sequences, Indexes and Synonyms	16
	1.6.1 Views	
	1.6.2 Sequences	
	1.6.3 Indexes and Synonyms	
1.7	Table Handling	19
1.8	Nested Queries	23
1.9	Summary	27
1.10	Solutions/Answer	28
1.11	Further Reading	34

1.0 INTRODUCTION

Database is an organised collection of information about an entity having controlled redundancy and serves multiple applications. DBMS (database management system) is an application software that is developed to create and manipulate the data in database. A query language can easily access a data in a database. SQL (Structured Query Language) is language used by most relational database systems. IBM developed the SQL language in mid-1979. All communication with the clients and the RDBMS, or between RDBMS is via SQL. Whether the client is a basic SQL engine or a disguised engine such as a GUI, report writer or one RDBMS talking to another, SQL statements pass from the client to the server. The server responds by processing the SQL and returning the results. The advantage of this approach is that the only network traffic is the initial query and the resulting response. The processing power of the client is reserved for running the application.

SQL is a data sub-language consisting of three built-in languages: Data definition language (DDL), Data manipulation language (DML) and Data control language (DCL). It is a fourth generation language. SQL has many more features and advantages. Let us discuss the SQL in more detail in this unit. It should be noted that many commercial DBMS may or may not implement all the details given in this unit. For example, MS-ACCESS does not support some of these features. Even some of the constructs may not be portable, please consult the DBMS Help for any such difference.

1.1 OBJECTIVES

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

- create, modify and delete database schema objects;
- update database using SQL command;
- retrieve data from the database through queries and sub-queries;
- handle join operations;
- control database access;
- deal with database objects like Tables, Views, Indexes, Sequences, and Synonyms using SQL.

1.2 WHAT IS SQL?

Structured Query Language (SQL) is a standard query language. It is commonly used with all relational databases for data definition and manipulation.

All the relational systems support SQL, thus allowing migration of database from one DBMS to another. In fact, this is one of the reasons of the major success of Relational DBMS. A user may be able to write a program using SQL for an application that involves data being stored in more than one DBMSs, provided these DBMS support standard SQL. This feature is commonly referred to as portability. However, not all the features of SQL implemented in one RDBMS are available in others because of customization of SQL. However, please note there is just ONE standard SQL.

SQL provides an interface where the user can specify "What" are the expected results. The query execution plan and optimisation is performed by the DBMS. The query plan and optimisation determines how a query needs to be executed. For example, if three tables X, Y and Z are to be joined together then which plan (X JOIN Y) and then Z or X JOIN (Y JOIN Z) may be executed. All these decisions are based on statistics of the relations and are beyond the scope of this unit.

SQL is called a non-procedural language as it just specifies what is to be dome rather than how it is to be done. Also, since SQL is a higher-level query language, it is closer to a language like English. Therefore, it is very user friendly.

The American National Standard Institute (ANSI) has designed standard versions of SQL. The first standard in this series was created in 1986. It was called SQL-86 or SQL1. This standard was revised and enhanced later and SQL-92 or SQL-2 was released. A newer standard of SQL is SQL3 which is also called SQL- 99. In this unit we will try to cover features from latest standards. However, some features may be found to be very specific to certain DBMSs.

Some of the important features of SQL are:

- It is a non procedural language.
- It is an English-like language.
- It can process a single record as well as sets of records at a time.
- It is different from a third generation language (C& COBOL). All SQL statements define what is to be done rather than how it is to be done.
- SQL is a data sub-language consisting of three built-in languages: Data definition language (DDL), Data manipulation language (DML) and Data control language (DCL).
- It insulates the user from the underlying structure and algorithm.
- SQL has facilities for defining database views, security, integrity constraints, transaction controls, etc.

There are many variants of SQL, but the standard SQL is the same for any DBMS environment. The following table shows the differences between SQL and one of its superset SQL*Plus which is used in Oracle. This will give you an idea of how various vendors have enhanced SQL to an environment. The non-standard features of SQL are not portable across databases, therefore, should not be used preferably while writing SQL queries.

Difference between SQL and SQL*Plus

SQL	SQL*Plus
SQL is a language	SQL *Plus is an environment
It is based on ANSI (American National Standards Institute) standard	It is oracle proprietary interface for executing SQL statements

SQL	
It is entered into the SQL buffer on one or more lines	It is entered one line at a time, not stored in the SQL buffer
SQL cannot be abbreviated	SQL*Plus can be abbreviated
It uses a termination character to execute command immediately	It does not require termination character. Commands are executed immediately.
SQL statements manipulate data and table definition in the database	It does not allow manipulation of values in the database

1.3 DATA DEFINITION LANGUAGE

As discussed in the previous block, the basic storage unit of a relational database management system is a table. It organises the data in the form of rows and columns. But what does the data field column of table store? How do you define it using SQL?

The Data definition language (DDL) defines a set of commands used in the creation and modification of schema objects such as tables, indexes, views etc. These commands provide the ability to create, alter and drop these objects. These commands are related to the management and administrations of the databases. Before and after each DDL statement, the current transactions are implicitly committed, that is changes made by these commands are permanently stored in the databases. Let us discuss these commands in more detail:

CREATE TABLE Command

Syntax:

```
CREATE TABLE <table name>(
Column_name1 data type (column width) [constraints],
Column_name2 data type (column width) [constraints],
Column_name3 data type (column width) [constraints],
……..
);
```

Where table name assigns the name of the table, column name defines the name of the field, data type specifies the data type for the field and column width allocates specified size to the field.

Guidelines for creation of table:

- Table name should start with an alphabet.
- In table name, blank spaces and single quotes are not allowed.
- Reserve words of that DBMS cannot be used as table name.
- Proper data types and size should be specified.
- Unique column name should be specified.

Column Constraints: NOT NULL, UNIQUE, PRIMARY KEY, CHECK, DEFAULT, REFERENCES,

On delete Cascade: Using this option whenever a parent row is deleted in a referenced table then all the corresponding child rows are deleted from the referencing table. This constraint is a form of referential integrity constraint.

Example 1:

```
CREATE TABLE product
    (
pno number (4) PRIMARY KEY,
pname char (20) NOT NULL,
qoh number (5) DEFAULT (100),
```

```
        class char (1) NOT NULL,
        rate number (8,2) NOT NULL,
        CHECK ((class='A' AND rate<1000) OR
        (class='B' AND rate>1000 AND rate<4500) OR
        (class='C' AND rate>4500))
    );
```

The command above creates a table. Primary key constraint ensures that product number (pno) is not null and unique (both are the properties of primary key). Please note the use of data type char (20). In many implementations of SQL on commercial DBMS like SQL server and oracle, a data type called varchar and varchar2 is used respectively. Varchar basically is variable length character type subject to a maximum specified in the declarations. We will use them at most of the places later.

Please note the use of check constraints in the table created above. It correlates two different attribute values.

Example 2:

```
CREATE TABLE prodtrans
(
pno number (4)
ptype char (1) CHECK (ptype in ('I','R','S')),
qty number (5)
FOREIGN KEY pno REFERENCES product (pno)
ON DELETE CASCADE);
```

In the table so created please note the referential constraint on the foreign key **pno** in **prodtrans** table to **product** table. Any product record if deleted from the product table will trigger deletion of all the related records (ON DELETE CASCADE) in the **prodtrans** table.

ALTER TABLE Command: This command is used for modification of existing structure of the table in the following situation:

- When a new column is to be added to the table structure.
- When the existing column definition has to be changed, i.e., changing the width of the data type or the data type itself.
- When integrity constraints have to be included or dropped.
- When a constraint has to be enabled or disabled.

Syntax

ALTER TABLE <table name> ADD (<column name> <data type>…);
ALTER TABLE <table name> MODIFY (<column name> <data type>…);
ALTER TABLE <table name> ADD CONSTRAINT <constraint name> < constraint type>(field name);
ALTER TABLE <table name> DROP<constraint name>;
ALTER TABLE <table name> ENABLE/DISABLE <constraint name>;

You need to put many constraints such that the database integrity is not compromised.

Example 3:

```
ALTER TABLE emp MODIFY (empno NUMBER (7));
```

DROP TABLE Command:

When an existing object is not required for further use, it is always better to eliminate it from the database. To delete the existing object from the database the following command is used.
Syntax:

DROP TABLE<table name>;

Example 4:

DROP TABLE emp;

1.4 DATA MANIPULATION LANGUAGE

Data manipulation language (DML) defines a set of commands that are used to query and modify data within existing schema objects. In this case commit is not implicit that is changes are not permanent till explicitly committed. DML statements consist of SELECT, INSERT, UPDATE and DELETE statements.

SELECT Statement

This statement is used for retrieving information from the databases. It can be coupled with many clauses. Let us discuss these clauses in more detail:

1. **Using Arithmetic operator**

Example 5:

```
SELECT ENAME, SAL, SAL+300
 FROM EMP;
```

2. **Operator Precedence**

The basic operators used in SQL are * / + -
Operators of the same priority are evaluated From Left to right
Parentheses are used to force prioritized evaluation.

Example 6:

```
SELECT ENAME, SAL, 12*SAL+100
FROM EMP;

SELECT ENAME, SAL, 12*(SAL+100)
FROM EMP;
```

3. **Using Column aliases**

Example 7:

To print column names as NAME and ANNUAL SALARY

```
SELECT ENAME "NAME", SAL*12 "ANNUAL SALARY"
FROM EMP;
```

4. **Concatenation operator**

Example 8:

Printing name and job as one string as column name employees:

```
SELECT ENAME||JOB "EMPLOYEES"
FROM EMP;
```

5. **Using Literal Character String**

Example 9:

To print <name> IS A <job> as one string with column name employee

```
SELECT ENAME || ' IS A ' || JOB AS "EMPLOYEE"
FROM EMP;
```

6. **To eliminate duplicate rows (distinct operator)**

Example 10:

```
SELECT DISTINCT DEPTNO
FROM EMP;
```

7. **Special comparison operator** used in where Clause

   a. **between. …and…**It gives range between two values (inclusive)

Example 11:

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL BETWEEN 1000 AND 1500;
```

   b. **In (list):** match any of a list of values

Example 12:

```
SELECT EMPNO, ENAME, SAL, MGR
FROM EMP
WHERE MGR IN (7902, 7566, 7788);
```

7902, 7566, and 7788 are Employee numbers

   c. **Like:** match a character pattern

- Like operator is used only with char and Varchar2 to match a pattern
- % Denotes zero or many characters
- _ Denotes one character
- Combination of % and_can also be used

Example 13:

(I) List the names of employees whose name starts with 's'

```
SELECT ENAME FROM EMP
WHERE ENAME LIKE 'S%';
```

(II) List the ename ending with 's'

```
SELECT ENAME FROM EMP
WHERE ENAME LIKE '%S';
```

(III) List ename having I as a second character

```
SELECT ENAME FROM EMP
WHERE ENAME LIKE '_I%';
```

d. **Is null operator**

Example 14:

to find employee whose manage-id is not specified

```
SELECT ENAME, MGR FROM EMP
WHERE MGR IS NULL;
```

8 **Logical Operators**

Rules of Precedence:

Order evaluated	Operator
1	All comparison operators

2	NOT
3	AND
4	OR

Example 15: To print those records of salesman or president who are having salary above 15000/-

```
Select ename, job, sal from emp
Where job = 'SALESMAN' or job = 'PRESIDENT'
And sal>15000;
```

The query formulation as above is incorrect for the problem statement. The correct formulation would be:

```
SELECT ENAME, JOB, SAL FROM EMP
WHERE (JOB = 'SALESMAN' OR JOB = 'PRESIDENT')
AND SAL>15000;
```

9. **Order by clause**

- It is used in the last portion of select statement
- By using this rows can be sorted
- By default it takes ascending order
- DESC: is used for sorting in descending order
- Sorting by column which is not in select list is possible
- Sorting by column Alias

Example 16:

```
SELECT EMPNO, ENAME, SAL*12 "ANNUAL"
FROM EMP
ORDER BY ANNUAL;
```

Example 17: Sorting by multiple columns; ascending order on department number and descending order of salary in each department.

```
SELECT ENAME, DEPTNO, SAL
FROM EMP
ORDER BY DEPTNO, SAL DESC;
```

10. **Aggregate functions**

- Some of these functions are count, min, max, avg.
- These functions help in getting consolidated information from a group of tuples.

Example 18: Find the total number of employees.

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

Example 19: Find the minimum, maximum and average salaries of employees of department D1.

```
SELECT MIN(SAL), MAX(SAL), AVG(SAL)
FROM EMP
WHERE DEPTNO = 'D1' ;
```

11. **Group By clauses**

- It is used to group database tuples on the basis of certain common attribute value such as employees of a department.
- WHERE clause still can be used, if needed.

Example 20: Find department number and Number of Employees working in that department.

```
SELECT DEPTNO, COUNT(EMPNO)
FROM EMP
GROUP BY DEPTNO;
```

Please note that while using group by and aggregate functions the only attributes that can be put in select clause are the aggregated functions and the attributes that have been used for grouping the information. For example, in the example 20, we cannot put ENAME attribute in the SELECT clause as it will not have a distinct value for the group. Please note the group here is created on DEPTNO.

12. Having clause

- This clause is used for creating conditions on grouped information.

Example 21: Find department number and maximum salary of those departments where maximum salary is more than Rs 20000/-.

```
SELECT DEPTNO, MAX(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING MAX(SAL) > 20000;
```

INSERT INTO command:

- Values can be inserted for all columns or for the selected columns
- Values can be given through sub query explained in section 1.8
- In place of values parameter substitution can also be used with insert.
- If data is not available for all the columns, then the column list must be included following the table name.

Example 22: Insert the employee numbers, an increment amount of Rs.500/- and the increment date-today (which is being entered through function SYSDATE) of all the managers into a table INCR (increment due table) from the employee file.

```
INSERT INTO INCR
SELECT EMPNO, 500, SYSDATE FROM EMP
WHERE JOB = 'MANAGER';
```

Example 23: Insert values in a table using parameter substitution (& operator is used for it 1, 2 are the field numbers).

```
INSERT INTO EMP
VALUES (&1,'&2','&3', &4, &5, &6,NULL, &7);
```

Please note these values needs to be supplied at run time.

UPDATE Command:

Syntax is

```
UPDATE <table name>
SET <column name> = <value>
WHERE <condition>;
```

Sub query in the UPDATE command:

Example 24: Double the commission for employees, who have got at least 2 increments.

```
UPDATE EMP
SET COMM = COMM * 2
WHERE 2 <= (            SELECT COUNT (*) FROM INCR
WHERE INCR.EMPNO = EMP.EMPNO
GROUP BY EMPNO);
```

Please note the use of subquery that counts the number of increments given to each employee stored in the INCR table. Please note the comparison, instead of ……>=2, we have written reverse of it as 2 <= …..

DELETE Command

In the following example, the deletion will be performed in EMP table.No deletion will be performed in the INCR table.

Example 25: Delete the records of employees who have got no increment.

```
DELETE FROM EMP
WHERE EMPNO NOT IN (SELECT EMPNO FROM INCR);
```

☞ Check Your Progress 1

1) What are the advantages of SQL? Can you think of some disadvantages of SQL?

…….

…….

…….

……..

2) Create the Room, Booking, and Guest tables using the integrity enhancement features of SQL with the following constraints:
   (a) Type must be one of Single, Double, or Family.
   (b) Price must be between Rs.100/- and Rs.1000/-.
   (c) roomNo must be between 1 and 100.
   (d) booking dateFrom and dateTo must be greater than today's date.
   (e) The same room cannot be double booked.
   (f) The same guest cannot have overlapping bookings.

…….

…….

…….

…….

…….

……..

3) Define the function of each of the clauses in the SELECT statement. What are the restrictions imposed on these clauses?

……..

……..

…….

…….

……..

4) Consider the supplier relations.

S

SNO (Supplier Number)	SNAME (Supplier Name)	STATUS	CITY
S1	Prentice Hall	30	Calcutta
S2	McGraw Hill	30	Mumbai
S3	Wiley	20	Chennai
S4	Pearson	40	Delhi
S5	Galgotia	10	Delhi

SP

SNO	PNO (Part Number)	Quantity
S1	P1	300
S1	P2	200
S2	P1	100
S2	P2	400
S3	P2	200
S4	P2	200

a) Get supplier numbers for suppliers with status > 20 and city is Delhi
b) Get Supplier Numbers and status for suppliers in Delhi in descending order of status.
c) Get all pairs of supplier numbers such that the two suppliers are located in the same city. (Hint: It is retrieval involving join of a table with itself.)
d) Get unique supplier names for suppliers who supply part P2 without using IN operator.
e) Give the same query above by using the operator IN.
f) Get part numbers supplied by more than one supplier. (Hint : It is retrieval with sub-query block, with inter block reference and same table involved in both blocks)
g) Get supplier numbers for suppliers who are located in the same city as supplier S1. (Hint: Retrieval with sub query and unqualified comparison operator).
h) Get supplier names for suppliers who supply part P1. (Hint : Retrieval using EXISTS)
i) Get part numbers for parts whose quantity is greater than 200 or are currently supplied by S2. (Hint: It is a retrieval using union).
j) Suppose for the supplier S5 the value for status is NULL instead of 10. Get supplier numbers for suppliers greater than 25. (Hint: Retrieval using NULL).
k) Get unique supplier numbers supplying parts. (Hint: This query is using the built-in function count).
l) For each part supplied, get the part number and the total quantity supplied for that part. (Hint: The query using GROUP BY).
m) Get part numbers for all parts supplied by more than one supplier. (Hint: It is GROUP BY with HAVING).
n) For all those parts, for which the total quantity supplied is greater than 300, get the part number and the maximum quantity of the part supplied in a single supply. Order the result by descending part number within those maximum quantity values.
o) Double the status of all suppliers in Delhi. (Hint: UPDATE Operation).
p) Let us consider the table TEMP has one column, called PNO. Enter into TEMP part numbers for all parts supplied by S2.
q) Add part p7.
r) Delete all the suppliers in Mumbai and also the suppliers concerned.

……

……

1.5 DATA CONTROL

The data control basically refers to commands that allow system and data privileges to be passed to various users. These commands are normally available to database administrator. Let us look into some data control language commands:

Create a new user:

CREATE USER < user name > IDENTIFIED BY < Password>

Example 26:

CREATE USER MCA12 IDENTIFIED BY W123

Grant: It is used to provide database access permission to users. It is of two types (1) system level permission (2) Object level permission.

Example 27:

GRANT CREATE SESSION TO MCA12;
(This command provides system level permission on creating a session – not portable)

GRANT SELECT ON EMP TO MCA12;
(Object level permission on table EMP)

GRANT SELECT, INSERT, UPDATE, DELETE ON EMP TO MCA12;

GRANT SELECT, UPDATE ON EMP TO MCA12, MCA13;
(Two users)
GRANT ALL ON EMP TO PUBLIC;
(All permission to all users, do not use it. It is very dangerous for database)

Revoke: It is used to cancel the permission granted.

Example 28:

REVOKE ALL ON EMP FROM MCA12;
(All permissions will be cancelled)
You can also revoke only some of the permissions.

Drop: A user-id can be deleted by using drop command.

Example 29:

DROP USER MCA12;

Accessing information about permissions to all users

(1) Object level permissions: With the help of data dictionary you can view the permissions to user. Let us take the table name from oracle. In oracle the name of the table containing these permissions is user_tab_privs.
DESCRIBE USER_TAB_PRIVS ;
SELECT * FROM USER_TAB_PRIVS;

(2) System level permissions: With the help of data dictionary you can see them. Let us take the table name as user_sys_privs (used in oracle).
DESCRIBE USER_SYS_PRIVS ;

SELECT * FROM USER_SYS_PRIVS ;

All these commands are very specific to a data base system and may be different on different DBMS.

1.6 DATABASE OBJECTS: VIEWS, SEQUENCES, INDEXES AND SYNONYMS

Some of the important concepts in a database system are the views and indexes. Views are a mechanism that can support data independence and security. Indexes help in better query responses. Let us discuss about them along with two more concepts sequences and synonyms in more details.

1.6.1 Views

A view is like a window through which data from tables can be viewed or changed. The table on which a view is based is called Base table. The view is stored as a SELECT statement in the data dictionary. When the user wants to retrieve data, using view. Then the following steps are followed.

1) View definition is retrieved from data dictionary table. For example, the view definitions in oracle are to be retrieved from table name USER-VIEWS.
2) Checks access privileges for the view base table.
3) Converts the view query into an equivalent operation on the underlying base table

Advantages:

- It restricts data access.
- It makes complex queries look easy.
- It allows data independence.
- It presents different views of the same data.

Type of views:

Simple views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain Functions	No	Yes
Contain groups of data	No	Yes
Data Manipulation	IS allowed	Not always

Let us look into some of the examples. To see all the details about existing views in Oracle:

SELECT* FROM USER_VIEWS;

Creating a view:

- A query can be embedded within the CREATE VIEW STATEMENT
- A query can contain complex select statements including join, groups and sub-queries
- A query that defines the view **cannot contain** an order by clause.
- DML operation (delete/modify/add) cannot be applied if the view contains any of the following:

Delete (You can't delete if view contains following)	*Modify (you cannot modify if view contains following)*	*Insert (you cannot insert if view contains following)*
• Group functions • A group by clause • A distinct keyword	• Group functions • A group by clause • A distinct keyword • Columns defined by Expressions	• Group functions • A group by clause • A distinct keyword • Columns defined by Expressions • There are Not Null Columns in the base tables that are not selected by view.

Example 30: Create a view named employee salary having minimum, maximum and average salary for each department.

```
CREATE VIEW EMPSAL (NAME, MINSAL, MAXSAL, AVGSAL) AS
        SELECT D.DNAME, MIN(E.SAL),MAX(E.SAL),AVG(E.SAL)
        FROM EMP E, DEPT D
        WHERE E.DEPTNO=D.DEPTNO
         GROUP BY D.DNAME;
```

To see the result of the command above you can give the following command:

```
SELECT * FROM EMPSAL;
You may get some sample output like:
NAME            MINSAL   MAXSA    AVGSAL
--------------  ---------  ---------  -------------
ACCOUNTING      1300       5000       2916.6667
RESEARCH        800        3000       2175
SALES           950        2850       1566.6667
```

To see the structure of the view so created, you can give the following command:

```
DESCRIBE EMPSAL;
 Name                          Null?    Type
---------------              ------------  ---------------------
 NAME                                       VARCHAR2 (14)
 MINSAL                                      NUMBER
 MAXSAL                                       NUMBER
 AVGSAL                                      NUMBER
```

Creating views with check option: This option restricts those updates of data values that cause records to go off the view. The following example explains this in more detail:

Example 31: To create a view for employees of Department = 30, such that user cannot change the department number from the view:

```
CREATE OR REPLACE VIEW EMPD30 AS
        SELECT EMPNO EMPL_NUM, ENAME NAME, SAL SALARY
        FROM EMP
        WHERE DEPTNO=30
        WITH CHECK OPTION;
```

Now the user cannot change the department number of any record of view EMPD30. If this change is allowed then the record in which the change has been made will go off the view as the view is only for department-30. This restricted because of use of WITH CHECK OPTION clause

Creating views with Read only option: In the view definition this option is used to ensure that no DML operations can be performed on the view.

Creating views with Replace option: This option is used to change the definition of the view without dropping and recreating it or regranting object privileges previously granted on it.

1.6.2 Sequences

Sequences:

- automatically generate unique numbers
- are sharable
- are typically used to create a primary key value
- replace application code
- speed up the efficiency of accessing sequence Values when cached in memory.

Example 32: Create a sequence named SEQSS that starts at 105, has a step of 1 and can take maximum value as 2000.

```
CREATE SEQUENCE SEQSS
START WITH 105
INCREMENT BY 1
 MAX VALUE 2000;
```

How the sequence so created is used? The following sequence of commands try to demonstrate the use of the sequence SEQSS.

Suppose a table person exists as:

```
        SELECT * FROM PERSON;
Output:        CODE     NAME          ADDRESS
                ------  --------------- ---------------
                104     RAMESH     MUMBAI
```

Now, if we give the command:

```
        INSERT INTO PERSON
                VALUES (SEQSS.NEXTVAL, &NAME, &ADDRESS)
```

On execution of statement above do the following input:

```
        Enter value for name: 'Rakhi'
        Enter value for address: 'New Delhi'
```

Now, give the following command to see the output:

```
        SELECT * FROM PERSON;
          CODE NAME          ADDRESS
        --------- --------------- -----------------
            104 RAMESH         MUMBAI
            105 Rakhi          NEW DELHI
```

The descriptions of sequences such as minimum value, maximum value, step or increment are stored in the data dictionary. For example, in oracle it is stored in the table user_sequences. You can see the description of sequences by giving the SELECT command.

Gaps in sequence values can occur when:

- A rollback occurs that is when a statement changes are not accepted.
- The system crashes
- A sequence is used in another table.

Modify a sequence:

```
ALTER SEQUENCE SEQSS
INCREMENT 2
MAXVALUE 3000;
```

Removing a sequence:

```
DROP SEQUENCE SEQSS;
```

1.6.3 Indexes and Synonyms

Some of the basic properties of indexes are:

- An Index is a schema Object
- Indexes can be created explicitly or automatically
- Indexes are used to speed up the retrieval of rows
- Indexes are logically and physically independent of the table. It means they can be created or dropped at any time and have no effect on the base tables or other indexes.
- However, when a table is dropped corresponding indexes are also dropped.

Creation of Indexes

Automatically: When a primary key or Unique constraint is defined in a table definition then a unique index is created automatically.

Manually: User can create non-unique indexes on columns to speed up access time to rows.

Example 33: The following commands create index on employee name and employee name + department number respectively.

```
CREATE INDEX EMP_ENAME_IDX ON EMP (ENAME);
CREATE INDEX EMP_MULTI_IDX ON EMP (ENAME, DEPTNO);
```

Finding details about created indexes: The data dictionary contains the name of index, table name and column names. For example, in Oracle a user-indexes and user-ind-columns view contains the details about user created indexes.

Remove an index from the data dictionary:

```
DROP INDEX EMP_ENAME_IDX;
```

Indexes cannot be modified.

Synonyms

It permits short names or alternative names for objects.

Example 34:

```
CREATE SYNONYM D30
FOR EMPD30;
```

Now if we give command:

```
SELECT * FROM D30;
```

The output will be:

```
NAME                 MINSAL    MAXSAL    AVGSAL
-------------------- ------------ -------------  --------------
ACCOUNTING           1300       5000       2916.6667
RESEARCH             800        3000       2175
SALES                950        2850       1566.6667
```

Removing a Synonym:

```
DROP SYNONYM D30;
```

1.7 TABLE HANDLING

In RDBMS more than one table can be handled at a time by using join operation. Join operation is a relational operation that causes two tables with a common domain to be combined into a single table or view. SQL specifies a join implicitly by referring the matching of common columns over which tables are joined in a WHERE clause. Two tables may be joined when each contains a column that shares a common domain with the other. The result of join operation is a single table. Selected columns from all the tables are included. Each row returned contains data from rows in the different input tables where values for the common columns match. An important rule of table handling is that there should be one condition within the WHERE clause for each pair of tables being joined. Thus if two tables are to be combined, one condition would be necessary, but if three tables (X, Y, Z) are to be combined then two conditions would be necessary because there are two pairs of tables (X-Y and Y-Z) OR (X-Z and Y-Z), and so forth. There are several possible types of joins in relational database queries. Four types of join operations are described below:

(1) **Equi Join**: A join in which the joining condition is based on equality between values in the common columns. Common columns appear (redundantly) in the result table. Consider the following relations:

- customer (<u>custid</u>, custname, ……..) and
- order (custid, ordered,……..)

<u>Example 35:</u> What are the names of all customers who have placed orders?

The required information is available in two tables, customer and order. The SQL solution requires joining the two table using equi join.

```
SELECT CUSTOMER.CUTOID, ORDER.CUSTOID,
                                    CUSTONAME, ORDERID
FROM CUSTOMER, ORDER
WHERE CUSTOMER.CUSTOID=ORDER.CUSTOID;
```

The output may be:

Customer.custoid	order.custoid	custoname	orderid
10	10	Pooja Enterprises	1001
12	12	Estern Enterprises	1002
3	3	Impressions	1003

(2) **Natural Join**: It is the same like Equi join except one of the duplicate columns is eliminated in the result table. The natural join is the most commonly used form of join operation.

<u>Example 36:</u>

```
SELECT CUSTOMER.CUTOID, CUSTONAME, ORDERID
FROM CUSTOMER, ORDER
WHERE CUSTOMER.CUSTOID=ORDER.CUSTOID;
```

Output:

custoid	custoname	orderid
10	Pooja Enterprises	1001
12	Estern Enterprises	1002
3	Impressions	1003

(3) **Outer Join**: The use of Outer Join is that it even joins those tuples that do not have matching values in common columns are also included in the result table. Outer join places null values in columns where there is not a match between

tables. A condition involving an outer join is that it cannot use the IN operator or cannot be linked to another condition by the OR operator.

Example 37: The following is an example of left outer join (which only considers the non-matching tuples of table on the left side of the join expression).

```
SELECT CUSTOMER.CUTOID, CUSTONAME, ORDERID
FROM CUSTOMER LEFT OUTER JOIN ORDER
WHERE CUSTOMER.CUSTOID =   ORDER.CUSTOID;
```

Output: The following result assumes a CUSTID in CUSTOMER table who have not issued any order so far.

```
CUSTOID            CUSTONAME   ORDERID
-------------------- ---------------------- ------------
   10              Pooja Enterprises      1001
   12              Estern Enterprises     1002
   3               Impressions            1003
   15              South Enterprises      NULL
```

The other types of outer join are the Right outer join or complete outer join.

(4) **Self-Join**: It is a join operation where a table is joined with itself. Consider the following sample partial data of EMP table:

EMPNO	ENAME	MGRID	…..
1	Nirmal	4	
2	Kailash	4	
3	Veena	1	
4	Boss	NULL	
…..	…..	…	

Example 38: Find the name of each employee's manager name.

```
SELECT WORKER.ENAME || 'WORK FOR' || MANAGER.ENAME
FROM EMP WORKER, EMP MANAGER
WHERE WORKER.MGR=MANAGER.EMPNO;
```

Output:

Nirmal works for Boss
Kailash works for Boss
Veena works for Nirmal

☞ Check Your Progress 2

1) Discuss how the Access Control mechanism of SQL works.

……..

……..

……..

……..

2) Consider Hotel schema consisting of three tables Hotel, Booking and Guest,

```
CREATE TABLE Hotel
        hotelNo        HotelNumber        NOT NULL,
        hotelName      VARCHAR(20)        NOT NULL,
        city           VARCHAR(50)        NOT NULL,

        PRIMARY KEY (hotelNo));
```

```
CREATE TABLE Booking(
hotelNo        HotelNumbers        NOT NULL,
guestNo        GuestNumbers        NOT NULL,
dateFrom       BookingDate         NOT NULL,
dateTo         BookingDate         NULL,
roomNo         RoomNumber          NOT NULL,

PRIMARY KEY (hotelNo, guestNo, dateFrom),
FOREIGN KEY (hotelNo) REFERENCES Hotel
        ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (guestNo) REFERENCES Guest
        ON DELETE NO ACTION ON UPDATE CASCADE,
FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
        ON DELETE NO ACTION ON UPDATE CASCADE);
CREATE TABLE Guest(
guestNo        GuestNumber         NOT NULL,
guestName      VARCHAR(20)         NOT NULL,
guestAddress   VARCHAR(50)         NOT NULL
PRIMARY KEY (guestno));

CREATE TABLE Room(
roomNo         RoomNumber          NOT NULL,
hotelNo        HotelNumbers        NOT NULL,
type           RoomType            NOT NULL DEFAULT 'S'
price          RoomPrice           NOT NULL,
PRIMARY KEY (roomNo, hotelNo),
FOREIGN KEY (hotelNo) REFERENCES Hotel
ON DELETE CASCADE ON UPDATE CASCADE);
```

Create a view containing the hotel name and the names of the guests staying at the hotel.

……

……

……

……

……

……

……

……..

3) Give the users Manager and Director full access to views HotelData and BookingOutToday, with the privilege to pass the access on to other users.

……

……

……

……

……

1.8 NESTED QUERIES

By now we have discussed the basic commands including data definition and data manipulations. Now let us look into some more complex queries in this section.

Sub-queries

Some of the basic issues of sub-queries are:

- A sub-query is a SELECT statement that is embedded in a clause of another SELECT statement. They are often referred to as a NESTED SELECT or SUB SELECT or INNER SELECT.

- The sub-query (inner query) executes first before the main query. The result of the sub-query is used by the main query (outer query).

- Sub-query can be placed in WHERE or HAVING or FROM clauses.

- Format of using sub-queries:

```
SELECT<select_list>
FROM<table>
WHERE expr OPERATOR
            (SELECT <select_list>
            FROM <TABLE>WHERE);
```

  Operator includes a comparison operator (single or multiple row operators)
  Single row operators: >, =, >=, <, <=, <>
  Multiple row operators: IN, ANY, ALL

- Order by clause cannot be used in sub-query, if specified it must be the last clause in the main select statement.

- Types of sub-queries:

  - Single row sub-query: It returns only one row from the inner select statement.
  - Multiple row sub-queries: it returns more than one row from the inner select statement
  - Multiple column sub-queries: it returns more than one column from the inner select statement.

  Single row operators are used with single row sub queries and multiple row operators are used with multiple row sub queries.

- The Outer and Inner queries can get data from different tables.

- Group Functions can be used in sub queries.

Consider the following partial relation EMP. Let us create some sub-queries for them

EMPNO	ENAME	JOB	SAL	DEPTNO
7566	Nirmal	MANAGER	2975	10
7788	Kailash	ANALYST	3000	10
7839	Karuna	PRESIDENT	5000	20
7902	Ashwin	ANALYST	3000	20
7905	Ashwini	MANAGER	4000	20

Example 39: Get the details of the person having the minimum salary.

```
SELECT ENAME, JOB, SAL
FROM EMP
 WHERE SAL =   (      SELECT MIN (SAL)
                      FROM EMP);
```

Output:

ENAME	JOB	SAL
Nirmal	MANAGER	2975

Example 40: Display the employees whose job title is the same as that of employee 7566 and salary is more than the salary of employee 7788.

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB  =   (      SELECT JOB
                      FROM EMP
                      WHERE EMPPNO = 7566)
          AND SAL >  (  SELECT SAL
                      FROM EMP
                      WHERE EMPPNO=7788);
```

Output: Job title for the employee 7566 happens to be 'MANAGER')

ENAME	JOB
Ashwini	MANAGER

Having Clause with sub queries: First we recollect the GROUP BYclause. The following query finds the minimum salary in each department.

```
SELECT DEPTNO, MIN(SAL)
FROM.EMP
GROUP BY DEPTNO;
```

Output:

DEPTNO	SAL
10	2975
20	3000

Example 41: To find the minimum salary in those departments whose minimum salary is greater than minimum salary of department number 10.

```
SELECT DEPTNO, MIN(SAL)
FROM EMP
GROUP BY DEPTNO
HAVING MIN(SAL) >  (        SELECT MIN (SAL)
                            FROM EMP
                            WHERE DEPTNO = 10);
```

Output:

DEPTNO	SAL
20	3000

Example 42: Find the name, department number and salary of employees drawing minimum salary in that department.

```
SELECT ENAME, SAL, DEPTNO
FROM EMP
WHERE SAL IN (SELECT MIN (SAL)
                    FROM EMP
                    GROUP BY DEPTNO);
```

Output:

ENAME	SAL	DEPTNO
Nirmal	2975	10
Ashwin	3000	20

Find the salary of employees employed as an ANALYST
SELECT SAL FROM EMPWHERE JOB= ' ANALYST '

Output:

SAL
3000
3000

Example 43: Find the salary of employees who are not 'ANALYST' but get a salary less than or equal to any person employed as 'ANALYST'.

```
SELECT EMPNO, ENAME, JOB, SAL
FROMEMP
WHERE SAL <= ANY ( SELECT SAL
                   FROM EMP WHERE JOB = 'ANALYST' )
    AND JOB<>'ANALYST' ;
```

Output:

EMPNO	ENAME	JOB	SAL
7566	Nirmal	MANAGER	2975

Find the average salary in each department

SELECT DEPTNO, AVG(SAL) FROM EMP GROUP BY DEPTNO;
Result:

DEPTNO	SAL
10	2987.5
20	4000

Example 44: Find out the employee who draws a salary more than the average salary of all the departments.

```
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE SAL> ALL (SELECT AVG (SAL)
                FROM EMP
                GROUP BY DEPTNO);
```

Output:

EMPNO	ENAME	JOB	SAL
7839	Karuna	PRESIDENT	5000

Example 45: Find the employee name, salary, department number and average salary of his/her department, for those employees whose salary is more than the average salary of that department.

```
SELECT A.ENAME, A.SAL, A.DEPTNO, B.AVGSAL
FROM EMP A, (   SELECT DEPTNO, AVG (SAL)  AVGSAL
                FROM EMP
                GROUP BY DEPTNO) B
WHERE A.DEPTNO=B.DEPTNO AND A.SAL> B. AVGSAL;
```

Output:

ENAME	SAL	DEPTNO	AVGSAL
Kailash	3000	10	2987.5
Karuna	5000	20	4000

Multiple column Queries:
Syntax:

```
SELECT COLUMN1, COL2,……
FROM TABLE
WHERE (COLUMN1, COL2, …) IN
                (SELECT COLUMN1, COL2,….
                 FROM TABLE
                 WHERE <CONDITION>);
```

Example 46: Find the department number, name, job title and salary of those people who have the same job title and salary as those are in department 10.

```
SELECT DEPTNO,ENAME, JOB, SAL
FROM EMP
WHERE (JOB, SAL) IN  (      SELECT JOB, SAL
                            FROM EMP
                            WHERE DEPTNO=10);
```

Output:

DEPTNO	ENAME	JOB	SAL
10	Nirmal	MANAGER	2975
10	Kailash	ANALYST	3000
20	Ashwin	ANALYST	3000

☞ Check Your Progress 3

1) What is the difference between a sub-query and a join? Under what circumstances would you not be able to use a sub-query?

……

……

……

2) Use the Hotel schema defined in question number 2 (Check Your Progress 2) and answer the following queries:

- List the names and addresses of all guests in Delhi, alphabetically ordered by name.
- List the price and type of all rooms at the GRAND Hotel.
- List the rooms that are currently unoccupied at the Grosvenor Hotel.
- List the number of rooms in each hotel.
- What is the most commonly booked room type for hotel in Delhi?
- Update the price of all rooms by 5%.

……

……

……

3) Consider the following Relational database.

Employees (eno, ename, address, basic_salary)
Projects (Pno, Pname, enos-of-staff-alotted)
Workin (pno, eno, pjob)

Two queries regarding the data in the above database have been formulated in SQL. Describe the queries in English sentences.

(i)
```
SELECT ename
FROM employees
WHERE eno IN ( SELECT eno
               FROM workin
               GROUP BY eno
 HAVING COUNT (*) =  (SELECT COUNT (*) FROM projects));
```

(ii)
```
SELECT Pname
FROM projects
WHERE Pno IN ( SELECT Pno
FROM projects
MINUS
GROUP BY eno
(SELECT DISTINCT Pno FROM workin));
```

……..

……..

…….

……..

……..

…….

1.9 SUMMARY

This unit has introduced the SQL language for relational database definition, manipulation and control. The SQL environment includes an instance of an SQL DBMS with accessible databases and associated users and programmers. Each schema definition that describes the database objects is stored in data dictionary/ system catalog. Information contained in system catalog is maintained by the DBMS itself, rather than by the users of DBMS.

The data definition language commands are used to define a database, including its creation and the creation of its tables, indexes and views. Referential integrity constraints are also maintained through DDL commands. The DML commands of SQL are used to load, update and query the database. DCL commands are used to establish user access to the database. SQL commands directly affect the base tables, which contain the raw data, or they may affect database view, which has been created. The basic syntax of an SQL SELECT statement contains the following keywords: SELECT, FROM, WHERE, ORDER BY, GROUP BY and HAVING.

SELECT determines which attributes will be displayed in the query result table. FROM determines which tables or views will be used in the query. WHERE sets the criteria of the query, including any joins of multiple tables, which are necessary. ORDER BY determines the order in which the result will be displayed. GROUP BY is used to categorize results. HAVING is used to impose condition with GROUP BY.

1.10 SOLUTIONS / ANSWERS

Check Your Progress 1

1)

Advantages

- A standard for database query languages
- (Relatively) Easy to learn
- Portability
- SQL standard exists
- Both interactive and embedded access
- Can be used by specialist and non-specialist.

Yes, SQL has disadvantages. However, they are primarily more technical with reference to Language features and relational model theories. We are just putting them here for reference purposes.

Disadvantages

- Impedance mismatch – mixing programming paradigms with embedded access
- Lack of orthogonality – many different ways to express some queries
- Language is becoming enormous (SQL-92 is 6 times larger than predecessor)
- Handling of nulls in aggregate functions is not portable
- Result tables are not strictly relational – can contain duplicate tuples, imposes an ordering on both columns and rows.

2. CREATE DOMAIN RoomType AS CHAR(1) ……*[Constraint (a)]*
CHECK(VALUE IN (S, F, D));

CREATE DOMAIN HotelNumbers AS HotelNumber
CHECK(VALUE IN (SELECT hotelNo FROM Hotel));
[An additional constraint for hotel number for the application]

CREATE DOMAIN RoomPrice AS DECIMAL(5, 2)
CHECK(VALUE BETWEEN 1000 AND 10000);

CREATE DOMAIN RoomNumber AS VARCHAR(4)
CHECK(VALUE BETWEEN '1' AND '100');

[Constraint (c), one additional character is kept instead of 3 we have used 4characters but no space wastage as varchar]

```
CREATE TABLE Room(
        roomNo      RoomNumber      NOT NULL,
        hotelNo     HotelNumbers    NOT NULL,
        type        RoomType        NOT NULL DEFAULT S,
        price       RoomPrice       NOT NULL,
        PRIMARY KEY (roomNo, hotelNo),
        FOREIGN KEY (hotelNo) REFERENCES Hotel
                ON DELETE CASCADE ON UPDATE CASCADE);
        CREATE DOMAIN GuestNumber AS CHAR(4);
```

```
CREATE TABLE Guest(
        guestNo         GuestNumber         NOT NULL,
        guestName       VARCHAR(20)         NOT NULL,
        guestAddress    VARCHAR(50)         NOT NULL);

CREATE DOMAIN GuestNumbers AS GuestNumber
        CHECK(VALUE IN (SELECT guestNo FROM Guest));
                [A sort of referential constraint expressed within domain]

CREATE DOMAIN BookingDate AS DATETIME
        CHECK(VALUE > CURRENT_DATE);            [constraint (d) ]

CREATE TABLE Booking(
        hotelNo         HotelNumbers        NOT NULL,
        guestNo         GuestNumbers        NOT NULL,
        dateFrom        BookingDate         NOT NULL,
        dateTo          BookingDate         NULL,
        roomNo          RoomNumber          NOT NULL,
        PRIMARY KEY (hotelNo, guestNo, dateFrom),
        FOREIGN KEY (hotelNo) REFERENCES Hotel
                ON DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (guestNo) REFERENCES Guest
                ON DELETE NO ACTION ON UPDATE CASCADE,
        FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
                ON DELETE NO ACTION ON UPDATE CASCADE,
        CONSTRAINT RoomBooked
                CHECK (NOT EXISTS ( SELECT *
                        FROM Booking b
                        WHERE b.dateTo > Booking.dateFrom AND
                        b.dateFrom < Booking.dateTo AND
                        b.roomNo = Booking.roomNo AND
                        b.hotelNo = Booking.hotelNo)),
        CONSTRAINT GuestBooked
                CHECK (NOT EXISTS ( SELECT *
                        FROM Booking b
                        WHERE b.dateTo > Booking.dateFrom AND
                        b.dateFrom < Booking.dateTo AND
                        b.guestNo = Booking.guestNo)));
```

3.

FROM	Specifies the table or tables to be used.
WHERE	Filters the rows subject to some condition.
GROUP BY	Forms groups of rows with the same column value.
HAVING	Filters the groups subject to some condition.
SELECT	Specifies which columns are to appear in the output.
ORDER BY	Specifies the order of the output.

If the SELECT list includes an aggregate function and no GROUP BY clause is being used to group data together, then no item in the SELECT list can include any reference to a column unless that column is the argument to an aggregate function.

When GROUP BY is used, each item in the SELECT list must be single-valued per group. Further, the SELECT clause may only contain:

- Column names.
- Aggregate functions.
- Constants.
- An expression involving combinations of the above.

All column names in the SELECT list must appear in the GROUP BY clause unless the name is used only in an aggregate function.

3. Please note that some of the queries are sub-queries and queries requiring join. The meaning of these queries will be clearer as you proceed further with the Unit.

a)
```
SELECT SNO
FROM S
WHERE CITY = 'Delhi'
AND STATUS > 20;
```

Result:

SNO
S4

b)
```
SELECT SNO, STATUS
FROM S
WHERE CITY = 'Delhi'
ORDER BY STATUS DESC;
```

Result:

SNO	STATUS
S4	40
S5	10

c)
```
SELECT FIRST.SNO, SECOND.SNO
FROM S FIRST, S SECOND
WHERE FIRST.CITY = SECOND.CITY AND FIRST.SNO <
SECOND.SNO;
```

Please note that if you do not give the condition after AND you will get some unnecessary tuples such as: (S4, S4), (S5, S4) and (S5, S5).

Result:

SNO	SNO
S4	S5

d)
```
SELECT DISTINCT SNAME
FROM S, SP
WHERE S.SNO = SP.SNO
AND SP.PNO = 'P2';
```

Result:

SNAME
Prentice Hall

McGraw Hill
Wiley
Pearson

```
OR
SELECT SNAME
FROM S
WHERE SNO = ANY (   SELECT SNO
                    FROM SP
                    WHERE PNO = 'P2');
```

e)
```
SELECT SNAME
FROM S
WHERE SNO IN (      SELECT SNO
                FROM SP
                WHERE PNO = 'P2' );
```

f)
```
SELECT DISTINCT PNO
FROM SP SPX
WHERE PNO IN (      SELECT PNO
                FROM SP
            WHERE SP.SNO = SPX.SNO AND SPX.SNO < SP.SNO );
```

This query can also be answered using count and group by. Please formulate that.

Result:

PNO
P1
P2

g)
```
SELECT SNO
FROM S
WHERE CITY = (      SELECT CITY
                    FROM S
                    WHERE SNO = 'S1' );
```

Result:

SNO
S1

h)
```
SELECT SNAME
FROM S
WHERE EXISTS (      SELECT *
                    FROM SP
                    WHERE SNO = S.SNO AND PNO = 'P1');
```

Result:

SNAME
Prentice Hall
McGraw Hill

i)
```
SELECT PNO
FROM SP
WHERE QUANTITY > 200 UNION (   SELECT PNO
                               FROM SP
                               WHERE SNO = S2 );
```

Result:

PNO
P1
P2

j)
```
SELECT SNO
FROM S
WHERE STATUS > 25 OR STATUS IS NULL;
```

Result:

SNO
S1
S2
S4
S5

k)
```
SELECT COUNT (DISTINCT SNO)
FROM SP;
```

Result: 4

l)
```
SELECT PNO, SUM(QUANTITY)
FROM SP
GROUP BY PNO;
```

Result:

PNO	SUM
P1	400
P2	1000

m)
```
SELECT PNO
FROM SP
GROUP BY PNO
HAVING COUNT(*) > 1 ;
```

The query is a same as that of part (f)

n)
```
SELECT PNO, MAX(QUANTITY)
FROM SP
WHERE QUANTITY > 200
GROUP BY PNO
HAVING SUM(QUANTITY) > 300
ORDER BY 2, PNO DESC;
```

o)
```
UPDATE  S
SET STATUS = 2 * STATUS
WHERE CITY = 'Delhi';
```

p)
```
INSERT INTO TEMP
SELECT PNO
FROM SP
WHERE SNO = 'S2';
```

q)
```
INSERT INTO SP( SNO,PNO,QUANTITY) < 'S5','P7',100> ;
```

Please note that part cannot be added without a supply in the present case.

Actually there should be another table for Parts

r) DELETE S, SP
WHERE SNO = (SELECT SNO
FROM S
WHERE CITY = 'Mumbai');

Check Your Progress 2

1) Each user has an authorization identifier (allocated by DBA).
Each object has an owner. Initially, only owner has access to an object but the owner can pass privileges to carry out certain actions on to other users via the GRANT statement and take away given privileges using REVOKE.

2) CREATE VIEW HotelData(hotelName, guestName) AS
SELECT h.hotelName, g.guestName
FROM Hotel h, Guest g, Booking b
WHERE h.hotelNo = b.hotelNo AND g.guestNo = b.guestNo AND
b.dateFrom <= CURRENT_DATE AND
b.dateTo >= CURRENT_DATE;

3) GRANT ALL PRIVILEGES ON HotelData
TO Manager, Director WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON BookingOutToday
TO Manager, Director WITH GRANT OPTION;

Check Your Progress 3

1) With a sub-query, the columns specified in the SELECT list are restricted to one table. Thus, cannot use a sub-query if the SELECT list contains columns from more than one table. But with a join operation SELECT list contains columns from more than two tables.

2) Answers of the queries are:

- SELECT guestName, guestAddress FROM Guest
WHERE address LIKE '%Delhi%'
ORDER BY guestName;

- SELECT price, type FROM Room
WHERE hotelNo =
(SELECT hotelNo FROM Hotel
WHERE hotelName = 'GRAND Hotel');

- SELECT * FROM Room r
WHERE roomNo NOT IN
(SELECT roomNo FROM Booking b, Hotel h
WHERE (dateFrom <= CURRENT_DATE AND
dateTo >= CURRENT_DATE) AND
b.hotelNo = h.hotelNo AND hotelName = 'GRAND Hotel');
- SELECT hotelNo, COUNT(roomNo) AS count
FROM Room
GROUP BY hotelNo;

- SELECT MAX(X)

```
FROM ( SELECT type, COUNT(type) AS X
FROM Booking b, Hotel h, Room r
WHERE r.roomNo = b.roomNo AND b.hotelNo = h.hotelNo AND
        city = 'LONDON'
GROUP BY type);
```

- UPDATE Room SET price = price*1.05;

3) (i) – Give names of employees who are working on all projects.

(ii) - Give names of the projects which are currently not being worked upon.

1.11 FURTHER READINGS

Fundamentals of DatabaseSystems; Almasri and Navathe; Pearson Education Limited; Fourth Edition; 2004.
A Practical Approach to Design, Implementation, and Management; Thomas Connolly and Carolyn Begg; Database Systems, Pearson Education Limited; Third Edition; 2004.
The Complete Reference; Kevin Lonely and George Koch; Oracle 9i, Tata McGraw-Hill; Fourth Edition; 2003.
Jeffrey A. Hoffer, Marry B. Prescott and Fred R. McFadden; Modern Database Management; Pearson Education Limited; Sixth Edition; 2004.

UNIT 2 TRANSACTIONS AND CONCURRENCY MANAGEMENT

Structure **Page Nos.**

2.0	Introduction	35
2.1	Objectives	35
2.2	The Transactions	35
2.3	The Concurrent Transactions	38
2.4	The Locking Protocol	42
	2.4.1 Serialisable Schedules	
	2.4.2 Locks	
	2.4.3 Two Phase Locking (2PL)	
2.5	Deadlock and its Prevention	49
2.6	Optimistic Concurrency Control	51
2.7	Summary	53
2.8	Solutions/ Answers	54

2.0 INTRODUCTION

One of the main advantages of storing data in an integrated repository or a database is to allow sharing of it among multiple users. Several users access the database or perform transactions at the same time. What if a user's transactions try to access a data item that is being used /modified by another transaction? This unit attempts to provide details on how concurrent transactions are executed under the control of DBMS. However, in order to explain the concurrent transactions, first we must describe the term transaction.

Concurrent execution of user programs is essential for better performance of DBMS, as concurrent running of several user programs keeps utilizing CPU time efficiently, since disk accesses are frequent and are relatively slow in case of DBMS. Also, a user's program may carry out many operations on the data returned from DB, but DBMS is only concerned about what data is being read /written from/ into the database. This unit discusses the issues of concurrent transactions in more detail.

2.1 OBJECTIVES

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

- describe the term CONCURRENCY;
- define the term transaction and concurrent transactions;
- discuss about concurrency control mechanism;
- describe the principles of locking and serialisability, and
- describe concepts of deadlock & its prevention.

2.2 THE TRANSACTIONS

A transaction is defined as the unit of work in a database system. Database systems that deal with a large number of transactions are also termed as transaction processing systems.

What is a transaction? Transaction is a unit of data processing. For example, some of the transactions at a bank may be withdrawal or deposit of money; transfer of money from A's account to B's account etc. A transaction would involve manipulation of one

or more data values in a database. Thus, it may require reading and writing of database value. For example, the withdrawal transactions can be written in pseudo code as:

Example 1:

```
; Assume that we are doing this transaction for person
;  whose account number is X.

TRANSACTION WITHDRAWAL (withdrawal_amount)
Begin transaction
        IF X exist then
                READ  X.balance
                IF      X.balance > withdrawal_amount
                        THEN  SUBTRACT withdrawal_amount
                                WRITE X.balance
                                 COMMIT
                ELSE
                DISPLAY "TRANSACTION CANNOT BE PROCESSED"
        ELSE  DISPLAY "ACCOUNT X DOES NOT EXIST"
End transaction;
```

Another similar example may be transfer of money from Account no x to account number y. This transaction may be written as:

Example 2:

```
; transfers transfer_amount from x's account to y's account
; assumes x&y both accounts exist

TRANSACTION (x, y, transfer_amount)
Begin transaction
        IF X AND Y exist then
                READ x.balance
                IF x.balance > transfer_amount THEN
                        x.balance = x.balance – transfer_amount
                        READ y.balance
                        y.balance = y.balance + transfer_amount
                        COMMIT
                ELSE DISPLAY ("BALANCE IN X NOT OK")
                      ROLLBACK
        ELSE DISPLAY ("ACCOUNT X OR Y DOES NOT EXIST")
End_transaction
```

Please note the use of two keywords here COMMIT and ROLLBACK. Commit makes sure that all the changes made by transactions are made permanent. ROLLBACK terminates the transactions and rejects any change made by the transaction. Transactions have certain desirable properties. Let us look into those properties of a transaction.

Properties of a Transaction

A transaction has four basic properties. These are:

- Atomicity
- Consistency
- Isolation or Independence
- Durability or Permanence

Atomicity: It defines a transaction to be a single unit of processing. In other words either a transaction will be done *completely* or *not at all*. In the transaction example 1 & 2 please note that transaction 2 is reading and writing more than one data items, the atomicity property requires either operations on both the data item be performed or not at all.

Consistency: This property ensures that a complete transaction execution takes a database from one consistent state to another consistent state. If a transaction fails even then the database should come back to a consistent state.

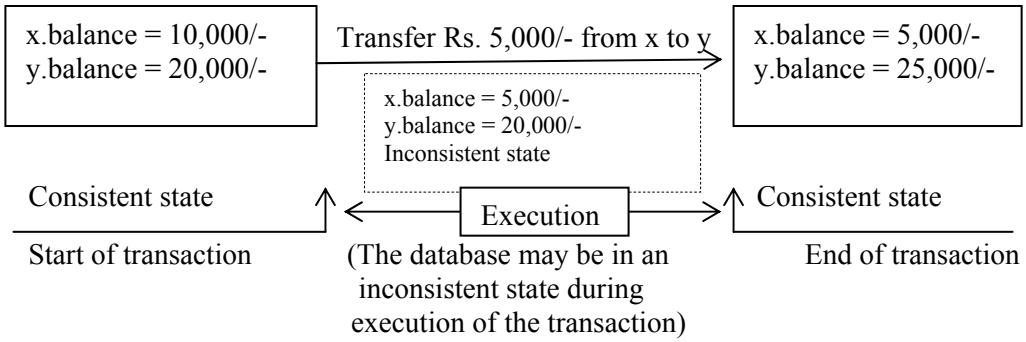


Figure 1: A Transaction execution

Isolation or Independence: The isolation property states that the updates of a transaction should not be visible till they are committed. Isolation guarantees that the progress of other transactions do not affect the outcome of this transaction. For example, if another transaction that is a withdrawal transaction which withdraws an amount of Rs. 5000 from X account is in progress, whether fails or commits, should not affect the outcome of this transaction. Only the state that has been read by the transaction last should determine the outcome of this transaction.

Durability: This property necessitates that once a transaction has committed, the changes made by it be never lost because of subsequent failure. Thus, a transaction is also a basic unit of recovery. The details of transaction-based recovery are discussed in the next unit.

A transaction has many states of execution. These states are displayed in *Figure 2*.

Error!

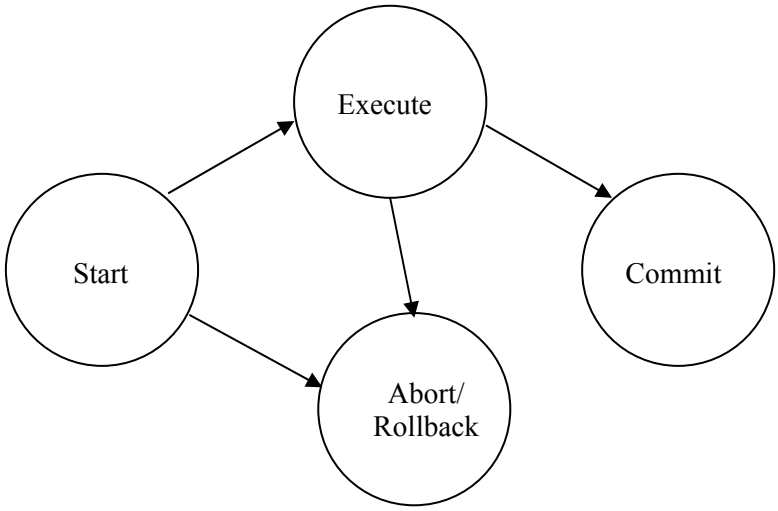


Figure 2: States of transaction execution

A transaction is started as a program. From the start state as the transaction is scheduled by the CPU it moves to the Execute state, however, in case of any system

error at that point it may also be moved into the Abort state. During the execution transaction changes the data values and database moves to an inconsistent state. On successful completion of transaction it moves to the Commit state where the durability feature of transaction ensures that the changes will not be lost. In case of any error the transaction goes to Rollback state where all the changes made by the transaction are undone. Thus, after commit or rollback database is back into consistent state. In case a transaction has been rolled back, it is started as a new transaction. All these states of the transaction are shown in *Figure 2*.

2.3 THE CONCURRENT TRANSACTIONS

Almost all the commercial DBMS support multi-user environment. Thus, allowing multiple transactions to proceed simultaneously. The DBMS must ensure that two or more transactions do not get into each other's way, i.e., transaction of one user doesn't effect the transaction of other or even the transactions issued by the same user should not get into the way of each other. Please note that concurrency related problem may occur in databases only if **two transactions are contending for the same data item and at least one of the concurrent transactions wishes to update a data value in the database**. In case, the concurrent transactions only read same data item and no updates are performed on these values, then it does NOT cause any concurrency related problem. Now, let us first discuss why you need a mechanism to control concurrency.

Consider a banking application dealing with checking and saving accounts. A Banking Transaction T1 for Mr. Sharma moves Rs.100 from his checking account balance X to his savings account balance Y, using the transaction T1:

Transaction T1:

```
A:  Read X
    Subtract 100
    Write X
B:  Read Y
    Add 100
    Write Y
```

Let us suppose an auditor wants to know the total assets of Mr. Sharma. He executes the following transaction:

Transaction T2:

```
Read X
Read Y
Display X+Y
```

Suppose both of these transactions are issued simultaneously, then the execution of these instructions can be mixed in many ways. This is also called the **Schedule**. Let us define this term in more detail.

A schedule S is defined as the sequential ordering of the operations of the 'n' interleaved transactions. A schedule maintains the order of operations within the individual transaction.

Conflicting Operations in Schedule: Two operations of different transactions conflict if they access the same data item AND one of them is a write operation.

For example, the two transactions TA and TB as given below, if executed in parallel, may produce a schedule:

TA	TB

READ X WRITE X

READ X WRITE X

SCHEDULE	TA	TB
READ X	READ X	
READ X		READ X
WRITE X		WRITE X
WRITE X	WRITE X	

One possible schedule for interleaved execution of TA and TB

Let us show you three simple ways of interleaved instruction execution of transactions T1 and T2. Please note that in the following tables the first column defines the sequence of instructions that are getting executed, that is the schedule of operations.

a) T2 is executed completely before T1 starts, then sum X+Y will show the correct assets:

Schedule	**Transaction T1**	**Transaction T2**	**Example Values**
Read X		Read X	X = 50000
Read Y		Read Y	Y= 100000
Display X+Y		Display X+Y	**150000**
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100

b) T1 is executed completely before T2 starts, then sum X+Y will still show the correct assets:

Schedule	**Transaction T1**	**Transaction T2**	**Example Values**
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100
Read X		Read X	X = 49900
Read Y		Read Y	Y= 100100
Display X+Y		Display X+Y	**150000**

c) Block A in transaction T1 is executed, followed by complete execution of T2, followed by the Block B of T1.

Schedule	**Transaction T1**	**Transaction T2**	**Example Values**
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read X		Read X	X = 49900
Read Y		Read Y	Y= 100000
Display X+Y		Display X+Y	**149900**
Read Y	Read Y		Y= 100000

Add 100	Add 100		100100
Write Y	Write Y		Y= 100100

In this execution an incorrect value is being displayed. This is because Rs.100 although removed from X, has not been put in Y, and is thus missing. Obviously, if T1 had been written differently, starting with block B and following up with block A, even then such an interleaving would have given a different but incorrect result.

Please note that for the given transaction there are many more ways of this interleaved instruction execution.

Thus, there may be a problem when the transactions T1 and T2 are allowed to execute in parallel. Let us define the problems of concurrent execution of transaction more precisely.

Let us assume the following transactions (assuming there will not be errors in data while execution of transactions)

Transaction T3 and T4: T3 reads the balance of account X and subtracts a withdrawal amount of Rs. 5000, whereas T4 reads the balance of account X and adds an amount of Rs. 3000

T3
READ X
SUB 5000
WRITE X

T4
READ X
ADD 3000
WRITE X

Problems of Concurrent Transactions

1. **Lost Updates:** Suppose the two transactions T3 and T4 run concurrently and they happen to be interleaved in the following way (assume the initial value of X as 10000):

T3	T4	Value of X	
		T3	T4
READ X		10000	
	READ X		10000
SUB 5000		5000	
	ADD 3000		13000
WRITE X		5000	
	WRITE X		13000

After the execution of both the transactions the value X is 13000 while the semantically correct value should be 8000. The problem occurred as the update made by T3 has been overwritten by T4. The root cause of the problem was the fact that both the transactions had read the value of X as 10000. Thus one of the two updates has been lost and we say that a **lost update** has occurred.

There is one more way in which the lost updates can arise. Consider the following part of some transactions:

T5	T6	Value of x originally 2000	
		T5	T6
UPDATE X		3000	
	UPDATE X		4000
ROLLBACK		2000	

Here T5 & T6 updates the same item X. Thereafter T5 decides to undo its action and rolls back causing the value of X to go back to the original value that was 2000. In this case also the update performed by T6 had got lost and a lost update is said to have occurred.

2. **Unrepeatable reads:** Suppose T7 reads X twice during its execution. If it did not update X itself it could be very disturbing to see a different value of X in its next read. But this could occur if, between the two read operations, another transaction modifies X.

T7	T8	Assumed value of X=2000	
		T7	T8
READ X		2000	
	UPDATE X		3000
READ X		3000	

Thus, the inconsistent values are read and results of the transaction may be in error.

3. **Dirty Reads:** T10 reads a value which has been updated by T9. This update has not been committed and T9 aborts.

T9	T10	Value of x old value = 200	
		T9	T10
UPDATE X		500	
	READ X		500
ROLLBACK		200	?

Here T10 reads a value that has been updated by transaction T9 that has been aborted. Thus T10 has read a value that would never exist in the database and hence the problem. Here the problem is primarily of isolation of transaction.

4. **Inconsistent Analysis:** The problem as shown with transactions T1 and T2 where two transactions interleave to produce incorrect result during an analysis by Audit is the example of such a problem. This problem occurs when more than one data items are being used for analysis, while another transaction has modified some of those values and some are yet to be modified. Thus, an analysis transaction reads values from the inconsistent state of the database that results in inconsistent analysis.

Thus, we can conclude that the prime reason of problems of concurrent transactions is that a transaction reads an inconsistent state of the database that has been created by other transaction.

But how do we ensure that execution of two or more transactions have not resulted in a concurrency related problem?

Well one of the commonest techniques used for this purpose is to restrict access to data items that are being read or written by one transaction and is being written by another transaction. This technique is called locking. Let us discuss locking in more detail in the next section.

☞ Check Your Progress 1

1) What is a transaction? What are its properties? Can a transaction update more than on data values? Can a transaction write a value without reading it? Give an example of transaction.

……

……

……

2) What are the problems of concurrent transactions? Can these problems occur in transactions which do not read the same data values?

……

……

……

3) What is a Commit state? Can you rollback after the transaction commits?

……

……

……

2.4 THE LOCKING PROTOCOL

To control concurrency related problems we use locking. A lock is basically a variable that is associated with a data item in the database. A lock can be placed by a transaction on a shared resource that it desires to use. When this is done, the data item is available for the exclusive use for that transaction, i.e., other transactions are locked out of that data item. When a transaction that has locked a data item does not desire to use it any more, it should unlock the data item so that other transactions can use it. If a transaction tries to lock a data item already locked by some other transaction, it cannot do so and waits for the data item to be unlocked. The component of DBMS that controls and stores lock information is called the Lock Manager. The locking mechanism helps us to convert a schedule into a serialisable schedule. We had defined what a schedule is, but what is a serialisable schedule? Let us discuss about it in more detail:

2.4.1 Serialisable Schedules

If the operations of two transactions conflict with each other, how to determine that no concurrency related problems have occurred? For this, serialisability theory has been developed. Serialisability theory attempts to determine the **correctness** of the schedules. The rule of this theory is:

"A schedule S of n transactions is serialisable if it is equivalent to some **serial schedule** of the same 'n' transactions".

A serial schedule is a schedule in which either transaction T_1 is completely done before T_2 or transaction T_2 is completely done before T_1. For example, the following figure shows the two possible serial schedules of transactions T_1 & T_2.

Schedule A: T2 followed by T1			Schedule B: T1 followed by T2		
Schedule	**T1**	**T2**	**Schedule**	**T1**	**T2**
Read X		Read X	Read X	Read X	
Read Y		Read Y	Subtract 100	Subtract 100	
Display X+Y		Display X+Y	Write X	Write X	
Read X	Read X		Read Y	Read Y	
Subtract 100	Subtract 100		Add 100	Add 100	
Write X	Write X		Write Y	Write Y	
Read Y	Read Y		Read X		Read X
Add 100	Add 100		Read Y		Read Y
Write Y	Write Y		Display X+Y		Display X+Y

Figure 3: Serial Schedule of two transactions

Schedule C: An Interleaved Schedule		
Schedule	**T1**	**T2**
Read X	Read X	
Subtract 100	Subtract 100	
Read X		Read X

Write X	Write X	
Read Y		Read Y
Read Y	Read Y	
Add 100	Add 100	
Display X+Y		Display X+Y
Write Y	Write Y	

Figure 4: An Interleaved Schedule

Now, we have to figure out whether this interleaved schedule would be performing read and write in the same order as that of a serial schedule. If it does, then it is equivalent to a serial schedule, otherwise not. In case it is not equivalent to a serial schedule, then it may result in problems due to concurrent transactions.

Serialisability

Any schedule that produces the same results as a serial schedule is called a serialisable schedule. But how can a schedule be determined to be serialisable or not? In other words, other than giving values to various items in a schedule and checking if the results obtained are the same as those from a serial schedule, is there an algorithmic way of determining whether a schedule is serialisable or not?

The basis of the algorithm for serialisability is taken from the notion of a serial schedule. There are two possible serial schedules in case of two transactions (T1- T2 OR T2 - T1). Similarly, in case of three parallel transactions the number of possible serial schedules is 3!, that is, 6. These serial schedules can be:

T1-T2-T3	T1-T3-T2	T2-T1-T3
T2-T3-T1	T3-T1-T2	T3-T2-T1

Using the notion of precedence graph, an algorithm can be devised to determine whether an interleaved schedule is serialisbale or not. In this graph, the transactions of the schedule are represented as the nodes. This graph also has directed edges. An edge from the node representing transactions Ti to node Tj means that there exists a **conflicting operation** between Ti and Tj and Ti precedes Tj in some conflicting operations. It has been proved that a serialisable schedule is the one that contains no cycle in the graph.

Given a graph with no cycles in it, there must be a serial schedule corresponding to it.

The steps of constructing a precedence graph are:

1. Create a node for every transaction in the schedule.
2. Find the precedence relationships in conflicting operations. Conflicting operations are (read-write) or (write-read) or (write–write) on the same data item in two different transactions. But how to find them?

   2.1 For a transaction Ti which *reads* an item A, find a transaction Tj that *writes* A later in the schedule. If such a transaction is found, draw an edge from Ti to Tj.
   2.2 For a transaction Ti which has *written* an item A, find a transaction Tj later in the schedule that *reads* A. If such a transaction is found, draw an edge from Ti to Tj.
   2.3 For a transaction Ti which has *written* an item A, find a transaction Tj that *writes* A later than Ti. If such a transaction is found, draw an edge from Ti to Tj.
3. If there is any cycle in the graph, the schedule is not serialisable, otherwise, find the equivalent serial schedule of the transaction by traversing the transaction nodes starting with the node that has no input edge.

Let us use this algorithm to check whether the schedule as given in Figure 4 is Serialisable. *Figure 5* shows the required graph. Please note as per step 1, we draw the two nodes for T1 and T2. In the schedule given in *Figure 4*, please note that the transaction T2 reads data item X, which is subsequently written by T1, thus there is an edge from T2 to T1 (clause 2.1). Also, T2 reads data item Y, which is subsequently written by T1, thus there is an edge from T2 to T1 (clause 2.1). However, that edge already exists, so we do not need to redo it. Please note that there are no cycles in the graph, thus, the schedule given in *Figure 4* is serialisable. The equivalent serial schedule (as per step 3) would be T2 followed by T1.

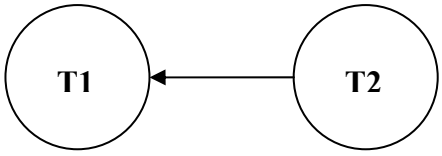


Figure 5: Test of Serialisability for the Schedule of Figure 4

Please note that the schedule given in part (c) of section 2.3 is not serialsable, because in that schedule, the two edges that exist between nodes T1 and T2 are:

- T1 writes X which is later read by T2 (clause 2.2), so there exists an edge from T1 to T2.
- T2 reads X which is later written by T1 (clause 2.1), so there exists an edge from T2 to T1.

Thus the graph for the schedule will be:

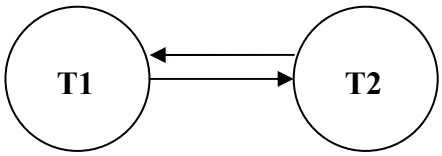


Figure 6: Test of Serialisability for the Schedule (c) of section 2.3

Please note that the graph above has a cycle T1-T2-T1, therefore it is not serialisable.

2.4.2 Locks

Serialisabilty is just a test whether a given interleaved schedule is ok or has a concurrency related problem. However, it does not ensure that the interleaved concurrent transactions do not have any concurrency related problem. This can be done by using locks. So let us discuss about what the different types of locks are, and then how locking ensures serialsability of executing transactions.

Types of Locks

There are two basic types of locks:

- Binary lock: This locking mechanism has two states for to a data item: locked or unlocked
- Multiple-mode locks: In this locking type each data item can be in three states read locked or shared locked, write locked or exclusive locked or unlocked.

Let us first take an example for binary locking and explain how it solves the concurrency related problems. Let us reconsider the transactions T1 and T2 for this purpose; however we will add to required binary locks to them.

Schedule	T1	T2
Lock X	Lock X	
Read X	Read X	
Subtract 100	Subtract 100	
Write X	Write X	
Unlock X	Unlock X	
Lock X		Lock X
Lock Y		Lock Y
Read X		Read X
Read Y		Read Y
Display X+Y		Display X+Y
Unlock X		Unlock X
Unlock Y		Unlock Y
Lock Y	Lock Y	
Read Y	Read Y	
Add 100	Add 100	
Write Y	Write Y	
Unlock Y	Unlock Y	

Figure 7: An incorrect locking implementation

Does the locking as done above solve the problem of concurrent transactions? No the same problems still remains. Try working with the old value. Thus, locking should be done with some logic in order to make sure that locking results in no concurrency related problem. One such solution is given below:

Schedule	T1	T2
Lock X	Lock X	
Lock Y	Lock Y	
Read X	Read X	
Subtract 100	Subtract 100	
Write X	Write X	
Lock X (issued by T2)	Lock X: denied as T1 holds the lock. The transaction T2 Waits and T1 continues.	
Read Y	Read Y	
Add 100	Add 100	
Write Y	Write Y	
Unlock X	Unlock X	
	The lock request of T2 on X can now be granted it can resumes by locking X	
Unlock Y	Unlock Y	
Lock Y		Lock Y
Read X		Read X
Read Y		Read Y
Display X+Y		Display X+Y
Unlock X		Unlock X
Unlock Y		Unlock Y

Figure 8: A correct but restrictive locking implementation

Thus, the locking as above when you obtain all the locks at the beginning of the transaction and release them at the end ensures that transactions are executed with no concurrency related problems. However, such a scheme limits the concurrency. We will discuss a two-phase locking method in the next subsection that provides sufficient concurrency. However, let us first discuss multiple mode locks.

Multiple-mode locks: It offers two locks: shared locks and exclusive locks. But why do we need these two locks? There are many transactions in the database system that never update the data values. These transactions can coexist with other transactions that update the database. In such a situation multiple reads are allowed on a data item, so multiple transactions can lock a data item in the shared or read lock. On the other hand, if a transaction is an updating transaction, that is, it updates the data items, it has to ensure that no other transaction can access (read or write) those data items that it wants to update. In this case, the transaction places an exclusive lock on the data items. Thus, a somewhat higher level of concurrency can be achieved in comparison to the binary locking scheme.

The properties of shared and exclusive locks are summarised below:

a) **Shared lock or Read Lock**

- It is requested by a transaction that wants to just read the value of data item.
- A shared lock on a data item does not allow an exclusive lock to be placed but permits any number of shared locks to be placed on that item.

b) **Exclusive lock**

- It is requested by a transaction on a data item that it needs to update.
- No other transaction can place either a shared lock or an exclusive lock on a data item that has been locked in an exclusive mode.

Let us describe the above two modes with the help of an example. We will once again consider the transactions T1 and T2 but in addition a transaction T11 that finds the total of accounts Y and Z.

Schedule	T1	T2	T11
S_Lock X		S_Lock X	
S_Lock Y		S_Lock Y	
Read X		Read X	
S_Lock Y			S_Lock Y
S_Lock Z			S_Lock Z
			Read Y
			Read Z
X_Lock X	X_Lock X. The exclusive lock request on X is denied as T2 holds the Read lock. The transaction T1 Waits.		
Read Y		Read Y	
Display X+Y		Display X+Y	
Unlock X		Unlock X	
X_Lock Y	X_Lock Y. The previous exclusive lock request on X is granted as X is unlocked. But the new exclusive lock request on Y is not granted as Y is locked by T2 and T11 in read mode. Thus T1 waits till both T2 and T11 will release the read lock on Y.		
Display Y+Z			Display Y+Z
Unlock Y		Unlock Y	
Unlock Y			Unlock Y
Unlock Z			Unlock Z
Read X	Read X		
Subtract 100	Subtract 100		
Write X	Write X		
Read Y	Read Y		
Add 100	Add 100		

Write Y	Write Y		
Unlock X	Unlock X		
Unlock Y	Unlock Y		

Figure 9: Example of Locking in multiple-modes

Thus, the locking as above results in a serialisable schedule. Now the question is can we release locks a bit early and still have no concurrency related problem? Yes, we can do it if we lock using two-phase locking protocol. This protocol is explained in the next sub-section.

2.4.3 Two Phase Locking (2PL)

The two-phase locking protocol consists of two phases:

Phase 1: The lock acquisition phase: If a transaction T wants to read an object, it needs to obtain the S (shared) lock. If T wants to modify an object, it needs to obtain X (exclusive) lock. No conflicting locks are granted to a transaction. **New locks on items can be acquired but no lock can be released till all the locks required by the transaction are obtained.**

Phase 2: Lock Release Phase: The existing locks can be released in any order but no new lock can be acquired **after a lock has been released**. The locks are held only till they are required.

Normally the locks are obtained by the DBMS. Any legal schedule of transactions that follows 2 phase locking protocol is guaranteed to be serialisable. The two phase locking protocol has been proved for it correctness. However, the proof of this protocol is beyond the scope of this Unit. You can refer to further readings for more details on this protocol.

There are two types of 2PL:

(1) The Basic 2PL
(2) Strict 2PL

The basic 2PL allows release of lock at any time after all the locks have been acquired. For example, we can release the locks in schedule of *Figure 8*, after we have Read the values of Y and Z in transaction 11, even before the display of the sum. This will enhance the concurrency level. The basic 2PL is shown graphically in *Figure 10.*

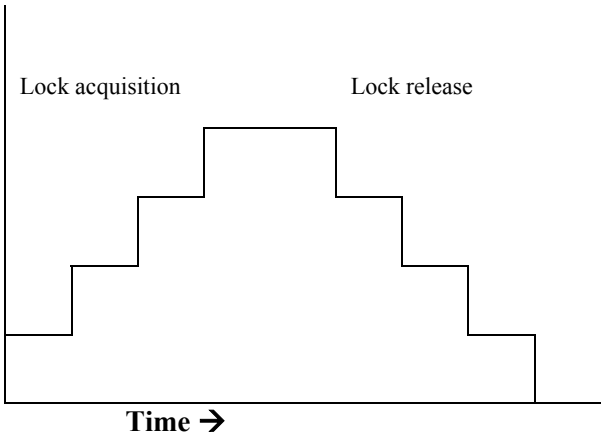


Figure 10: Basic Two Phase Locking

However, this basic 2PL suffers from the problem that it can result into loss of atomic / isolation property of transaction as theoretically speaking once a lock is released on a

data item it can be modified by another transaction before the first transaction commits or aborts.

To avoid such a situation we use strict 2PL. The strict 2PL is graphically depicted in *Figure 11*. However, the basic disadvantage of strict 2PL is that it restricts concurrency as it locks the item beyond the time it is needed by a transaction.

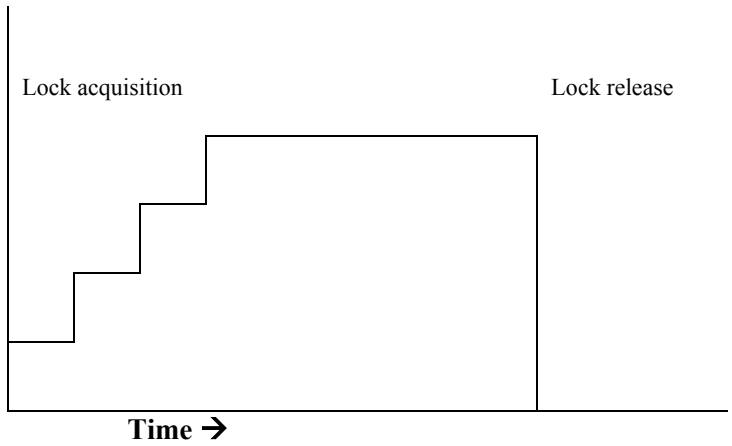


Figure 11: Strict Two Phase Locking

Does the 2PL solve all the problems of concurrent transactions? No, the strict 2PL solves the problem of concurrency and atomicity, however it introduces another problem: "Deadlock". Let us discuss this problem in next section.

☞ Check Your Progress 2

1) Let the transactions T1, T2, T3 be defined to perform the following operations:

   T1: Add one to A
   T2: Double A
   T3: Display A on the screen and set A to one.

   Suppose transactions T1, T2, T3 are allowed to execute concurrently. If A has initial value zero, how many possible correct results are there? Enumerate them.

   ……

   ……

2) Consider the following two transactions, given two bank accounts having a balance A and B.

   Transaction T1: Transfer Rs. 100 from A to B

   Transaction T2: Find the multiple of A and B.

   Create a non-serialisable schedule.

   ……

   ……

3) Add lock and unlock instructions (exclusive or shared) to transactions T1 and T2 so that they observe the serialisable schedule. Make a valid schedule.

   ……

   ……

2.5 DEADLOCK AND ITS PREVENTION

As seen earlier, though 2PL protocol handles the problem of serialisability, but it causes some problems also. For example, consider the following two transactions and a schedule involving these transactions:

TA	TB
X_lock A	X_lock A
X_lock B	X_lock B
:	:
:	:
Unlock A	Unlock A
Unlock B	Unlock B

Schedule

T1: X_lock A
T2: X_lock B
T1: X_lock B
T2: X_lock A

As is clearly seen, the schedule causes a problem. After T1 has locked A, T2 locks B and then T1 tries to lock B, but unable to do so waits for T2 to unlock B. Similarly, T2 tries to lock A but finds that it is held by T1 which has not yet unlocked it and thus waits for T1 to unlock A. At this stage, neither T1 nor T2 can proceed since both of these transactions are waiting for the other to unlock the locked resource.

Clearly the schedule comes to a halt in its execution. The important thing to be seen here is that both T1 and T2 follow the 2PL, which guarantees serialisability. So whenever the above type of situation arises, we say that a deadlock has occurred, since two transactions are **waiting for a condition that will never occur**.

Also, the deadlock can be described in terms of a directed graph called a "*wait for*" graph, which is maintained by the lock manager of the DBMS. This graph G is defined by the pair (V, E). It consists of a set of vertices/nodes V is and a set of edges/arcs E. Each transaction is represented by node and an arc from Ti → Tj, if Tj holds a lock and Ti is waiting for it. When transaction Ti requests a data item currently being held by transaction Tj then the edge Ti → Tj is inserted in the "wait for" graph. This edge is removed only when transaction Tj_j is no longer holding the data item needed by transaction Ti.

A deadlock in the system of transactions occurs, if and only if the wait-for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks, a periodic check for cycles in graph can be done. For example, the "wait-for" for the schedule of transactions TA and TB as above can be made as:

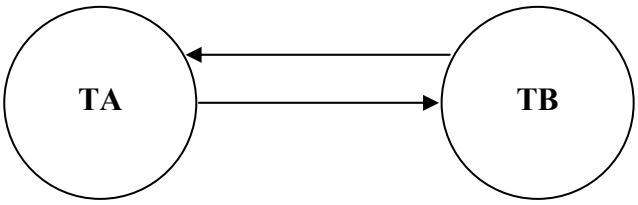


Figure 12: Wait For graph of TA and TB

In the figure above, TA and TB are the two transactions. The two edges are present between nodes TA and TB since each is waiting for the other to unlock a resource held by the other, forming a cycle, causing a deadlock problem. The above case shows a direct cycle. However, in actual situation more than two nodes may be there in a cycle.

A deadlock is thus a situation that can be created because of locks. It causes transactions to wait forever and hence the name deadlock. A deadlock occurs because of the following conditions:

a) Mutual exclusion: A resource can be locked in exclusive mode by only one transaction at a time.
b) Non-preemptive locking: A data item can only be unlocked by the transaction that locked it. No other transaction can unlock it.
c) Partial allocation: A transaction can acquire locks on database in a piecemeal fashion.
d) Circular waiting: Transactions lock part of data resources needed and then wait indefinitely to lock the resource currently locked by other transactions.

In order to prevent a deadlock, one has to ensure that at least one of these conditions does not occur.

A deadlock can be prevented, avoided or controlled. Let us discuss a simple method for deadlock prevention.

Deadlock Prevention

One of the simplest approaches for avoiding a deadlock would be to acquire all the locks at the start of the transaction. However, this approach restricts concurrency greatly, also you may lock some of the items that are not updated by that transaction (the transaction may have if conditions).Thus, better prevention algorithm have been evolved to prevent a deadlock having the basic logic: *not to allow circular wait to occur*. This approach rolls back some of the transactions instead of letting them wait.

There exist two such schemes. These are:

"Wait-die" scheme: The scheme is based on non-preventive technique. It is based on a simple rule:

```
If Ti requests a database resource that is held by Tj
        then if Ti has a smaller timestamp than that of Tj
                it is allowed to wait;
        else Ti aborts.
```

A timestamp may loosely be defined as the system generated sequence number that is unique for each transaction. Thus, a smaller timestamp means an older transaction. For example, assume that three transactions T1, T2 and T3 were generated in that sequence, then if T1requests for a data item which is currently held by transaction T2, it is allowed to wait as it has a smaller time stamping than that of T1. However, if T3 requests for a data item which is currently held by transaction T2, then T3 is rolled back (die).

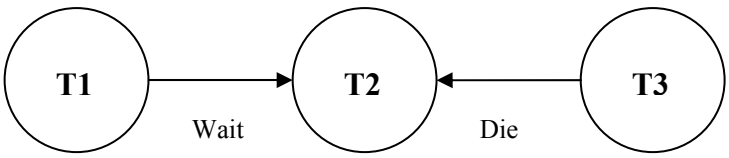


Figure 13: Wait-die Scheme of Deadlock prevention

"Wound-wait" scheme: It is based on a preemptive technique. It is based on a simple rule:

If Ti requests a database resource that is held by Tj
then if Ti has a larger timestamp (Ti is younger) than that of Tj
it is allowed to wait;
else Tj is wounded up by Ti.

For example, assume that three transactions T1, T2 and T3 were generated in that sequence, then if T1requests for a data item which is currently held by transaction T2, then T2 is rolled back and data item is allotted to T1 as T1 has a smaller time stamping than that of T2. However, if T3 requests for a data item which is currently held by transaction T2, then T3 is allowed to wait.

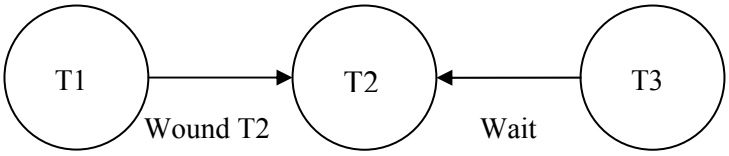


Figure 14: Wound-wait Scheme of Deadlock prevention

It is important to see that whenever any transaction is rolled back, it would not make a starvation condition, that is no transaction gets rolled back repeatedly and is never allowed to make progress. Also both "wait-die" & "wound-wait" scheme avoid starvation. The number of aborts & rollbacks will be higher in wait-die scheme than in the wound-wait scheme. But one major problem with both of these schemes is that these schemes may result in unnecessary rollbacks. You can refer to further readings for more details on deadlock related schemes.

2.6 OPTIMISTIC CONCURRENCY CONTROL

Is locking the only way to prevent concurrency related problems? There exist some other methods too. One such method is called an Optimistic Concurrency control. Let us discuss it in more detail in this section.

The basic logic in optimistic concurrency control is to allow the concurrent transactions to update the data items assuming that the concurrency related problem will not occur. However, we need to reconfirm our view in the validation phase. Therefore, the optimistic concurrency control algorithm has the following phases:

a) READ Phase: A transaction T reads the data items from the database into its private workspace. All the updates of the transaction can only change the local copies of the data in the private workspace.

b) VALIDATE Phase: Checking is performed to confirm whether the read values have changed during the time transaction was updating the local values. This is performed by comparing the current database values to the values that were read in the private workspace. In case, the values have changed the local copies are thrown away and the transaction aborts.

c) WRITE Phase: If validation phase is successful the transaction is committed and updates are applied to the database, otherwise the transaction is rolled back.

Some of the terms defined to explain optimistic concurrency contents are:

- write-set(T): all data items that are written by a transaction T
- read-set(T): all data items that are read by a transaction T

- Timestamps: for each transaction T, the start-time and the end time are kept for all the three phases.

More details on this scheme are available in the further readings. But let us show this scheme here with the help of the following examples:

Consider the set for transaction T1 and T2.

T1		T2	
Phase	**Operation**	**Phase**	**Operation**
-	-	Read	Reads the read set (T2). Let say variables X and Y and performs updating of local values
Read	Reads the read set (T1) lets say variable X and Y and performs updating of local values	-	-
Validate	Validate the values of (T1)	-	-
-	-	Validate	Validate the values of (T2)
Write	Write the updated values in the database and commit	-	-
-	-	Write	Write the updated values in the database and commit

In this example both T1 and T2 get committed. Please note that Read set of T1 and Read Set of T2 are both disjoint, also the Write sets are also disjoint and thus no concurrency related problem can occur.

T1	T2	T3
Operation	Operation	Operation
Read R(A)	--	--
--	Read R(A)	--
--	--	Read (D)
--	--	Update(D)
--	--	Update (A)
--	--	Validate (D,A) finds OK Write (D,A), COMMIT
--	Validate(A):Unsuccessful Value changed by T3	--
Validate(A):Unsuccessful Value changed by T3	--	--
ABORT T1	--	--
--	Abort T2	--

In this case both T1 and T2 get aborted as they fail during validate phase while only T3 is committed. Optimistic concurrency control performs its checking at the transaction commits point in a validation phase. The serialization order is determined by the time of transaction validation phase.

☞ Check Your Progress 3

1) Draw suitable graph for following locking requests, find whether the transactions are deadlocked or not.

T1: S_lock A	--	--
--	T2: X_lock B	--
--	T2: S_lock C	--

--	--	T3: X_lock C
--	T2: S_lock A	--
T1: S_lock B	--	--
T1: S_lock A	--	--
--	--	T3: S_lock A
All the unlocking requests start from here		

……

……

……

……

……

……

……

……

2) What is Optimistic Concurrency Control?

……

……

……

……

……

……

2.7 SUMMARY

In this unit you have gone through the concepts of transaction and Concurrency Management. A transaction is a sequence of many actions. Concurrency control deals with ensuring that two or more users do not get into each other's way, i.e., updates of transaction one doesn't affect the updates of other transactions.

Serializability is the generally accepted criterion for correctness for the concurrency control. It is a concept related to concurrent schedules. It determines how to analyse whether any schedule is serialisable or not. Any schedule that produces the same results as a serial schedule is a serialisable schedule.

Concurrency Control is usually done via locking. If a transaction tries to lock a resource already locked by some other transaction, it cannot do so and waits for the resource to be unlocked.

Locks are of two type a) shared lock b) Exclusive lock. Then we move on to a method known as Two Phase Locking (2PL). A system is in a deadlock state if there exist a set of transactions such that every transaction in the set is waiting for another transaction in the set. We can use a deadlock prevention protocol to ensure that the system will never enter a deadlock state.

Finally we have discussed the method Optimistic Concurrency Control, another concurrency management mechanism.

2.8 SOLUTIONS / ANSWERS

Check Your Progress 1

1) A transaction is the basic unit of work on a Database management system. It defines the data processing on the database. IT has four basic properties:

   a. Atomicity: transaction is done completely or not at all.
   b. Consistency: Leaves the database in a consistent state
   c. Isolation: Should not see uncommitted values
   d. Durability: Once committed the changes should be reflected.

   A transaction can update more than one data values. Some transactions can do writing of data without reading a data value.

   A simple transaction example may be: Updating the stock inventory of an item that has been issued. Please create a sample pseudo code for it.

2) The basic problems of concurrent transactions are:

   - Lost updates: An update is overwritten
   - Unrepeatable read: On reading a value later again an inconsistent value is found.
   - Dirty read: Reading an uncommitted value
   - Inconsistent analysis: Due to reading partially updated value.

   No these problems cannot occur if the transactions do not read the same data values. The conflict occurs only if one transaction updates a data value while another is reading or writing the data value.

3) Commit state is defined as when transaction has done everything correctly and shows the intent of making all the changes as permanent. No, you cannot rollback after commit.

Check Your Progress 2

1) There are six possible results, corresponding to six possible serial schedules:

Initially:	A = 0
T1-T2-T3:	A = 1
T1-T3-T2:	A = 2
T2-T1-T3:	A = 1
T2-T3-T1:	A = 2
T3-T1-T2:	A = 4
T3-T2-T1:	A = 3

2)

Schedule	T1	T2
Read A	Read A	
A = A - 100	A = A - 100	
Write A	Write A	
Read A		Read A
Read B		Read B

Read B	Read B	
Result = A * B		Result = A * B
Display Result		Display Result
B = B + 100	B = B + 100	
Write B	Write B	

Please make the precedence graph and find out that the schedule is not serialisable.

3)

Schedule	**T1**	**T2**
Lock A	Lock A	
Lock B	Lock B	
Read A	Read A	
A = A - 100	A = A - 100	
Write A	Write A	
Unlock A	Unlock A	
Lock A		Lock A: Granted
Lock B		Lock B: Waits
Read B	Read B	
B = B + 100	B = B + 100	
Write B	Write B	
Unlock B	Unlock B	
Read A		Read A
Read B		Read B
Result = A * B		Result = A * B
Display Result		Display Result
Unlock A		Unlock A
Unlock B		Unlock B

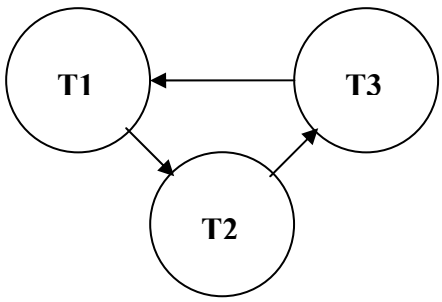
You must make the schedules using read and exclusive lock and a schedule in strict 2PL.

Check Your Progress 3

1) The transaction T1 gets the shared lock on A, T2 gets exclusive lock on B and Shared lock on A, while the transactions T3 gets exclusive lock on C.

- Now T2 requests for shared lock on C which is exclusively locked by T3, so cannot be granted. So T2 waits for T3 on item C.
- T1 now requests for Shared lock on B which is exclusively locked by T2, thus, it waits for T2 for item B. The T1 request for shared lock on C is not processed.
- Next T3 requests for exclusive lock on A which is share locked by T1, so it cannot be granted. Thus, T3 waits for T1 for item A.

The Wait for graph for the transactions for the given schedule is:



Since there exists a cycle, therefore, the schedule is deadlocked.

2) The basic philosophy for optimistic concurrency control is the optimism that nothing will go wrong so let the transaction interleave in any fashion, but to avoid any concurrency related problem we just validate our assumption before we make changes permanent. This is a good model for situations having a low rate of transactions.

UNIT 3 DATABASE RECOVERY AND SECURITY

Structure **Page Nos.**

3.0	Introduction	57
3.1	Objectives	57
3.2	What is Recovery?	57
	3.2.1 Kinds of failures	
	3.2.2 Failure controlling methods	
	3.2.3 Database errors	
3.3	Recovery Techniques	61
3.4	Security & Integrity	66
	3.4.1 Relationship between Security and Integrity	
	3.4.2 Difference between Operating System and Database Security	
3.5	Authorisation	68
3.6	Summary	71
3.7	Solutions/Answers	71

3.0 INTRODUCTION

In the previous unit of this block, you have gone through the concepts of transactions and Concurrency management. In this unit we will introduce two important issues relating to database management systems.

A computer system suffers from different types of failures. A DBMS controls very critical data of an organisation and therefore must be reliable. However, the reliability of the database system is linked to the reliability of the computer system on which it runs. In this unit we will discuss recovery of the data contained in a database system following failure of various types and present the different approaches to database recovery. The types of failures that the computer system is likely to be subjected to include failures of components or subsystems, software failures, power outages, accidents, unforeseen situations and natural or man-made disasters. Database recovery techniques are methods of making the database consistent till the last possible consistent state. The aim of recovery scheme is to allow database operations to be resumed after a failure with minimum loss of information at an economically justifiable cost.

The second main issue that is being discussed in this unit is Database security. "Database security" is protection of the information contained in the database against unauthorised access, modification or destruction. The first condition for security is to have Database integrity. "Database integrity" is the mechanism that is applied to ensure that the data in the database is consistent.

Let us discuss all these terms in more detail in this unit.

3.1 OBJECTIVES

At the end of this unit, you should be able to:

- describe the terms RECOVERY and INTEGRITY;
- describe Recovery Techniques;
- define Error and Error detection techniques, and
- describe types of Authorisation.

3.2 WHAT IS RECOVERY?

During the life of a transaction, that is, a after the start of a transaction but before the transaction commits, several changes may be made in a database state. The database during such a state is in an inconsistent state. What happens when a failure occurs at this stage? Let us explain this with the help of an example:

Assume that a transaction transfers Rs.2000/- from A's account to B's account. For simplicity we are not showing any error checking in the transaction. The transaction may be written as:

Transaction T1:

```
READ A
A = A – 2000
WRITE A
————————→ Failure
READ B
B = B + 2000
WRITE B
COMMIT
```

What would happen if the transaction fails after account A has been written back to database? As far as the holder of account A is concerned s/he has transferred the money but that has never been received by account holder B.

Why did this problem occur? Because although a transaction is considered to be atomic, yet it has a life cycle during which the database gets into an inconsistent state and failure has occurred at that stage.

What is the solution? In this case where the transaction has not yet committed the changes made by it, the partial updates need to be undone.

How can we do that? By remembering information about a transaction such as when did it start, what items it updated etc. All such details are kept in a log file. We will study about log in Section 3.3. But first let us analyse the reasons of failure.

Failures and Recovery

In practice several things might happen to prevent a transaction from completing. Recovery techniques are used to bring database, which does not satisfy consistency requirements, into a consistent state. If a transaction completes normally and commits then all the changes made by the transaction on the database are permanently registered in the database. They should not be lost (please recollect the durability property of transactions given in Unit 2). But, if a transaction does not complete normally and terminates abnormally then all the changes made by it should be discarded. An abnormal termination of a transaction may be due to several reasons, including:

a) user may decide to abort the transaction issued by him/ her
b) there might be a deadlock in the system
c) there might be a system failure.

The recovery mechanisms must ensure that a consistent state of database can be restored under all circumstances. In case of transaction abort or deadlock, the system remains in control and can deal with the failure but in case of a system failure the system loses control because the computer itself has failed. Will the results of such failure be catastrophic? A database contains a huge amount of useful information and

any system failure should be recognised on the restart of the system. The DBMS should recover from any such failures. Let us first discuss the kinds of failure for identifying how to recover.

3.2.1 Kinds of Failures

The kinds of failures that a transaction program during its execution can encounter are:

1) **Software failures:** In such cases, a software error abruptly stops the execution of the current transaction (or all transactions), thus leading to losing the state of program excution and the state/ contents of the buffers. But what is a buffer? A buffer is the portion of RAM that stores the partial contents of database that is currently needed by the transaction. The software failures can further be subdivided as:
   a) Statement or application program failure
   b) Failure due to viruses
   c) DBMS software failure
   d) Operating system failure

   A Statement of program may cause abnormal termination if it does not execute completely. This happens if during the execution of a statement, an integrity constraint gets violated. This leads to abnormal termination of the transaction due to which any prior updates made by the transaction may still get reflected in the database leaving it in an inconsistent state.

   A failure of transaction can occur if some code in a transaction program leads to its
   abnormal termination. For example, a transaction can go into an infinite loop. In such a case the only way to break the loop is to abort the program. Similarly, the failure can be traced to the operating system or DBMS and transactions are aborted abruptly. Thus part of the transaction that was executed before abort may cause some updates in database, and hence the database is updated only partially which leads to an inconsistent state of database.

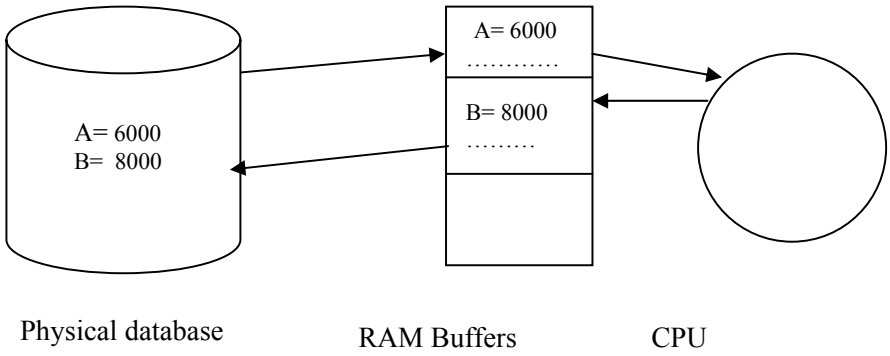
2) **Hardware failure:** Hardware failures are those failures when some hardware chip or disk fails. This may result in loss of data. One such problem can be that a disk gets damaged and cannot be read any more. This may be due to many reasons. For example, a voltage fluctuation in the power supply to the computer makes it go off or some bad sectors may come on disk or there is a disk crash. In all these cases, the database gets into an inconsistent state.

3) **External failure:** A failure can also result due to an external cause, such as fire, earthquakes, floods, etc. The database must be duly backed up to avoid problems occurring due to such failures.

In practice software failures are more common than hardware failures. Fortunately, recovery from software failures is much quicker.

The basic unit of recovery is a transaction. But, how are the transactions handled during recovery? Consider that some transactions are deadlocked, then at least one of these transactions has to be restarted to break the deadlock and thus the partial updates made by such restarted program in the database need to be **undone** so that the database does not go to an inconsistent state. So the transaction may have to be rolled back which makes sure that the transaction does not bring the database to an inconsistent state. This is one form of recovery. Let us consider a case when a transaction has committed but the changes made by the transaction have not been communicated to permanently stored physical database. A software failure now occurs and the contents of the CPU/ RAM are lost. This leaves the database in an inconsistent state. Such failure requires that on restarting the system the database be brought to a consistent state using **redo** operation. The redo operation makes the

changes made by the transaction again to bring the system to a consistent state. The database system then can be made available to the users. The point to be noted here is that the database updates are performed in the buffer in the memory. *Figure 1* shows some cases of undo and redo. You can create more such cases.



	Physical Database	RAM	Activity
Case 1	A=6000 B=8000	A=4000 B=8000	Transaction T1 has just changed the value in RAM. Now it aborts, value in RAM is lost. No problem. But we are not sure that the physical database has been written back, so must undo.
Case 2	A=4000 B=8000	A=4000 B=8000	The value in physical database has got updated due to buffer management, now the transaction aborts. The transaction must be undone.
Case 3	A=6000 B=8000	A=4000 B=10000 Commit	The value B in physical database has not got updated due to buffer management. In case of failure now when the transaction has committed. The changes of transaction must be redone to ensure transfer of correct values to physical database.

Figure 1: Database Updates And Recovery

3.2.2 Failure Controlling Methods

Failures can be handled using different recovery techniques that are discussed later in the unit. But the first question is do we really need recovery techniques as a failure control mechanism? The recovery techniques are somewhat expensive both in terms of time and in memory space for small systems. In such a case it is more beneficial to better avoid the failure by some checks instead of deploying recovery technique to make database consistent. Also, recovery from failure involves manpower that can be used in some other productive work if failure can be avoided. It is, therefore, important to find out some general precautions that help in controlling failure. Some of these precautions may be:

- having a regulated power supply.
- having a better secondary storage system such as RAID.
- taking periodic backup of database states and keeping track of transactions after each recorded state.
- properly testing the transaction programs prior to use.
- setting important integrity checks in the databases as well as user interfaces etc.

However, it may be noted that if the database system is critical it must use a DBMS that is suitably equipped with recovery procedures.

3.2.3 Database Errors

An error is said to have occurred if the execution of a command to manipulate the database cannot be successfully completed either due to inconsistent data or due to state of program. For example, there may be a command in program to store data in database. On the execution of command, it is found that there is no space/place in database to accommodate that additional data. Then it can be said that an error has occurred. This error is due to the physical state of database storage.

Broadly errors are classified into the following categories:

1) **User error:** This includes errors in the program (e.g., Logical errors) as well as errors made by online users of database. These types of errors can be avoided by applying some check conditions in programs or by limiting the access rights of online users e.g., read only. So only updating or insertion operation require appropriate check routines that perform appropriate checks on the data being entered or modified. In case of an error, some prompts can be passed to user to enable him/her to correct that error.

2) **Consistency error:** These errors occur due to the inconsistent state of database caused may be due to wrong execution of commands or in case of a transaction abort. To overcome these errors the database system should include routines that check for the consistency of data entered in the database.

3) **System error:** These include errors in database system or the OS, e.g., deadlocks. Such errors are fairly hard to detect and require reprogramming the erroneous components of the system software.

Database errors can result from failure or can cause failure and thus will require recovery. However, one of the main tasks of database system designers is to make sure that errors minimised. These concepts are also related to database integrity and have also been discusses in a later section.

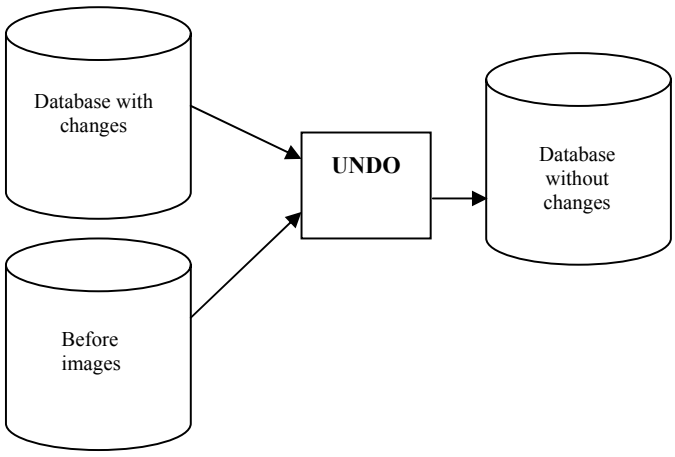
Comment: These are really the normal precautions one would take anyway – nothing specific to controlling database failures.

3.3 RECOVERY TECHNIQUES

After going through the types of failures and database errors, let us discuss how to recover from the failures. Recovery can be done using/restoring the previous consistent state (backward recovery) or by moving forward to the next consistent state as per the committed transactions (forward recovery) recovery. Please note that a system can recover from software and hardware failures using the forward and backward recovery only if the system log is intact. What is system log? We will discuss it in more detail, but first let us define forward and backward recovery.

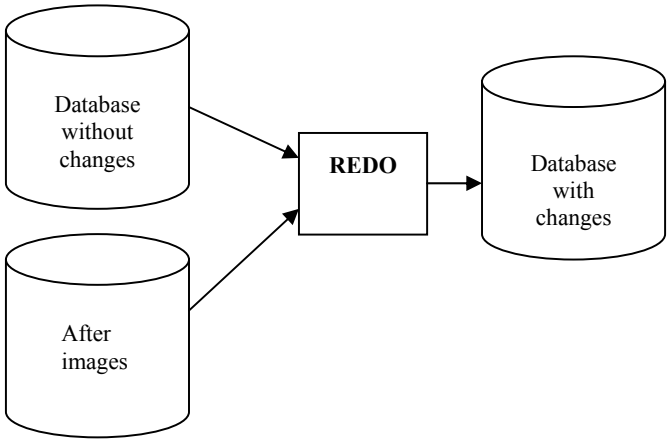
1) **Backward Recovery (UNDO)**

In this scheme the uncommitted changes made by a transaction to a database are undone. Instead the system is reset to the previous consistent state of database that is free from any errors.



2) **Forward Recovery (Redo)**

In this scheme the committed changes made by a transaction are reapplied to an earlier copy of the database.



In simpler words, when a particular error in a system is detected, the recovery system makes an accurate assessment of the state of the system and then makes the appropriate adjustment based on the anticipated results - had the system been error free.

One thing to be noted is that the Undo and Redo operations must be idempotent, i.e., executing them several times must be equivalent to executing them once. This characteristic is required to guarantee correct behaviour of database even if a failure occurs during the recovery process.

Depending on the above discussed recovery scheme, several types of recovery methods have been used. However, we define the most important recovery schemes used in most of the commercial DBMSs

Log based recovery

Let us first define the term transaction log in the context of DBMS. A transaction log is a record in DBMS that keeps track of all the transactions of a database system that update any data values in the database. A log contains the following information about a transaction:

- A transaction begin marker
- The transaction identification: The transaction id, terminal id or user id etc.

- The operations being performed by the transaction such as update, delete, insert.
- The data items or objects that are affected by the transaction including name of the table, row number and column number.
- The before or previous values (also called UNDO values) and after or changed values (also called REDO values) of the data items that have been updated.
- A pointer to the next transaction log record, if needed.
- The COMMIT marker of the transaction.

In a database system several transactions run concurrently. When a transaction commits, the data buffers used by it need not be written back to the physical database stored on the secondary storage as these buffers may be used by several other transactions that have not yet committed. On the other hand, some of the data buffers that may have updates by several uncommitted transactions might be forced back to the physical database, as they are no longer being used by the database. So the transaction log helps in remembering which transaction did which changes. Thus the system knows exactly how to separate the changes made by transactions that have already committed from those changes that are made by the transactions that did not yet commit. Any operation such as begin transaction, insert /delete/update and end transaction (commit), adds information to the log containing the transaction identifier and enough information to undo or redo the changes.

But how do we recover using log? Let us demonstrate this with the help of an example having three concurrent transactions that are active on ACCOUNTS table as:

Transaction T1	**Transaction T2**	**Transaction T3**
Read X	Read A	Read Z
Subtract 100	Add 200	Subtract 500
Write X	Write A	Write Z
Read Y		
Add 100		
Write Y		

Figure 2: The sample transactions

Assume that these transactions have the following log file (hypothetical) at a point:

Transaction Begin Marker	**Transaction Id**	**Operation on ACCOUNTS table**	**UNDO values (assumed)**	**REDO values**	**Transaction Commit Marker**
Y	T1	Sub on X Add on Y	500 800	400 Not done yet	N
Y	T2	Add on A	1000	1200	N
Y	T3	Sub on Z	900	400	Y

Figure 3: A sample (hypothetical) Transaction log

Now assume at this point of time a failure occurs, then how the recovery of the database will be done on restart.

Values	**Initial**	**Just before the failure**	**Operation Required for recovery**	**Recovered Database Values**
X	500	400 (assuming update has been done in physical database also)	UNDO	500
Y	800	800	UNDO	800

A	1000	1000 (assuming update has not been done in physical database)	UNDO	1000
Z	900	900 (assuming update has not been done in physical database)	REDO	400

Figure 4: The database recovery

The selection of REDO or UNDO for a transaction for the recovery is done on the basis of the state of the transactions. This state is determined in two steps:

- Look into the log file and find all the transactions that have started. For example, in *Figure 3*, transactions T1, T2 and T3 are candidates for recovery.
- Find those transactions that have committed. REDO these transactions. All other transactions have not committed so they should be rolled back, so UNDO them. For example, in *Figure 3* UNDO will be performed on T1 and T2; and REDO will be performed on T3.

Please note that in Figure 4 some of the values may not have yet been communicated to database, yet we need to perform UNDO as we are not sure what values have been written back to the database.
But how will the system recover? Once the recovery operation has been specified, the system just takes the required REDO or UNDO values from the transaction log and changes the inconsistent state of database to a consistent state. (Please refer to *Figure 3* and *Figure 4*).

Let us consider several transactions with their respective start & end (commit) times as shown in *Figure 5*.

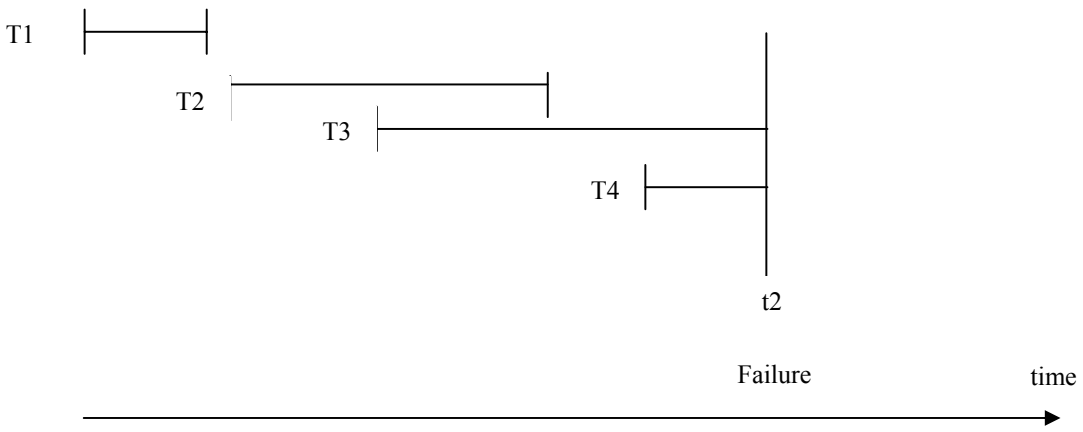


Figure 5: Execution of Concurrent Transactions

In the figure above four transactions are executing concurrently, on encountering a failure at time t2, the transactions T1 and T2 are to be REDONE and T3 and T4 will be UNDONE. But consider a system that has thousands of parallel transactions then all those transactions that have been committed may have to be redone and all uncommitted transactions need to be undone. That is not a very good choice as it requires redoing of even those transactions that might have been committed even hours earlier. So can we improve on this situation? Yes, we can take checkpoints. *Figure 6* shows a checkpoint mechanism:

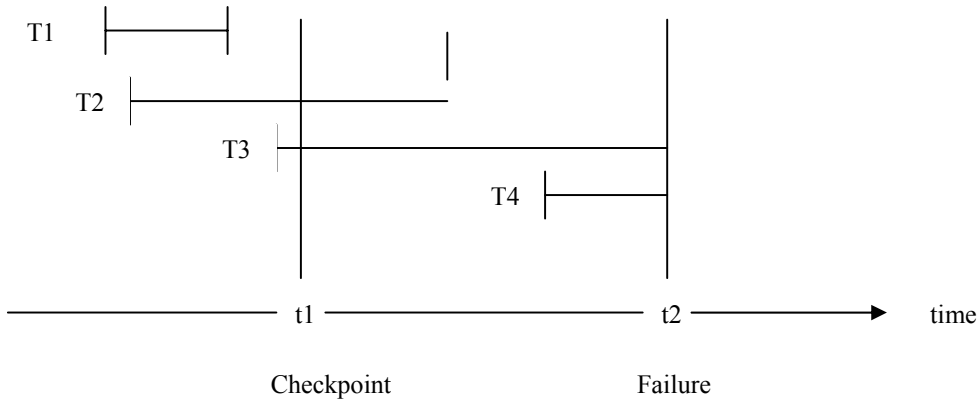


Figure 6: Checkpoint In Transaction Execution

A checkpoint is taken at time t_1 and a failure occurs at time t_2. Checkpoint transfers all the committed changes to database and all the system logs to stable storage (it is defined as the storage that would not be lost). At restart time after the failure the stable check pointed state is restored. Thus, we need to only REDO or UNDO those transactions that have completed or started after the checkpoint has been taken. The only possible disadvantages of this scheme may be that during the time of taking the checkpoint the database would not be available and some of the uncommitted values may be put in the physical database. To overcome the first problem the checkpoints should be taken at times when system load is low. To avoid the second problem some systems allow some time to the ongoing transactions to complete without restarting new transactions.

In the case of *Figure 6* the recovery from failure at t_2 will be as follows:

- The transaction T1 will not be considered for recovery as the changes made by it have already been committed and transferred to physical database at checkpoint t_1.
- The transaction T2 since it has not committed till the checkpoint t_1 but have committed before t_2, will be REDONE.
- T3 must be UNDONE as the changes made by it before checkpoint (we do not know for sure if any such changes were made prior to checkpoint) must have been communicated to the physical database. T3 must be restarted with a new name.
- T4 started after the checkpoint, and if we strictly follow the scheme in which the buffers are written back only on the checkpoint, then nothing needs to be done except restarting the transaction T4 with a new name.

The restart of a transaction requires the log to keep information of the new name of the transaction and maybe give higher priority to this newer transaction.

But one question is still unanswered that is during a failure we lose database information in RAM buffers, we may also lose log as it may also be stored in RAM buffers, so how does log ensure recovery?

The answer to this question lies in the fact that for storing transaction log we follow a **Write Ahead Log Protocol.** As per this protocol, the transaction logs are written to stable storage before any item is updated. Or more specifically it can be stated as; **the undo potion of log is written to stable storage prior to any updates and redo portion of log is written to stable storage prior to commit.**

Log based recovery scheme can be used for any kind of failure provided you have stored the most recent checkpoint state and most recent log as per write ahead log

protocol into the stable storage. Stable storage from the viewpoint of external failure requires more than one copy of such data at more than one location. You can refer to the further readings for more details on recovery and its techniques.

☞ Check Your Progress 1

1) What is the need of recovery? What is it the basic unit of recovery?

……

……

2) What is a checkpoint? Why is it needed? How does a checkpoint help in recovery?

……

……

3) What are the properties that should be taken into consideration while selecting recovery techniques?

……

……...……..……..

3.4 SECURITY AND INTEGRITY

After going through the concepts of database recovery in the previous section, let us now deal with an important concept that helps in minimizing consistency errors in database systems. These are the concepts of database security and integrity.

Information security is the protection of information against unauthorised disclosure, alteration or destruction. Database security is the protection of information that is maintained in a database. It deals with ensuring only the "right people" get the right to access the "right data". By right people we mean those people who have the right to access/update the data that they are requesting to access/update with the database. This should also ensure the confidentiality of the data. For example, in an educational institution, information about a student's grades should be made available only to that student, whereas only the university authorities should be able to update that information. Similarly, personal information of the employees should be accessible only to the authorities concerned and not to everyone. Another example can be the medical records of patients in a hospital. These should be accessible only to health care officials.

Thus, one of the concepts of database security is primarily a specification of access rules about who has what type of access to what information. This is also known as the problem of Authorisation. These access rules are defined at the time database is defined. The person who writes access rules is called the authoriser. The process of ensuring that information and other protected objects are accessed only in authorised ways is called access control. There may be other forms of security relating to physical, operating system, communication aspects of databases. However, in this unit, we will confine ourselves mainly to authorisation and access control using simple commands.

The term integrity is also applied to data and to the mechanism that helps to ensure its consistency. Integrity refers to the avoidance of accidental loss of consistency. Protection of database contents from unauthorised access includes legal and ethical issues, organization policies as well as database management policies. To protect database several levels of security measures are maintained:

1) **Physical:** The site or sites containing the computer system must be physically secured against illegal entry of unauthorised persons.

2) **Human**: An Authorisation is given to a user to reduce the chance of any information leakage and unwanted manipulations.

3) **Operating System:** Even though foolproof security measures are taken to secure database systems, weakness in the operating system security may serve as a means of unauthorised access to the database.

4) **Network:** Since databases allow distributed or remote access through terminals or network, software level security within the network software is an important issue.

5) **Database system:** The data items in a database need a fine level of access control. For example, a user may only be allowed to read a data item and is allowed to issue queries but would not be allowed to deliberately modify the data. It is the responsibility of the database system to ensure that these access restrictions are not violated. Creating database views as discussed in Unit 1 Section 1.6.1 of this block is a very useful mechanism of ensuring database security.

Comment: Can also talk of views to restrict read access.

To ensure database security requires implementation of security at all the levels as above. The Database Administrator (DBA) is responsible for implementing the database security policies in a database system. The organisation or data owners create these policies. DBA creates or cancels the user accounts assigning appropriate security rights to user accounts including power of granting and revoking certain privileges further to other users.

3.4.1 Relationship between Security and Integrity

Database security usually refers to access, whereas database integrity refers to avoidance of accidental loss of consistency. But generally, the turning point or the dividing line between security and integrity is not always clear. *Figure 7* shows the relationship between data security and integrity.

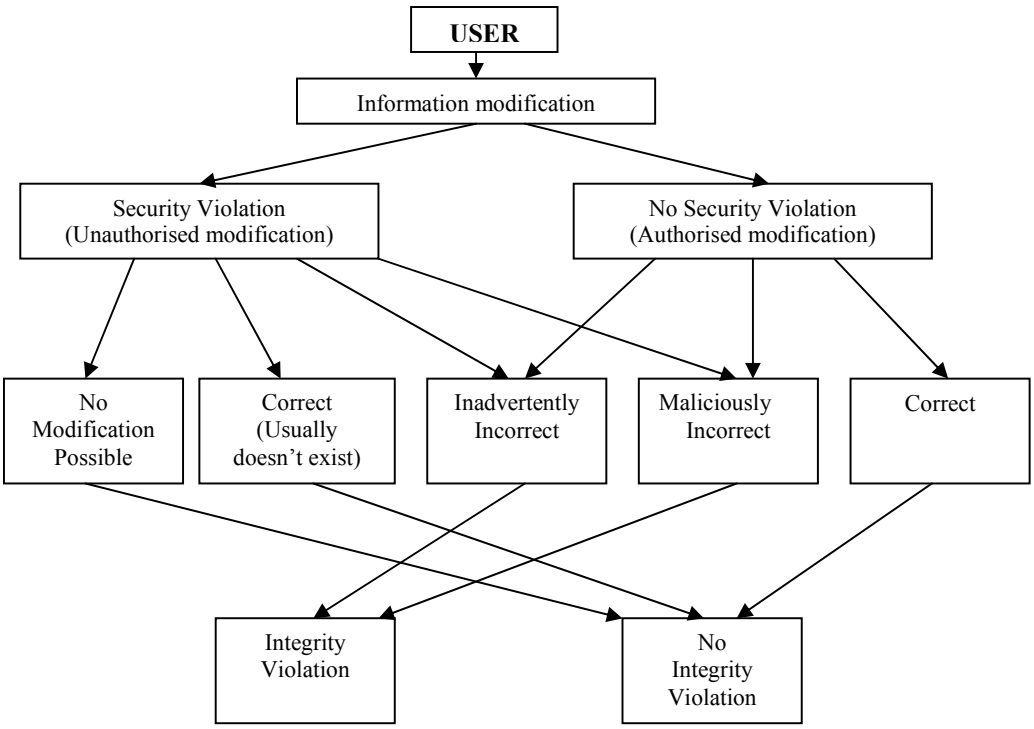


Figure 7: Relationship between security and Integrity

3.4.2 Difference between Operating System and Database Security

Security within the operating system can be implemented at several levels ranging from passwords for access to system, to the isolation of concurrent executing processes with the system. However, there are a few differences between security measures taken at operating system level as compared to those that of database system. These are:

- Database system protects more objects, as the data is persistent in nature. Also database security is concerned with different levels of granularity such as file, tuple, an attribute value or an index. Operating system security is primarily concerned with the management and use of resources.
- Database system objects can be complex logical structures such as views, a number of which can map to the same physical data objects. Moreover different architectural levels viz. internal, conceptual and external levels, have different security requirements. Thus, database security is concerned with the semantics – meaning of data, as well as with its physical representation. Operating system can provide security by not allowing any operation to be performed on the database unless the user is authorized for the operation concerned.

Figure 8 shows the architecture of a database security subsystem that can be found in any commercial DBMS.

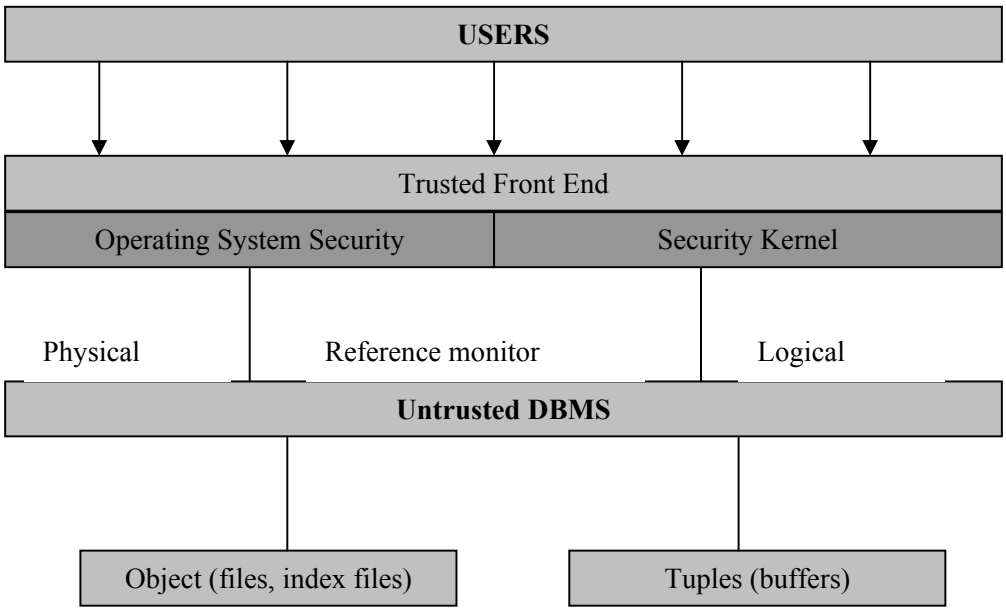


Figure 8: Database Security subsystem

3.5 AUTHORISATION

Authorisation is the culmination of the administrative policies of the organisation. As the name specifies, authorisation is a set of rules that can be used to determine which user has what type of access to which portion of the database. The following forms of authorisation are permitted on database items:

1) **READ**: it allows reading of data object, but not modification, deletion or insertion of data object.
2) **INSERT**: allows insertion of new data, but not the modification of existing data, e.g., insertion of tuple in a relation.
3) **UPDATE**: allows modification of data, but not its deletion. But data items like primary-key attributes may not be modified.

4) **DELETE**: allows deletion of data only.

A user may be assigned all, none or a combination of these types of Authorisation, which are broadly called access authorisations.

In addition to these manipulation operations, a user may be granted control operations like

1) Add: allows adding new objects such as new relations.

2) Drop: allows the deletion of relations in a database.

3) Alter: allows addition of new attributes in a relations or deletion of existing attributes from the database.

4) Propagate Access Control: this is an additional right that allows a user to propagate the access control or access right which s/he already has to some other, i.e., if user A has access right R over a relation S, then if s/he has propagate access control, s/he can propagate her/his access right R over relation S to another user B either fully or part of it. In SQL you can use WITH GRANT OPTION for this right.

You must refer to Section 1.5 of Unit 1 of this block for the SQL commands relating to data and user control.

The ultimate form of authority is given to the database administrator. He is the one who may authorize new users, restructure the database and so on. The process of Authorisation involves supplying information known only to the person the user has claimed to be in the identification procedure.

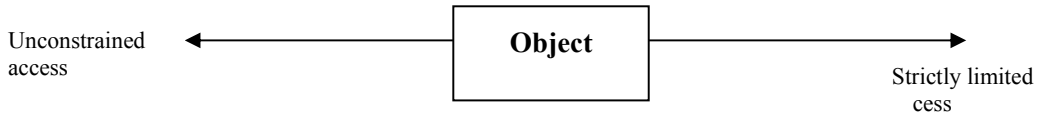
A basic model of Database Access Control

Models of database access control have grown out of earlier work on protection in operating systems. Let us discuss one simple model with the help of the following example:

Security problem: Consider the relation:

Employee (Empno, Name, Address, Deptno, Salary, Assessment)

Assuming there are two users: Personnel manager and general user . What access rights may be granted to each user? One extreme possibility is to grant an unconstrained access or to have a limited access.



One of the most influential protection models was developed by Lampson and extended by Graham and Denning. This model has 3 components:

1) A set of objects: where objects are those entities to which access must be controlled.

2) A set of subjects: where subjects are entities that request access to objects.

3) A set of all access rules: which can be thought of as forming an access (often referred to as authorisation matrix).

Let us create a sample authorisation matrix for the given relation:

Object / Subject	Empno	Name	Address	Deptno	Salary	Assessment
Personnel Manager	Read	All	All	All	All	All
General User	Read	Read	Read	Read	Not accessible	Not accessible

As the above matrix shows, Personnel Manager and general user are the two subjects. Objects of database are Empno, Name, Address, Deptno, Salary and Assessment. As per the access matrix, Personnel Manager can perform any operation on the database of an employee except for updating the Empno that may be self-generated and once given can never be changed. The general user can only read the data but cannot update, delete or insert the data into the database. Also the information about the salary and assessment of the employee is not accessible to the general user.

In summary, it can be said that the basic access matrix is the representation of basic access rules. These rules may be implemented using a view on which various access rights may be given to the users.

☞ Check Your Progress 2

1) What are the different types of data manipulation operations and control operations?

……

……

……

……

……

……

……

2) What is the main difference between data security and data integrity?

……

……

……

……

……

3) What are the various aspects of security problem?

……

……

……

……

4) Name the 3 main components of Database Access Control Model?

……

……

……

……

3.6 SUMMARY

In this unit we have discussed the recovery of the data contained in a database system after failures of various types. The types of failures that the computer system is likely to be subject to include that of components or subsystems, software failures, power outages, accidents, unforeseen situations, and natural or man-made disasters. Database recovery techniques are methods of making the database fault tolerant. The aim of the recovery scheme is to allow database operations to be resumed after a failure with the minimum loss of information and at an economically justifiable cost.

The basic technique to implement database recovery is by using data redundancy in the form of logs, and archival copies of the database. Checkpoint helps the process of recovery.

Security and integrity concepts are crucial since modifications in a database require the replacement of the old values. The DBMS security mechanism restricts users to only those pieces of data that are required for the functions they perform. Security mechanisms restrict the type of actions that these users can perform on the data that is accessible to them. The data must be protected from accidental or intentional (malicious) corruption or destruction. In addition, there is a privacy dimension to data security and integrity.

Security constraints guard against accidental or malicious tampering with data; integrity constraints ensure that any properly authorized access, alteration, deletion, or insertion of the data in the database does not change the consistency and validity of the data. Database integrity involves the correctness of data and this correctness has to be preserved in the presence of concurrent operations, error in the user's operation and application programs, and failures in hardware and software.

3.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Recovery is needed to take care of the failures that may be due to software, hardware and external causes. The aim of the recovery scheme is to allow database operations to be resumed after a failure with the minimum loss of information and at an economically justifiable cost. One of the common techniques is log-based recovery. A transaction is the basic unit of recovery.

2) A checkpoint is a point when all the database updates and logs are written to stable storage. A checkpoint ensures that not all the transactions need to be REDONE or UNDONE. Thus, it helps in faster recovery from failure. You should create a sample example, for checkpoint having a sample transaction log.

3) The following properties should be taken into consideration:

- Loss of data should be minimal
- Recovery should be quick
- Recovery should be automatic
- Affect small portion of database.

Check Your Progress 2

1) Data manipulation operations are:

   - Read
   - Insert
   - Delete
   - Update

   Data control operations are:
   - Add
   - Drop
   - Alter
   - Propagate access control

2) Data security is the protection of information that is maintained in database against unauthorised access, modification or destruction. Data integrity is the mechanism that is applied to ensure that data in the database is correct and consistent.

3)
   - Legal, social and ethical aspects
   - Physical controls
   - Policy questions
   - Operational problems
   - Hardware control
   - Operating system security
   - Database administration concerns

4) The three components are:

   - Objects
   - Subjects
   - Access rights

UNIT 4 DISTRIBUTED AND CLIENT SERVER DATABASES

Structure **Page Nos.**

4.0	Introduction	73
4.1	Objectives	73
4.2	Need for Distributed Database Systems	73
4.3	Structure of Distributed Database	74
4.4	Advantages and Disadvantages of DDBMS	78
	4.4.1 Advantages of Data Distribution	
	4.4.2 Disadvantages of Data Distribution	
4.5	Design of Distributed Databases	81
	4.5.1 Data Replication	
	4.5.2 Data Fragmentation	
4.6	Client Server Databases	87
	4.6.1 Emergence of Client Server Architecture	
	4.6.2 Need for Client Server Computing	
	4.6.3 Structure of Client Server Systems	
	4.6.4 Advantages of Client Server Systems	
4.7	Summary	91
4.8	Solutions/Answers	92

4.0 INTRODUCTION

In the previous units, we have discussed the basic issues relating to Centralised database systems. This unit discusses the distributed database systems which are primarily relational and one important implementation model: the client server model. This unit focuses on the basic issues involved in the design of distributed database designs. The unit also discusses some very basic concepts; in respect of client server computing including the basic structure, advantages/disadvantages etc. It will be worth mentioning here that almost all the commercial database management systems follow such a model of implementation. Although in client server model one of the concepts is the concept of distribution but it is primarily distribution of roles on server computers or rather distribution of work. So a client server system may be a centralised database.

4.1 OBJECTIVES

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

- differentiate DDBMS and conventional DBMS;
- define Network topology for DDBMS;
- define the concepts of horizontal and vertical fragmentation;
- define the concepts and rates of client server computing, and
- identify the needs of all these systems.

4.2 NEED FOR DISTRIBUTED DATABASE SYSTEMS

A *distributed database* is a set of database stored on multiple computers that appears to applications as a single database. As a result, an application can simultaneously access and modify the data in several databases in a network. Each database in the

system is controlled by its local server but cooperates to maintain the consistency of the global distributed database. The computers in a distributed system communicate with each other through various communication media, such as high-speed buses or telephone line. They don't share main memory, nor a clock, although, to work properly many applications on different computers might have to synchronise their clocks. In some cases the absolute time might be important and the clocks might have to be synchronised with atomic clocks.

The processors in a distributed system may vary in size and function such as small microcomputers, workstation, minicomputers, and large general-purpose computer system. These processors are referred to by sites, nodes, computers, and so on, depending on the context in which they are mentioned. We mainly use the term site, in order to emphasise the physical distribution of these systems.

A distributed database system consists of a collection of sites, each of which may participate in the execution of transactions, which access data at one site, or several sites. The main difference between centralised and distributed database systems is that, in the former, the data resides in one single Centralised control, while in the latter, the data resides in several sets under the control of local distributed DBMS components which are under the control of one DBMS. As we shall see, this distribution of data is the cause of many difficulties that will be addressed in this unit.

Independent or decentralised systems were normally used in earlier days. There was duplication of hardware and other facilities. Evolution of computer systems also led to incompatible procedures and lack of management control. A centralised database system was then evolved. In a centralised database the DBMS and data reside at a single database instance. Although for recovery purposes we keep redundant database information yet it is under the control of a single DBMS. A further enhancement of the centralised database system may be to provide access to centralised data from a number of distributed locations through a network. In such a system a site failure except the central site will not result in total system failure. Although, communication technology has greatly improved yet the centralised approach may create problems for an organization that has geographically dispersed operations and data has to be accessed from a centralised database. Some of the problems may be:

a. loss of messages between a local and central site;
b. failure of communication links between local and central site. This would make the system unreliable;
c. excessive load on the central site would delay queries and accesses. A single site would have to bear a large number of transactions and hence would require large computing systems.

The problems as above could be addressed using distributed database systems. It improves the reliability and sharability of data and the efficiency of data access. Distributed Database Systems can be considered a system connected to intelligent remote devices each of which can itself act as a local database repository. All data is accessible from each site. The distributed system increases the efficiency of access because multiple of sites can co-ordinate efficiently to respond to a query and control & processing is limited to this DBMS.

4.3 STRUCTURE OF DISTRIBUTED DATABASE

A distributed database system consists of a collection of sites, each of which maintains a local databases system. Each site is able to process local transactions, those transactions that access data only in that single site. In addition, a site may

participate in the execution of global transactions, those transactions that access data at several sites. The architecture of Distributed Database systems is given in *Figure 1*

Figure 1: **Three different database system architectures. (a) No database sharing architecture. (b) A networked architecture with a centralised database. (c) A distributed database architecture**

The execution of global transactions on the distributed architecture requires communication among the sites. *Figure 2* illustrates a representative distributed database system architecture having transactions. Please note that both the transactions shown are global in nature as they require data from both the sites.

Figure 2: Distributed Database Schema and Transactions

Some of the key issues involving DDBMS are:

- How the data is distributed?
- Is this data distribution hidden from the general users?
- How is the integration of data and its control including security managed locally and globally?
- How are the distributed database connected?

Well we will provide the basic answers to most of these questions during the course of this unit. However, for more details, you may refer to further readings.

The sites in a distributed system can be connected physically in a variety of ways. The various topologies are represented as graphs whose nodes correspond to sites. An edge from node A to node B corresponds to a direct connection between the two sites.

Some of the most common connection structures are depicted in *Figure 3*. The major differences among these configurations involve:

- **Installation cost.** The cost of physically linking the sites in the system
- **Communication cost.** The cost in time and money to send a message from site A to site B.
- **Reliability.** The frequency with which a link or site fails.
- **Availability.** The degree to which data can be accessed despite failure of some links or sites.

These differences play an important role in choosing the appropriate mechanism for handling the distribution of data The sites of a distributed database system may be distributed physically either over a large geographical area (such as all over India), or over a small geographical area such as a single building or a number of adjacent buildings). The former type of network is based on wide area network, while the latter uses local-area network.

Figure 3: Some interconnection Networks

Since the sites in wide area networks are distributed physically over a large geographical area, the communication links are likely to be relatively slow and less reliable as compared with local-area networks. Typical wide area links are telephone lines, microwave links, and satellite channels. Many newer enhanced communication technologies including fiber optics are also used for such links. The local-area network sites are close to each other, communication links are of higher speed and lower error rate than their counterparts in wide area networks. The most common links are twisted pair, base band coaxial, broadband coaxial, and fiber optics.

A Distributed Transaction

Let us illustrate the concept of a distributed transaction by considering a banking system consisting of three branches located in three different cities. Each branch has its own computer with a database consisting of all the accounts maintained at that branch. Each such installation is thus a site. There also exists one single site which maintains information about all the other branches of the bank. Suppose that the database systems at the various sites are based on the relational model. Each branch maintains its portion of the relation: DEPOSIT (DEPOSIT-BRANCH) where

DEPOSIT-BRANCH = (branch-name, account-number, customer-name, balance)

A site containing information about the four branches maintains the relation branch-details, which has the schema as:

BRANCH-DETAILS (branch-name, Financial_ health, city)

There are other relations maintained at the various sites which are ignored for the purpose of our example.

A local transaction is a transaction that accesses accounts in one single site, at which the transaction was initiated. A global transaction, on the other hand, is one which either accesses accounts in a site different from the one at which the transaction was initiated, or accesses accounts in several different sites. To illustrate the difference between these two types of transactions, consider the transaction to add Rs.5000 to account number 177 located at the Delhi branch. If the transaction was initiated at the Delhi branch, then it is considered local; otherwise, it is considered global. A transaction to transfer Rs.5000 from account 177 to account 305, which is located at the Bombay branch, is a global transaction since accounts in two different sites are accessed as a result of its execution. A transaction finding the total financial standing of all branches is global.

What makes the above configuration a distributed database system are the facts that:

- The various sites may be locally controlled yet are aware of each other.
- Each site provides an environment for executing both local and global transactions.

However, a user need not know whether on underlying application is distributed or not.

4.4 ADVANTAGES AND DISADVANTAGES OF DDBMS

There are several reasons for building distributed database systems, including sharing of data, reliability and availability, and speedup of query processing. However, along with these advantages come several disadvantages, including software development

cost, greater potential for bugs, and increased processing overheads. In this section, we shall elaborate briefly on each of these.

4.4.1 Advantages of Data Distribution

The primary advantage of distributed database systems is the ability to share and access data in a reliable and efficient manner.

Data sharing and Distributed Control

The geographical distribution of an organization can be reflected in the distribution of the data; if a number of different sites are connected to each other, then a user at one site may be able to access data that is available at another site. The main advantage here is that the user need not know the site from which data is being accessed. Data can be placed at the site close to the users who normally use that data. The local control of data allows establishing and enforcement of local policies regarding use of local data. A global database administrator (DBA) is responsible for the entire system. Generally, part of this responsibility is given to the local administrator, so that the local DBA can manage the local DBMS. Thus in the distributed banking system, it is possible for a user to get his/her information from any branch office. This external mechanism would, in effect to a user, look to be a single centralised database.

The primary advantage of accomplishing data sharing by means of data distribution is that each site is able to retain a degree of control over data stored locally. Depending upon the design of the distributed database system, each local administrator may have a different degree of autonomy. This is often a major advantage of distributed databases. In a centralised system, the database administrator of the central site controls the database and, thus, no local control is possible.

Reflects organisational structure

Many organizations are distributed over several locations. If an organisation has many offices in different cities, databases used in such an application are distributed over these locations. Such an organisation may keep a database at each branch office containing details of the staff that work at that location, the local properties that are for rent, etc. The staff at a branch office will make local inquiries to such data of the database. The company headquarters may wish to make global inquiries involving the access of data at all or a number of branches.

Improved Reliability

In a centralised DBMS, a server failure terminates the operations of the DBMS. However, a failure at one site of a DDBMS, or a failure of a communication link making some sites inaccessible, does not make the entire system inoperable. Distributed DBMSs are designed to continue to function despite such failures. In particular, if data are replicated in several sites, a transaction needing a particular data item may find it at several sites. Thus, the failure of a site does not necessarily imply the shutdown of the system.

The failure of one site must be detected by the system, and appropriate action may be needed to recover from the failure. The system must no longer use the services of the failed site. Finally, when the failed site recovers or is repaired, mechanisms must be available to integrate it smoothly back into the system. The recovery from failure in distributed systems is much more complex than in a centralised system.

Improved availability

The data in a distributed system may be replicated so that it exists at more than one site. Thus, the failure of a node or a communication link does not necessarily make the data inaccessible. The ability of most of the systems to continue to operate despite the failure of one site results in increased availability which is crucial for database

systems used for real-time applications. For example, loss of access to data in an airline may result in the loss of potential ticket buyers to competitors.

Improved performance

As the data is located near the site of its demand, and given the inherent parallelism due to multiple copies, speed of database access may be better for distributed databases than that of the speed that is achievable through a remote centralised database. Furthermore, since each site handles only a part of the entire database, there may not be the same contention for CPU and I/O services as characterized by a centralised DBMS.

Speedup Query Processing

A query that involves data at several sites can be split into sub-queries. These sub-queries can be executed in parallel by several sites. Such parallel sub-query evaluation allows faster processing of a user's query. In those cases in which data is replicated, queries may be sent to the least heavily loaded sites.

Economics

It is now generally accepted that it costs less to create a system of smaller computers with the equivalent power of a single large computer. It is more cost-effective to obtain separate computers. The second potential cost saving may occurs where geographically remote access to distributed data is required. In such cases the economics is to minimise cost due to the data being transmitted across the network for data updates as opposed to the cost of local access. It may be more economical to partition the application and perform the processing locally at application site.

Modular growth

In distributed environments, it is easier to expand. New sites can be added to the network without affecting the operations of other sites, as they are somewhat independent. This flexibility allows an organisation to expand gradually. Adding processing and storage power to the network can generally result in better handling of ever increasing database size. A more powerful system in contrast, a centralised DBMS, would require changes in both the hardware and software with increasing size and more powerful DBMS to be procured.

4.4.2 Disadvantages of Data Distribution

The primary disadvantage of distributed database systems is the added complexity required to ensure proper coordination among the sites. This increased complexity takes the form of:

- **Higher Software development cost:** Distributed database systems are complex to implement and, thus, more costly. Increased complexity implies that we can expect the procurement and maintenance costs for a DDBMS to be higher than those for a centralised DBMS. In addition to software, a distributed DBMS requires additional hardware to establish a network between sites. There are ongoing communication costs incurred with the use of this network. There are also additional maintenance costs to manage and maintain the local DBMSs and the network.

- **Greater potential for bugs:** Since the sites of a distributed system operate concurrently, it is more difficult to ensure the correctness of algorithms. The art of constructing distributed algorithms is an active and important area of research.

- **Increased processing overhead:** The exchange of messages and the additional computation required to achieve coordination among the sites is an overhead that does not arise in centralised systems.

- **Complexity:** A distributed DBMS that is reliable, available and secure is inherently more complex than a centralised DBMS. Replication of data discussed in the next section, also adds to complexity of the distributed DBMS. However, adequate data replication is necessary to have availability, reliability, and performance.

- **Security:** In a centralised system, access to the data can be easily controlled. However, in a distributed DBMS not only does access to replicated data have to be controlled in multiple locations, but also the network needs to be secured. Networks over a period have become more secure, still there is a long way to go.

- **Lack of standards and experience:** The lack of standards has significantly limited the potential of distributed DBMSs. Also, there are no tools or methodologies to help users convert a centralised DBMS into a distributed DBMS.

- **General Integrity control more difficult:** Integrity is usually expressed in terms of constraints or the rules that must be followed by database values. In a distributed DBMS, the communication and processing costs that are required to enforce integrity constraints may be very high as the data is stored at various sites. However, with better algorithms we can reduce such costs.

- **Purpose:** General-purpose distributed DBMSs have not been widely accepted. We do not yet have the same level of experience in industry as we have with centralised DBMSs.

- **Database design more complex:** Besides the normal difficulties of designing a centralised database, the design of a distributed database has to take account of fragmentation of data, allocation of fragments to specific sites, and data replication. The designer of distributes systems must balance the advantages against the disadvantages of distribution of data.

☞ **Check Your Progress 1**

1) What are the advantages of distributed databases over centralised Databases?

……

……

……

2) Differntiate between global and local transaction.

……

……

……

4.5 DESIGN OF DISTRIBUTED DATABASES

The distributed databases are primarily relational at local level. So a local database schema is the same as that of a centralised database design. However, a few more dimensions have been added to the design of distributed database. These are:

- **Replication:** It is defined as a copy of a relation. Each replica is stored at a different site. The alternative to replication is to store only one copy of a relation which is not recommended in distributed databases.

- **Fragmentation:** It is defined as partitioning of a relation into several fragments. Each fragment can be stored at a different site.

The distributed database design is a combination of both these concepts. Let us discuss them in more detail in the following subsections.

4.5.1 Data Replication

"If a relation R has its copies stored at two or more sites, then it is considered replicated".

But why do we replicate a relation?

There are various advantages and disadvantages of replication of relation

- **Availability:** A site containing the copy of a replicated relation fails, even then the relation may be found in another site. Thus, the system may continue to process at least the queries involving just read operation despite the failure of one site. Write can also be performed but with suitable recovery algorithm.

- **Increased parallelism:** Since the replicated date has many copies a query can be answered from the least loaded site or can be distributed. Also, with more replicas you have greater chances that the needed data is found on the site where the transaction is executing. Hence, data replication can minimise movement of data between sites.

- **Increased overheads on update:** On the disadvantage side, it will require the system to ensure that all replicas of a relation are consistent. This implies that all the replicas of the relation need to be updated at the same time, resulting in increased overheads. For example, in a banking system, if account information is replicated at various sites, it is necessary that the balance in a particular account should be the same at all sites.

The problem of controlling concurrent updates by several transactions to replicated data is more complex than the centralised approach of concurrency control. The management of replicas of relation can be simplified by choosing one of the replicas as the primary copy. For example, in a banking system, the primary copy of an account may be the site at which the account has been opened. Similarly, in an airline reservation system, the primary copy of the flight may be associated with the site at which the flight originates. The replication can be classified as:

Complete replication: It implies maintaining of a complete copy of the database at each site. Therefore, the reliability and availability and performance for query response are maximized. However, storage costs, communication costs, and updates are expensive. To overcome some of these problems relation, snapshots are sometimes used. A snapshot is defined as the copy of the data at a given time. These copies are updated periodically, such as, hourly or weekly. Thus, snapshots may not always be up to date.

Selective replication: This is a combination of creating small fragments of relation and replicating them rather than a whole relation. The data should be fragmented on need basis of various sites, as per the frequency of use, otherwise data is kept at a centralised site. The objective of this strategy is to have just the advantages of the other approach but none of the disadvantages. This is the most commonly used strategy as it provides flexibility.

4.5.2 Data Fragmentation

"Fragmentation involves decomposing the data in relation to non-overlapping component relations".

Why do we need to fragment a relation? The reasons for fragmenting a relation are:

Use of partial data by applications: In general, applications work with views rather than entire relations. Therefore, it may be more appropriate to work with subsets of relations rather than entire data.

Increases efficiency: Data is stored close to most frequently used site, thus retrieval would be faster. Also, data that is not needed by local applications is not stored, thus the size of data to be looked into is smaller.

Parallelism of transaction execution: A transaction can be divided into several sub-queries that can operate on fragments in parallel. This increases the degree of concurrency in the system, thus allowing transactions to execute efficiently.

Security: Data not required by local applications is not stored at the site, thus no unnecessary security violations may exist.

But how do we carry out fragmentation? Fragmentation may be carried out as per the following rules:

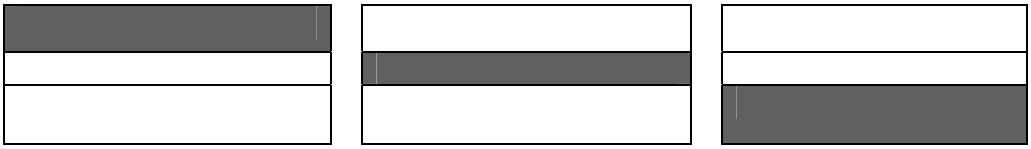
a) **Completeness:** This rule ensures that there is no loss of data during fragmentation. If a relation is decomposed into fragments, then each data item must appear in at least one fragment.

b) **Reconstruction:** This rule ensures preservation of functional dependencies. It should be possible to define a relational operation that will reconstruct the relation from its fragments.

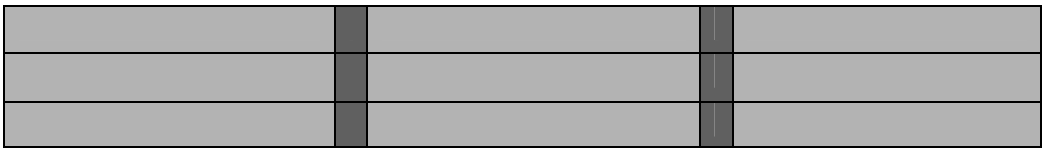
c) **Disjointness:** *A* data item that appears in a fragment should not appear in any other fragment. However, a typical fragmentation called vertical fragmentation is an exception to this rule. In vertical fragmentation the primary key attributes must be repeated to allow reconstruction of original relation. This rule ensures minimization of data redundancy.

What are the different types of fragmentation?

There are two main types of fragmentation: **horizontal** and **vertical**. Horizontal fragments, as the name suggests are subsets of tuples and vertical fragments are subsets of attributes (refer to Figure 4). There are also two other types of fragmentation: mixed, and derived–a type of horizontal fragmentation, are just introduced. A detailed discussion on them is beyond the scope of this unit.



(a) Horizontal Fragmentation



(b) Vertical Fragmentation

Figure 4: Horizontal and Vertical Fragmentation

Horizontal Fragmentation

Horizontal fragmentation groups together the tuples in a relation that are collectively used by the important transactions. A horizontal fragment is produced by specifying a WHERE clause condition that performs a restriction on the tuples in the relation. It can also be defined using the *Selection* operation of the relational algebra.

Example:

Let us illustrate horizontal fragmentation with the help of an example.

DEPOSIT (branch-code, account-number, customer-name, balance)

A sample relation instance of the relation DEPOSIT is shown in *Figure 5*.

Branch-code	**Account number**	**Customer name**	**Balance**
1101	3050	Suresh	5000
1101	2260	Swami	3360
1102	1170	Swami	2050
1102	4020	Khan	10000
1101	1550	Khan	620
1102	4080	Khan	1123
1102	6390	Khan	7500

Figure 5: Sample DEPOSIT relation

Mathematically a fragment may be defined as a selection on the global relation R. The reconstruction of the relation R can be obtained by taking the union of all fragments.

So let us decompose the table in Figure 5 into horizontal fragments. Let us do these fragments on the branch-code as 1101 and 1102

DEPOSIT1 obtained by selection on branch-code as 1101			
Branch-code	**Account number**	**Customer name**	**Balance**
1101	3050	Suresh	5000
1101	2260	Swami	3360
1101	1550	Khan	620
DEPOSIT2 obtained by selection on branch- code as 1102			
Branch-code	**Account number**	**Customer name**	**Balance**
1102	1770	Swami	2050
1102	4020	Khan	10000
1102	4080	Khan	1123
1102	6390	Khan	7500

Figure 6: Horizontal fragmentation of relation DEPOSIT

The two fragments can be defined in relational algebra as:

$$\text{DEPOSIT1} = \sigma_{\text{branch-code= 1101}} (\text{DEPOSIT})$$

$$\text{DEPOSIT2} = \sigma_{\text{branch-code= 1102}} (\text{DEPOSIT})$$

These two fragments are shown in *Figure 6*. Fragment 1 can be stored in the branch whose code is 1101 while the second fragment can be stored at branch 1102.

In our example, the fragments are disjoint. However, by changing the selection predicates used to construct the fragments; we may have overlapping horizontal fragments. This is a form of data replication.

Vertical Fragmentation

Vertical fragmentation groups together only those attributes in a relation that are used jointly by several important transactions. A vertical fragment is defined using the **Projection** operation of the relational algebra. In its most simple form, vertical

fragmentation is the same as that of decomposition. In general, a relation can be constructed on taking Natural join of all vertical fragments.

More generally, vertical fragmentation is accomplished by adding a special attribute called a tuple-number to the scheme R. A tuple-number is a physical or logical address for a tuple. Since each tuple in R must have a unique address, the tuple-number attribute is a key to the new fragments obtained (please refer to *Figure 7*).

Branch-code	Account number	Customer name	Balance	Tuple-number
1101	3050	Suresh	5000	1
1101	2260	Swami	3360	2
1102	1170	Swami	2050	3
1102	4020	Khan	10000	4
1101	1550	Khan	620	5
1102	4080	Khan	1123	6
1102	6390	Khan	7500	7

Figure 7: The relation DEPOSIT of figure 5 with tuple- numbers

This relation now can be decomposed into two fragments as: shows a vertical decomposition of the scheme Deposit-scheme tuple number into:

$$\text{DEPOSIT3} = \prod_{\text{(branch-code. customer-name, tuple-number)}} (\text{DEPOSIT})$$

$$\text{DEPOSIT4} = \prod_{\text{(account-number, balance, tuple-number)}} (\text{DEPOSIT})$$

The example of Figure7 on this basis would become:

DEPOSIT3		
Branch-code	**Customer-name**	**Tuple-number**
1101	Suresh	1
1101	Swami	2
1102	Swami	3
1102	Khan	4
1101	Khan	5
1102	Khan	6
1102	Khan	7
DEPOSIT4		
Account number	**Balance**	**Tuple-number**
3050	5000	1
2260	3360	2
1170	2050	3
4020	10000	4
1550	620	5
4080	1123	6
6390	7500	7

Figure 8: Vertical fragmentation of relation DEPOSIT

How can we reconstruct the original relation from these two fragments? By taking natural join of the two vertical fragments on tuple-number. The tuple number allows direct retrieval of the tuples without the need for an index. Thus, this natural join may be computed much more efficiently than typical natural join.

However, please note that as the tuple numbers are system generated, therefore they should not be visible to general users. If users are given access to tuple-number, it becomes impossible for the system to change tuple addresses.

Mixed fragmentation

Sometimes, horizontal or vertical fragmentation of a database schema by itself is insufficient to adequately distribute the data for some applications. Instead, **mixed** or **hybrid** fragmentation is required. Mixed fragmentation consists of a horizontal fragment that is vertically fragmented, or a vertical fragment that is then horizontally fragmented.

Derived horizontal fragmentation

Some applications may involve a join of two or more relations. If the relations are stored at different locations, there may be a significant overhead in processing the join. In such cases, it may be more appropriate to ensure that the relations, or fragments of relations, that are joined together are at the same location. We can achieve this using derived horizontal fragmentation.

After going through the basis of concepts relating to distributed database systems, let us sketch the process of design of distributed database systems.

A step-wise distributed database design methodology

Following is a step-wise methodology for distributed database design.

(1) Examine the nature of distribution. Find out whether an organisation needs to have a database at each branch office, or in each city, or possibly at a regional level. It has direct implication from the viewpoint of fragmentation. For example, in case database is needed at each branch office, the relations may fragment on the basis of branch number.
(2) Create a detailed global E-R diagram, if so needed and identify relations from entities.
(3) Analyse the most important transactions in the system and identify where horizontal or vertical fragmentation may be desirable and useful.
(4) Identify the relations that are not to be fragmented. Such relations will be replicated everywhere. From the global ER diagram, remove the relations that are not going to be fragmented.
(5) Examine the relations that are on one-side of a relationship and decide a suitable fragmentation schema for these relations. Relations on the many-side of a relationship may be the candidates for derived fragmentation.
(6) During the previous step, check for situations where either vertical or mixed fragmentation would be needed, that is, where the transactions require access to a subset of the attributes of a relation.

☞ Check Your Progress 2

1) What are the rules while fragmenting data?

……

……

……

……

2) What is Data Replication? What are its advantages & disadvantages?

……

……

……

……

……

……

……

4.6 CLIENT SERVER DATABASES

The concept behind the Client/Server systems is concurrent, cooperative processing. It is an approach that presents a single systems view from a user's viewpoint. It involves processing on multiple, interconnected machines. It provides coordination of activities in a manner transparent to end-users. Remember, client-server database is distribution of activities into clients and a server. It may have a centralised or distributed database system at the server backend. It is primarily a very popular commercial database implementation model.

4.6.1 Emergence of Client Server Architecture

Some of the pioneering work that was done by some of the relational database vendors allowed the computing to be distributed on multiple computers on network using contemporary technologies involving:

- Low Cost, High Performance PCs and Servers
- Graphical User Interfaces
- Open Systems
- Object-Orientation
- Workgroup Computing
- EDI and E-Mail
- Relational Databases
- Networking and Data Communication.

4.6.2 Need for Client Server Computing

Client/Server (C/S) architecture involves running the application on multiple machines in which each machine with its component software handles only a part of the job. Client machine is basically a PC or a workstation that provides presentation services and the appropriate computing, connectivity and interfaces while the server machine provides database services, connectivity and computing services to multiple users. Both client machines and server machines are connected to the same network. As the number of users grows, client machines can be added to the network while as the load on the database server increases more servers can be connected through the network. Thus, client-server combination is a scalable combination. Server machines are more powerful machines database services to multiple client requests.

The client-server systems are connected though the network. This network need not only be the Local Area Network (LAN). It can also be the Wide Area Network (WAN) across multiple cities. The client and server machines communicate through standard application program interfaces (called API), and remote procedure calls (RPC). The language through which RDBMS based C/S environment communicate is the structured query language (SQL).

4.6.3 Structure of Client Server Systems

In client/server architecture, clients represent users who need services while servers provide services. Both client and server are a combination of hardware and software. Servers are separate logical objects that communicate with clients over a network to perform tasks together. A client makes a request for a service and receives a reply to that request. A server receives and processes a request, and sends back the required response. The client/server systems may contain two different types of architecture - 2-Tier and 3-Tier Client/Server Architectures

Every client/server application contains three functional units:

- Presentation logic which provides the human/machine interaction (the user interface). The presentation layer handles input from the keyboard, mouse, or

other input devices and provides output in the form of screen displays. For example, the ATM machine of a bank provides such interfaces.

- Business logic is the functionality provided to an application program. For example, software that enables a customer to request to operate his/her balance on his/her account with the bank is business logic. It includes rules for withdrawal, for minimum balance etc. It is often called business logic because it contains the business rules that drive a given enterprise.

- The bottom layer provides the generalized services needed by the other layers including file services, print services, communications services and database services. One example of such a service may be to provide the records of customer accounts.

These functional units can reside on either the client or on one or more servers in the application:

Figure 9: A client-server system

In general two popular client/server systems are:

- 2-Tier client Server Models

- 3-Tier client server Model

2-Tier Client/Server Models

Initial two-tier (client/server) applications were developed to access large databases available on the server side and incorporated the rules used to manipulate the data with the user interface into the client application. The primary task of the server was simply to process as many requests for data storage and retrieval as possible.

Two-tier client/server provides the user system interface usually on the desktop environment to its users. The database management services are usually on the server that is a more powerful machine and services many clients. Thus, 2-Tier client-server architecture splits the processing between the user system interface environment and the database management server environment. The database management server also provides stored procedures and triggers. There are a number of software vendors that provide tools to simplify the development of applications for the two-tier client/server architecture.

In 2-tier client/server applications, the business logic is put inside the user interface on the client or within the database on the server in the form of stored procedures. This results in division of the business logic between the client and server. File servers and database servers with stored procedures are examples of 2-tier architecture.

The two-tier client/server architecture is a good solution for distributed computing. Please note the use of words distributed computing and not distributed databases. A 2-tier client server system may have a centralised database management system or distributed database system at the server or servers. A client group of clients on a LAN can consist of a dozen to 100 people interacting simultaneously. A client server system does have a number of limitations. When the number of users exceeds 100, performance begins to deteriorate. This limitation is a result of the server maintaining a connection via communication messages with each client, even when no work is being done.

A second limitation of the two-tier architecture is that implementation of processing management services using vendor proprietary database procedures restricts flexibility and choice of DBMS for applications. The implementations of the two-tier architecture provides limited flexibility in moving program functionality from one server to another without manually regenerating procedural code.

Figure 10: 2-tier and 3 tier client server systems

Some of the major functions performed by the client of a two-tier application are: present a user interface, gather and process user input, perform the requested processing, report the status of the request.

This sequence of commands can be repeated as many times as necessary. Because servers provide only access to the data, the client uses its local resources to perform most of the processing. The client application must contain information about where the data resides and how it is organised in the server database. Once the data has been retrieved, the client is responsible for formatting and displaying it to the user.

3-tier architecture

As the number of clients increases the server would be filled with the client requests. Also, because much of the processing logic was tied to applications, changes in business rules lead to expensive and time-consuming alterations to source code. Although the ease and flexibility of two-tier architecture tools continue to drive many small-scale business applications, the need for faster data access and rapid developmental and maintenance timelines has persuaded systems developers to seek out a new way of creating distributed applications.

The three-tier architecture emerged to overcome the limitations of the two tier architecture (also referred to as the multi-tier architecture). In the three-tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. The middle tier may consist of transaction processing monitors, message servers, or application servers.

The middle-tier can perform queuing, application execution, and database staging. For example, on a middle tier that provides queuing, the client can deliver its request to the middle layer and simply gets disconnected because the middle tier will access the data and return the answer to the client. In addition the middle layer adds scheduling and prioritisation for various tasks that are currently being performed.

The three-tier client/server architecture has improved performance for client groups with a large number of users (in the thousands) and improves flexibility when compared to the two-tier approach. Flexibility is in terms of simply moving an application on to different computers in three-tier architecture. It has become as simple as “drag and drop” tool. Recently, mainframes have found a new use as servers in three-tier architectures:

In 3-tier client/server applications, the business logic resides in the middle tier, separate from the data and user interface. In this way, processes can be managed and deployed separately from the user interface and the database. Also, 3-tier systems can integrate data from multiple sources.

4.6.4 Advantages of Client Server Computing

Client/Server systems have following advantages:

- They provide low cost and user-friendly environment
- They offer expandability that ensures that the performance degradation is not so much with increased load.
- They allow connectivity with the heterogeneous machines deals with real time data feeders like ATMs, Numerical machines.
- They allow enforcement of database management activities including security, performance, backup, integrity to be a part of the database server machine. Thus, avoiding the requirement to write a large number of redundant piece of code dealing with database field validation and referential integrity.

- One major advantage of the client/server model is that by allowing multiple users to simultaneously access the same application data, updates from one computer are instantly made available to all computers that had access to the server.

Client/server systems can be used to develop highly complex multi-user database applications being handled by any mainframe computer.

Since PCs can be used as clients, the application can be connected to the spreadsheets and other applications through Dynamic Data Exchange (DDE) and Object Linking and Embedding (OLE). If the load on database machine grows, the same application can be run on a slightly upgraded machine provided it offers the same version of RDBMSs.

A more detailed discussion on these topics is beyond the scope of this unit. You can refer to further readings for more details.

☞ Check your Progress 3

1) What are the advantages of Client/Server Computing?

……

……

2) Describe the Architecture of Client/Server system.

……

……

4.7 SUMMARY

A distributed database system consists of a collection of sites, each of which maintains a local database system. Each site is able to process local transactions–those transactions that access data only at that single site. In addition, a site may participate in the execution of global transactions–those transactions that access data at several sites. The execution of global transactions requires communication among the sites.

There are several reasons for building distributed database systems, including sharing of data, reliability and availability, and speed of query processing. However, along with these advantages come several disadvantages, including software development cost, greater potential for bugs, and increased processing overheads. The primary disadvantage of distributed database systems is the added complexity required to ensure proper co-ordination among the sites.

There are several issues involved in storing a relation in the distributed database, including replication and fragmentation. It is essential that the system minimise the degree to which a user needs to be aware of how a relation is stored.

Companies that have moved out of the mainframe system to Client/Server architecture have found three major advantages:

- Client/Server technology is more flexible and responsive to user needs
- A significant reduction in data processing costs
- An increase in business competitiveness as the market edge turns towards merchandising.

4.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1) The advantages of DDBMS are:

- The primary advantages of distributed database systems is the ability to share and access data in a reliable and efficient manner. A user at one site may be able to access data that is available at another site.
- Each site is able to retain a degree of control over data stored locally.
- It reflects organisational structure
- If distributed over several locations then improved availability of DBMS. A failure of a communication link making some sites inaccessible does not make the entire system inoperable. The ability of most of the systems to continue to operate despite failure of onesite results in increased availability, which is crucial for database systems used for real-time applications.
- Speed of database access may be better than that achievable from a remote centralised database. Parallel computation allows faster processing of a user's query.
- It costs much less to create a system of smaller computers with lower equivalent a single large computer. It is much easier to expand. New sites can be added to the network without affecting the operations of other sites.

2)

Global Transactions	Local Transactions
Involves multiple sites	Can be performed locally
Uses data communication links	Performed locally
Takes longer to execute, in general	Faster
Performance depends on global schema objects available	Performance is governed by local schema objects like local indexes
Just 20-25 % of total	Most transactions are local

Check Your Progress 2

1) Fragmentation cannot be carried out haphazardly. There are three rules that must be followed during fragmentation.

   a) Completeness: If a relation R is decomposed into fragments R1, R2,.. Rn, each data item that can be found in R must appear in at least one fragment.

   b) Reconstruction: It must be possible to define a relational operational that will reconstruct the relation R from the fragments.

   c) Disjointness: If a data item d appears in fragment R1 then it should not appear in any other fragment. Vertical fragmentation is the exception to this rule, where primary key attributes must be repeated to allow reconstruction.

2) If relation R is replicated a copy of relation R is stored in two or more sites. There are a number of advantages and disadvantages to replication.

Availability: If one of the sites containing relation R fails, then the relation R may found in another site.

Increased parallelism: In cases where the majority of access to the relation R results in only the reading of the relation, several sites can process queries involving R in parallel.

Increasing overhead on update: The system must ensure that all replicas of a relation R are consistent, otherwise erroneous computations may result. This implies that whenever R is updated, this update must be propagated to all sites containing replicas, resulting in increased overheads.

Check Your Progress 3

1) Advantages of client/server computing are as follows:

- C/S computing caters to low-cost and user-friendly environment.
- It offers expandability.
- It ensures that the performance degradation is not so much with increased load.
- It allows connectivity with the heterogeneous machines and also with real time data feeders.
- It allows server enforced security, performance, backup, integrity to be a part of the database machine avoiding the requirement to write a large number of redundant piece of code dealing with database field validation and referential integrity, allows multiple users to simultaneously access the same application data.
- Updates from one computer are instantly made available to all computers that had access to the server.
- It can be used to develop highly complex multi-user database applications being handled by any mainframe computer. If the load on database machine grows, the same application can be run on a slightly upgraded machine.

2) **2-Tier client/Server Models**

With two-tier client/server, the user system interface is usually located in the user's desktop environment and the database management services are usually in a server. Processing management is spilt between the user system interface environment and the database management server environment. The database management server provides stored procedures and triggers. In 2-tier client/server applications, the business logic is buried inside the user interface on the client or
within the database on the server in the form of stored procedures. Alternatively, the business logic can be divided between client and server.

3-tier architecture

The three-tier architecture (also referred to as the multi-tier architecture) emerged to overcome the limitations of the two-tier architecture. In the three-tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing, application execution, and database staging. In addition the middle layer adds scheduling and prioritization for work in progress. The three-tier client/server architecture has been shown to improve performance for groups with a large number of users (in the thousands) and improves flexibility when compared to the two-tier approach.

In 3-tier client/server application, the business logic resides in the middle tier, separate from the data and user interface. In this way, processes can be managed and deployed separately from the user interface and the database. Also, 3-tier systems can integrate data from multiple sources.

SOCIAS-IGNOU/P.O.10.5T/MAY, 2004