UNIT I RELATIONAL DATABASES

Purpose of Database System – Views of data – Data Models – Database System Architecture – Introduction to relational databases – Relational Model – Keys – Relational Algebra – SQL fundamentals – Advanced SQL features – Embedded SQL– Dynamic SQL

INTRODUCTION

DATABASE

Database is collection of data which is related by some aspect. Data is collection of facts and figures which can be processed to produce information. Mostly data represents recordable facts. Data aids in producing information which is based on facts. A database management system stores data, in such a way which is easier to retrieve, manipulate and helps to produce information.

So a database is a collection of related data that we can use for

- Defining - specifying types of data
- Constructing - storing & populating
- Manipulating - querying, updating, reporting

DISADVANTAGES OF FILE SYSTEM OVER DB

In the early days, File-Processing system is used to store records. It uses various files for storing the records.

Drawbacks of using file systems to store data:

- Data redundancy and inconsistency
  -Multiple file formats, duplication of information in different files
- Difficulty in accessing data
  Need to write a new program to carry out each new task
- Data isolation — multiple files and formats
- Integrity problems
  - Hard to add new constraints or change existing ones
- Atomicity problem
  -Failures may leave database in an inconsistent state with partial updates carried Out. E.g. transfer of funds from one account to another should either complete or not happen at all
- Concurrent access anomalies
  - Concurrent accessed needed for performance
- Security problems

Database systems offer solutions to all the above problems

PURPOSE OF DATABASE SYSTEM

The typical file processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. A file processing system has a number of major disadvantages.

- Data redundancy and inconsistency
- Difficulty in accessing data
- Data isolation – multiple files and formats
- Integrity problems
- Atomicity of updates
- Concurrent access by multiple users
- Security problems

1.Data redundancy and inconsistency:

In file processing, every user group maintains its own files for handling its data processing applications.

Example:

Consider the UNIVERSITY database. Here, two groups of users might be the course registration personnel and the accounting office. The accounting office also keeps data on registration and related billing information, whereas the registration office keeps track of student courses and grades.Storing the same data multiple times is called data redundancy.This redundancy leads to several problems.

•Need to perform a single logical update multiple times.
•Storage space is wasted.
•Files that represent the same data may become inconsistent.

Data inconsistency is the various copies of the same data may no larger Agree. **Example:**

One user group may enter a student's birth date erroneously as JAN-19-1984, whereas the other user groups may enter the correct value of JAN-29-1984.

2. Difficulty in accessing data

File processing environments do not allow needed data to be retrieved in a convenient and efficient manner.

3.Data isolation

Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

4.Integrity problems

The data values stored in the database must satisfy certain types of consistency constraints. **Example:**
The balance of certain types of bank accounts may never fall below a prescribed amount . Developers enforce these constraints in the system by addition appropriate code in the various application programs

5.Atomicity problems

Atomic means the transaction must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file processing system.

Example:

Consider a program to transfer $50 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account A but was not credited to account B, resulting in an inconsistent database state.

6.Concurrent access anomalies

For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to
provide because data may be accessed by many different application programs that have not been coordinated previously.

Example: When several reservation clerks try to assign a seat on an airline flight, the system should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger.

7. Security problems

Enforcing security constraints to the file processing system is difficult.

APPLICATION OF DATABASE

Database Applications

- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grades
- Sales: customers, products, purchases
- Manufacturing: production, inventory, orders, supply chain
- Human resources: employee records, salaries, tax deductions
- Telecommunication: Call History, Billing
- Credit card transactions: Purchase details,Statements

VIEWS OF DATA

It refers that how database is actually stored in database, what data and structure of data used by database for data. So describe all this database provides user with views and these are

- **Data abstraction**
- **Instances and schemas**

Data abstraction

As a data in database are stored with very complex data structure so when user come and want to access any data, he will not be able to access data if he has go through this data structure. So to simplify the interaction of user and database, DBMS hides some information which is not of user interest, a this is called data abstraction:- **So developer hides complexity from user and store abstract view of data.**

Data abstraction has three level of abstractions

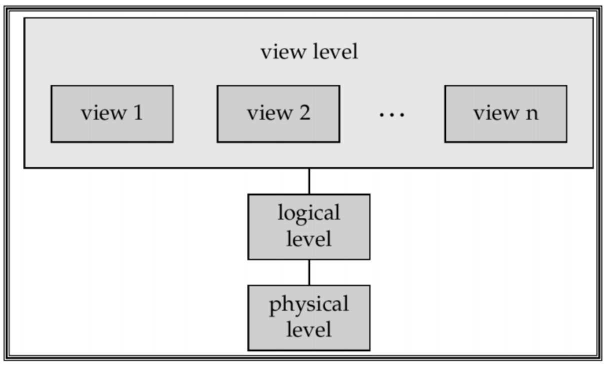
- **level / internal level**
- **Logical level / conceptual level**
- **view level / external level**

Physical level:- this is the lowest level of data abstraction which describe How data is actual stored in database. This level basically describe the data structure and access path /indexing use for accessing file.

Logical level:- The next level of abstraction describe what data are stored in the database and what are the relationship existed among those of data.

View level:- In this level user only interact with database and the complexity remain unview . user see data and there may be many views of one data like chart and graph.

View of Data



Levels of Abstraction

DATA MODELS IN DBMS

A **Data Model** is a logical structure of Database. It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

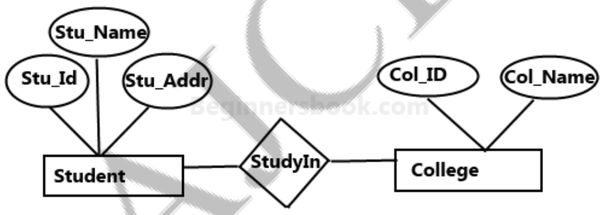
Types of Data Models:

Object based logical Models – Describe data at the conceptual and view levels.

1. E-R Model

An **entity–relationship model (ER model)** is a systematic way of describing and defining a business process. An ER model is typically implemented as a database. The main components of E-R model are: entity set and relationship set.

A sample E-R Diagram:



Sample E-R Diagram

2. **Object oriented Model**

An object data model is a data model based on object-oriented programming, associating methods (procedures) with objects that can benefit from class hierarchies. Thus, "objects" are levels of abstraction that include attributes and behavior

Record based logical Models – Like Object based model, they also describe data at the conceptual and view levels. These models specify logical structure of database with records, fields and attributes.

1. **Relational Model**

In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records.

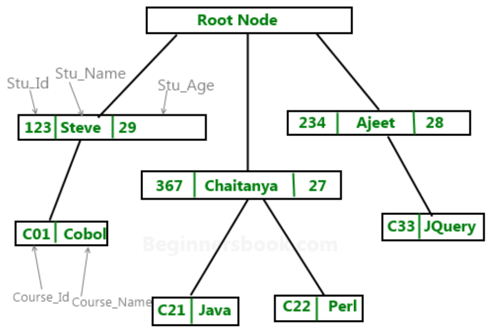
Sample relationship Model: Student table with 3 columns and three records.

Stu_Id	Stu_Name	Stu_Age
111	Ashish	23
123	Saurav	22
169	Lester	24

2. **Hierarchical Model**

In hierarchical model, data is organized into a tree like structure with each record is having one parent record and many children. The main drawback of this model is that, it can have only one to many relationships between nodes.

Sample Hierarchical Model Diagram:



3. **Network Model** – Network Model is same as hierarchical model except that it has graph-like structure rather than a tree-based structure. Unlike hierarchical model, this model allows each record to have more than one parent record.

Physical Data Models – These models describe data at the lowest level of abstraction.

Three Schema Architecture

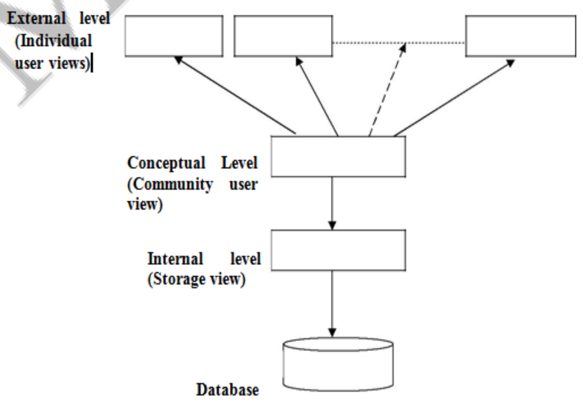
The goal of the three schema architecture is to separate the user applications and the physical database. The schemas can be defined at the following levels:

1. **The internal level** – has an internal schema which describes the physical storage structure of the database. Uses a physical data model and describes the complete details of data storage and access paths for the database.

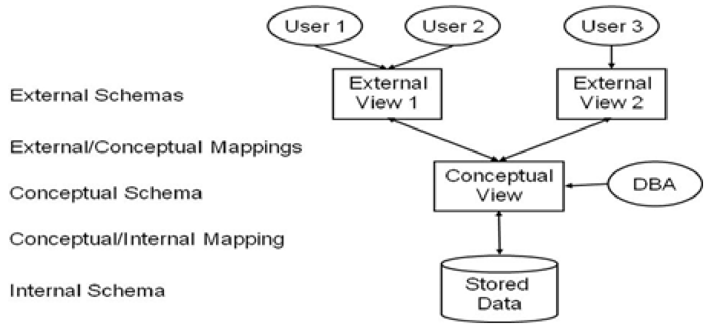
2. **The conceptual level** – has a conceptual schema which describes the structure of the database for users. It hides the details of the physical storage structures, and concentrates on describing entities, data types, relationships, user operations and constraints. Usually a representational data model is used to describe the conceptual schema.

3. **The External or View level** – includes external schemas or user vies. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. Represented using the representational data model.

The three schema architecture is used to visualize the schema levels in a database. The three schemas are only descriptions of data, the data only actually exists is at the physical level.


External level (Individual user views)
Conceptual Level (Community user view)
Internal level (Storage view)
Database

Database Architecture



COMPONENTS OF DBMS

Database Users

Users are differentiated by the way they expect to interact with the system

- Application programmers
- Sophisticated users
- Naïve users
- Database Administrator
- Specialized users etc,.

Application programmers:

Professionals who write application programs and using these application programs they interact with the database system

Sophisticated users :

These user interact with the database system without writing programs, But they submit queries to retrieve the information

Specialized users:

Who write specialized database applications to interact with the database system.

Naïve users:

Interacts with the database system by invoking some application programs that have been written previously by application programmers

Eg : people accessing database over the web

Database Administrator:

Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.

- ➢ Schema definition
- ➢ Access method definition
- ➢ Schema and physical organization modification
- ➢ Granting user authority to access the database
- ➢ Monitoring performance

Storage Manager

The Storage Manager include these following components/modules

- ➢ Authorization Manager
- ➢ Transaction Manager
- ➢ File Manager
- ➢ Buffer Manager

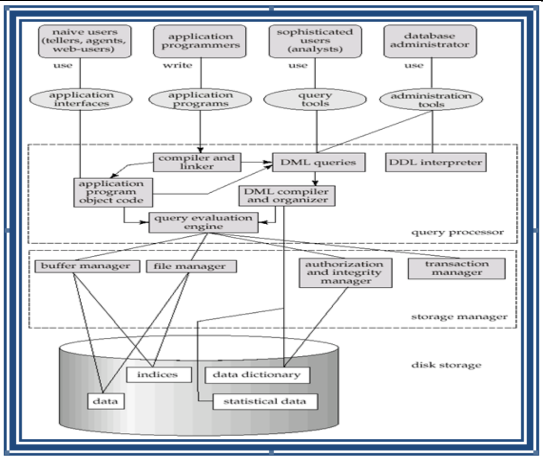
- ❖ Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- ❖ **The storage manager is responsible to the following tasks:**
  - ➢ interaction with the file manager
  - ➢ efficient storing, retrieving and updating of data

Authorization Manager

- ➢ Checks whether the user is an authorized person or not
- ➢ Test the satisfaction of integrity constraints

Transaction Manager

Responsible for concurrent transaction execution It ensures that the database remains in a consistent state despite of the system failure



EVOLUTION OF RDBMS

Before the acceptance of Codd's Relational Model, database management systems was just an ad hoc collection of data designed to solve a particular type of problem, later extended to solve more basic purposes. This led to complex systems, which were difficult to understand, install, maintain and use. These database systems were plagued with the following problems:

- They required large budgets and staffs of people with special skills that were in short supply.
- Database administrators' staff and application developers required prior preparation to access these database systems.
- End-user access to the data was rarely provided.
- These database systems did not support the implementation of business logic as a DBMS responsibility.

Hence, the objective of developing a relational model was to address each and every one of the shortcomings that plagued those systems that existed at the end of the 1960s decade, and make DBMS products more widely appealing to all kinds of users.

The existing relational database management systems offer powerful, yet simple solutions for a wide variety of commercial and scientific application problems. Almost every industry uses relational systems to store, update and retrieve data for operational, transaction, as well as decision support systems.

RELATIONAL DATABASE

A relational database is a database system in which the database is organized and accessed according to the relationships between data items without the need for any consideration of physical orientation and relationship. Relationships between data items are expressed by means of **tables**.

It is a tool, which can help you store, manage and disseminate information of various kinds. It is a collection of objects, tables, queries, forms, reports, and macros, all stored in a computer program all of which are inter-related.

It is a method of structuring data in the form of records, so that relations between different entities and attributes can be used for data access and transformation.

RELATIONAL DATABASE MANAGEMENT SYSTEM

A Relational Database Management System (RDBMS) is a system, which allows us to perceive data as tables (and nothing but tables), and ***operators*** necessary to manipulate that data are at the user's disposal.

Features of an RDBMS

The features of a relational database are as follows:

- ➢ The ability to create multiple relations (tables) and enter data into them
- ➢ An interactive query language
- ➢ Retrieval of information stored in more than one table
- ➢ Provides a *Catalog* or *Dictionary*, which itself consists of tables (called *system* tables)

Basic Relational Database Terminology

Catalog:

A catalog consists of all the information of the various schemas (external, conceptual and internal) and also all of the corresponding mappings (external/conceptual, conceptual/internal).

It contains detailed information regarding the various objects that are of interest to the system itself; e.g., tables, views, indexes, users, integrity rules, security rules, etc.

In a relational database, the entities of the ERD are represented as *tables* and their attributes as the *columns* of their respective tables in a database schema.

It includes some important terms, such as:

- *Table*: Tables are the basic storage structures of a database where data about something in the real world is stored. It is also called a *relation or an entity.*

- *Row:* Rows represent collection of data required for a particular entity. In order to identify each row as unique there should be a *unique identifier* called the *primary key*, which allows no duplicate rows. For example in a library every member is unique and hence is given a membership number, which uniquely identifies each member. A row is also called a *record* or a *tuple.*
- *Column:* Columns represent characteristics or attributes of an entity. Each attribute maps onto a column of a table. Hence, a column is also known as an *attribute*.
- *Relationship:* Relationships represent a logical link between two tables. A relationship is depicted by a *foreign key* column.
- Degree: number of attributes
- Cardinality: number of tuples
- An attribute of an entity has a particular value. The set of possible values That a given attribute can have is called its *domain*.

KEYS AND THEIR USE

Key: An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity set.

Super Key: If we add additional attributes to a key, the resulting combination would still uniquely identify an instance of the entity set. Such augmented keys are called super keys.

Primary Key: It is a minimum super key.

It is *a unique* ***identifier*** *for the table* (a column or a column combination with the property that at any given time no two rows of the table contain the same value in that column or column combination).

Foreign Key: A foreign key is a field (or collection of fields) in one table that uniquely identifies a row of another table. In simpler words, the foreign key is defined in a second table, but it refers to the primary key in the first table.

Candidate Key: There may be two or more attributes or combinations of attributes that uniquely identify an instance of an entity set. These attributes or combinations of attributes are called candidate keys.

Secondary Key: A secondary key is an attribute or combination of attributes that may not be a candidate key, but that classifies the entity set on a particular characteristic. Any key consisting of a single attribute is called a **simple key,** while that consisting of a combination of attributes is called a **composite key**.

Referential Integrity

Referential Integrity can be defined as an integrity constraint that specifies that the value (or existence) of an attribute in one relation depend on the value (or existence) of an attribute in the same or another relation. Referential integrity in a relational database is consistency between coupled tables. It is usually enforced by the combination of a primary key and a foreign key. For referential integrity to hold, any field in a table that is declared a foreign key can contain only values from a parent table's primary key field. For instance, deleting a record that contains a value referred to by a foreign key in another table would break referential integrity.

Relational Model

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts

Tables − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Tuple − A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema − A relation schema describes the relation name (table name), attributes, and their names.

Relation key − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain − Every attribute has some pre-defined value scope, known as attribute domain.

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints −

- Key constraints
- Domain constraints
- Referential integrity constraints

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key**for that relation. If there are more than one such minimal subsets, these are called ***candidate keys***.

Key constraints force that −

- in a relation with a key attribute, no two tuples can have identical values for key attributes.
- a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

Domain Constraints

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-

Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages − relational algebra and relational calculus.

Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

$\sigma_{subject = "database"}$(Books)

Output − Selects tuples from books where subject is 'database'.

$\sigma_{\text{subject = "database" and price = "450"}}$(Books)

Output − Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{\text{subject = "database" and price = "450" or year > "2010"}}$(Books)

Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1, A2, An}$ (r)

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

$\prod_{\text{subject, author}}$ (Books)

Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

It performs binary union between two given relations and is defined as −

r ∪ s = { t | t ∈ r or t ∈ s}

Notation − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.

- Duplicate tuples are automatically eliminated.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − r − s

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

$\sigma_{author = 'tutorialspoint'}$(Books X Articles)

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation − ρ_x (E)

Where the result of expression **E** is saved with name of **x**.

Additional operations are −

- Set intersection
- Assignment
- Natural join

SQL FUNDAMENTALS:

SQL is a standard computer language for accessing and manipulating databases.

What is SQL?

- SQL stands for **S**tructured **Q**uery **L**anguage
- SQL allows you to access a database
- SQL is an ANSI standard computer language
- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert new records in a database
- SQL can delete records from a database
- SQL can update records in a database
- SQL is easy to learn

SQL is a Standard - BUT....

SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. SQL statements are used to retrieve and update data in a database. SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.Unfortunately, there are many different versions of the SQL language, but to be in compliance with the ANSI standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

Note: Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

SQL Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.Below is an example of a table called "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).

SQL Queries
With SQL, we can query a database and have a result set returned.
A query like this:

```
SELECT LastName FROM Persons
```

Gives a result set like this:

LastName
Hansen
Svendson
Pettersen

Note: Some database systems require a semicolon at the end of the SQL statement. We don't use the semicolon in our tutorials.

SQL Data Manipulation Language (DML)
SQL (Structured Query Language) is a syntax for executing queries. But the SQL language also includes a syntax to update, insert, and delete records.
These query and update commands together form the Data Manipulation Language (DML) part of SQL:

- **SELECT** - extracts data from a database table
- **UPDATE** - updates data in a database table
- **DELETE** - deletes data from a database table
- **INSERT INTO** - inserts new data into a database table

SQL Data Definition Language (DDL)
The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. We can also define indexes (keys), specify links between tables, and impose constraints between database tables.
The most important DDL statements in SQL are:

- **CREATE TABLE** - creates a new database table
- **ALTER TABLE** - alters (changes) a database table
- **DROP TABLE** - deletes a database table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

The SQL SELECT Statement
The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).
Syntax

```
SELECT column_name(s)
FROM table_name
```

Note: SQL statements are not case sensitive. SELECT is the same as select.

SQL SELECT Example
To select the content of columns named "LastName" and "FirstName", from the database table called "Persons", use a SELECT statement like this:

```
SELECT LastName,FirstName FROM Persons
```

The database table "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The result

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Select All Columns
To select all columns from the "Persons" table, use a * symbol instead of column names, like this:

```
SELECT * FROM Persons
```

Result

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The Result Set

The result from a SQL query is stored in a result-set. Most database software systems allow navigation of the result set with programming functions, like: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc.
Programming functions like these are not a part of this tutorial. To learn about accessing data with function calls,

Semicolon after SQL Statements?

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.
Some SQL tutorials end each SQL statement with a semicolon. Is this necessary? We are using MS Access and SQL Server 2000 and we do not have to put a semicolon after each SQL statement, but some database programs force you to use it.

The SELECT DISTINCT Statement

The DISTINCT keyword is used to return only distinct (different) values.
The SELECT statement returns information from table columns. But what if we only want to select distinct elements?
With SQL, all we need to do is to add a DISTINCT keyword to the SELECT statement:

Syntax

```
SELECT DISTINCT column_name(s)
FROM table_name
```

Using the DISTINCT keyword

To select ALL values from the column named "Company" we use a SELECT statement like this:

```
SELECT Company FROM Orders
```

"Orders" table

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

Result

Company
Sega
W3Schools
Trio
W3Schools

Note that "W3Schools" is listed twice in the result-set.
To select only DIFFERENT values from the column named "Company" we use a SELECT DISTINCT statement like this:

```
SELECT DISTINCT Company FROM Orders
```

Result:

Company
Sega
W3Schools
Trio

Now "W3Schools" is listed only once in the result-set.

The WHERE clause is used to specify a selection criterion.

The WHERE Clause

To conditionally select data from a table, a WHERE clause can be added to the SELECT statement.

Syntax

```
SELECT column FROM table
WHERE column operator value
```

With the WHERE clause, the following operators can be used:

Operator	**Description**
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

Note: In some versions of SQL the <> operator may be written as !=

Using the WHERE Clause

To select only the persons living in the city "Sandnes", we add a WHERE clause to the SELECT statement:

```
SELECT * FROM Persons
WHERE City='Sandnes'
```

"Persons" table

LastName	**FirstName**	**Address**	**City**	**Year**
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

Result

LastName	**FirstName**	**Address**	**City**	**Year**
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980

Using Quotes

Note that we have used single quotes around the conditional values in the examples.

SQL uses single quotes around text values (most database systems will also accept double quotes). Numeric values should not be enclosed in quotes.

For text values:

```
This is correct:
SELECT * FROM Persons WHERE FirstName='Tove'
This is wrong:
SELECT * FROM Persons WHERE FirstName=Tove
```

For numeric values:

```
This is correct:
SELECT * FROM Persons WHERE Year>1965
This is wrong:
SELECT * FROM Persons WHERE Year>'1965'
```

The LIKE Condition

The LIKE condition is used to specify a search for a pattern in a column.

Syntax

```
SELECT column FROM table
WHERE column LIKE pattern
```

A "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

Using LIKE

The following SQL statement will return persons with first names that start with an 'O':

```
SELECT * FROM Persons
WHERE FirstName LIKE 'O%'
```

The following SQL statement will return persons with first names that end with an 'a':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%a'
```

The following SQL statement will return persons with first names that contain the pattern 'la':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%la%'
```

The INSERT INTO Statement

The INSERT INTO statement is used to insert new rows into a table.

Syntax

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

You can also specify the columns for which you want to insert data:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Insert a New Row

This "Persons" table:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

And this SQL statement:

```
INSERT INTO Persons
VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

Will give this result:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Insert Data in Specified Columns

This "Persons" table:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

And This SQL statement:

```
INSERT INTO Persons (LastName, Address)
VALUES ('Rasmussen', 'Storgt 67')
```

Will give this result:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

Null (no value …not space not empty)

The UPDATE statement is used to modify the data in a table.

Syntax

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Person:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

Update one Column in a Row

We want to add a first name to the person with a last name of "Rasmussen":

```
UPDATE Person SET FirstName = 'Nina'
WHERE LastName = 'Rasmussen'
```

Result:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Storgt 67	

Update several Columns in a Row

We want to change the address and add the name of the city:

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

Result:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

The DELETE Statement

The DELETE statement is used to delete rows in a table.

Syntax

```
DELETE FROM table_name
WHERE column_name = some_value
```

Person:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Delete
Drop

Delete a Row

"Nina Rasmussen" is going to be deleted:

```
DELETE FROM Person WHERE LastName = 'Rasmussen'
```

Result

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name
or
DELETE * FROM table_name
```

The ORDER BY keyword is used to sort the result.

Sort the Rows

The ORDER BY clause is used to sort the rows.

Orders:

Company	**OrderNumber**
Sega	3412
ABC Shop	5678
W3Schools	2312
W3Schools	6798

Example

To display the company names in alphabetical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company  ASC (asending)
```

Result:

Company	**OrderNumber**
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

Example

To display the company names in alphabetical order AND the OrderNumber in numerical order:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company, OrderNumber
```

Result:

Company	**OrderNumber**
ABC Shop	5678
Sega	3412
W3Schools	2312
W3Schools	6798

Aggregate functions

Aggregate functions operate against a collection of values, but return a single value.

Note: If used among many other expressions in the item list of a SELECT statement, the SELECT must have a GROUP BY clause!!

"Persons" table (used in most examples)

Name	**Age**
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Aggregate functions in MS Access

Function	**Description**
AVG(column)	Returns the average value of a column
COUNT(column)	Returns the number of rows (without a NULL value) of a column
COUNT(*)	Returns the number of selected rows
FIRST(column)	Returns the value of the first record in a specified field
LAST(column)	Returns the value of the last record in a specified field
MAX(column)	Returns the highest value of a column
MIN(column)	Returns the lowest value of a column
STDEV(column)	
STDEVP(column)	
SUM(column)	Returns the total sum of a column
VAR(column)	

VARP(column)	

Aggregate functions in SQL Server

Function	**Description**
AVG(column)	Returns the average value of a column
BINARY_CHECKSUM	
CHECKSUM	
CHECKSUM_AGG	
COUNT(column)	Returns the number of rows (without a NULL value) of a column
COUNT(*)	Returns the number of selected rows
COUNT(DISTINCT column)	Returns the number of distinct results
FIRST(column)	Returns the value of the first record in a specified field (not supported in SQLServer2K)
LAST(column)	Returns the value of the last record in a specified field (not supported in SQLServer2K)
MAX(column)	Returns the highest value of a column
MIN(column)	Returns the lowest value of a column
STDEV(column)	
STDEVP(column)	
SUM(column)	Returns the total sum of a column
VAR(column)	
VARP(column)	

Scalar functions

Scalar functions operate against a single value, and return a single value based on the input value.

Useful Scalar Functions in MS Access

Function	**Description**
UCASE(c)	Converts a field to upper case
LCASE(c)	Converts a field to lower case
MID(c,start[,end])	Extract characters from a text field
LEN(c)	Returns the length of a text field
INSTR(c,char)	Returns the numeric position of a named character within a text field
LEFT(c,number_of_char)	Return the left part of a text field requested
RIGHT(c,number_of_char)	Return the right part of a text field requested
ROUND(c,decimals)	Rounds a numeric field to the number of decimals specified
MOD(x,y)	Returns the remainder of a division operation

Aggregate functions (like SUM) often need an added GROUP BY functionality.

GROUP BY

GROUP BY... was added to SQL because aggregate functions (like SUM) return the aggregate of all column values every time they are called, and without the GROUP BY function it was impossible to find the sum for each individual group of column values.

The syntax for the GROUP BY function is:

```
SELECT column,SUM(column) FROM table GROUP BY column
```

GROUP BY Example

This "Sales" Table:

Company	**Amount**
W3Schools	5500
IBM	4500
W3Schools	7100

And This SQL:

```
SELECT Company, SUM(Amount) FROM Sales
```

Returns this result:

Company	SUM(Amount)
W3Schools	17100
IBM	17100
W3Schools	17100

The above code is invalid because the column returned is not part of an aggregate. A GROUP BY clause will solve this problem:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
```

Returns this result:

Company	SUM(Amount)
W3Schools	12600
IBM	4500

HAVING…

HAVING... was added to SQL because the WHERE keyword could not be used against aggregate functions (like SUM), and without HAVING... it would be impossible to test for result conditions.

The syntax for the HAVING function is:

```
SELECT column,SUM(column) FROM table
GROUP BY column
HAVING SUM(column) condition value
```

This "Sales" Table:

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

This SQL:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
HAVING SUM(Amount)>10000
```

Returns this result

Company	SUM(Amount)
W3Schools	12600

EMBEDDED SQL

Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language, which is known as a host language. Because the host language cannot parse SQL, the inserted SQL is parsed by an embedded SQL preprocessor.

Embedded SQL is a robust and convenient method of combining the computing power of a programming language with SQL's specialized data management and manipulation capabilities.

Structure of embedded SQL

Structure of embedded SQL defines step by step process of establishing a connection with DB and executing the code in the DB within the high level language.

Connection to DB

This is the first step while writing a query in high level languages. First connection to the DB that we are accessing needs to be established. This can be done using the keyword CONNECT. But it has to precede with 'EXEC SQL' to indicate that it is a SQL statement.

EXEC SQL CONNECT db_name;

EXEC SQL CONNECT HR_USER; //connects to DB HR_USER

Once connection is established with DB, we can perform DB transactions. Since these DB transactions are dependent on the values and variables of the host language. Depending on their values, query will be written and executed. Similarly, results of DB query will be returned to the host language which will be captured by the variables of host language. Hence we need to declare the variables to pass the value to the query and get the values from query. There are two types of variables used in the host language.

- **Host variable :** These are the variables of host language used to pass the value to the query as well as to capture the values returned by the query. Since SQL is dependent on host language we have to use variables of host language and such variables are known as host variable. But these host variables should be declared within the SQL area or within SQL code. That means compiler should be able to differentiate it from normal C variables. Hence we have to declare host variables within BEGIN DECLARE and END DECLARE section. Again, these declare block should be enclosed within EXEC SQL and ';'.

```
EXEC SQL BEGIN DECLARE SECTION;
        int STD_ID;
        char STD_NAME [15];
        char ADDRESS[20];
EXEC SQL END DECLARE SECTION;
```

We can note here that variables are written inside begin and end block of the SQL, but they are declared using C code. It does not use SQL code to declare the variables. Why? This is because they are host variables – variables of C language. Hence we cannot use SQL syntax to declare them. Host language supports almost all the datatypes from int, char, long, float, double, pointer, array, string, structures etc.

When host variables are used in a SQL query, it should be preceded by colon – ':' to indicate that it is a host variable. Hence when pre-compiler compiles SQL code, it substitutes the value of host variable and compiles.

EXEC SQL SELECT * FROM STUDENT WHERE STUDENT_ID =:STD_ID;

The following code is a simple embedded SQL program, written in C. The program illustrates many, but not all, of the embedded SQL techniques. The program prompts the user for an order number, retrieves the customer number, salesperson, and status of the order, and displays the retrieved information on the screen.

```
int main() {
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    int OrderID;         /* Employee ID (from user)        */
    int CustID;          /* Retrieved customer ID          */
    char SalesPerson[10] /* Retrieved salesperson name     */
    char Status[6]       /* Retrieved order status         */
  EXEC SQL END DECLARE SECTION;

  /* Set up error processing */
  EXEC SQL WHENEVER SQLERROR GOTO query_error;
  EXEC SQL WHENEVER NOT FOUND GOTO bad_number;

  /* Prompt the user for order number */
  printf ("Enter order number: ");
  scanf_s("%d", &OrderID);

  /* Execute the SQL query */
  EXEC SQL SELECT CustID, SalesPerson, Status
    FROM Orders
    WHERE OrderID = :OrderID
    INTO :CustID, :SalesPerson, :Status;
```

```
        /* Display the results */
        printf ("Customer number:  %d\n", CustID);
        printf ("Salesperson: %s\n", SalesPerson);
        printf ("Status: %s\n", Status);
        exit();

      query_error:
        printf ("SQL error: %ld\n", sqlca->sqlcode);
        exit();

      bad_number:
        printf ("Invalid order number.\n");
        exit();
      }
```

DYNAMIC SQL

The main disadvantage of embedded SQL is that it supports only static SQLs. If we need to build up queries at run time, then we can use dynamic sql. That means if query changes according to user input, then it always better to use dynamic SQL. Like we said above, the query when user enters student name alone and when user enters both student name and address, is different. If we use embedded SQL, one cannot implement this requirement in the code. In such case dynamic SQL helps the user to develop query depending on the values entered by him, without making him know which query is being executed. It can also be used when we do not know which SQL statements like Insert, Delete update or select needs to be used, when number of host variables is unknown, or when datatypes of host variables are unknown or when there is direct reference to DB objects like tables, views, indexes are required.
However this will make user requirement simple and easy but it may make query lengthier and complex. That means depending upon user inputs, the query may grow or shrink making the code flexible enough to handle all the possibilities. In embedded SQL, compiler knows the query in advance and pre-compiler compiles the SQL code much before C compiles the code for execution. Hence embedded SQLs will be faster in execution. But in the case of dynamic SQL, queries are created, compiled and executed only at the run time. This makes the dynamic SQL little complex, and time consuming.
Since query needs to be prepared at run time, in addition to the structures discussed in embedded SQL, we have three more clauses in dynamic SQL. These are mainly used to build the query and execute them at run time.

PREPARE

Since dynamic SQL builds a query at run time, as a first step we need to capture all the inputs from the user. It will be stored in a string variable. Depending on the inputs received from the user, string variable is appended with inputs and SQL keywords. These SQL like string statements are then converted into SQL query. This is done by using PREPARE statement.
For example, below is the small snippet from dynamic SQL. Here sql_stmt is a character variable, which holds inputs from the users along with SQL commands. But is cannot be considered as SQL query as it is still a sting value. It needs to be converted into a proper SQL query which is done at the last line using PREPARE statement. Here sql_query is also a string variable, but it holds the string as a SQL query.

EXECUTE

This statement is used to compile and execute the SQL statements prepared in DB.
EXEC SQL EXECUTE sql_query;

EXECUTE IMMEDIATE

This statement is used to prepare SQL statement as well as execute the SQL statements in DB. It performs the task of PREPARE and EXECUTE in a single line.
EXEC SQL EXECUTE IMMEDIATE :sql_stmt;
Dynamic SQL will not have any SELECT queries and host variables. But it can be any other SQL statements like insert, delete, update, grant etc. But when we use insert/ delete/ updates in this type, we cannot use host variables. All the input values will be hardcoded. Hence the SQL statements can be directly executed using EXECUTE IMMEDIATE rather than using PREPARE and then EXECUTE.

EXEC SQL EXECUTE IMMEDIATE 'GRANT SELECT ON STUDENT TO Faculty';
EXEC SQL EXECUTE IMMEDIATE 'DELETE FROM STUDENT WHERE STD_ID = 100';
EXEC SQL EXECUTE IMMEDIATE 'UPDATE STUDENT SET ADDRESS = 'Troy' WHERE STD_ID =100';

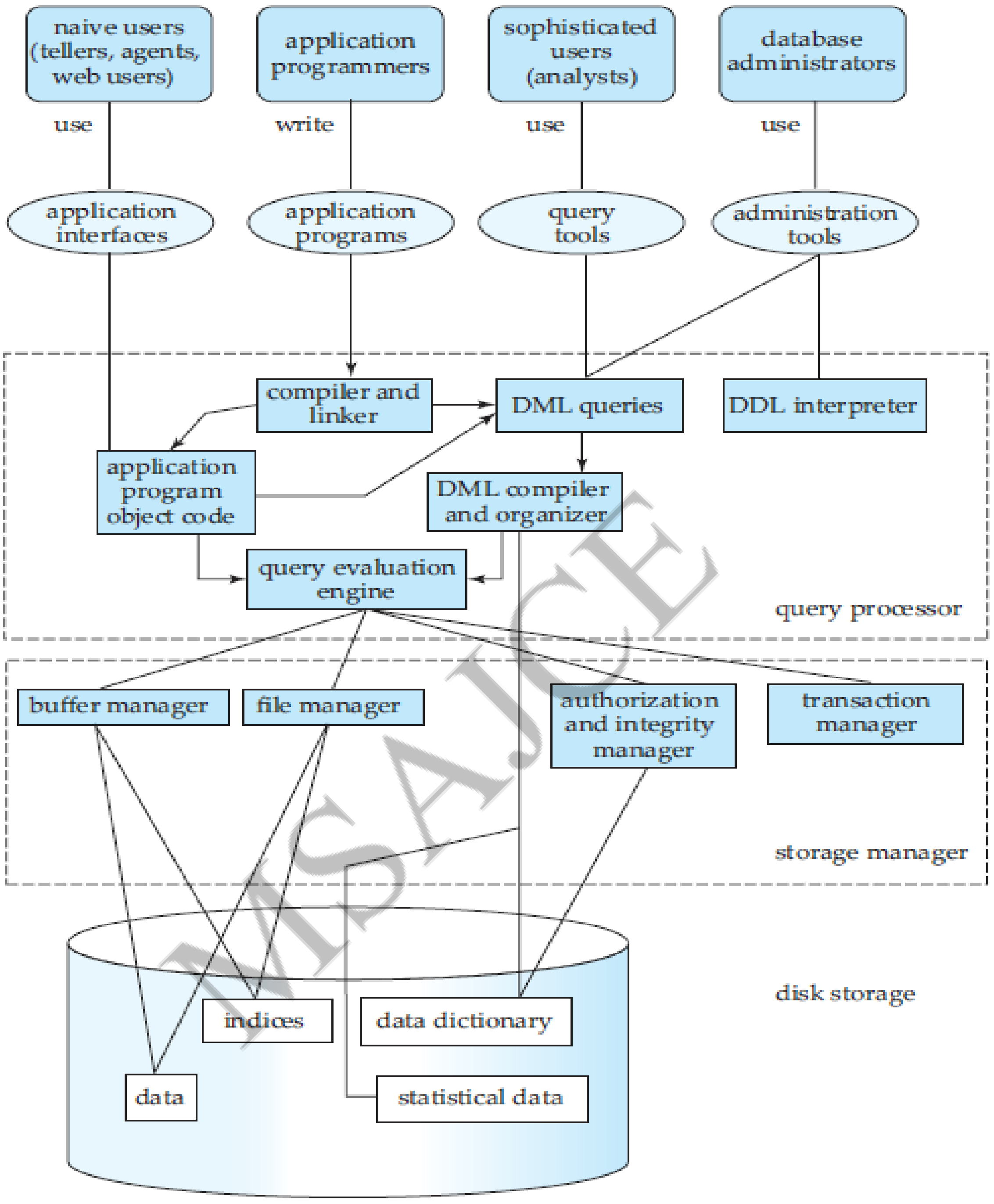


Figure 1.5 System structure.

UNIT II DATABASE DESIGN

Entity-Relationship model – E-R Diagrams – Enhanced-ER Model – ER-to-Relational Mapping – Functional Dependencies – Non-loss Decomposition – First, Second, Third Normal Forms, Dependency Preservation – Boyce/Codd Normal Form – Multi-valued Dependencies and Fourth Normal Form – Join Dependencies and Fifth Normal Form

DATABASE DESIGN

A well-designed database shall:

- Eliminate Data Redundancy: the same piece of data shall not be stored in more than one place. This is because duplicate data not only waste storage spaces but also easily lead to inconsistencies.
- Ensure Data Integrity and Accuracy

Entity-Relationship Data Model

- Classical, popular conceptual data model
- First introduced (mid 70's) as a (relatively minor) improvement to the relational model: pictorial diagrams are easier to read than relational database schemas
- Then evolved as a popular model for the first conceptual representation of data structures in the process of database design

ER Model: Entity and Entity Set

Considering the above example, **Student** is an entity, **Teacher** is an entity, similarly, **Class**, **Subject**etc are also entities.

An Entity is generally a real-world object which has characteristics and holds relationships in a DBMS.

If a Student is an Entity, then the complete dataset of all the students will be the **Entity Set**

ER Model: Attributes

If a Student is an Entity, then student's roll no., student's name, student's age, student's gender etc will be its attributes.

An attribute can be of many types, here are different types of attributes defined in ER database model:

1. **Simple attribute**: The attributes with values that are atomic and cannot be broken down further are simple attributes. For example, student's age.
2. **Composite attribute**: A composite attribute is made up of more than one simple attribute. For example, student's address will contain, house no., street name, pincode etc.
3. **Derived attribute**: These are the attributes which are not present in the whole database management system, but are derived using other attributes. For example, average age of students in a class.
4. **Single-valued attribute**: As the name suggests, they have a single value.
5. **Multi-valued attribute**: And, they can have multiple values.

ER Model: Relationships

When an Entity is related to another Entity, they are said to have a relationship. For example, A ClassEntity is related to Student entity, because students study in classes, hence this is a relationship.

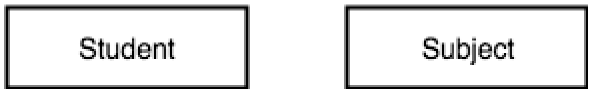
Depending upon the number of entities involved, a degree is assigned to relationships.

For example, if 2 entities are involved, it is said to be Binary relationship, if 3 entities are involved, it is said to be Ternary relationship, and so on.

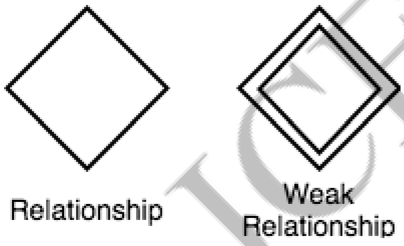
Working with ER Diagrams
ER Diagram is a visual representation of data that describes how data is related to each other. In ER Model, we disintegrate data into entities, attributes and setup relationships between entities, all this can be represented visually using the ER diagram.

Components of ER Diagram
Entitiy, Attributes, Relationships etc form the components of ER Diagram and there are defined symbols and shapes to represent each one of them.
Let's see how we can represent these in our ER Diagram.

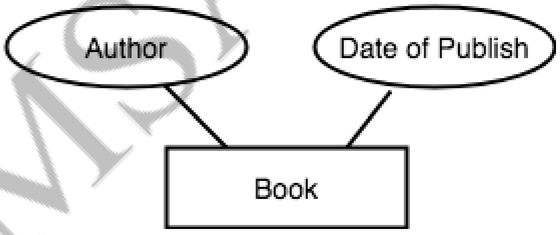
Entity
Simple rectangular box represents an Entity.



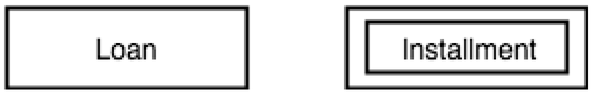
Relationships between Entities - Weak and Strong
Rhombus is used to setup relationships between two or more entities.



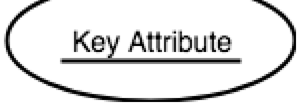
Attributes for any Entity
Ellipse is used to represent attributes of any entity. It is connected to the entity.



Weak Entity
A weak Entity is represented using double rectangular boxes. It is generally connected to another entity.

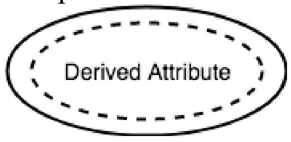


Key Attribute for any Entity
To represent a Key attribute, the attribute name inside the Ellipse is underlined.

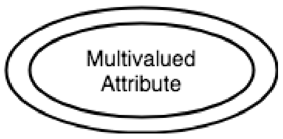


Derived Attribute for any Entity
Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth.

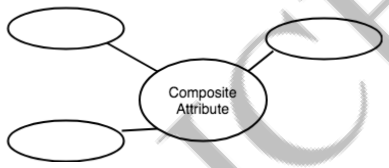
To represent a derived attribute, another dotted ellipse is created inside the main ellipse.



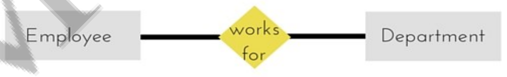
Multivalued Attribute for any Entity
Double Ellipse, one inside another, represents the attribute which can have multiple values.



Composite Attribute for any Entity
A composite attribute is the attribute, which also has attributes.

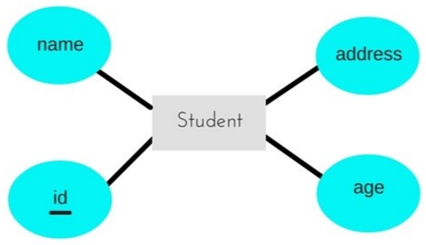


ER Diagram: Entity
An Entity can be any object, place, person or class. In ER Diagram, an entity is represented using rectangles. Consider an example of an Organisation- Employee, Manager, Department, Product and many more can be taken as entities in an Organisation.



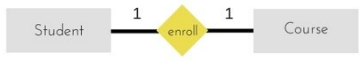
The yellow rhombus in between represents a relationship.

ER Diagram: Key Attribute
Key attribute represents the main characteristic of an Entity. It is used to represent a Primary key. Ellipse with the text underlined, represents Key Attribute.



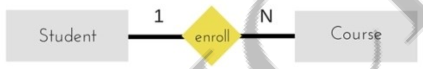
ER Diagram: Binary Relationship
Binary Relationship means relation between two Entities. This is further divided into three types.

One to One Relationship
This type of relationship is rarely seen in real world.

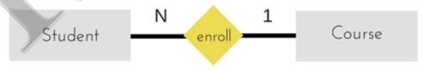


The above example describes that one student can enroll only for one course and a course will also have only one Student. This is not what you will usually see in real-world relationships.

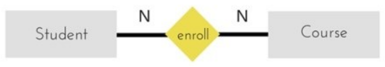
One to Many Relationship
The below example showcases this relationship, which means that 1 student can opt for many courses, but a course can only have 1 student. Sounds weird! This is how it is.



Many to One Relationship
It reflects business rule that many entities can be associated with just one entity. For example, Student enrolls for only one Course but a Course can have many Students.

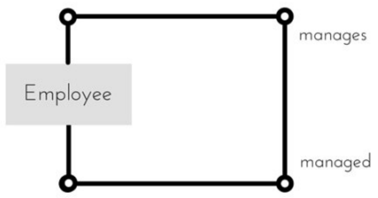


Many to Many Relationship



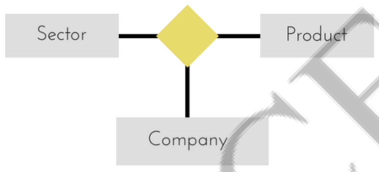
The above diagram represents that one student can enroll for more than one courses. And a course can have more than 1 student enrolled in it.

ER Diagram: Recursive Relationship
When an Entity is related with itself it is known as Recursive Relationship.



ER Diagram: Ternary Relationship
Relationship of degree three is called Ternary relationship.

A Ternary relationship involves three entities. In such relationships we always consider two entites together and then look upon the third.



- The above relationship involves 3 entities.
- Company operates in Sector, producing some Products.

For example, in the diagram above, we have three related entities, Company, Product and Sector. To understand the relationship better or to define rules around the model, we should relate two entities and then derive the third one.
A Company produces many Products/ each product is produced by exactly one company.
A Company operates in only one Sector / each sector has many companies operating in it.
Considering the above two rules or relationships, we see that although the complete relationship involves three entities, but we are looking at two entities at a time.

The Enhanced ER Model

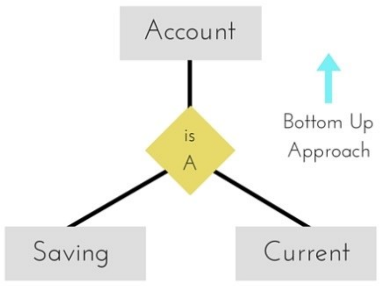
As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modelling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.
Hence, as part of the Enhanced ER Model, along with other improvements, three new concepts were added to the existing ER Model, they were:

1. Generalization
2. Specialization
3. Aggregration

Generalization

Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.

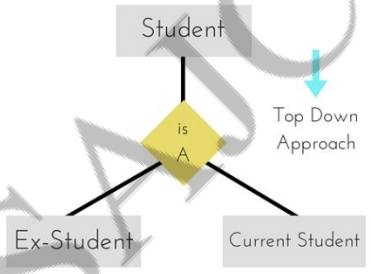
It's more like Superclass and Subclass system, but the only difference is the approach, which is bottom-up. Hence, entities are combined to form a more generalised entity, in other words, sub-classes are combined to form a superclass.



For example, Saving and Current account types entities can be generalised and an entity with name Account can be created, which covers both.

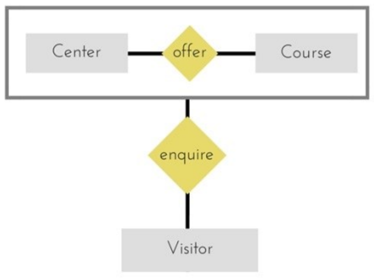
Specialization

Specialization is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, a higher level entity may not have any lower-level entity sets, it's possible.



Aggregation

Aggregation is a process when relation between two entities is treated as a single entity.



In the diagram above, the relationship between Center and Course together, is acting as an Entity, which is in relationship with another entity Visitor. Now in real world, if a Visitor or a Student visits a Coaching Center, he/she will never enquire about the center only or just about the course, rather he/she will ask enquire about both.

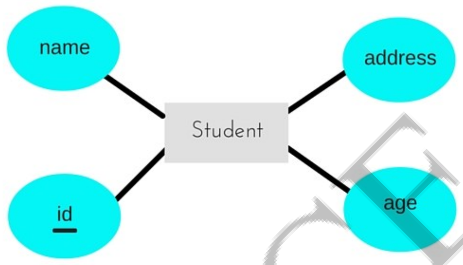
ER Model to Relational Model

ER Model can be represented using ER Diagrams which is a great way of designing and representing the database design in more of a flow chart form.

It is very convenient to design the database using the ER Model by creating an ER diagram and later on converting it into relational model to design your tables.
Not all the ER Model constraints and components can be directly transformed into relational model, but an approximate schema can be derived.
Few examples of ER diagrams and convert it into relational model schema, hence creating tables in RDBMS.

Entity becomes Table

Entity in ER Model is changed into tables, or we can say for every Entity in ER model, a table is created in Relational Model.
And the attributes of the Entity gets converted to columns of the table.
And the primary key specified for the entity in the ER model, will become the primary key for the table in relational model.
For example, for the below ER Diagram in ER Model,



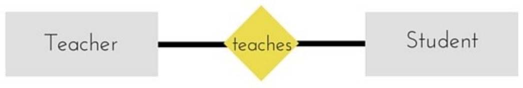
A table with name Student will be created in relational model, which will have 4 columns, id, name, age, address and id will be the primary key for this table.
Table:Student

id	name	age	address

Relationship becomes a Relationship Table

In ER diagram, we use diamond/rhombus to represent a relationship between two entities. In Relational model we create a relationship table for ER Model relationships too.

In the ER diagram below, we have two entities Teacher and Student with a relationship between them.



As discussd above, entity gets mapped to table, hence we will create table for Teacher and a table for Student with all the attributes converted into columns.

Now, an additional table will be created for the relationship, for example StudentTeacher or give it any name you like. This table will hold the primary key for both Student and Teacher, in a tuple to describe the relationship, which teacher teaches which student.

If there are additional attributes related to this relationship, then they become the columns for this table, like subject

name.
Also proper foreign key constraints must be set for all the tables.

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

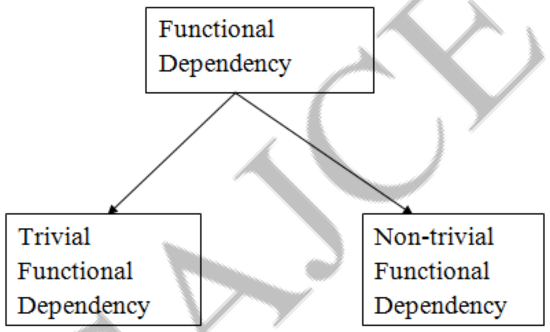
Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

Emp_Id → Emp_Name

Types of Functional dependency


Functional Dependency
Trivial Functional Dependency
Non-trivial Functional Dependency

Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

Consider a table with two columns Employee_Id and Employee_Name.

{Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as Employee_Id is a subset of {Employee_Id, Employee_Name}.

3. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

Non-trivial functional dependency

A → B has a non-trivial functional dependency if B is not a subset of A.

When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

ID → Name,

Name → DOB

Normalization of Database

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic

approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.
Normalization is used for mainly two purposes,

- Eliminating reduntant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Problems Without Normalization

If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized. To understand these anomalies let us take an example of a Student table.

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields branch, hod(Head of Department) and office_tel is repeated for the students who are in the same branch in the college, this is Data Redundancy.

Insertion Anomaly

Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as NULL.
Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.
These scenarios are nothing but Insertion anomalies.

Updation Anomaly

What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updation anomaly.

Deletion Anomaly

In our Student table, two different informations are kept together, Student information and Branch information.
Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information.
This is Deletion anomaly.

Normalization Rule

Normalization rules are divided into the following normal forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Fourth Normal Form
6. Fifth Normal Form

First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

Rules for First Normal Form

The first normal form expects you to follow a few simple rules while designing your database, and they are:

Rule 1: Single Valued Attributes

Each column of your table should be single valued which means they should not contain multiple values. We will explain this with help of an example later, let's see the other rules for now.

Rule 2: Attribute Domain should not change

This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.
For example: If you have a column dob to save date of births of a set of people, then you cannot or you must not save 'names' of some of them in that column along with 'date of birth' of others in that column. It should hold only 'date of birth' for all the records/rows.

Rule 3: Unique name for Attributes/Columns

This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data.
If one or more columns have same name, then the DBMS system will be left confused.

Rule 4: Order doesn't matters

This rule says that the order in which you store the data in your table doesn't matter.

EXAMPLE

Create a table to store student data which will have student's roll no., their name and the name of subjects they have opted for.
Here is the table, with some sample data added to it.

roll_no	name	subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

The table already satisfies 3 rules out of the 4 rules, as all our column names are unique, we have stored data in the order we wanted to and we have not inter-mixed different type of data in columns.

But out of the 3 different students in our table, 2 have opted for more than 1 subject. And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value.
It's very simple, because all we have to do is break the values into atomic values.
Here is our updated table and it now satisfies the First Normal Form.

roll_no	name	subject
101	Akon	OS
101	Akon	CN
103	Ckon	Java
102	Bkon	C
102	Bkon	C++

By doing so, although a few values are getting repeated but values for the subject column are now atomic for each record/row. Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Second Normal Form (2NF)

For a table to be in the Second Normal Form,
1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

Dependency

Let's take an example of a Student table with columns student_id, name, reg_no(registration number), branch and address(student's home address).

student_ id	name	reg_n o	branc h	addre ss

In this table, student_id is the primary key and will be unique for every row, hence we can use student_id to fetch any row of data from this table
Even for a case, where student names are same, if we know the student_id we can easily fetch the correct record.

student_id	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat

Hence we can say a Primary Key for a table is the column or a group of columns(composite key) which can uniquely identify each record in the table.

I can ask from branch name of student with student_id 10, and I can get it. Similarly, if I ask for name of student with student_id 10 or 11, I will get it. So all I need is student_id and every other column depends on it, or can be fetched using it.This is Dependency and we also call it Functional Dependency.

Partial Dependency

Now that we know what dependency is, we are in a better state to understand what partial dependency is.
For a simple table like Student, a single column like student_id can uniquely identfy all the records in a table. But this is not true all the time. So now let's extend our example to see if more than 1 column together can act

as a primary key.
Let's create another table for Subject, which will have subject_id and subject_name fields and subject_id will be the primary key.

subject_id	subject_name
1	Java
2	C++
3	Php

Now we have a Student table with student information and another table Subject for storing subject information.
Let's create another table Score, to store the marks obtained by students in the respective subjects. We will also be saving name of the teacher who teaches that subject along with marks.

score_id	student_id	subject_id	marks	teacher
1	10	1	70	Java Teacher
2	10	2	75	C++ Teacher
3	11	1	80	Java Teacher

In the score table we are saving the **student_id** to know which student's marks are these and **subject_id** to know for which subject the marks are for.

Together, student_id + subject_id forms a **Candidate Key** which can be the **Primary key**.

To get me marks of student with student_id 10, can you get it from this table? No, because you don't know for which subject. And if I give you subject_id, you would not know for which student. Hence we need student_id + subject_id to uniquely identify any row.

But where is Partial Dependency?

Now if you look at the Score table, we have a column names teacher which is only dependent on the subject, for Java it's Java Teacher and for C++ it's C++ Teacher & so on.

Now as we just discussed that the primary key for this table is a composition of two columns which is student_id & subject_id but the teacher's name only depends on subject, hence the subject_id, and has nothing to do with student_id.

This is Partial Dependency, where an attribute in a table depends on only a part of the primary key and not on the whole key.

How to remove Partial Dependency?

There can be many different solutions for this, but out objective is to remove teacher's name from Score table.
The simplest solution is to remove columns teacher from Score table and add it to the Subject table. Hence, the Subject table will become:

subject_id	subject_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

And our Score table is now in the second normal form, with no partial dependency.

score_id	student_id	subject_id	marks
1	10	1	70
2	10	2	75
3	11		

Third Normal Form (3NF)

A table is said to be in the Third Normal Form when,

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.

So let's use the same example, where we have 3 tables, **Student**, **Subject** and **Score**.

Student Table

student_id	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat
12	Bkon	09-WY	IT	Rajasthan

Subject Table

subject_id	subject_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

Score Table

In the Score table, we need to store some more information, which is the exam name and total marks, so let's add 2 more columns to the Score table.

score_id	student_id	subject_id	marks
1	10	1	70
2	10	2	75
3	11	1	80

Transitive Dependency

With exam_name and total_marks added to our Score table, it saves more data now. Primary key for the Score table is a composite key, which means it's made up of two attributes or columns → student_id + subject_id.
The new column exam_name depends on both student and subject. For example, a mechanical engineering student will have Workshop exam but a computer science student won't. And for some subjects you have Practical exams and for some you don't. So we can say that exam_name is dependent on both student_id and subject_id.
And what about our second new column total_marks? Does it depend on our Score table's primary key?

Well, the column total_marks depends on exam_name as with exam type the total score changes. For example, practicals are of less marks while theory exams are of more marks.
But, exam_name is just another column in the score table. It is not a primary key or even a part of the primary key, and total_marks depends on it.
This is Transitive Dependency. When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.

How to remove Transitive Dependency

Again the solution is very simple. Take out the columns exam_name and total_marks from Score table and put them in an Exam table and use the exam_id wherever required.

Score Table: In 3rd Normal Form

score_id	student_id	subject_id	marks	exam_id

The new Exam table

exam_id	exam_name	total_marks
1	Workshop	200
2	Mains	70
3	Practicals	30

Advantage of removing Transitive Dependency

The advantage of removing transitive dependency is,

- Amount of data duplication is reduced.
- Data integrity achieved.

Boyce and Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency (X → Y), X should be a super Key.In simple words, it means, that for a dependency A → B, A cannot be a non-prime attribute, if B is a prime attribute.

Example

College enrolment table with columns student_id, subject and professor.

student_id	subject	professor
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Chash
104	Java	P.Java

In the table above:
One student can enroll for multiple subjects. For example, student with student_id 101, has opted for subjects - Java & C++

- For each subject, a professor is assigned to the student.
- And, there can be multiple professors teaching one subject like Java.

What do you think should be the Primary Key?

Well, in the table above student_id, subject together form the primary key, because
using student_id and subject, we can find all the columns of the table.
One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.
Hence, there is a dependency between subject and professor here, where subject depends on the professor name.
This table satisfies the 1st Normal form because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.
This table also satisfies the 2nd Normal Form as there is no Partial Dependency.
And, there is no Transitive Dependency, hence the table also satisfies the 3rd Normal Form.
But this table is not in Boyce-Codd Normal Form.
Why this table is not in BCNF?
In the table above, student_id, subject form primary key, which means subject column is a prime attribute.
But, there is one more dependency, professor → subject.
And while subject is a prime attribute, professor is a non-prime attribute, which is not allowed by BCNF.
How to satisfy BCNF?
To make this relation(table) satisfy BCNF, we will decompose this table into two tables, student table and professor table.
Below we have the structure for both the tables.
Student Table

student_id	p_id
101	1
101	2

Professor Table

p_id	professor	subject
1	P.Java	Java
2	P.Cpp	C++

And now, this relation satisfy Boyce-Codd Normal Form.

Fourth Normal Form (4NF)

A table is said to be in the Fourth Normal Form when,

1. It is in the Boyce-Codd Normal Form.
2. And, it doesn't have Multi-Valued Dependency.

Multi-valued Dependency

A table is said to have multi-valued dependency, if the following conditions are true,

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

Example

Below we have a college enrolment table with columns s_id, course and hobby.

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

From the table above, student with s_id 1 has opted for two courses, Science and Maths, and has two hobbies, Cricket and Hockey.

You must be thinking what problem this can lead to, right?

Well the two records for student with s_id 1, will give rise to two more records, as shown below, because for one student, two hobbies exists, hence along with both the courses, these hobbies should be specified.

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
1	Science	Hockey
1	Maths	Cricket

And, in the table above, there is no relationship between the columns course and hobby. They are independent of each other.

So there is multi-value dependency, which leads to un-necessary repetition of data and other anomalies as well.

How to satisfy 4th Normal Form?

To make the above relation satify the 4th normal form, we can decompose the table into 2 tables.

CourseOpted Table

s_id	course
1	Science
1	Maths
2	C#
2	Php

Hobbies Table,

s_id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

Now this relation satisfies the fourth normal form.

A table can also have functional dependency along with multi-valued dependency. In that case, the functionally dependent columns are moved in a separate table and the multi-valued dependent columns are moved to separate tables.

Fifth Normal Form (5NF)

A database is said to be in 5NF, if and only if,

1. It's in 4NF
2. If we can decompose table further to eliminate redundancy and anomaly, and when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data or any new record set should not arise. In simple words, joining two or more decomposed table should not lose records nor create new records.

What is Join Dependency

If a table can be recreated by joining multiple tables and each of this table have a subset of the attributes of the table, then the table is in Join Dependency. It is a generalization of Multivalued DependencyJoin Dependency can be related to 5NF, wherein a relation is in 5NF, only if it is already in 4NF and it cannot be decomposed further.

Example

<Employee>

EmpName	**EmpSkills**	**EmpJob** (Assigned Work)
Tom	Networking	EJ001
Harry	Web Development	EJ002
Katie	Programming	EJ002

The above table can be decomposed into the following three tables; therefore it is not in 5NF:

<EmployeeSkills>

EmpName	**EmpSkills**
Tom	Networking
Harry	Web Development
Katie	Programming

<EmployeeJob>

EmpName	**EmpJob**
Tom	EJ001
Harry	EJ002
Katie	EJ002

<JobSkills>

EmpSkills	**EmpJob**
Networking	EJ001
Web Development	EJ002
Programming	EJ002

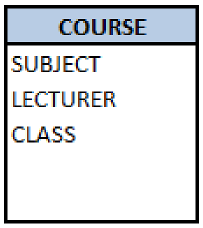
Our Join Dependency:

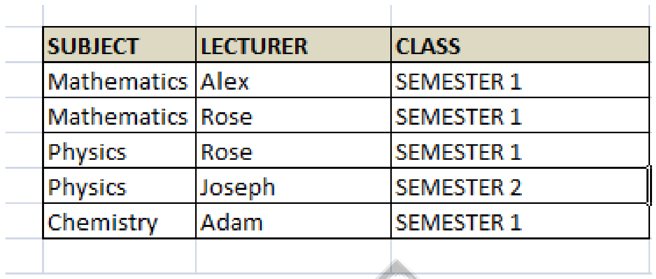
{(EmpName, EmpSkills), (EmpName, EmpJob), (EmpSkills, EmpJob)}

The above relations have join dependency, so they are not in 5NF. That would mean that a join relation of the above three relations is equal to our original relation **<Employee>**.

FIFTH NORMAL FORM EXAMPLE

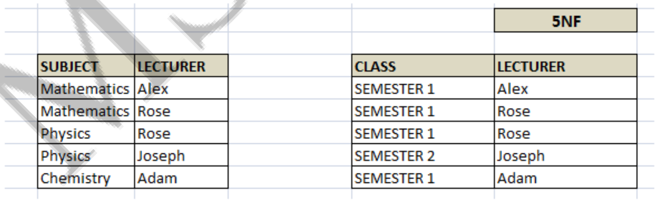
Consider an example of different Subjects taught by different lecturers and the lecturers taking classes for different semesters.
Note: Please consider that Semester 1 has Mathematics, Physics and Chemistry and Semester 2 has only Mathematics in its academic year!!





SUBJECT	LECTURER	CLASS
Mathematics	Alex	SEMESTER 1
Mathematics	Rose	SEMESTER 1
Physics	Rose	SEMESTER 1
Physics	Joseph	SEMESTER 2
Chemistry	Adam	SEMESTER 1

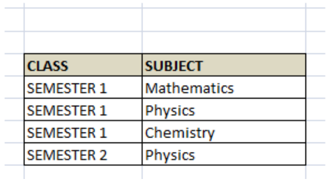
In above table, Rose takes both Mathematics and Physics class for Semester 1, but she does not take Physics class for Semester 2. In this case, combination of all these 3 fields is required to identify a valid data. Imagine we want to add a new class - Semester3 but do not know which Subject and who will be taking that subject. We would be simply inserting a new entry with Class as Semester3 and leaving Lecturer and subject as NULL. As we discussed above, it's not a good to have such entries. Moreover, all the three columns together act as a primary key, we cannot leave other two columns blank!
Hence we have to decompose the table in such a way that it satisfies all the rules till 4NF and when join them by using keys, it should yield correct record. Here, we can represent each lecturer's Subject area and their classes in a better way. We can divide above table into three - (SUBJECT, LECTURER), (LECTURER, CLASS), (SUBJECT, CLASS)



5NF

SUBJECT	LECTURER
Mathematics	Alex
Mathematics	Rose
Physics	Rose
Physics	Joseph
Chemistry	Adam

CLASS	LECTURER
SEMESTER 1	Alex
SEMESTER 1	Rose
SEMESTER 1	Rose
SEMESTER 2	Joseph
SEMESTER 1	Adam



CLASS	SUBJECT
SEMESTER 1	Mathematics
SEMESTER 1	Physics
SEMESTER 1	Chemistry
SEMESTER 2	Physics

Now, each of combinations is in three different tables. If we need to identify who is teaching which subject to which semester, we need join the keys of each table and get the result.
For example, who teaches Physics to Semester 1, we would be selecting Physics and Semester1 from table 3 above, join with table1 using Subject to filter out the lecturer names. Then join with table2 using Lecturer to get correct lecturer name. That is we joined key columns of each table to get the correct data. Hence there is no lose or new data - satisfying 5NF condition.

UNIT III

3 Transactions

Syllabus

Transaction Concepts - ACID Properties - Schedules - Serializability - Transaction support in SQL - Need for Concurrency - Concurrency control - Two Phase Locking- Timestamp - Multiversion - Validation and Snapshot isolation - Multiple Granularity locking - Deadlock Handling - Recovery Concepts - Recovery based on deferred and immediate update - Shadow paging - ARIES Algorithm.

Contents

Part I : Introduction to Transactions

3.1 Transaction Concepts

AU : Dec.-14, Marks 4

Definition : A transaction can be defined as a **group of tasks** that form a single logical unit.

For example - Suppose we want to withdraw ₹ 100 from an account then we will follow following operations :

1) Check account balance
2) If sufficient balance is present request for withdrawal.
3) Get the money
4) Calculate Balance = Balance − 100
5) Update account with new balance.

The above mentioned **four steps** denote **one transaction.**

In a database, each transaction should maintain ACID property to meet the consistency and integrity of the database.

Review Question

1. *Write a short note on – Transaction Concept.* **AU : Dec.-14, Marks 4**

3.2 ACID Properties

AU : May-14,18,19, Dec.-19, Marks 15

1) Atomicity :

- This property states that each transaction must be considered as a **single unit** and **must be completed fully or not completed at all.**
- No transaction in the database is left half completed.
- Database should be in a state either before the transaction execution or after the transaction execution. It **should not be in a state 'executing'.**
- For example - In above mentioned withdrawal of money transaction all the five steps must be completed fully or none of the step is completed. Suppose if transaction gets failed after step 3, then the customer will get the money but the balance will not be updated accordingly. The state of database should be either at before ATM withdrawal (i.e customer without withdrawn money) or after ATM withdrawal (i.e. customer with money and account updated). This will make the system in consistent state.

2) Consistency :

- The database must remain in consistent state after performing any transaction.
- For example : In ATM withdrawal operation, the balance must be updated appropriately after performing transaction. Thus the database can be in consistent state.

3) Isolation :

- In a database system where **more than one transaction** are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.
- No transaction will affect the existence of any other transaction.
- **For example :** If a bank manager is checking the account balance of particular customer, then manager should see the balance either before withdrawing the money or after withdrawing the money. This will make sure that each individual transaction is completed and any other dependent transaction will get the consistent data out of it. Any failure to any transaction will not affect other transaction in this case. Hence it makes all the transactions consistent.

4) Durability :

- The database should be **strong enough** to handle any **system failure.**
- If there is any set of insert /update, then it should be able to handle and commit to the database.
- If there is any **failure,** the database should be able to **recover** it to the consistent state.
- For example : In ATM withdrawal example, if the system failure happens after Customer getting the money then the system should be strong enough to update Database with his new balance, after system recovers. For that purpose the system has to keep the **log of each transaction and its failure.** So when the system recovers, it should be able to know when a system has failed and if there is any pending transaction, then it should be updated to Database.

Review Questions

1. *Explain with an example the properties that must be satisfied by transaction.* **AU : May-18, Marks 7**
2. *Explain the ACID properties of transaction.* **AU : May-14, Marks 8**

3. *Discuss in detail about the ACID properties of transaction.* **AU : May-19, Marks 15**
4. *Discuss the properties of a transaction that ensure integrity of data in the database system.* **AU : Dec 19, Marks 4**

AU : May-14,18, Dec.-11,19, Marks 8

3.3 Transaction States

- Each transaction has following five states :

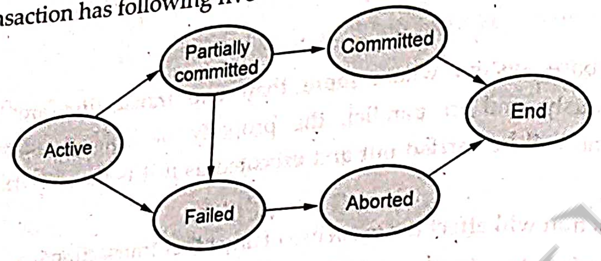


Fig. 3.3.1 : Transaction states

1) **Active :** This is the first state of transaction. For example : insertion, deletion or updation of record is done here. But data is not saved to database.
2) **Partially Committed :** When a transaction executes its final operation, it is said to be in a partially committed state.
3) **Failed :** A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
4) **Aborted :** If a transaction is failed to execute, then the database recovery system will make sure that the database is in its previous consistent state. If not, it brings the database to consistent state by aborting or rolling back the transaction.
5) **Committed :** If a transaction executes all its operations successfully, it is said to be committed. This is the last step of a transaction, if it executes without fail.

Example 3.3.1 *Define a transaction. Then discuss the following with relevant examples :*
1. A read only transaction 2. A read write transaction 3. An aborted transaction **AU : Dec.-11, Marks 8**

Solution :

(1) Read only transaction

T1
Read(A)
Read(B)
Display(A-B)

(2) A read write transaction

T1
Read(A)
A=A+100
Write(A)

(3)

T1	T2	
Read(A)		Assume A=100
A=A+50		A=150
Write(A)		
	Read(A)	A=150
	A=A+100	A=250
RollBack		A=100 (restore back to original value which is before Transaction T1)
	Write(A)	

Review Questions

1. *During execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through which transaction may pass. Explain why each state transaction may occur ?* **AU : May-18, Marks 6**
2. *With a neat sketch explain the states of transaction.* **AU : May-14, Marks 8**
3. *Brief the states of a transaction with a neat diagram.* **AU : Dec 19, Marks 4**

3.4 Schedules

Schedule is an order of multiple transactions executing in concurrent environment. Following figure represents the types of schedules.

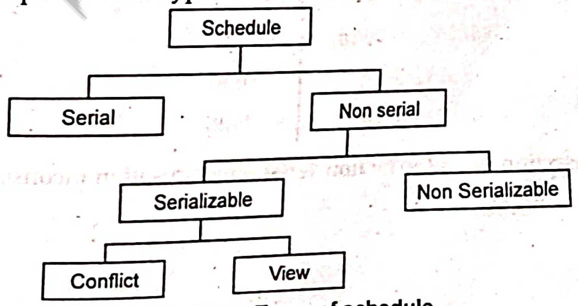


Fig. 3.4.1 : Types of schedule

Serial schedule : The schedule in which the transactions execute one after the other is called serial schedule. It is consistent in nature. For example : Consider following two transactions T_1 and T_2

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

All the operations of transaction T_1 on data items A and then B executes and then in transaction T_2 all the operations on data items A and B execute. The **R stands for Read** operation and **W stands for write** operation.

Non serial schedule : The schedule in which operations present within the transaction are intermixed. This may lead to conflicts in the result or inconsistency in the resultant data. For example-

Consider following two transactions,

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(B)
R(A)	
W(B)	
	R(B)
	W(B)

The above transaction is said to be **non serial** which result in inconsistency or conflicts *in the data.*

3.5 Serializability

AU : May-15,18,19, Dec.-15, Marks 13

- When multiple transactions run concurrently, then it may lead to inconsistency of data (i.e. change in the resultant value of data from different transactions).
- Serializability is a concept that helps to identify which non serial schedule and find the transaction equivalent to serial schedule.
- For example :

T_1	A	B	T_2
Initial Value	100	100	
A=A-10			
W(A)			
B=B+10			
W(B)			
	90	110	
			A=A-10
			W(A)
	80	110	

- In above transactions initially T_1 will read the values from database as A = 100, B = 100 and modify the values of A and B. But transaction T_2 will read the modified value i.e. 90 and will modify it to 80 and perform write operation. Thus at the end of transaction T_1 value of A will be 90 but at end of transaction T_2 value of A will be 80. Thus conflicts or inconsistency occurs here. This sequence can be converted to a sequence which may give us consistent result. This process is called **serializability.**

Serial schedule : The schedule in which the transactions execute one after the other is called serial schedule. It is consistent in nature. For example : Consider following two transactions T_1 and T_2

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

All the operations of transaction T_1 on data items A and then B executes and then in transaction T_2 all the operations on data items A and B execute. The **R stands for Read** operation and **W stands for write** operation.

Non serial schedule : The schedule in which operations present within the transaction are intermixed. This may lead to conflicts in the result or inconsistency in the resultant data. For example-

Consider following two transactions,

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(B)
R(A)	
W(B)	
	R(B)
	W(B)

The above transaction is said to be **non serial** which result in inconsistency or conflicts in the data.

3.5 Serializability

AU : May-15,18,19, Dec.-15, Marks 13

- When multiple transactions run concurrently, then it may lead to inconsistency of data (i.e. change in the resultant value of data from different transactions).
- Serializability is a concept that helps to identify which non serial schedule and find the transaction equivalent to serial schedule.
- For example :

T_1	A	B	T_2
Initial Value	100	100	
A=A-10			
W(A)			
B=B+10			
W(B)			
	90	110	
			A=A-10
			W(A)
	80	110	

- In above transactions initially T_1 will read the values from database as A = 100, B = 100 and modify the values of A and B. But transaction T_2 will read the modified value i.e. 90 and will modify it to 80 and perform write operation. Thus at the end of transaction T_1 value of A will be 90 but at end of transaction T_2 value of A will be 80. Thus conflicts or inconsistency occurs here. This sequence can be converted to a sequence which may give us consistent result. This process is called **serializability**.

Difference between Serial schedule and Serializable schedule

Serial schedule	Serializable schedule
No concurrency is allowed in serial schedule.	**Concurrency** is allowed in serializable schedule.
In serial schedule, if there are two transactions executing at the same time and no interleaving of operations is permitted, then following can be the possibilities of execution – (i) Execute all the operations of transactions T1 in a sequence and then execute all the operations of transactions T2 in a sequence. (ii) Execute all the operations of transactions T2 in a sequence and then execute all the operations of transactions T1 in a sequence.	In serializable schedule, if there are two transactions executing at the same time and interleaving of operations is allowed there can be different possible orders of executing an individual operation of the transactions.

Example of serial schedule

T1	T2
Read(A)	
A=A-50	
Write(A)	
Read(B)	
B=B+100	
Write(B)	
	Read(A)
	A=A+10
	Write(A)

Example of serializable schedule

T1	T2
Read(A)	
A=A-50	
Write(A)	
	Read(B)
	B=B+100
	Write(B)
Read(B)	
Write(B)	

- There are **two types of serializabilities : conflict** serializability and view serializability

3.5.1 Conflict Serializability

- ***Definition :*** Suppose T_1 and T_2 are two transactions and I_1 and I_2 are the instructions in T_1 and T_2 respectively. Then these two transactions are said to be conflict Serializable, if both the instruction access the data item d, and at least one of the instruction is write operation.

- **What is conflict ? :** In the definition **three conditions** are specified for a conflict in conflict serializability –

1) There should be **different transactions**

2) The **operations** must be performed on **same data items**

3) **One of the operation** must be the **Write(W)** operation

- We can test a given schedule for conflict serializability by constructing a **precedence graph** for the schedule, and by searching for absence of cycles in the graph.
- Predence graph is a directed graph, consisting of G=(V,E) where V is set of vertices and E is set of edges. The set of vertices consists of all the transactions participating in the schedule. The set of edges consists of all edges Ti →Tj for which one of three conditions holds :

1. Ti executes write(Q) before Tj executes read(Q).
2. Ti executes read(Q) before Tj executes write(Q).
3. Ti executes write(Q) before Tj executes write(Q).

- A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called **topological sorting.**

Testing for serializability

Following method is used for testing the serializability : To test the conflict serializability we can draw a graph G = (V,E) where V = vertices which represent the number of transactions. E = edges for conflicting pairs.

Step 1 : Create a node for each transaction.

Step 2 : Find the conflicting pairs(RW, WR, WW) on the same variable(or data item) by different transactions.

Step 3 : Draw edge for the given schedule. Consider following cases

1. Ti executes write(Q) before Tj executes read(Q), then draw edge from Ti to Tj.
2. Ti executes read(Q) before Tj executes write(Q) , then draw edge from Ti to Tj
3. Ti executes write(Q) before Tj executes write(Q), , then draw edge from Ti to Tj

Step 4 : Now, if precedence graph is cyclic then it is a non conflict serializable schedule and if the precedence graph is acyclic then it is conflict serializable schedule.

Example 3.5.1 *Consider the following two transactions and schedule (time goes from top to bottom). Is this schedule conflict-serializable ? Explain why or why not.*

T_1	T_2
R(A)	
W(A)	
	R(A)
	R(B)
R(B)	
W(B)	

Solution :

Step 1 : To check whether the schedule is conflict serializable or not we will check from **top to bottom**. Thus we will start reading from top to bottom as

T_1: R(A) -> T_1:W(A) ->T_2:R(A) -> T_2:R(B) ->T_1:R(B)->T_1:W(B)

Step 2 : We will find **conflicting operations**. Two operations are called as conflicting operations if all the following conditions hold true for them-

i) Both the **operations belong to different transactions.**

ii) Both the operations are on **same data item.**

iii) **At least one** of the two operations is a **write operation**

From above given example in the top to bottom scanning we find the conflict as T_1:W(A) ->T_2:R(A).

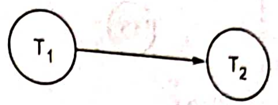
i) Here note that there are two different transactions T_1 and T_2,

ii) Both work on same data item i.e. A and

iii) One of the operation is write operation

Step 3 : We will build a precedence graph by drawing one node from each transaction. In above given scenario as there are two transactions, there will be two nodes namely T_1 and T_2

Step 4 : Draw the edge between conflicting transactions. For example in above given scenario, the conflict occurs while moving from T_1:W(A) to T_2:R(A). Hence edge must be from T_1 to T_2.



Step 5 : Repeat the step 4 while reading from top to bottom. Finally the **precedence graph** will be as follows

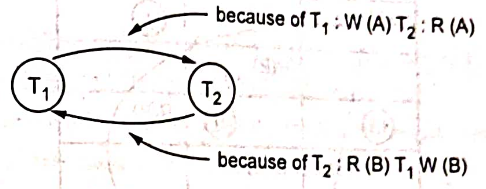

because of T1 : W (A) T2 : R (A)
T1
T2
because of T2 : R (B) T1 W (B)

Fig. 3.5.1 : Precedence graph

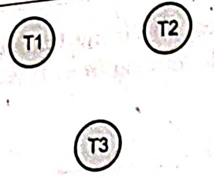
Step 6 : Check if **any cycle** exists in the graph. Cycle is a path using which we can start from one node and reach to the same node. If the is cycle found then schedule is not conflict serializable. In the step 5 we get **a graph with cycle**, that means given schedule is **not conflict serializable.**

Example 3.5.2 *Check whether following schedule is conflict serializable or not. If it is not conflict serializable then find the serializability order.*

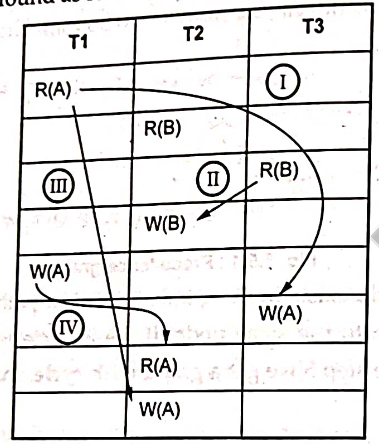
T_1	T_2	T_3
R(A)		
	R(B)	
		R(B)
	W(B)	
W(A)		
		W(A)
	R(A)	
	W(A)	

Solution :

Step 1 : We will read from top to bottom, and build a precedence graph for conflicting entries. We will build a precedence graph by drawing one node from each transaction. In above given scenario as there are three transactions, there will be two nodes namely T1 T2, and T3

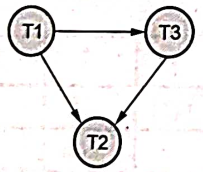


Step 2 : The conflicts are found as follows -



T1	T2	T3
R(A)		I
	R(B)	
III	II	R(B)
	W(B)	
W(A)		
IV		W(A)
	R(A)	
	W(A)	

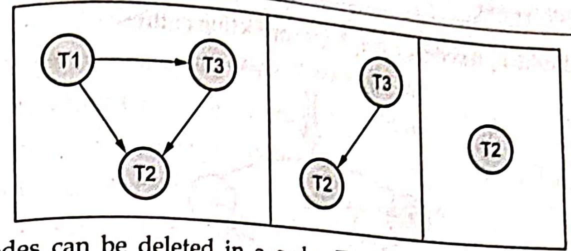
Step 3 : The precedence graph will be as follows -



Step 4 : As there is **no cycle** in the precedence graph, the given sequence is **conflict serializable.** Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

Step 5 : A **serializability order** of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called **topological sorting.**

Step 6 : Find the vertex which has no incoming edge which is **T1.** If we delete T1 node then **T3** is a node that has no incoming edge. If we delete T3, then **T2** is a node that has no incoming edge.



Thus the nodes can be deleted in a order T1, T3 and T2. Hence the order will be T1-T3-T2

Example 3.5.3 *Check whether the below schedule is conflict serializable or not.*
{B2,r2(X),b1,r1(X),W1(X),r1(Y),W1(Y),W2(X),e1,C1,e2,C2}

Solution : b2 and b1 represents **begin transaction 2** and **begin transaction 1**. Similarly, e1 and e2 represents **end transaction 1** and **end transaction 2.**

We will rewrite the schedule as follows -

T_1	T_2
	$r_2(X)$
$r_1(X)$	
$W_1(X)$	
$r_1(Y)$	
$W_1(Y)$	
	$W_2(X)$

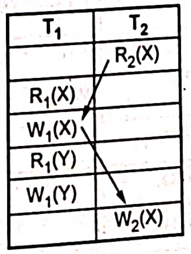
Step 1 : We will find **conflicting operations.** Two operations are called as conflicting operations if all the following conditions hold true for them -

i) Both the **operations belong to different transactions.**

ii) Both the operations are on **same data item.**

iii) **At least one** of the two operations is **a write operation.**

The conflicting entries are as follows -



T_1	T_2
	$R_2(X)$
$R_1(X)$	
$W_1(X)$	
$R_1(Y)$	
$W_1(Y)$	
	$W_2(X)$

Step 2 : Now we build a precedence graph for conflicting entries.

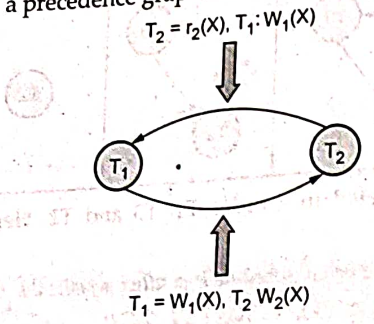


Fig. 3.5.2

As there are two transactions only two nodes are present in the graph.

Step 3 : We get a **graph with cycle**, that means given schedule is **not conflict serializable.**

Example 3.5.4 *Consider the three transactions T1, T2, and T3 and schedules S1 and S2 given below. Determine whether each schedule is serializable or not ? If a schedule is serializable write down the equivalent serial schedule(S).*

T1: R1(x) R1(z);W1(x);

T2: R2(x);R2(y);W2(z);W2(y)

T3:R3(x);R3(y);W3(y);

S1: R1(x);R2(z);R1(z);R3(x);R3(y);W1(x);W3(y);R2(y);W2(z);W2(y);

S2: R1(x);R2(z);R3(x);R1(z);R2(y);R3(y);W1(x);W2(z);W3(y);W2(y);

Solution : Step 1 : We will represent the schedule S1 as follows

T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

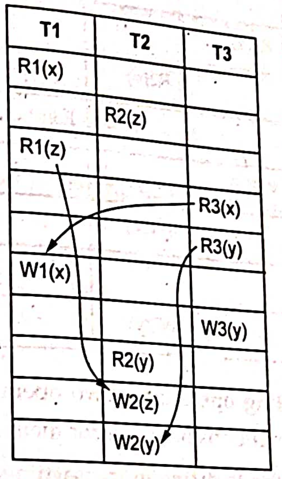
Step (a) : We will find **conflicting operations.** Two operations are called as conflicting operations if all the following conditions hold true for them -

i) Both the **operations belong to different transactions.**

ii) Both the operations are on **same data item.**

iii) **At least one** of the two operations is a **write operation**

The conflicting entries are as follows -



T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

Step (b) : Now we will draw precedence graph as follows -

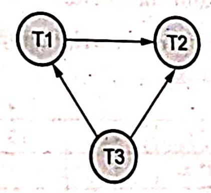


Fig. 3.5.3 : Precedence graph

As there is **no cycle** in the precedence graph, the given sequence is **conflict serializable.** Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

Step (c) : A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called **topological sorting.**

Step (d) : Find the vertex which has no incoming edge which is T3. If we delete T3, then T1 is the edge that has no incoming edge. Finally find the vertex having no outgoing edge which is T2. Hence the order will be T3- T1-T2.

Step 2 : Now we build a precedence graph for conflicting entries.

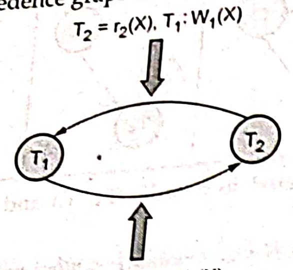


Fig. 3.5.2

As there are two transactions only two nodes are present in the graph.

Step 3 : We get a **graph with cycle**, that means given schedule is **not conflict serializable.**

Example 3.5.4 *Consider the three transactions T1, T2, and T3 and schedules S1 and S2 given below. Determine whether each schedule is serializable or not ? If a schedule is serializable write down the equivalent serial schedule(S).*

T1: R1(x) R1(z);W1(x);

T2: R2(x);R2(y);W2(z);W2(y)

T3:R3(x);R3(y);W3(y);

S1: R1(x);R2(z);R1(z);R3(x);R3(y);W1(x);W3(y);R2(y);W2(z);W2(y);

S2: R1(x);R2(z);R3(x);R1(z);R2(y);R3(y);W1(x);W2(z);W3(y);W2(y);

Solution : Step 1 : We will represent the schedule S1 as follows

T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

Step (a) : We will find **conflicting operations.** Two operations are called as conflicting operations if all the following conditions hold true for them -

i) Both the **operations belong to different transactions.**

ii) Both the operations are on **same data item.**

iii) **At least one** of the two operations is a **write operation**

The conflicting entries are as follows -

T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

Step (b) : Now we will draw precedence graph as follows -

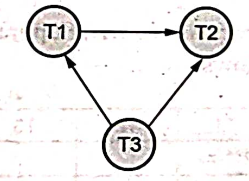


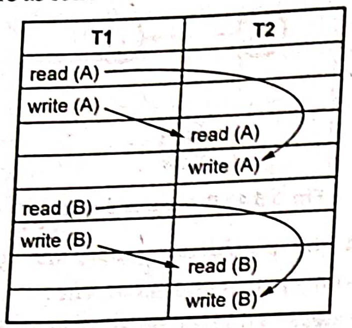
Fig. 3.5.3 : Precedence graph

As there is **no cycle** in the precedence graph, the given sequence is **conflict serializable.** Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

Step (c) : A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called **topological sorting.**

Step (d) : Find the vertex which has no incoming edge which is T3. If we delete T3, then T1 is the edge that has no incoming edge. Finally find the vertex having no outgoing edge which is T2. Hence the order will be T3- T1-T2.

The conflicting entries are as follows -



T1	T2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Step 2 : Now we will build precedence graph as follows -

$$\begin{tikzcd} T1 \arrow[r] & T2 \end{tikzcd}$$

Step 3 : There is **no cycle** in the precedence graph. That means this schedule is conflict serializable. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow the following steps to find the serializable order.

1) Find the vertex which has no incoming edge which is T1.
2) Then find the vertex having no outgoing edge which is T2. In between them there is no other transaction.
3) Hence the order will be T1-T2.

3.5.2 View Serializability

- If a given schedule is found to be view equivalent to some serial schedule, then it is called as a view serializable schedule.
- **View Equivalent Schedule :** Consider two schedules S_1 and S_2 consisting of transactions T_1 and T_2 respectively, then schedules S_1 and S_2 are said to be view equivalent schedule if it satisfies following three conditions :
  - o If transaction T_1 reads a data item A from the database initially in schedule S_1, then in schedule S_2 also, T_1 must perform the initial read of the data item X from the database. This is same for all the data items. In other words - the initial reads must be same for all data items.
  - o If data item A has been updated at last by transaction T_i in schedule S_1, then in schedule S_2 also, the data item A must be updated at last by transaction T_i.

  - o If transaction Ti reads a data item that has been updated by the transaction T_j in schedule S_1, then in schedule S_2 also, transaction T_i must read the same data item that has been updated by transaction T_j. In other words the Write-Read sequence must be same.

Sr.No.	Conflict serializability	View serializability
1	Every conflict serializable is view serializable.	Every view serializable schedule is not necessarily conflict serializable.
2.	It is easy to test conflict serializability.	It is complex to test view serializability.

Steps to check whether the given schedule is view serializable or not

Step 1 : If the schedule is conflict serializable then it is surely view serializable because conflict serializability is a restricted form of view serializability.

Step 2 : If it is not conflict serializable schedule then check whether there exist any blind write operation. The blind write operation is a write operation without reading a value. If there does not exist any blind write then that means the given schedule is not view serializable. In other words if a blind write exists then that means schedule may or may not be view conflict.

Step 3 : Find the view equivalence schedule

Example 3.5.6 *Consider the following schedules for checking if these are view serializable or not.*

T_1	T_2	T_3
		W(C)
	R(A)	
	W(B)	
R(C)		
		W(B)
W(B)		

Solution : i) The initial read operation is performed by T_2 on data item A or by T_1 on data item C. Hence we will begin with T_2 or T_1. We will choose T_2 at the beginning.

ii) The final write is performed by T_1 on the same data item B. Hence T_1 will be at the last position.

iii) The data item C is written by T_3 and then it is read by T_1. Hence T_3 should appear before T_1.

Thus we get the order of schedule of view serializability as $T_2 - T_1 - T_3$

Example 3.5.7 *Consider following two transactions :*

```
T1 : read(A)
     read(B)
     if A=0 then B:=B+1;
     write(B)
T2 : read(B);
     read(A);
     if B=0 then A:=A+1;
     write(A)
```

Let consistency requirement be A=0 V B=0 with A=B=0 the initial values.

1) *Show that every serial execution involving these two transactions preserves the consistency of the Database ?*

2) *Show a concurrent execution of T_1 and T_2 that produces a non serializable schedule ?*

3) *Is there a concurrent execution of T_1 and T_2 that produces a serializable schedule ?*

Solution : 1) There are two possible executions: $T_1 \rightarrow T_2$ or $T_2 \rightarrow T_1$

Consider case $T_1 \rightarrow T_2$ then

A	B
0	0
0	1
0	1

0 V 0 = 0
0 V 1 = 1
1 V 0 = [illegible]

$A \vee B = A$ OR $B = F \vee T = T$. This means consistency is met.

Consider case $T_2 \rightarrow T_1$ then

A	B
0	0
1	0
1	0

$A \vee B = A$ OR $B = F \vee T = T$. This means consistency is met.

(2) The concurrent execution means interleaving of transactions T_1 and T_2. It can be

T_1	T_2
R(A)	
	R(B)
	R(A)
R(B) If A=0 then B=B+1	If B=0 then A=A+1 W(A)
W(B)	

This is a non-serializable schedule.

(3) There is no concurrent execution resulting in a serializable schedule.

Example 3.5.8 *Test serializability of the following schedule :*

i) $r_1(x);r_3(x);w_1(x);r_2(x);w_3(x)$ ii) $r_3(x);r_2(x);w_3(x);r_1(x);w_1(x)$

Solution : i) $r_1(x);r_3(x);w_1(x);r_2(x);w_3(x)$

The r_1 represents the read operation of transaction T_1, w_3 represents the write operation on transaction T_3 and so on. Hence from given sequence the schedule for three transactions can be represented as follows :

T_1	T_2	T_3
$r_1(x)$		
		$r_3(x)$
$w_1(x)$		
	$r_2(x)$	
		$w_3(x)$

Step 1 : We will use the precedence graph method to check the serializability. As there are three transactions, three nodes are created for each transaction.

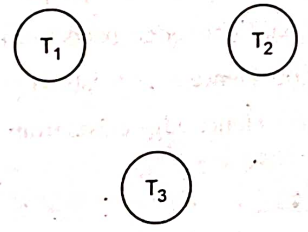


Fig. 3.5.5 : Nodes

Step 2 : We will read from top to bottom. Initially we read $r_1(x)$ and keep on moving bottom in search of write operation. Here all the transactions work on same data item i.e. x. Now we get a write operation in T_3 as $w_3(x)$. Hence the dependency is from T_1 to T_3. Therefore we draw edge from T_1 to T_3.

Similarly, for $r_3(x)$ we get $w_1(x)$ pair. Hence there will be edge from T_3 to T_1. Continuing in this fashion we get the precedence graph as

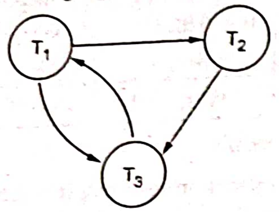


Fig. 3.5.6 : Precedence Graph

Step 3 : As cycle exists in the above precedence graph, we conclude that it is **not serializable.**

ii) $r_3(x);r_2(x);w_3(x);r_1(x);w_1(x)$

From the given sequence the schedule can be represented as follows :

T_1	T_2	T_3
		$r_3(x)$
	$r_2(x)$	
		$w_3(x)$
$r_1(x)$		
$w_1(x)$		

Step 1 : Read the schedule from top to bottom for pair of operations. For $r_3(x)$ we get $w_1(x)$ pair. Hence edge exists from T_3 to T_1 in precedence graph.

There is a pair from $r_2(x) : w_3(x)$. Hence edge exists from T_2 to T_3.

There is a pair from $r_2(x) : w_1(x)$. Hence edge exists from T_2 to T_1.

There is a pair from $w_3(x): r_1(x)$. Hence edge exists from T_3 to T_1.

Step 2 : The precedence graph will then be as follows -

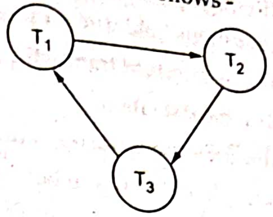
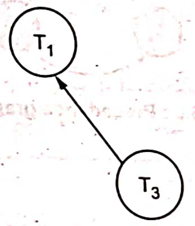


Fig. 3.5.7 : Precedence graph

Step 3 : As there is no cycle in the above graph, the given schedule is **serializable.**

Step 4 : The searializability order for consistent schedule will be obtained by applying topological sorting on above drawn precedence graph. This can be achieved as follows,

Sub-Step 1 : Find the node having no incoming edge. We obtain T_2 is such a node. Hence T_2 is at the beginning of the serializability sequence. Now delete T_2. The Graph will be



Sub-Step 2 : Repeat sub-Step 1, We obtain T_3 and T_1 nodes as a sequence.

Thus we obtain the sequence of transactions as T_2, T_3 and T_1. Hence the serializability order is

$r_2(x);r_3(x);w_3(x):r_1(x);w_1(x)$

Example 3.5.9 *Consider the following schedules. The actions are listed in the order they are scheduled, and prefixed with the transaction name.*

S1 : T1 : R(X), T2 : R(X), T1 : W(Y), T2 : W(Y) T1 : R(Y), T2 : R(Y)

S2 : T3 : W(X), T1 : R(X), T1 : W(Y), T2 : R(Z), T2 : W(Z) T3 : R(Z)

For each of the schedules, answer the following questions :

i) What is the precedence graph for the schedule ?

ii) Is the schedule conflict-serializable ? If so, what are all the conflict equivalent serial schedules ?

iii) Is the schedule view-serializable ? If so, what are all the view equivalent serial schedules ?

AU : May-15, Marks 2 + 7 + 7

Solution : I) We will find **conflicting operations**. Two operations are called as conflicting operations if all the following conditions hold true for them -

- **Both the operations belong to different transactions.**
- Both the operations are on **same data item.**
- At least one of the two operations is a write operation

For S1: From above given example in the top to bottom scanning we find the conflict as

- o **T1 : W(Y), T2 : W(Y) and**
- o **T2 : W(Y), T1 : R(Y)**

Hence we will build the precedence graph. Draw the edge between conflicting transactions. For example in above given scenario, the conflict occurs while moving from T_1:W(Y) to T_2:W(Y). Hence edge must be from T_1 to T_2. Similarly for second conflict, there will be the edge from T2 to T1

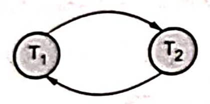


Fig. 3.5.8 : Precedence graph for S1

For S2: The conflicts are

- o **T3 : W(X), T1 : R(X)**
- o **T2 : W(Z) T3 : R(Z)**

Hence the precedence graph is as follows -

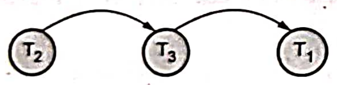


Fig. 3.5.9 : Precedence graph for S2

(ii)

- o S1 is **not conflict-serializable** since the dependency **graph has a cycle.**
- o S2 is conflict-serializable as the dependency graph is acylic. The order T2-T3-T1 is the only equivalent **serial order.**

(iii)

- o S1 is **not view serializable.**
- o S2 is trivially view-serializable as it is conflict serializable. The **only serial order** allowed is

T2-T3-T1.

Example 3.5.10 *Check whether following schedule is view serializable or not. Justify your answer. (Note : T1 and T2 are transactions). Also explain the concept of view equivalent schedules and conflict equivalent schedule considering the example schedule given below :*

T1	T2
read (A) *A: = A - 50*	
write (A)	
	read (A) *temp: = A*0.1* *A: = A - temp*
	write (A)
read (B) *B: = B + 50*	
write (B)	
	read (B) *B: = B + temp*
	write (B)

Solution :

Step 1 : We will first find if the given schedule is conflict serializable or not. For that purpose, we will find the conflicting operations. These are as shown below -

T1	T2
read (A) A : = A – 50	
write (A)	
	read (A) temp : = A*0.1 A : = A – temp
	write (A)
read (B) B : = B + 50	
write (B)	
	read (B) B : = B + temp
	write (B)

The precedence graph is as follows -

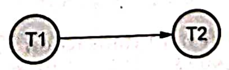


Fig. 3.5.10 : Precedence graph

As there exists no cycle, the schedule is conflict serializable. The possible serializability order can be T1 - T2

Now we check it for view serializability. As we get the serializability order as T1 - T2, we will find the view equivalence with the given schedule as serializable schedule.

Let S be the given schedule as given in the problem statement. Let the serializable schedule is S'={T1,T2}. These two schedules are represented as follows :

T1	T2
read (A) A: = A - 50	
write (A)	
	read (A) temp: = A*0.1 A: = A - temp
	write (A)
read (B) B: = B + 50	
write (B)	
	read (B) B: = B + temp
	write (B)

Schedule S

T1	T2
read (A) A: = A - 50	
write (A)	
read (B) B: = B + 50	
write (B)	
	read (A) temp: = A*0.1 A: = A - temp
	write (A)
	read (B) B: = B + temp
	write (B)

Schedule S'

Now we will check the equivalence between them using following conditions -

(1) Initial Read

In schedule S initial read on A is in transaction T1. Similarly initial read on B is in transaction T1.

Similarly in schedule S', initial read on A is in transaction T1. Similarly initial read on B is in transaction T1.

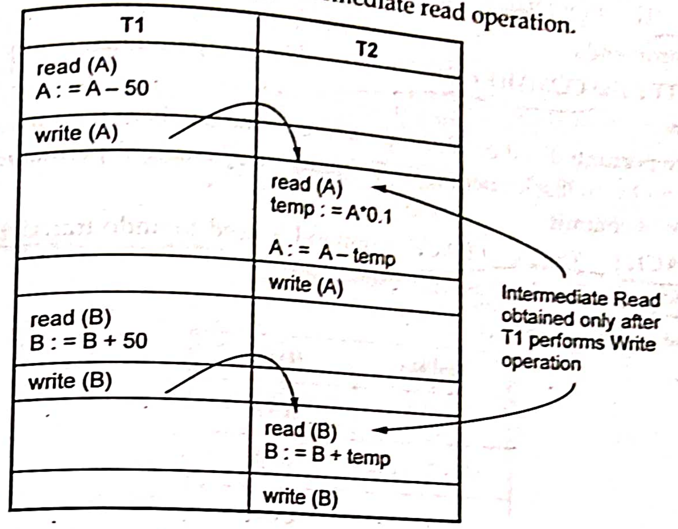
(2) Final Write

In schedule S final write on A is in transaction T2. Similarly final write on B is in transaction T2.

In schedule S' final write on A is in transaction T2. Similarly final write on B is in transaction T2

(3) Intermediate Read

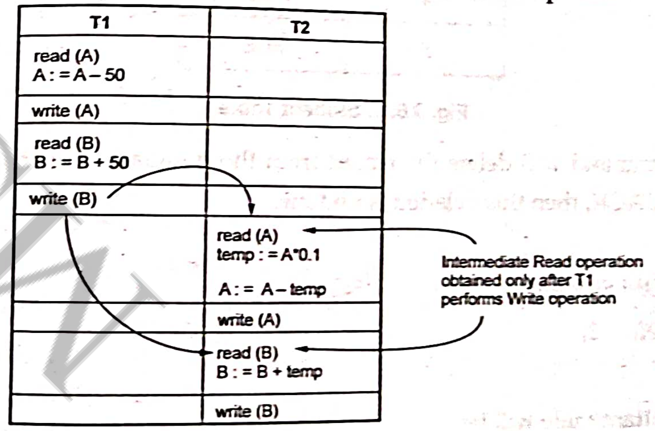
Consider schedule S for finding intermediate read operation.



T1	T2
read (A) A : = A – 50	
write (A)	
	read (A) temp : = A*0.1 A : = A – temp
	write (A)
read (B) B : = B + 50	
write (B)	
	read (B) B : = B + temp
	write (B)

Intermediate Read obtained only after T1 performs Write operation

Similarly consider schedule S' for finding intermediate read operation.



T1	T2
read (A) A : = A – 50	
write (A)	
read (B) B : = B + 50	
write (B)	
	read (A) temp : = A*0.1 A : = A – temp
	write (A)
	read (B) B : = B + temp
	write (B)

Intermediate Read operation obtained only after T1 performs Write operation

In both the schedules S and S', the intermediate read operation is performed by T2 only after T1 performs write operation.

Thus all the above three conditions get satisfied. Hence given schedule is view serializable.

Review Questions

1. *Explain Conflict serializability and view serializability.* **AU : May-18, Marks 6, Dec.-15, Marks 8**
2. *Discuss in detail about the testing of serializability.* **AU : May-19, Marks 13**

3.6 Transaction Support in SQL

The COMMIT, ROLLBACK, and SAVEPOINT are collectively considered as Transaction Commands

(1) COMMIT : The COMMIT command is used to save permanently any transaction to database.

When we perform, Read or Write operations to the database then those changes can be undone by rollback operations. To make these changes permanent, we should make use of **commit**

(2) ROLLBACK : The ROLLBACK command is used to undo transactions that have not already saved to database. For example

Consider the database table as

RollNo	Name
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Fig. 3.6.1 : Student Table

Following command will delete the record from the database, but if we immediately performs ROLLBACK, then this deletion is undone.

For instance -

```
DELETE  FROM Student
WHERE RollNo = 2;
ROLLBACK;
```

Then the resultant table will be

RollNo	Name
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

(3) SAVEPOINT : A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction. The SAVEPOINT can be created as

SAVEPOINT savepoint_name;

Then we can ROLLBACK to SAVEPOIT as

ROLLBACK TO savepoint_name;

For example – Consider Student table as follows -

RollNo	Name
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Fig. 3.6.2 : Student Table

Consider Following commands

```
SQL> SAVEPOINT S1
SQL>DELETE FROM Student
Where RollNo=2;
SQL> SAVEPOINT S2
SQL>DELETE FROM Student
Where RollNo=3;
SQL> SAVEPOINT S3
SQL>DELETE FROM Student
Where RollNo=4
SQL> SAVEPOINT S4
SQL>DELETE FROM Student
Where RollNo=5
SQL> ROLLBACK TO S3;
```

Then the resultant table will be

RollNo	Name
1	AAA
2	BBB
3	CCC

Thus the effect of deleting the record having RollNo 2, and RollNo3 is undone.

Part II : Concurrency Control

3.7 Concurrency Control

AU : May-19, Marks 15

- One of the fundamental properties of a transaction is **isolation**.
- When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved.
- A database can have multiple transactions running at the same time. This is called **concurrency**.
- To preserve the isolation property, the system must control the interaction among the concurrent transactions; this control is achieved through one of a variety of mechanisms called **concurrency control schemes.**
- **Definition of concurrency control :** A mechanism which ensures that simultaneous execution of more than one transactions does not lead to any database inconsistencies is called **concurrency control mechanism.**
- The concurrency control can be achieved with the help of various protocols such as - lock based protocol, Deadlock handling, Multiple Granularity, Timestamp based protocol, and validation based protocols.

Review Question

1. *What is concurrency control ? How it is implemented in DBMS ? Briefly elaborate diagrams and examples.* **AU : May-19, Marks 15**

3.8 Need for Concurrency

AU : May-17,19, Marks 15

- Following are the purposes of concurrency control -
  - **To ensure isolation**
  - **To resolve read-write or write-write conflicts**
  - **To preserve consistency of database**
- Concurrent execution of transactions over shared database creates several data integrity and consistency problems - these are

(1) Lost update problem : This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

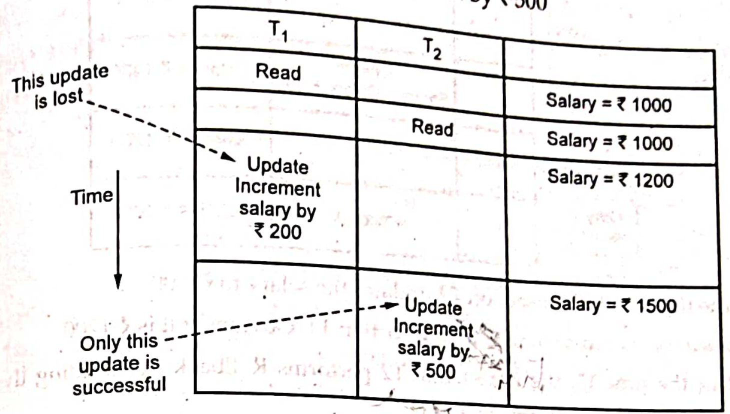
For example – Consider following transactions

(1) Salary of Employee is read during transaction T1.

(2) Salary of Employee is read by another transaction T2.

(3) During transaction T1, the salary is incremented by ₹ 200

(4) During transaction T2, the salary is incremented by ₹ 500



T_1	T_2	
Read		Salary = ₹ 1000
	Read	Salary = ₹ 1000
Update Increment salary by ₹ 200		Salary = ₹ 1200
	Update Increment salary by ₹ 500	Salary = ₹ 1500

The result of the above sequence is that the update made by transaction T1 is completely lost. Therefor this problem is called as lost update problem.

(2) Dirty read or Uncommited read problem : The dirty read is a situation in which one transaction reads the data immediately after the write operation of previous transaction

T_1	T_2
R(A)	
A=A+50	
W(A)	
	R(A)
	A=A-20
	W(A)
	Commit
Commit	

Dirty read

For example – Consider following transactions -

Assume initially salary is = ₹ 1000

Time	T_1	T_1	
	...	...	Salary = ₹ 1000
t_1		Update Salary = Salary + 200	Salary = ₹ 1200
t_2 (Dirty Read)	Read		Salary = ₹ 1200
t_3		Rollback	Salary = ₹ 1000

(1) At the time t1, the transaction T2 updates the salary to ₹1200

(2) This salary is read at time t2 by transaction T1. Obviously it is ₹ 1200

(3) But at the time t3, the transaction T2 performs Rollback by undoing the changes made by T1 and T2 at time t1 and t2.

(4) Thus the salary again becomes = ₹ 1000. This situation leads to **Dirty Read** or **Uncommited Read** because here the read made at time t2**(immediately after update of another transaction)** becomes a dirty read.

(3) Non-repeatable read problem

This problem is also known as **inconsistent analysis problem**. This problem occurs when a particular transaction sees two different values for the same row within its lifetime. For example –

Time	T_1	T_2	
t_1	Read		Salary = ₹ 1000
t_2	.	Update salary from ₹ 1000 to ₹ 1200	Salary = ₹ 1200
t_3		Commit	
t_4	Read		Salary = ₹ 1200

(1) At time t_1, the transaction T_1 reads the salary as ₹ 1000

(2) At time t_2 the transaction T_2 reads the same salary as ₹ 1000 and updates it to ₹1200

(3) Then at time t_3, the transaction T_2 gets committed.

(4) Now when the transaction T_1 reads the same salary at time t_4, it gets different value than what it had read at time t_1. Now, transaction T_1 cannot repeat its reading operation. Thus inconsistent values are obtained.

Hence the name of this problem is non-repeatable read or inconsistent analysis problem.

(4) Phantom read problem

The phantom read problem is a special case of non repeatable read problem.

This is a problem in which one of the transaction makes the changes in the database system and due to these changes another transaction can not read the data item which it has read just recently. For example –

Time	T_1	T_2	
t_1	Read		Salary = ₹ 1000
t_2		Read	Salary = ₹ 1000
t_3	Delete salary		No salary
t_4		Read	"Can not find salary"

(1) At time t_1, the transaction T_1 reads the value of salary as ₹ 1000

(2) At time t_2, the transaction T_2 reads the value of the same salary as ₹ 1000

(3) At time t_3, the transaction T_1 deletes the variable salary.

(4) Now at time t_4, when T_2 again reads the salary it gets error. Now transaction T_2 can not identify the reason why it is not getting the salary value which is read just few time back.

This problem occurs due to changes in the database and is called phantom read problem.

Review Questions

1. *Discuss the violations caused by each of the following: dirty read, non repeatable read and phantoms with suitable example.* **AU : May-17, Marks 13**
2. *What is concurrency control ? How it is implemented in DBMS ? Briefly elaborate diagrams and examples.* **AU : May-19, Marks 15**

3.9 Locking Protocols

AU : Dec-15,17, May-16, Marks 16

3.9.1 Why Do We Need Locks ?

- One of the method to ensure the **isolation** property in transactions is to require that data items be accessed in a mutually exclusive manner. That means, while one transaction is accessing a data item, no other transaction can modify that data item.
- The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a **lock on that item.**
- Thus the lock on the operation is required to ensure the isolation of transaction.

3.9.2 Simple Lock Based Protocol

- **Concept of Protocol :** The lock based protocol is a mechanism in which there is exclusive use of locks on the data item for current transaction.
- **Types of Locks :** There are two types of locks used -

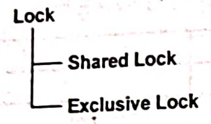


Fig. 3.9.1 : Types of locks

i) **Shared Lock :** The shared lock is used for reading data items only. It is denoted by **Lock-S**. This is also called as **read lock**.

ii) **Exclusive Lock :** The exclusive lock is used for both read and write operations. It is denoted as **Lock-X**. This is also called as **write lock**.

- The **compatibility matrix** is used while working on set of locks. The **concurrency control manager** checks the compatibility matrix before granting the lock. If the two modes of transactions are compatible to each other then only the lock will be granted.
- In a set of locks may consists of shared or exclusive locks. Following matrix represents the compatibility between modes of locks.

	S	X
S	T	F
X	F	F

Fig. 3.9.2 : Compatibility matrix for locks

Here T stands for True and F stands for False. If the control manager get the compatibility mode as True then it grant the lock otherwise the lock will be denied.

- **For example :** If the transaction T_1 is holding a shared lock in data item A, then the control manager can grant the shared lock to transaction T_2 as compatibility is True.
  But it cannot grant the exclusive lock as the compatibility is false. In simple words if transaction T_1 is reading a data item A then same data item A can be read by another transaction T_2 but cannot be written by another transaction.
- Similarly if an exclusive lock (i.e. lock for read and write operations) is hold on the data item in some transaction then no other transaction can acquire Shared or

exclusive lock as the compatibility function denotes F. That means of some transaction is writing a data item A then another transaction can not read or write that data item A.

Hence the rule of thumb is

i) **Any number** of transactions can hold **shared lock** on an item.

ii) But **exclusive lock can be hold by only one transaction.**

- **Example of a schedule denoting shared and exclusive locks :** Consider following schedule in which initially A=100. We deduct 50 from A in T_1 transaction and Read the data item A in transaction T_2. The scenario can be represented with the help of locks and concurrency control manager as follows :

Exclusive Lock

Shared Lock

T_1	T_2	Concurrency control manager
Lock-X(A)		
		Grant X(A,T1) because in T1 there is write operation.
R(A)		
A=A-50		
W(A)		
Unlock(A)		
	Lock-S(A)	
		Grant S(A,T2) because in T2 there is Read operation
	R(A)	
	Unlock(A)	

Review Questions

1. *State and explain the lock based concurrency control with suitable example.* **AU : Dec-17, Marks 13, May-16, Marks 16**
2. *What is Concurrency control ? How is implemented in DBMS ? Illustrate with suitable example.* **AU : Dec-15, Marks 8**

3.10 Two Phase Locking

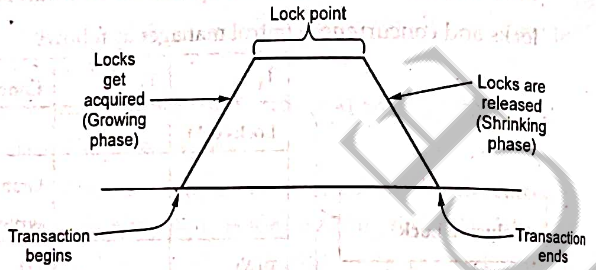
AU : May-14,18, Dec.-11,16,19, Marks 9

- The two phase locking is a protocol in which there are two phases :
  i) Growing phase (Locking phase) : **It is a phase in which the transaction may obtain locks but does not release any lock.**
  ii) Shrinking phase (Unlocking phase) : **It is a phase in which the transaction may release the locks but does not obtain any new lock.**
- **Lock Point :** The last lock position or first unlock position is called lock point. For example

Lock(A)
Lock(B)
Lock(C)
....
...
...

$\left.\begin{matrix} \\ \\ \end{matrix}\right\}$ **Lock Point**

Unlock(A)
Unlock(B)
Unlock(C)


Lock point
Locks get acquired (Growing phase)
Locks are released (Shrinking phase)
Transaction begins
Transaction ends

For example –

Consider following transactions

T1	T2
Lock-X(A)	Lock-S(B)
Read(A)	Read(B)
A=A-50	Unlock-S(B)
Write(A)	
Lock-X(B)	
Unlock-X(A)	
B=B+100	Lock-S(A)
Write(B)	Read(A)
Unlock-X(B)	Unlock-S(A)

The important rule for being a two phase locking is - All Lock operations precede all the unlock operations.

In above transactions T1 is in two phase locking mode but transaction T2 is not in two phase locking. Because in T2, the Shared lock is acquired by data item B, then data item B is read and then the lock is released. Again the lock is acquired by data item A , then the data item A is read and the lock is then reloased. Thus we get lock-unlock-lock-unlock sequence. Clearly this is not possible in two phase locking.

Example 3.10.1 *Prove that two phase locking guarantees serializability.* **AU : Dec.-11, Marks 8**

Solution :

- o Serializability is mainly an issue of handling write operation. Because any inconsistency may only be created by **write** operation.
- o Multiple reads on a database item can happen parallely.
- o 2-Phase locking protocol restricts this unwanted read/write by applying **exclusive lock.**
- o Moreover, when there is an **exclusive lock** on an item it will **only be released in shrinking phase.** Due to this restriction there is no chance of getting any inconsistent state.

The serializability using two phase locking can be understood with the help of following example

Consider two transactions

T_1	T_2
R(A)	
	R(A)
R(B)	
W(B)	

Step 1 : Now we will apply two phase locking. That means we will apply locks in growing and shrinking phase

T_1	T_2
Lock-S(A)	
R(A)	
	Lock-S(A)
	R(A)
Lock-X(B)	
R(B)	
W(B)	
Unlock-X(B)	
	Unlock-S(A)

Note that above schedule is serializable as it prevents interference between two transactions.

The serializability order can be obtained based on the **lock point**. The lock point is either last lock operation position or first unlock position in the transaction.

The last lock position is in T_1, then it is in T_2. Hence the serializability will be T_1->T_2 based on lock points. Hence The **serializability sequence** can be **R1(A);R2(A);R1(B);W1(B)**

Limitations of Two Phase Locking Protocol

The two phase locking protocol leads to two problems – deadlock and cascading roll back.

(1) Deadlock : The deadlock problem can not be solved by two phase locking. Deadlock is a situation in which when two or more transactions have got a lock and waiting for another locks currently held by one of the other transactions.

For example

T1	T2
Lock-X(A)	Lock-X(B)
Read(A)	Read(B)
A=A-50	B=B+100
Write(A)	Write(B)
Delayed, wait for T2 to release Lock on B	Delayed, wait for T1 to release Lock on A

(2) Cascading Rollback : Cascading rollback is a situation in which a single transaction failure leads to a series of transaction rollback. For example -

T1	T2	T3
Read(A)		
Read(B)		
C=A+B		
Write(C)		
	Read(C)	
	Write(C)	
		Read(C)

When T1 writes value of C then only T2 can read it. And when T2 writes the value of C then only transaction T3 can read it. But if the transaction T1 gets failed then automatically transactions T2 and T3 gets failed.

The simple two phase locking does not solve the cascading rollback problem. To solve the problem of cascading Rollback two types of two phase locking mechanisms can be used.

3.10.1 Types of Two Phase Locking

(1) Strict two phase locking : The strict 2PL protocol is a basic two phase protocol but all **the exclusive mode locks be held until the transaction commits.** That means in other words all the **exclusive locks are unlocked** only after the **transaction is committed.** That also means that if T_1 has exclusive lock, then T_1 will release the exclusive lock only after commit operation, then only other transaction is allowed to read or write. For example - Consider two transactions

T_1	T_2
W(A)	
	R(A)

If we apply the locks then

T_1	T_2
Lock-X(A)	
W(A)	
Commit	
Unlock(A)	
	Lock-S(A)
	R(A)
	Unlock-S(A)

Thus only after commit operation in T_1, we can unlock the exclusive lock. This ensures the strict serializability.

Thus compared to basic two phase locking protocol, the advantage of strict 2PL protocol is it ensures strict serializability.

(2) Rigorous two phase locking : This is stricter two phase locking protocol. Here all locks are to be held until the transaction commits. The transactions can be seriealized in the order in which they commit.

example - Consider transactions

T_1
R(A)
R(B)
W(B)

If we apply the locks then

T_1
Lock-S(A)
R(A)
Lock-X(B)
R(B)
W(B)
Commit
Unlock(A)
Unlock(B)

Thus the above transaction uses rigorous two phase locking mechanism

Example 3.10.2 *Consider the following two transactions :*

T1:read(A)

Read(B);

If A=0 then B=B+1;

Write(B)

T2:read(B); read(A)

If B=0 then A=A+1

Write(A)

Add lock and unlock instructions to transactions T1 and T2, so that they observe two phase locking protocol. Can the execution of these transactions result in deadlock ?

AU : Dec.-16, Marks 6

Solution :

T1	T2
Lock-S(A)	Lock-S(B)
Read(A)	Read(B)
Lock-X(B)	Lock-X(A)
Read(B)	Read(A)
if A=0 then B=B+1	if B=0 then A=A+1
Write(B)	Write(A)
Unlock(A)	Unlock(B)
Commit	Commit
Unlock(B)	Unlock(A)

This is lock-unlock instruction sequence help to satisfy the requirements for strict two phase locking for the given transactions.

The execution of these transactions result in deadlock. Consider following partial execution scenario which leads to deadlock.

T1	T2
Lock-S(A)	Lock-S(B)
Read(A)	Read(B)
Lock-X(B)	Lock-X(A)
Now it will wait for T2 to release exclusive lock on A	Now it will wait for T1 to release exclusive lock on B

3.10.2 Lock Conversion

Lock conversion is a mechanism in two phase locking mechanism - which allows conversion of shared lock to exclusive lock or exclusive lock to shared lock.

Method of Conversion :

First Phase :

- o can acquire a lock-S on item
- o can acquire a lock-X on item
- o can convert a lock-S to a lock-X **(upgrade)**

Second Phase :

- o can release a lock-S
- o can release a lock-X
- o can convert a lock-X to a lock-S **(downgrade)**

This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

For example – Consider following two transactions -

T_1	T_2
R(A)	R(A)
R(B)	R(B)
...	
R(C)	
...	
W(A)	

Here if we start applying locks, then we must apply the **exclusive lock on data item A,** because we have to read as well as write on data item A. Another transaction T2 does not get shared lock on A until transaction T1 performs write operation on A. Since transaction T1 needs exclusive lock only at the end when it performs write operation on A, it is better if T1 could initially lock A in shared mode and then later change it to exclusive mode lock when it performs write operation. In such situation, the lock conversion mechanism becomes useful.

When we convert the shared mode lock to exclusive mode then it is called **upgrading** and when we convert exclusive mode lock to shared mode then it is called **downgrading.**

Also note that upgrading takes place only in **growing phase** and downgrading takes place only in shrinking phase. Thus we can refine above transactions using lock conversion mechanism as follows –

T_1	T_2
Lock-S(A)	
R(A)	
	Lock-S(A)
	R(A)
Lock-S(B)	
R(B)	
	Lock-S(B)
	R(B)
	Unlock(A)
	Unlock(B)
...	
Lock-S(C)	
R(C)	
...	
Upgrade(A)	
W(A)	
Unlock(A)	
Unlock(B)	
Unlock(C)	

Review Questions

1. *What is concurrency control ? Explain two phase locking protocol with an example.* **AU : May-18, Marks 7**
2. *Illustrate two phase locking protocol with an example.* **AU : May-14, Dec.-16, Marks 6**
3. *Discuss elaborately the two phase locking protocol that ensures serializability.* **AU : Dec.-19, Marks 9**

3.11 Timestamp Based Protocol

- The time stamp ordering protocol is a scheme in which the order of transaction is decided in advance based on their timestamps. Thus the schedules are serialized according to their timestamps.
- The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.
- A larger timestamp indicates a more recent transaction or it is also called as **younger transaction** while lesser timestamp indicates **older transaction.**
- Assume a collection of data items that are accessed, with read and write operations, by transactions.
- For each data item X the DBMS maintains the following values :
  - o **RTS(X):** The Timestamp on which object X was last read (by some transaction T_i , i.e., RTS(X):=TS(T_i)) [Note that: RTS stands for Read Time Stamp]
  - o **WTS(X):** The Timestamp on which object X was last written (by some transaction T_j , i.e., WTS(X):=TS(T_j)) [Note that: WTS stands for Write Time Stamp]
- For the following algorithms we use the following assumptions: - A data item X in the database has a **RTS(X) and WTS(X).** These are actually the timestamps of read and write operations performed on data item X at latest time.
- A transaction T attempts to perform some action (read or write) on data item X on some timestamp and we call that timestamp as **TS(T).**
- By timestamp ordering algorithm we need to decide whether transaction T has to be aborted or T can continue execution.

Basic Timestamp Ordering Algorithm

Case 1 (Read) : Transaction T issues a **read(X) operation**

i) If TS(T) < WTS(X), then read(X) is rejected. T has to abort and be rejected.

ii) If WTS(X) ≤ TS(T), then execute read(X) of T and update RTS(X).

Case 2 (Write) : Transaction T issues a **write(X) operation**

i) If TS(T) < RTS(X) or if TS(T) < WTS(X), then write is rejected

ii) If RTS(X) ≤ TS(T) or WTS(X) ≤ TS(T), then execute write(X) of T and update WTS(X).

Example for Case 1 (Read operation)

(i) Suppose we have two transactions T1 and T2 with timestamps 10 sec and 20 sec respectively.

10 Sec T_1	20 Sec T_2
R(X)	
	W(X)
R(X)	

RTS(X) and WTS(X) is initially = 0

Then RTS(X)=10, when transaction T_1 executes

After that WTS(X) =20 when transaction T_2 executes

Now if Read operation R(X) occurs on transaction T_1 at TS(T_1) = 10 then

TS(T_1) i.e. 10 <WTS(X) i.e. 20, hence we have to reject second read operation on T_1 i.e.

10 Sec T_1	20 Sec T_2
R(X)	
	W(X)
R(X)	

This read operation gets rejected as it occurs at older timestamp than the write operation

(ii) Suppose we have two transactions T1 and T2 with timestamps 10 sec and 20 sec respectively.

10 Sec T_1	20 Sec T_2
W(X)	
	R(X)

RTS(X) and WTS(X) is initially = 0

Then WTS(X) =10 as transaction T_1 executes.

Now if Read operation R(X) occurs on transaction T_2 at $TS(T_2) = 20$ then

$TS(T_2)$ i.e. 20 >WTS(X) which is 10, hence we accept read operation on T_2. The transaction T_2 will perform read operation and now RTS will be updated as

RTS(X)=20

Example for Case 2 (Write Operation)

(i) Suppose we have two transactions T_1 and T_2 with timestamps 10 sec and 20 sec respectively.

10 Sec T_1	20 Sec T_2
R(X)	
	W(X)
W(X)	

RTS(X) and WTS(X) is initially = 0

Then RTS(X)=10, when transaction T_1 executes

After that WTS(X) =20 when transaction T_2 executes

Now if Write operation W(X) occurs on transaction T_1 at $TS(T_1) = 10$ then

$TS(T_1)$ i.e. 10 <WTS(X), hence we have to reject second write operation on T_1 i.e.

10 Sec T_1	20 Sec T_2
R(X)	
	W(X)
W(X)	

This write operation gets rejected as it occurs at older timestamp than the write operation at transaction T_2

(ii) Suppose we have two transactions T_1 and T_2 with timestamps 10 sec and 20 sec respectively.

10 Sec T_1	20 Sec T_2
W(X)	
	W(X)

RTS(X) and WTS(X) is initially = 0

Then WTS(X) =10 as transaction T_1 executes.

Now if write operation W(X) occurs on transaction T_2 at $TS(T_2) = 20$ then

$TS(T_2)$ i.e. 20 >WTS(X) which is 10, hence we accept write operation on T_2. The transaction T_2 will perform write operation and now WTS will be updated as

WTS(X) = 20

Advantages and disadvantages of time stamp ordering

Advantages

(1) Schedules are serializable

(2) No waiting for transaction and hence there is no deadlock situation.

Disadvantages

(1) Schedules are not recoverable once transactions occur.

(2) Same transaction may be continuously aborted or restarted.

3.12 Multi-version Concurrency Control

- The DBMS maintains multiple physical versions of single logical object in the database.
- When a transaction writes to an object, the database creates a new version of that object.
- When a transaction reads an object, it reads the newest version that exists when the transaction started.
- In this technique, the writers don't block readers or readers don't block writer.
- This scheme makes use of :

  i) Locking protocol and ii) Timestamp protocol.
- The multiversion is now used in almost all database management system as a modern technique of concurrency control.

3.13 Validation and Snapshot Isolation

3.13.1 Validation Based Protocol

- The optimistic concurrency control algorithm is basically a validation based protocol.
- It works in **three phases** -
  - o **Read phase :**
    - In this phase, transaction T is read.
    - The values of various data items are read and stored in temporary variables.
    - All the operations are then performed on temporary variables without updating the actual database.
  - o **Validation phase :**
    - In this phase, the temporary variable value is validated against the actual data in the database and it checked whether the transaction T follows serializability or not.
  - o **Write phase**
    - If the transaction T is validated then only the temporary results are written to database, otherwise the system rolls back.
- Each phase has following **different timestamps**
  - o **Start (T_i) :**
    - It contains the timestamp when T_i starts the execution.
  - o **Validation (T_i) :**
    - It contains the timestamp when transaction Ti finishes the read phase and starts its validation phase.
  - o **Finish (T_i) :**
    - It contains the timestamp when transaction T_i finishes its write phase.
- With the help of timestamp in validation phase, this protocol determines if the transaction will commit or rollback. Hence
  TS (T_i) = validation (T_i).
- The serializability is determined at the validation process, it can't be determined in advance.
- While executing the transactions, this protocol gives **greater degree of concurrency when there are less number of conflicts.** That is because the serializability order is

not pre-decided (validated and then executed) and relatively less transactions will have to be rolled back.

3.13.2 Snapshot Isolation

- The snapshot isolation is a **multi-version concurrency control technique.**
- In snapshot isolation, we can imagine that **each transaction is given its own version, or snapshot,** of the database when it begins. It reads data from this private version and is thus isolated from the updates made by other transactions.
- If the transaction updates the database, that update appears only in its own version, not in the actual database itself.
- Information about these updates is saved so that the updates can be applied to the "real" database if the transaction commits.
- When a transaction T enters the partially committed state, it then proceeds to the committed state only if no other concurrent transaction has modified data that T intends to update. Transactions that, as a result, cannot commit abort instead.
- Snapshot isolation ensures that attempts to read data never need to wait.

3.14 Multiple Granularity Locking

- In database management system there are **data at various levels** depending upon the **data sizes.**
- For example : A database contains a table and a table contains rows and each row contains some data value. Thus different levels of data can be database, tables, records.

This can be represented in the form of tree. For example.

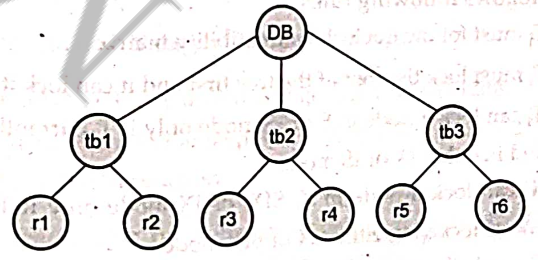


Fig. 3.14.1 : Multiple granularity

- Each node **can be locked** individually. When a transaction locks a node, in either shared or exclusive mode, the transaction also **implicitly locks** all the descendants of that node in the same lock mode.

- **For example** - if transaction T_1 gets an explicit lock on table tb1 in exclusive mode, then it has an implicit lock in exclusive mode on all the records belonging to that table. It does not need to lock the individual records r_1 and r_2.
- A **new mode of lock** is introduced along with exclusive and shared locks. This is called **intention mode lock**. Thus In addition to S and X lock modes, there are three **additional lock modes** with multiple granularity :
  - o **Intention-Shared (IS) :** Explicit locking at a lower level of the tree but only with shared locks.
  - o **Intention-Exclusive (IX) :** Explicit locking at a lower level with exclusive or shared locks.
  - o **Shared and Intention-Exclusive (SIX) :** The sub-tree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive mode locks.
- The **compatibility matrix** for these lock modes are described below :

	IS	IX	S	SIX	X
IS	True	True	True	True	False
IX	True	True	False	False	False
S	True	False	True	False	False
SIX	True	False	False	False	False
X	False	False	False	False	False

- With help of intention lock modes on the transactions the multiple-granularity locking protocol ensures serializability.
- The protocol follows **following rules** -

1) Transaction T_i must follow the lock-compatibility matrix.
2) Transaction T_i must lock the root of the tree first and it can lock it in any mode.
3) Transaction T_i can lock a node in S or IS mode only if T_i currently has the parent of the node locked in either IX or IS mode.
4) Transaction T_i can lock a node in X, SIX, or IX mode only if T_i currently has the parent of the node locked in either IX or SIX mode.
5) Transaction T_i can lock a node only if T_i has not previously unlocked any node. That means T_i is two phase.
6) Transaction T_i can unlock a node only if T_i currently has none of the children of the node locked.

Advantages of multiple granularity protocol

1) It enhances **concurrency**.
2) It ensures **serializability**.
3) It keeps **track of what to lock** and **how to lock.**
4) This type of locking can be hierarchically represented.
5) It makes it easy to decide which data item to be locked and which data item need to be unlocked.

3.15 Isolation Levels

- The consistency of the database is maintained with the help of isolation property(one of the property from ACID properties) of transaction.
- The transaction should take place in a system in such a way that it is the only transaction that is accessing the resources in a database system at particular instance.

As we know, concurrent execution of transaction over shared database creates various problems. Following are three commonly occurring anomalies

(1) Dirty read : This occurs when we read the uncommitted data. We should not read the uncommitted data because it causes many errors in the database.

For example – Let there are two transactions T1 and T2.

Step 1 : Before T1 and T2 perform any operation **Salary = 1000 Rs.**

Step 2 : T2 Writes the salary as **Salary =1200 Rs.**

Step 3 : T1 Reads immediately the **Salary** as **1200 Rs.**

Step 4 : T2 rolls back and now **Salary =1000** and commits.

Step 5 : Now if T1 reads the Salary then it will get the value as 1000

That means the read operation at step 3 is a **dirty read** operation because T1 read the data before T2 commits the transaction

(2) Non repeatable read : This problem occurs when a particular transaction sees two different values for the same row within its lifetime.

For example - Let, there are two transactions T1 and T2

Step 1 : At time t1, the transaction T1 reads the **Salary = 1000 Rs.**

Step 2 : At time t2 the transaction T2 reads the same salary = 1000 Rs and updates it to 1200 Rs.

Step 3 : Then at time t3, the transaction T2 gets committed.

Step 4 : Now when the transaction T1 reads the same salary at time t4, it gets different value(as **Rs.1200**) than what it had read(Rs.1000) at time t1. Now, transaction T1 cannot repeat its reading operation.

Thus inconsistent values are obtained.

(3) Phantom read : It is a special case of non repeatable read. This is a problem in which one of the transaction makes the changes in the database system and due to these changes another transaction can not read the data item which it has read just recently.

For example - Let, there are two transactions T1 and T2

Step 1 : At time t1, the transaction T1 reads the value of **Salary = 1000 Rs.**

Step 2 : At time t2, the transaction T2 reads the value of the same salary as **1000 Rs.**

Step 3 : At time t3, the transaction T1 **deletes the variable salary.**

Step 4 : Now at time t4, when T2 again reads the salary it **gets error.** Now transaction T2 cannot identify the reason why it is not getting the salary value which is read just few time back.

Hence isolation levels are defined so that any two transactions can execute concurrently and integrity of data can be maintained.

- There are four levels of transaction isolation defined by SQL -
  - o **Serializable :**
    - This is the **Highest isolation level.**
    - **Serializable execution is defined** to be an execution of operations in which concurrently executing transactions **appears to be serially executing.**
  - o **Repeatable Read :**
    - This is the **most restrictive isolation level.**
    - The transaction **holds read locks on all rows** it references.
    - It holds **write locks on all rows** it inserts, updates, or deletes.
    - Since other transaction cannot read, update or delete these rows, it avoids non repeatable read.

- o **Read Committed :**
  - This isolation level allows only **committed data to be read.**
  - Thus it does **not allows dirty read** (i.e. one transaction reading of data immediately after written by another transaction).
  - The transaction hold a **read or write lock** on the current row, and thus prevent other rows from reading, updating or deleting it.
- o **Read Uncommitted :**
  - It is **lowest isolation level.**
  - In this level, one transaction may read not yet committed changes made by other transaction.
  - This level **allows dirty reads.**

In this level **transactions are not isolated** from each other.

To summerize, the above isolation levels -

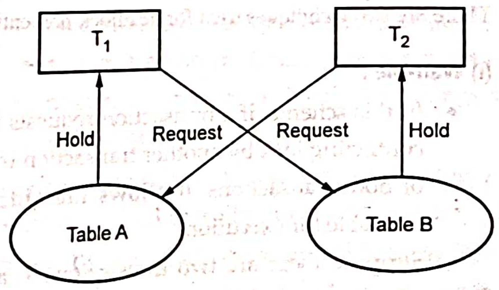
- SERIALIZABLE - dirty reads, non-repeatable reads and phantoms not allowed; all schedules must be serializable
- REPEATABLE READ - dirty reads, non-repeatable reads not allowed, but phantoms allowed
- READ COMMITTED - dirty reads not allowed, but non-repeatable reads and phantoms allowed
- READ UNCOMMITTED - dirty reads, non-repeatable reads and phantoms allowed

3.16 Deadlock Handling

AU : May-09, Dec.-14,15,16,19, Marks 16

Deadlock is a specific concurrency problem in which two transactions depend on each other for something.

For example – Consider that transaction T1 holds a lock on some rows of table **A** and needs to update some rows in the **B** table. Simultaneously, transaction T2 holds locks on some rows in the **B** table and needs to update the rows in the **A** table held by Transaction T1.



Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. This situation is called **deadlock** in DBMS.

Definition : Deadlock can be formally defined as - " A system is in deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set. "

There are four conditions for a deadlock to occur

A deadlock may occur if all the following conditions holds true.

1. **Mutual exclusion condition :** There must be at least one resource that cannot be used by more than one process at a time.
2. **Hold and wait condition :** A process that is holding a resource can request for additional resources that are being held by other processes in the system.
3. **No preemption condition :** A resource cannot be forcibly taken from a process. Only the process can release a resource that is being held by it.
4. **Circular wait condition :** A condition where one process is waiting for a resource that is being held by second process and second process is waiting for third process ....so on and the last process is waiting for the first process. Thus making a circular chain of waiting.

Deadlock can be handled using two techniques –

1. Deadlock Prevention
2. Deadlock Detection and deadlock recovery

1. Deadlock prevention :

For large database, deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that deadlock never occur. The DBMS analyzes the operations whether they can create deadlock situation or not, If they do, that transaction is never allowed to be executed.

There are two techniques used for deadlock prevention –

(i) Wait-Die :

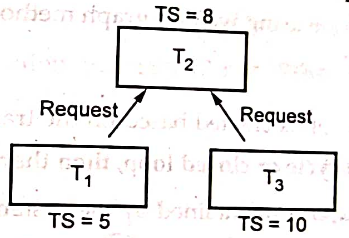
- In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It **allows the older transaction to wait** until the resource is available for execution.
- Suppose there are two transactions T_i and T_j and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS :

- o Check if $TS(T_i) < TS(T_j)$ - If T_i is the older transaction and T_j has held some resource, then T_i is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
- o Check if $TS(T_i) < TS(T_j)$ - If T_i is older transaction and has held some resource and if T_j is waiting for it, then T_j is killed and restarted later with the random delay but with the same timestamp.

Timestamp is a way of assigning priorities to each transaction when it starts. If timestamp is lower then that transaction has higher priority. **That means oldest transaction has highest priority.**

For example -

Let T1 is a transaction which requests the data item acquired by Transaction T2. Similarly T3 is a transaction which requests the data item acquired by transaction T2.



Here TS(T1) i.e. Time stamp of T1 is less than TS(T3). In other words T1 is older than T3. Hence T1 is made to **wait** while T3 is **rolledback.**

(ii) Wound - wait :

- o In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After some delay, the younger transaction is restarted but with the same timestamp.
- o If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until older releases it.

Suppose T1 needs a resource held by T2 and T3 also needs the resource held by T2, with TS(T1)=5, TS(T2)=8 and TS(T3)=10, then T1 being older waits and T3 being younger dies. After the some delay, the younger transaction is restarted but with the same timestamp.

This ultimately prevents a deadlock to occur.

To summarize

	Wait-Die	Wound-wait
Older transaction needs a data item held by younger transaction	older transaction waits	younger transaction dies.
Younger transaction needs a data item held by older transaction	Younger transaction dies	Younger transaction dies.

2. Deadlock detection :

- In deadlock detection mechanism, an algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred or not. If deadlock is occurrence is detected, then the system must try to recover from it.
- Deadlock detection is done using wait for graph method.

Wait for graph

- o In this method, a graph is created based on the transaction and their lock. If the created **graph has a cycle or closed loop, then there is a deadlock.**
- o The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.
- o This graph consists of a pair G = (V, E), where V is a set of vertices and E is a set of edges.
- o The set of vertices consists of all the transactions in the system.
- o When transaction T_i requests a data item currently being held by transaction T_j, then the edge $T_i \rightarrow T_j$ is inserted in the wait-for graph. This edge is removed only when transaction T_j is no longer holding a data item needed by transaction T_i.

For example - Consider following transactions, We will draw a wait for graph for this scenario and check for deadlock.

T1	T2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	

We will use three rules for designing the wait-for graph -

Rule 1 : If T1 has **Read** operation and then T2 has **Write** operation then draw an edge T1->T2.

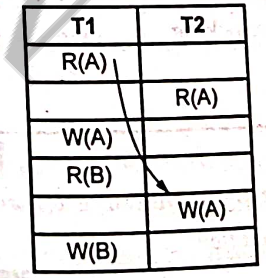
Rule 2 : If T1 has **Write** operation and then T2 has **Read** operation then draw an edge T1->T2

Rule 3 : If T1 has **Write** operation and then T2 has **Write** operation then draw an edge T1->T2

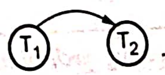
Let us draw wait-for graph

Step 1 : Draw vertices for all the transactions.

Step 2 : We find the Read-Write pair from two different transactions reading from top to bottom. If such as pair is found then we will add the edges between corresponding directions. For instance -

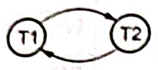


T1	T2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	



Step 3 :

T1	T2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	



As **cycle is detected** in the wait-for graph there is no need to further process. The **deadlock** is present in this transaction scenario.

Example 3.16.1 *Give an example of a scenario where two phase locking leads to deadlock.* **AU : May-09, Marks 4**

Solution : Following scenario of execution of transactions can result in deadlock.

T_1	T_2
Lock – S (A)	
	Lock – S (B)
	Read (B)
Read (A)	
Lock – X (B)	
	Lock – X (A)

These two instructions cause a deadlock situation.

In above scenario, transaction T_1 makes an exclusive lock on data item B and then transaction T_2 makes an exclusive lock on data item A. Here unless and until T_1 does not give up the lock (i.e. unlock) on B; T_2 cannot read / write it. Similarly unless and until T_2 does not give up the lock on A; T_1 cannot read or write on A.

This is a purely deadlock situation in two phase locking.

Review Questions

1. *Outline deadlock handling mechanisms.* **AU : Dec-16, Marks 7**
2. *What is deadlock ? How does it occur ? How transactions can be written to*
   (i) Avoid deadlock. (ii) Guarantee correct execution.
   Illustrate with suitable example. **AU : Dec.-15, Marks 16**
3. *Write short note on- Deadlock* **AU : Dec.-14, Marks 4**
4. *Narrate the actions that are considered for deadlock detection and the recovery from deadlock.* **AU : Dec 19, Marks 9**

Part III : Recovery

3.17 Recovery Concepts

- An integral part of a database system is a recovery scheme that can **restore the database** to the consistent state that existed before the failure.
- The recovery scheme must also provide **high availability;** that means, it must minimize the time for which the database is not usable after a failure.

3.17.1 Failure Classification

Various types of failures are -

1. **Transaction Failure :** Following are two types of errors due to which the transaction gets failed.
   - **Logical Error :**
     i) This error is caused due to internal conditions such as bad input, data not found, overflow of resource limit and so on.
     ii) Due to logical error the transaction can not be continued.
   - **System Error :**
     i) When the system enters in an undesired state and then the transaction can not be continued then this type of error is called as system error.
2. **System Crash :** The situation in which there is a hardware malfunction, or a bug in the database software or the operating system, and because of which there is a loss of the content of volatile storage, and finally the transaction processing come to a halt is called system crash.
3. **Disk Failure :** A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. The backup of data is maintained on the secondary disks or DVD to recover from such failure.

3.17.2 Storage

A DBMS stores the data on external storage because the amount of data is very huge and must persist across program executions.

The storage structure is a memory structure in the system. It has following categories -

1) Volatile :

- Volatile memory is a primary memory in the system and is placed along with the CPU.

- These memories can store only small amount of data, but they are very fast. For example - main memory, cache memory.
- A volatile storage cannot survive system crashes.
- That means data in these memories will be lost on failure.

2) Non Volatile :

Non volatile memory is a secondary memory and is huge in size. For example : Hard disk, Flash memory, magnetic tapes.

These memories are designed to withstand system crashes.

3) Stable :

- Information residing in stable storage is never lost.
- To implement stable storage, we replicate the information in several nonvolatile storage media (usually disk) with independent failure modes.

Stable Storage Implementation

- Stable storage is a kind of storage on which the information residing on it is never lost.
- Although stable storage is theoretically impossible to obtain it can be approximately built by applying a technique in which **data loss is almost impossible.**
- That means the information is **replicated** in several nonvolatile storage media with **independent failure modes.**
- Updates must be done with care to ensure that a failure during an update to stable storage **does** not cause a loss of information.

3.17.3 Recovery with Concurrent Transactions

There are four ways for recovery with concurrent transactions :

1) **Interaction with concurrency control :** In this scheme recovery depends upon the concurrency control scheme which is used for the transaction. If a transaction gets failed, then rollback and undo all the updates performed by transaction.

2) **Transaction Rollback :** In this scheme, if the transaction gets failed, then the failed transaction can be rolled back with the help of **log**. The system scans the log backward, and with the help of log entries the system can restore the data items.

3) **Checkpoints :** The checkpoints are used to reduce the number of log records.

4) **Restart recovery :** When the system recovers from the crash it constructs two lists : Undo-list and Redo-list. The undo list consists of transactions to be undone. The redo list consists of transactions to be redone. These two lists are handled as follows

Step 1 : Initially both the lists are empty.

Step 2 : The system scans the log entries from backwards. It scans each entry until it finds **first checkpoint record.**

3.17.4 Shadow Copy Technique

- In the **shadow-copy scheme,** a transaction that wants to update the database first creates a complete **copy of the database.**
- All updates are done on the new database copy, leaving the original copy, the shadow copy, untouched.
- If at any point the transaction has to be aborted, the system merely deletes the new copy. The old copy of the database has not been affected.
- The current copy of the database is identified by a pointer, called **db-pointer,** which is stored on disk.
- If the transaction **partially commits,** it is committed as follows :
  - o First, the operating system is asked to make sure that all pages of the new copy of the database have been written out to disk.
  - o After the operating system has written all the pages to disk, the database system updates the pointer **db-pointer** to point to the new copy of the database
  - o The new copy then becomes the current copy of the database.
  - o The old copy of the database is then deleted.
    - ▪ The transaction is said to have been **committed** at the point where the **updated db-pointer** is written to disk.
    - ▪ The disk system guarantees that it will update **db-pointer** atomically, as long as we make sure that **db-pointer** lies entirely in a single sector.

Where is shadow copy technique used ?

- o Shadow copy schemes are commonly used by text editors.
- o Shadow copying can be used for small databases.

3.18 Recovery Techniques

To ensure atomicity despite failures, we first output information describing the modifications to stable storage without modifying the database itself.

There are two approaches are used for recovery -

1) Log based Recovery

2) Shadow paging

3.19 Recovery based on Deferred and Immediate Update AU : May-19, Marks 15

Before discussing the recovery algorithms(deferred and immediate update), let us see the concept of Log and REDO and UNDO operations.

3.19.1 Concept of Log

- Log is the most commonly used structure for recording the modifications that is to be made in the actual database. Hence during the recovery procedure a log file is maintained.
- A log record maintains four types of operations. Depending upon the type of operations there are four types of log records -
  1. **<Start>** Log record : It is represented as <T_i, Start>
  2. **<Update>** Log record
  3. **<Commit>** Log record : It is represented as <T_i, Commit>
  4. **<Abort>** Log record : It is represented as <T_i, Abort>
- The log contains various fields as shown in following Fig. 3.19.1. This structure is for <update> operation

Transaction ID(T_i)	Data Item Name	Old Value of Data Item	New Value of Data Item

Fig. 3.19.1

- For example : The sample log file is

<T_1, Start>
< T_1,a,10,20>
<T_1, Commit>

Here 10 represents the old value before commit operation and 20 is the new value that needs to be updated in the database after commit operation.

- The log must be maintained on the stable storage and the entries in the log file are maintained before actually updating the physical database.
- There are two approaches used for log based recovery technique - Deferred Database Modification and Immediate Database Modification.

3.19.2 REDO and UNDO Operation

During transaction execution, the updates are recorded only in the log and in the cache buffers. After the transaction reaches its commit point and the log is force written to disk and the updates are recorded in the database.

In order to maintain the atomicity of transaction, the operations can be redone or undone.

UNDO : This is an operation in which we restore all the old values (BFIM - BeFore Modification Image) onto the disk. This is called **roll-back operation.**

REDO : This is an operation in which all the modified values(AFIM - AFter Modification Image) are restored onto the disk. This is called **roll-forward operation.**

These operations are recorded in the log as they happen.

Difference between UNDO and REDO

Sr. No.	UNDO	REDO
1.	Makes a change go away.	Reproduces a change.
2.	Used for rollback and read consistency.	Used for rolling forward the changes.
3.	Protects the database from inconsistent reads.	Protects from data loss.

3.19.3 Write Ahead Logging Rule

- Before a block of data in main memory is output to the database, all log records pertaining to data in that block must have been output to stable storage. This rule is called the **Write-Ahead Logging (WAL)**
- This rule is necessary because - In the event of a crash or ROLLBACK, the original content contained in the rollback journal is played back into the database file to revert the database file to its original state.

3.19.4 Deferred Database Modification

- In this technique, the database is not updated immediately.
- Only log file is updated on each transaction.
- When the transaction reaches to its commit point, then only the database is physically updated from the log file.
- In this technique, if a transaction fails before reaching to its commit point, it will not have changed database anyway. Hence there is no need for the UNDO operation. The REDO operation is required to record the operations from log file to physical database. Hence deferred database modification technique is also called as NO UNDO / REDO algorithm.
- **For example :**

Consider two transactions T_1 and T_2 as follows :

T_1	T_2
Read (A, a)	Read (C, c)
a = a – 10	c = c – 20
Write (A, a)	Write (C, c)
Read (B, b)	
b = b + 10	
Write (B, b)	

If T_1 and T_2 are executed serially with initial values of A = 100, B = 200 and C = 300, then the state of log and database if crash occurs

a) Just after write (B, b)

b) Just after write (C, c)

c) Just after <T_2, commit>

The result of above 3 scenarios is as follows :

Initially the log and database will be

Log	Database
<T_1, Start>	
<T_1, A, 90>	
<T_1, B, 210>	
<T_1, Commit>	
	A = 90
	B = 210
<T_2, Start>	
<T_2, C, 280>	
<T_2, Commit>	
	C = 280

a) Just after write (B, b)

Just after write operation, no commit record appears in log. Hence no write operation is performed on database. So database retains only old values. Hence A = 100 and B = 200 respectively.

Thus the system comes back to original position and no redo operation take place. The incomplete transaction of T_1 can be deleted from log.

b) Just after write (C, c)

The state of log records is as follows

Note that crash occurs before T_2 commits. At this point T_1 is completed successfully, so new values of A and B are written from log to database. But as T_2 is not committed, there is no redo (T_2) and the incomplete transaction T_2 can be deleted from log.

The redo (T_1) is done as < T_1, commit> gets executed. Therefore A = 90, B = 210 and C = 300 are the values for database.

c) **Just after < T_2, commit>**

The log records are as follows :

< T_1, Start>

< T_1, A, 90>

< T_1, B, 210>

< T_1, Commit>

< T_2, Start>

< T_2, 6, 280>

< T_2, Commit>

← **Crash occurs here**

Clearly both T_1 and T_2 reached at commit point and then crash occurs. So both redo (T_1) and redo (T_2) are done and updated values will be A = 90, B = 210, C = 280.

3.19.5 Immediate Database Modification

In this technique, the database is updated during the execution of transaction even before it reaches to its commit point.

If the transaction gets failed before it reaches to its commit point, then the a **ROLLBACK Operation** needs to be done to bring the database to its earlier consistent state. That means the effect of operations need to be undone on the database. For that purpose both Redo and Undo operations are both required during the recovery. This technique is known as **UNDO / REDO** technique.

For example : Consider two transaction T_1 and T_2 as follows :

T_1	T_2
Read(A,a)	Read(C, c)
a = a –10	c = c – 20
Write(A, a)	Write(C, c)
Read(B, b)	
b = b+10	
Write(B, b)	

Here T_1 and T_2 are executed serially. Initially A = 100, B = 200 and C = 300

If the crash occurs after

i) Just after Write(B, b) **ii) Just after Write(C, c)** **iii) Just after <T_2,Commit>**

Then using the immediate Database modification approach the result of above three scenarios can be elaborated as follows :

The contents of **log** and **database** is as follows :

Log	Database
<T_1,Start>	
<T_1,A,100,90>	A = 90
<T_1,B,200,210>	B = 210
<T_1,Commit>	
<T_2,Start>	
<T_2,C,300,280>	C = 280
<T_2,Commit>	

The recovery scheme uses two recovery techniques -

i) **UNDO (T_i) :** The transaction T_i needs to be undone if the log contains <T_i,Start> but does not contain <T_i,Commit>. In this phase, it restores the values of all data items updated by T_i to the old values.

ii) **REDO (T_i) :** The transaction T_i needs to be redone if the log contains both <T_i,Start> and <T_i,Commit>. In this phase, the data item values are set to the new values as per the transaction. After a failure has occurred log record is consulted to determine which transaction need to be redone.

a) **Just after Write (B, b) :** When system comes back from this crash, it sees that there is <T_1, Start> but no <T_1, Commit>. Hence T_1 must be undone. That means old values of A and B are restored. Thus old values of A and B are taken from log and both the transaction T_1 and T_2 are re-executed.

b) **Just after Write (C, c) :** Here both the redo and undo operations will occur.

c) **Undo :** When system comes back from this crash, it sees that there is <T_2, Start> but no <T_2, Commit>. Hence T_2 must be undone. That means old values of C is restored.

Thus old value of C is taken from log and the transaction T_2 is re-executed.

d) **Redo :** The transaction T_1 must be done as log contains both the <T_1,Start> and <T_1,Commit>

So A = 90, B = 210 and C = 300

e) **Just after <T_2, Commit> :** When the system comes back from this crash, it sees that there are two transaction T_1 and T_2 with both start and commit points. That means T_1 and T_2 need to be redone. So A = 90, B = 210 and C = 280

Example 3.19.1 *Suppose there is a database system that never fails. Is a recovery manager require for this system ? Why ?*

Solution :

1) Yes. Even-though the database system never fails, the recovery manager is required for this system.

2) During the transaction processing some transactions might be aborted. Such transactions must be rolled back and then the schedule is continued further.

3) Thus to perform the rollbacks of aborted transactions recovery manager is required.

Review Question

1. *Explain deferred and immediate modification versions of the log based recovery scheme.* **AU : May- 19, Marks 15**

3.20 Shadow Paging

- Shadow paging is a recovery scheme in which database is considered to be made up of number of fixed size disk pages.
- A directory or a page table is constructed with n number of pages where each ith page points to the ith database page on the disk.

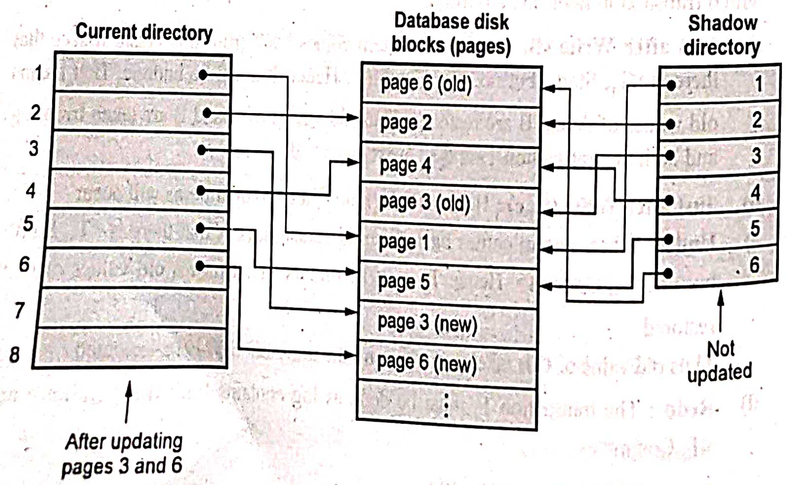


Fig. 3.20.1 : Demonstration of shadow paging

- The directory can be kept in the main memory.
- When a transaction begins executing, the **current directory**-whose entries point to the most recent or current database pages on disk-is copied into a another directory called **shadow directory.**
- The shadow directory is then saved on **disk** while the current directory is used by the transaction.
- During the execution of transaction, the shadow directory is never modified.
- When a write operation is to be performed then the **new copy** of modified database page is created but the old copy of database page is never overwritten. This newly created database page is written somewhere else.
- The current directory will point to newly modified web page and the shadow page directory will point to the old web page entries of database disk.
- When the failure occurs then the modified database pages and current directory is discarded.
- The state of database before the failure occurs is now available through the shadow directory and this state can be recovered using shadow directory pages.
- This technique does not require any UNDO/REDO operation.

3.21 ARIES Algorithm

- ARIES is a recovery algorithm.
- It stands for Algorithm for Recovery and Isolation Exploiting Semantics.
- It is based on Write Ahead Log (WAL) protocol.
- When database crashes during some transaction processing, we have the logs that got saved to the disk.
- ARIES has 3 phases that occur in the following order -

1) Analysis :

- Scan the log from start to reconstruct the transaction and dirty page table. Dirty pages contain data that has been changed but not yet written to disk.
- The active transactions which were present at the time of crash are identified.
- During analysis phase the log is scanned forward from the checkpoint record to construct snapshot of what system looks like at the time of crash.

2) Redo :

- This phase is started only after completion of analysis phase.
- The log is read forward and each update is redone.

3) Undo :

- This phase is started after redo phase.
- The log is scanned backward and updates to corresponding active transactions are undone.

Advantages :

1) It is simple and flexible.
2) It supports concurrency control protocol.
3) Independent recovery of every page.

3.22 Two Marks Questions with Answers

Q.1 What is a transaction ? AU : May-04, Dec.05

Ans. : A transaction can be defined as a **group of tasks** that form a single logical unit.

Q.2 What does time to commit mean ? AU : May-04

Ans. :

- The COMMIT command is used to save permanently any transaction to database.
- When we perform, Read or Write operations to the database then those changes can be undone by rollback operations. To make these changes permanent, we should make use of **commit**

Q.3 What are the various properties of transaction that the database system maintains to ensure integrity of data. AU : Dec.-04

OR

Q.4 What are ACID properties ? AU : May-05,06,08,13,15,Dec.-07,14,17

Ans. : In a database, each transaction should maintain ACID property to meet the consistency and integrity of the database. These are

(1) Atomicity (2) Consistency (3) Isolation (4) Durability

Q.5 Give the meaning of the expression ACID transaction. AU : Dec.-08

Ans. : The expression ACID transaction represents the transaction that follows the ACID Properties.

Q.6 State the atomicity property of a transaction. AU : May-09,13

Ans. : This property states that each transaction must be considered as a **single unit and must be completed fully or not completed at all.**

No transaction in the database is left half completed.

Q.7 What is meant by concurrency control ? AU : Dec.-15

Ans. : A mechanism which ensures that simultaneous execution of more than one transactions does not lead to any database inconsistencies is called **concurrency control mechanism.**

Q.8 State the need for concurrency control. AU : Dec.-17

OR

Q.9 Why is it necessary to have control of concurrent execution of transactions ? How is it made possible ? AU : Dec.-02

Ans. : Following are the purposes of concurrency control –

- o To ensure isolation :
- o To resolve read-write or write-write conflicts :
- o To preserve consistency of database :

Q.10 List commonly used concurrency control techniques. AU : Dec.-11

Ans. : The commonly used concurrency control techniques are –

i) Lock ii) Timestamp

iii) Snapshot Isolation

Q.11 What is meant by serializability? How it is tested? AU : May-14,18, Dec.-14,16

Ans. : Serializability is a concept that helps to identify which non serial schedule and find the transaction equivalent to serial schedule.

It is tested using precedence graph technique.

Q.12 What is serializable schedule ? AU : May-17

Ans. : The schedule in which the transactions execute one after the other is called serial schedule. It is consistent in nature. For example : Consider following two transactions T_1 and T_2

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

All the operations of transaction T_1 on data items A and then B executes and then in transaction T_2 all the operations on data items A and B execute. The R stands for Read operation and W stands for write operation.

Q.13 When are two schedules conflict equivalent ? **AU : Dec.-08**

Ans. : Two schedules are conflict equivalent if :

- They contain the same set of the transaction.
- every pair of conflicting actions is ordered the same way.

For example –

Non Serial Schedule

T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Serial Schedule

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2. Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.

Hence both of the above the schedules are conflict equivalent.

Q.14 Define two phase locking. **AU : May-13**

Ans. : The two phase locking is a protocol in which there are two phases :

i) **Growing Phase (Locking Phase) :** It is a phase in which the transaction may obtain locks but does not release any lock.

ii) **Shrinking Phase (Unlocking Phase) :** It is a phase in which the transaction may release the locks but does not obtain any new lock.

Q.15 What is the difference between shared lock and exclusive lock? **AU : May-18**

Ans. :

Shared Lock	Exclusive Lock
Shared lock is used for when the transaction wants to perform read operation.	Exclusive lock is used when the transaction wants to perform both read and write operation.
Multiple shared lock can be set on a transactions simultaneously.	Only one exclusive lock can be placed on a data item at a time.
Using shared lock data item can be viewed.	Using exclusive lock data can be inserted or deleted.

Q.16 What type of lock is needed for insert and delete operations. **AU : May-17**

Ans. : The exclusive lock is needed to insert and delete operations.

Q.17 What benefit does strict two-phase locking provide ? What disadvantages result ? **AU : May-06, 07, Dec.-07**

Ans. : **Benefits :**

1. This ensure that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits and preventing other transaction from reading that data.
2. This protocol solves dirty read problem.

Disadvantage :

1. Concurrency is reduced.

Q.18 What is rigorous two phase locking protocol ? **AU : Dec.-13**

Ans. : This is stricter two phase locking protocol. Here all locks are to be held until the transaction commits.

Q.19 Differentiate strict two phase locking and rigourous two phase locking protocol. **AU : May-16**

Ans. :

- In Strict two phase locking protocol all the exclusive mode locks be held until the transaction commits.
- The rigourous two phase locking protocol is stricter than strict two phase locking protocol. Here all locks are to be held until the transaction commits.

Q.20 Define deadlock. AU : May-08,09,14

Ans. : Deadlock is a situation in which when two or more transactions have got a lock and waiting for another locks currently held by one of the other transactions.

Q.21 List four conditions for deadlock. AU : Dec.-16

Ans. : 1. Mutual exclusion condition 2. Hold and wait condition

3. No preemption condition 4. Circular wait condition

Q.22 Why is recovery needed ? AU : May-09

Ans. :

- A recovery scheme that can **restore the database** to the consistent state that existed before the failure.
- Due to recovery mechanism, there is high availability of database to its users.

Q.23 What are states of transaction ? AU : May-19

Ans. : Various states of transaction are - (1) Active, (2) Partially Committed (3) Failed (4) Aborted (5) Committed.

Q.24 What is meant by log based recovery ?

Ans. : Log is a most commonly used data structure for recording the modifications that can be made to actual database.

Log based recovery is a technique in which a log of each transaction is maintained in some stable storage so that if failure occurs then it can be recovered from there.

Q.25 List the responsibilities of a DBMS has whenever a transaction is submitted to the system for execution. AU : Dec.-19

Ans. : The system is responsible for making sure that - (1) Either all the operations in the transaction are completed successfully and effect is recorded permanently in the database. (2) The transaction, has no effect whatsoever on the database or on the database or on any other transaction.

Q.26 Brief any two violations that may occur if a transaction executes a lower isolation level than serializable. AU : Dec.-19

Ans. : (1) For non-repeatable Read the phantom read is allowed.

(2) For read committed non-repeatable reads and phantom reads are allowed.

UNIT IV

4 Implementation Techniques

Syllabus

RAID - File Organization - Organization of Records in Files - Data dictionary Storage - Column Oriented Storage - Indexing and Hashing - Ordered Indices - B+ tree Index Files - B tree Index Files - Static Hashing - Dynamic Hashing - Query Processing Overview - Algorithms for Selection, Sorting and join operations - Query optimization using Heuristics - Cost Estimation.

Contents

Part I : File Organization and Indexing

4.1 RAID

AU : Dec.-06,13,14,16,19, May-15,16,17,19, Marks 16

- RAID stands for Redundant Array of Independent Disks. This is a technology in which multiple secondary disks are connected together to increase the performance, data redundancy or both.
- For achieving the data redundancy - in case of disk failure, if the same data is also backed up onto another disk, we can retrieve the data and go on with the operation.
- It consists of an array of disks in which multiple disks are connected to achieve different goals.
- The **main advantage** of RAID, is the fact that, to the operating system the array of disks can be presented as a single disk.

Need for RAID

- o RAID is a technology that is used to **increase the performance.**
- o It is used for **increased reliability** of data storage.
- o An array of multiple disks accessed in parallel will give greater throughput than a single disk.
- o With multiple disks and a suitable redundancy scheme, your system can stay up and running when a disk fails, and even while the replacement disk is being installed and its **data restored.**

Features

(1) RAID is a technology that contains the set of physical disk drives.

(2) In this technology, the operating system views the separate disks as a single logical disk.

(3) The data is distributed across the physical drives of the array.

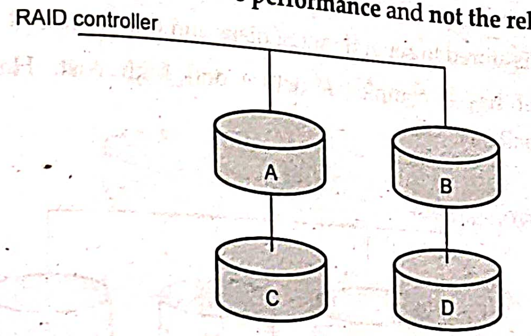
(4) In case of disk failure, the parity information can be helped to recover the data.

4.1.1 RAID Levels

Level : RAID 0

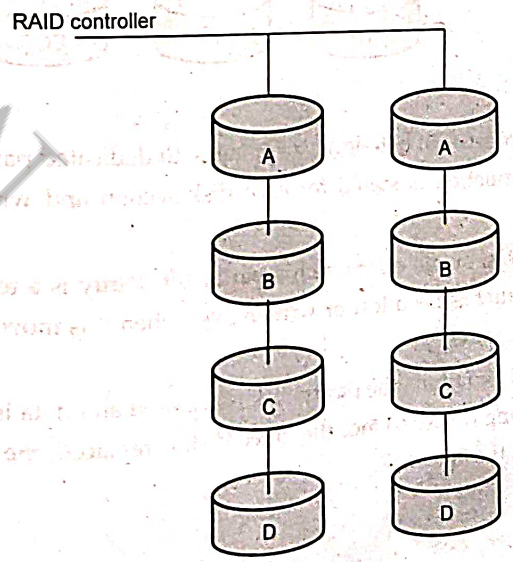
- o In this level, data is broken down into blocks and these blocks are stored across all the disks.
- o Thus **striped array of disks** is implemented in this level. For instance in the following figure blocks "A B" form a stripe.

- o There is no duplication of data in this level so once a block is lost then there is no way recover it.
- o The main **priority of this level is performance** and **not the reliability.**



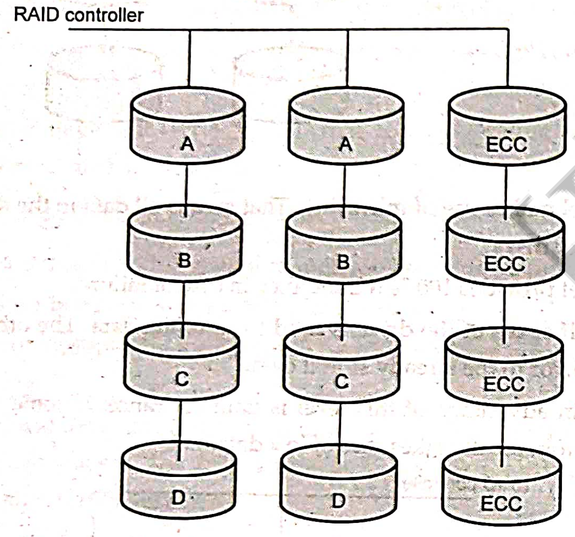
Level : RAID 1

- o This level makes use of **mirroring.** That means all data in the drive is duplicated to another drive.
- o This level provides 100 % redundancy in case of failure.
- o Only half space of the drive is used to store the data. The other half of drive is just a mirror to the already stored data.
- o The main advantage of this level is fault tolerance. If some disk fails then the other automatically takes care of lost data.



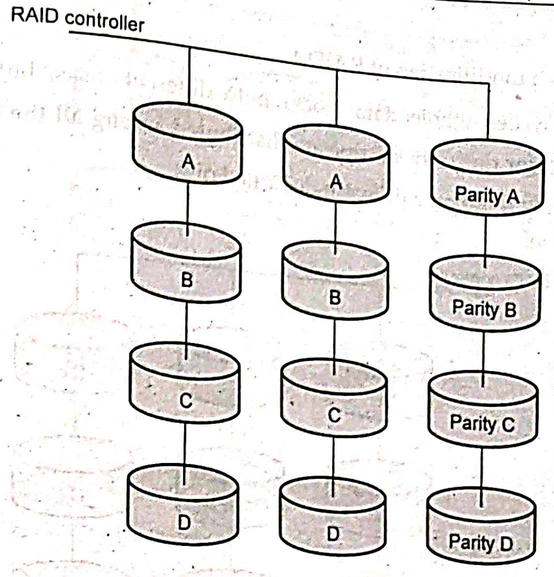
Level : RAID 2

- This level makes use of **mirroring as well as stores Error Correcting Codes (ECC)** for its data striped on different disks.
- The data is stored in separate set of disks and ECC is stored another set of disks.
- This level has a complex structure and high cost. Hence it is not used commercially.



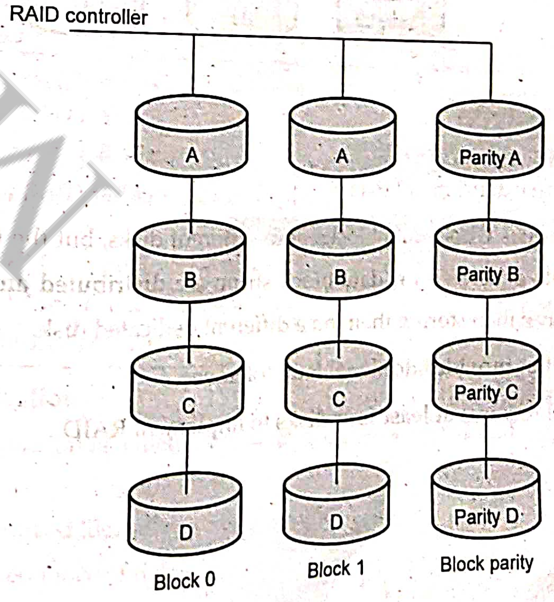
Level : RAID 3

- This level consists of byte-level stripping with dedicated parity. In this level, the parity information is stored for each disk section and written to a dedicated parity drive.
- We can detect single errors with a **parity bit.** Parity is a technique that checks whether data has been lost or written over when it is moved from one place in storage to another.
- In case of disk failure, the parity disk is accessed and data is reconstructed from the remaining devices. Once the failed disk is replaced, the missing data can be restored on the new disk.



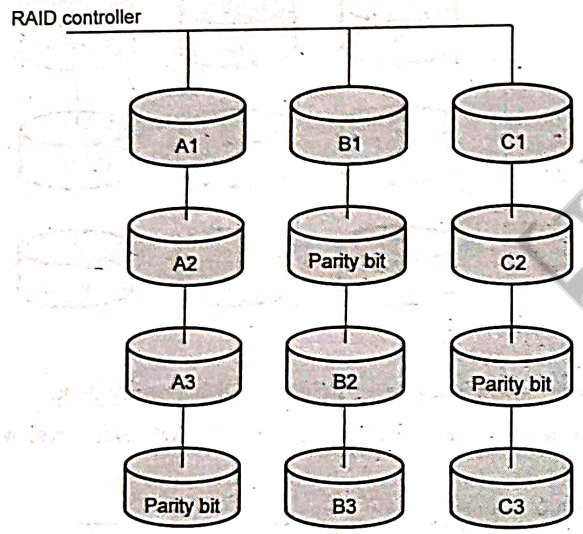
Level : RAID 4

- RAID 4 consists of block-level stripping with a parity disk.
- Note that level 3 uses byte-level striping, whereas level 4 uses block-level striping.



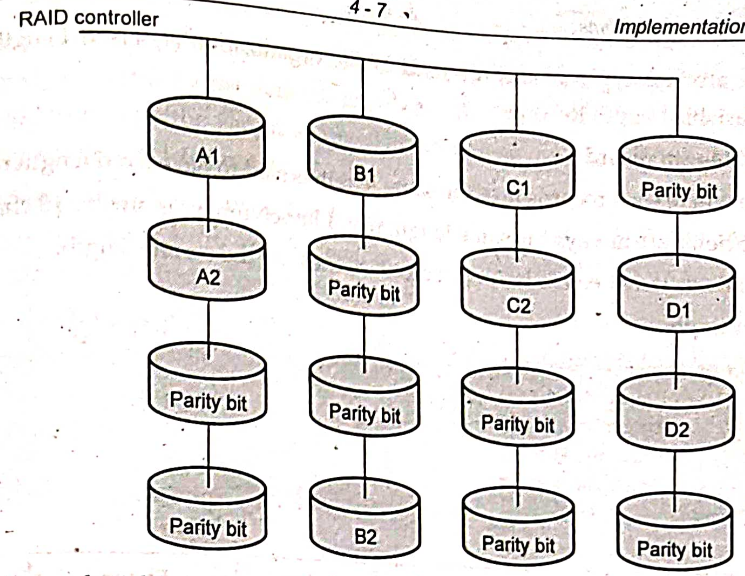
Level : RAID 5

- o RAID 5 is a modification of RAID 4.
- o RAID 5 writes whole data blocks onto different disks, but the **parity bits** generated for data block stripe are **distributed among all the data disks** rather than storing them on a different dedicated disk.



RAID : Level 6

- o RAID 6 is a extension of Level 5
- o RAID 6 writes whole data blocks onto different disks, but the **two independent parity bits** generated for data block stripe are **distributed among all the data disks** rather than storing them on a different dedicated disk.
- o Two parities provide additional fault tolerance.
- o This level requires **at least four disks** to implement RAID.



The factors to be taken into account in choosing a RAID level are :

Monetary cost of extra disk-storage requirements.

1. Performance requirements in terms of number of I/O operations.
2. Performance when a disk has failed.
3. Performance during rebuild

Review Questions

1. *Explain what a RAID system is ? How does it improve performance and reliability. Discuss the level 3 and level 4 of RAID.* **AU : May-17, Marks (3+4+6)**
2. *Explain the concept of RAID.* **AU : Dec.-16, Marks 6, Dec.-14, Marks 8**
3. *Briefly explain RAID and RAID levels.* **AU : Dec.-06, Marks 10, Dec.-13, May-15, 16, Marks 16**
4. *What is RAID ? List the different levels in RAID technology and explain its features.*
5. *What is RAID ? Briefly discuss about RAID.* **AU : May-19, Marks 13**
6. *Explain various levels of RAID Systems.* **AU : Dec.-19, Marks 10**

4.2 File Organization

AU : Dec.-08, Marks 10

- A file organization is a method of **arranging records** in a file when the file is stored on disk.
- A file is organized logically as a sequence of records.
- Record is a sequence of fields.

- There are two types of records used in file organization (1) Fixed Length Record (2) Variable Length Record.

(1) Fixed length record

- A file where each records is of the same length is said to have fixed length records.
- Some fields are always the same length (e.g. PhoneNumber is always 10 characters).
- Some fields may need to be 'padded out' so they are the correct length.
- For example -

```
type Employee = record
        EmpNo varchar(4)
        Ename varchar(10)
        Salary integer(5)
        Phone varchar(10)
End
```

EmpNo	EName	Salary	Phone
1111	AAA	1000	1111111111
2222	BBB	2000	2222222222
3333	CCC	3000	3333333333
4444	DDD	4000	4444444444

For instance the first record of example file can be stored as

1	1	1	1	A	A	A								1
0	0	0		1	1	1	1	1	1	1	1	1	1	

Thus total 29 bytes are required to store.

Advantage :

- Access is fast because the computer knows where each record starts.

Disadvantage :

(1) Due to fixed size, some larger sized record may cross the block boundaries. That means part of record will be stored in one block and other part of the record may be stored in some another block. Thus we may require two block access for each read or write.

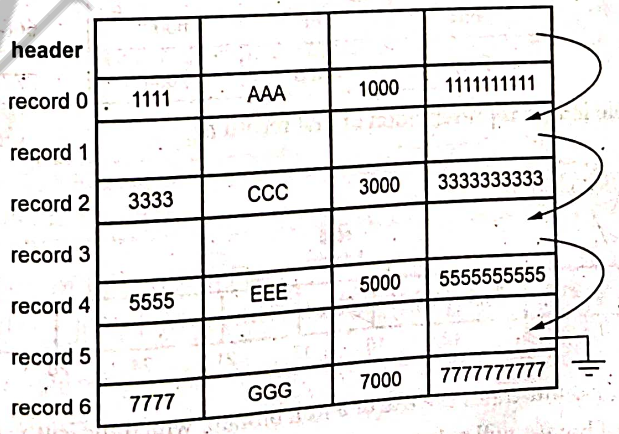
(2) It is difficult to delete the record from this structure. If some intermediate record is deleted from the file then the vacant space must be occupied by next subsequent records.

- When a record is deleted,we could move the record that came after it into the space formerly occupied by the deleted record. But this may require moving of multiple records to occupy the vacant space. This is an undesirable solution to fill up the vacant space of deleted records.
- Another approach is to use a **file header.** At the beginning of the file, we allocate a certain number of bytes as a file header. The header will contain information such as address of the first record whose contents are deleted. We use this first record to store the address of the second available record and so on. Thus the stored record addresses are referred as **pointer** while the deleted records thus form a linked list which is called as **free list.**

For example - Consider the employee record in a file is -

	Emp No	Ename	Salary	Phone
record 0	1111	AAA	1000	1111111111
record 1	2222	BBB	2000	2222222222
record 2	3333	CCC	3000	3333333333
record 3	4444	DDD	4000	4444444444
record 4	5555	EEE	5000	5555555555
record 5	6666	FFF	6000	6666666666
record 6	7777	GGG	7000	7777777777

The representation of records maintaining free list after deletion of record 1,3 and 5



header				
record 0	1111	AAA	1000	1111111111
record 1				
record 2	3333	CCC	3000	3333333333
record 3				
record 4	5555	EEE	5000	5555555555
record 5				
record 6	7777	GGG	7000	7777777777

- On insertion of a new record, we use the record pointed to by the header. We change the header pointer to point to the next available record. If no space is available, we add the new record to the end of the file.

(2) Variable length record

- Variable-length records arise in a database in several ways :

(i) Storage of multiple record types in a file.

(ii) Record types that allow variable lengths for one or more fields.

(iii) Record types that allow repeating fields such as arrays or multisets.

- The representation of a record with variable-length attributes typically has **two parts :**

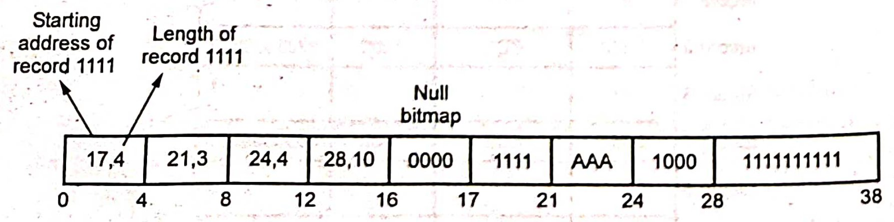
a) an initial part with **fixed length attributes**. This initial part of the record is represented by a pair (offset, length). The offset denotes the starting address of the record while the length represents the actual length of the length.

b) followed by data for variable length attributes.

- For example - Consider the employee records stored in a file as

record 0	1111	AAA	1000	1111111111
record 1	2222	BBB	2000	2222222222
record 2	3333	CCC	3000	3333333333
record 3	4444	DDD	4000	4444444444
record 4	5555	EEE	5000	5555555555
record 5	6666	FFF	6000	6666666666
record 6	7777	GGG	7000	7777777777

- The variable length representation of first record is,



- The figure also illustrates the use of a null bitmap, which indicates which attributes of the record have a null value.

- The variable length record can be stored in blocks. A specialized structure called **slotted page structure** is commonly used for organizing the records within a block. This structure is as shown by following Fig. 4.2.1.

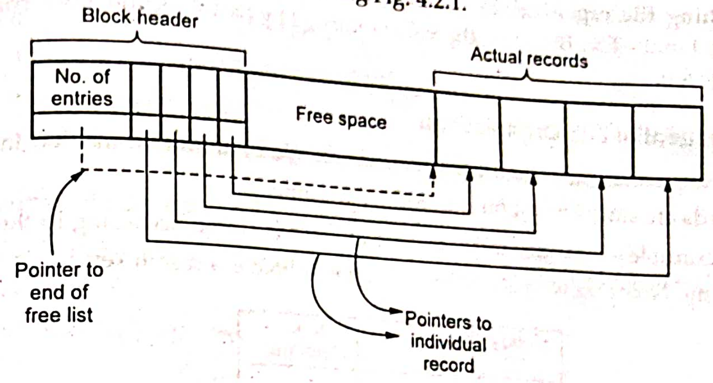


Fig. 4.2.1

- This structure can be described as follows -
  - o At the beginning of each block there is a block header which contains -
    - ▪ Total number of entries (i.e. records).
    - ▪ Pointer to end of free list.
    - ▪ Followed by an array of entries that contain size and location of each record.
  - o The actual records are stored in contiguous block. Similarly the free list is also maintained as continuous block.
  - o When a new record is inserted then the space is allocated from the block of free list.
  - o Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.

Review Question

1. *Describe different types of file organization. Explain using a sketch of each of the, with their advantages and disadvantages.* **AU : Dec.-08, Marks 10**

4.3 Organization of Records in Files

There are three commonly used approaches of organizing records in file -

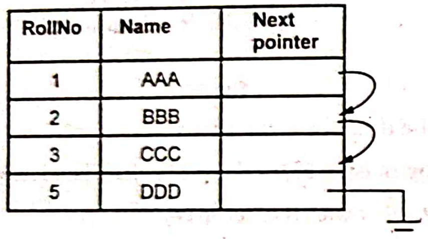
(1) Heap file organization : Any record can be placed anywhere in the file where there is a space. There is no ordering for placing the records in the file. Generally single file is used.

(2) Sequential file organization : Records are stored in sequential order based on the value of search key.

(3) Hashing file organization : A hash function is used to obtain the location of the record in the file. Based on the value returned by hash function, the record is stored in the file.

4.3.1 Sequential File Organization

- The sequential file organization is a simple file organization method in which the records are stored based on the search key value.
- For example - Consider following set of records stored according to the RollNo of student. Note that we assume here that the RollNo is a search key.

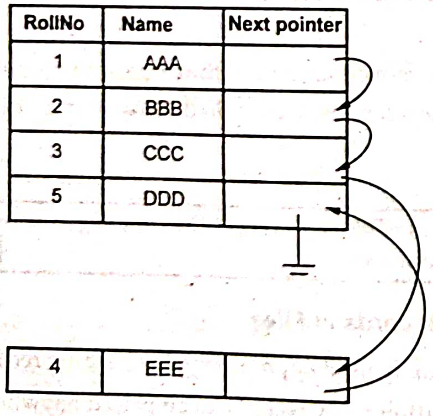


RollNo	Name	Next pointer
1	AAA	
2	BBB	
3	CCC	
5	DDD	

Now if we want to insert following record

4	EEE

Then, we will insert it in sorted order of RollNo and adjust the pointers accordingly.



RollNo	Name	Next pointer
1	AAA	
2	BBB	
3	CCC	
5	DDD	
4	EEE	

Fig. 4.3.1

- However, maintaining physical sequential order is very difficult as there can be several insertions and deletions.
- We can maintain the deletion by next pointer chain.
- For insertion following rules can be applied -
  - o If there is free space insert record there.
  - o If no free space, insert the record in an overflow block.
  - o In either case, pointer chain must be updated.

4.3.2 Multi-table Clustering File Organization

In a multitable clustering file organization, records of several different relations are stored in the same file.

For example - Following two tables **Student and Course**

Sname	Marks
Ankita	55
Ankita	67
Ankita	86
Prajkta	91

Sname	Cname	City
Ankita	ComputerSci	Chennai
Prajkta	Electronics	Pune

The multitable clustering organization for above tables is,

Ankita	ComputerSci	Chennai
Ankita	55	
Ankita	67	
Ankita	86	
Prajkta	Electronics	Pune
Prajkta	91	

This type of file organization is good for join operations such as Student ⋈ Course. This file organization results in variable size records.

The pointer chain can be added to the above records to keep track of address of next record. It can be as shown in following Fig. 4.3.2.

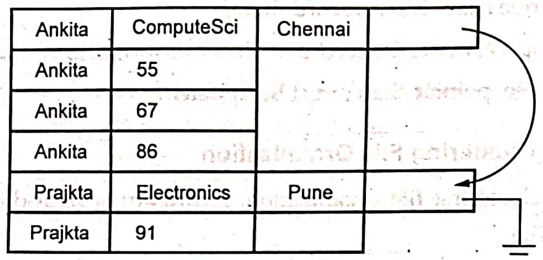


Fig. 4.3.2 Multi-table clustering

4.4 Data Dictionary Storage

- **Definition :** Data dictionary is mini database management system that manages the metadata.
- Data dictionaries are helpful to the database administrators in management of database.
- The general structure of data dictionary is as shown in the Fig. 4.4.1.
- The data in the database dictionary is maintained by several programs and generates the reports if required.
- The data dictionary is integrated with the database systems in which the data is controlled by the data dictionaries and is made available to the DBMS software.
- Following type of **information is maintained** by data dictionary -

a) Description of the schema of the database.

b) Detailed information about the physical database design.

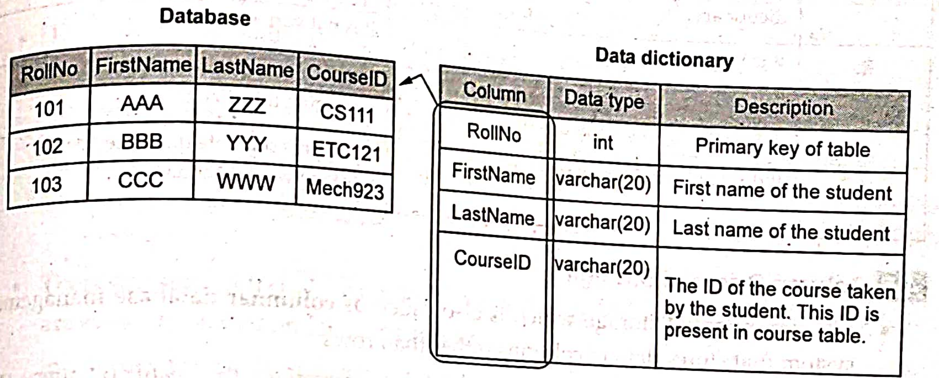
c) Description of database users, their roles and their access rights.

d) Description of database transactions.

e) The description of the relationship between database transactions and data items referenced by them.

f) Information about the usage statistics. That means, how many times the queries are raised to the database, how many transactions are made by the DBMS.

- For example - Consider a Student database, in which various fields are RollNo, FirstName, LastName, and CourseID. The data dictionary for this database maintains the information about this database. The data dictionary contains the column names, Data type of each field and the description of each column of the database.



Database

RollNo	FirstName	LastName	CourseID
101	AAA	ZZZ	CS111
102	BBB	YYY	ETC121
103	CCC	WWW	Mech923

Data dictionary

Column	Data type	Description
RollNo	int	Primary key of table
FirstName	varchar(20)	First name of the student
LastName	varchar(20)	Last name of the student
CourseID	varchar(20)	The ID of the course taken by the student. This ID is present in course table.

Fig. 4.4.1 : Representation of data dictionary

4.4.1 Active and Passive Data Dictionaries

Active data dictionary

- Active Data dictionary is managed automatically by the database management system.
- They are consistent with current structure.
- In the active data dictionary, when any modification or changes is executed by the DBMS, then this dictionary it also gets modified by the DBMS automatically.
- Most of the active data dictionaries are derived from system catalog.

Passive data dictionary

- Passive data dictionary is used only for documentation purpose.
- Passive dictionary is a self-contained application and set of files used for documenting the data processing environment.
- The process of maintaining or modification of the database is manual.
- It is managed by the users of the database systems.

Difference between active and passive data dictionary

Sr. No.	Active data dictionary	Passive data dictionary
1.	The database management system automatically maintains the active data dictionary.	The passive data dictionary is modified whenever the structure of database gets changed.
2.	It is very consistent.	It is not very consistent.
3.	The database management systems automatically manages this dictionary.	The users are responsible for manually managing this dictionary.
4.	It does not require separate database	It requires separate database for working with dictionary.

4.5 Column Oriented Storage

- Column oriented storage which is also called as **columnar database** management system that stores data in columns rather than rows.
- The advantage of column oriented storage is to retrieve the result of query very **efficiently.**
- It also improves disk I/O performances.
- **Example -**

Row - oriented

ID	Name	Subject	Marks
1	AAA	English	87
2	BBB	Maths	95
3	CCC	History	83

Column - oriented

ID	Name
1.	AAA
2.	BBB
3.	CCC

ID	Subject	Marks
1.	English	87
2.	Maths	95
3.	History	83

- As column oriented database store data by columns instead of rows, it can store more data in smaller amount of memory.
- The data retrieval is done column by column only the columns that need to be required are retrieved. This makes it possible for efficient retrieval of data. Also, large amount of data can be handled.
- It is mainly used in data analytics, business intelligence and data warehousing.

4.6 Indexing and Hashing

AU : Dec.-11, Marks 6

- An index is a **data structure** that **organizes data records** on the disk to make the **retrieval** of data **efficient.**
- The search key for an index is collection of one or more fields of records using which we can efficiently retrieve the data that satisfy the search conditions.
- The indexes are required to speed up the search operations on file of records.
- There are two types of indices -
  - o **Ordered Indices :** This type of indexing is based on sorted ordering values.
  - o **Hash Indices :** This type of indexing is based on uniform distribution of values across range of buckets. The address of bucket is obtained using the hash function.
- There are several techniques of for using indexing and hashing. These techniques are evaluated based on following factors -
  - o **Access Types :** It supports various types of access that are supported efficiently.
  - o **Access Time :** It denotes the time it takes to find a particular data item or set items.
  - o **Insertion Time :** It represents the time required to insert new data item.
  - o **Deletion Time :** It represents the time required to delete the desired data item.
  - o **Space overhead :** The space is required to occupy the index structure. But allocating such extra space is worth to achieve improved performance.

Example 4.4.1 *Since indices speed query processing. Why might they not be kept on several search keys? List as many reasons as possible.* **AU : Dec.-11, Marks 6**

Solution : Reasons for not keeping several search indices include :

a. Every index requires additional CPU time and disk I/O overhead during inserts and deletions.

b. Indices on non-primary keys might have to be changed on updates, although an index on the primary key might not (as updates typically do not modify the primary key attributes).

c. Each extra index requires additional storage space.

d. For queries which involve conditions on several search keys, efficiency might not be bad even if only some of the keys have indices on them.

Therefore database performance is improved less by adding indices when many indices already exist.

4.7 Ordered Indices

AU : Dec.-04,08,15, May 06, Marks 10

4.7.1 Primary and Clustered Indices

Primary index :

- An index on a set of fields that includes the primary key is called a primary index. The primary index file should be always in sorted order.
- The primary indexing is always done when the data file is arranged in sorted order and primary indexing contains the primary key as its search key.
- Consider following scenario in which the primary index consists of few entries as compared to actual data file.

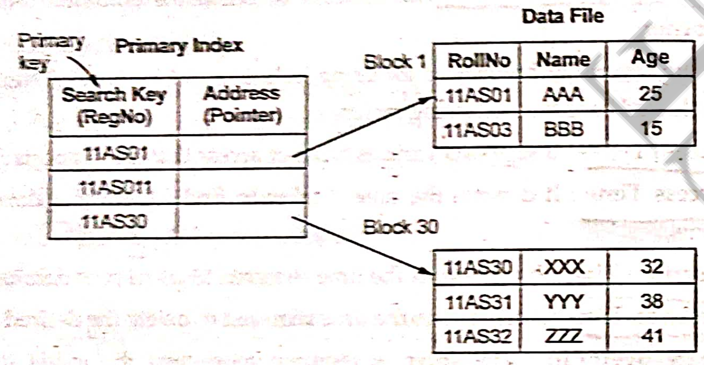


Fig. 4.7.1 : Example of primary index

- Once if you are able to locate the first entry of the record containing block, other entries are stored continuously. For example if we want to search a record for RegNo 11AS32 we need not have to search for the entire data file. With the help of primary index structure we come to know the location of the record containing the RegNo 11AS30, now when the first entry of block 30 is located, then we can easily locate the entry for 11AS32.
- We can apply binary search technique. Suppose there are n = 300 blocks in a main data file then the number of accesses required to search the data file will be $\log_2 n + 1 = (\log_2 300) + 1 \approx 9$
- If we use primary index file which contains at the most n = 3 blocks then using binary search technique, the number of accesses required to search using the primary index file will be $\log_2 n + 1 = (\log_2 3) + 1 \approx 3$
- This shows that using primary index the access time can be deduced to great extent.

Clustered index :

- In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as clustering index.
- When a file is organized so that the ordering of data records is the same as the ordering of data entries in some index then say that index is clustered, otherwise it is an unclustered index.
- Note that, the data file need to be in sorted order.
- Basically, records with similar characteristics are grouped together and indexes are created for these groups.
- For example, students studying in each semester are grouped together. i.e.; 1st semester students, 2nd semester students, 3rd semester students etc. are grouped.

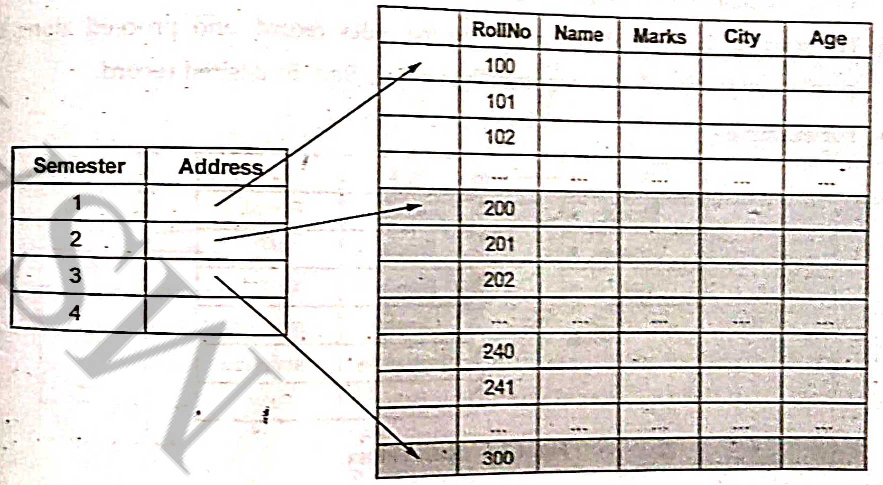


Fig. 4.7.2 : Clustered index

4.7.2 Dense and Sparse Indices

There are two types of ordered indices :

1) **Dense index :**

- An index record appears for every search key value in file.
- This record contains search key value and a pointer to the actual record.

- For example :

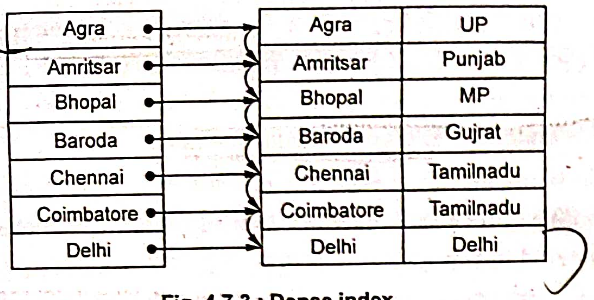


Fig. 4.7.3 : Dense index

2) Sparse index :

- Index records are created only for some of the records.
- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along the pointers in the file (that is, sequentially) until we find the desired record.
- For example -

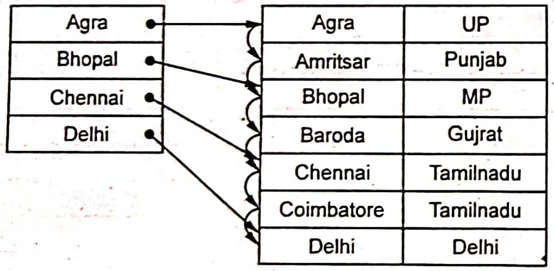


Fig. 4.7.4 : Sparse index

4.7.3 Single and Multilevel Indices

Single level indexing :

- A single-level index is an auxiliary file that makes it more efficient to search for a record in the data file.
- The index is usually specified on one field of the file (although it could be specified on several fields).

- Each index can be in the following form.

Search Key	Pointer to Record

- The index file usually occupies considerably less disk blocks than the data file because its entries are much smaller.
- A binary search on the index yields a pointer to the file record.
- The types of single level indexing can be primary indexing, clustering index or secondary indexing.
- Example : Following Fig. 4.7.5 represents the single level indexing -

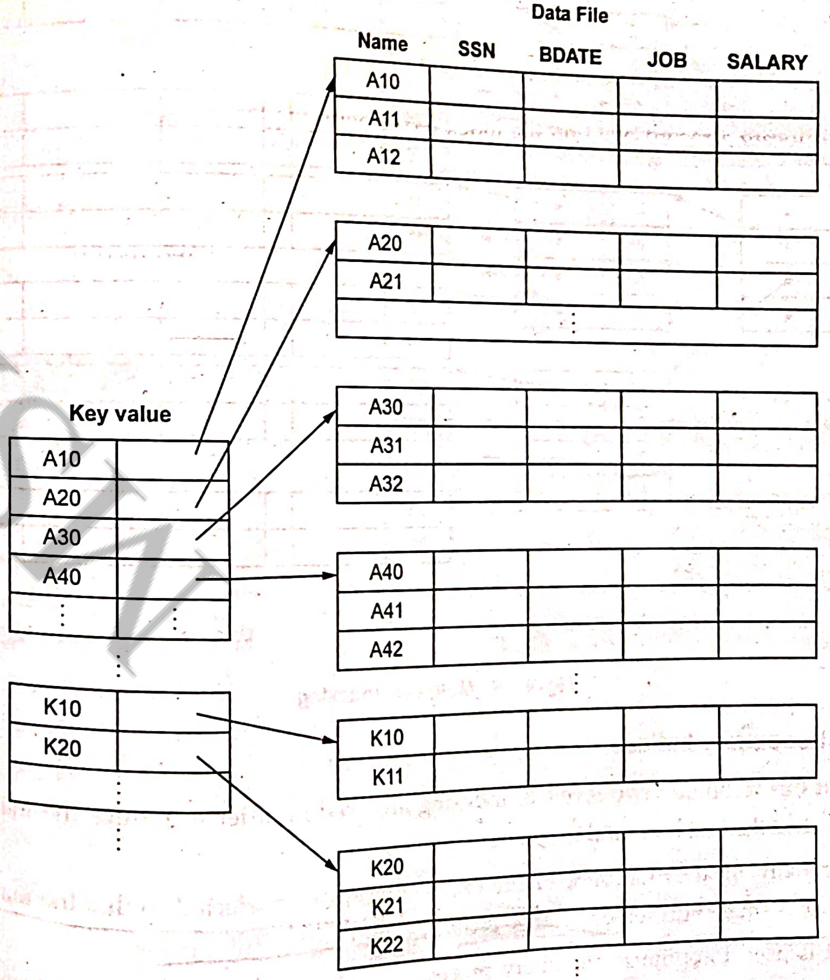


Fig. 4.7.5 : Single level indexing

Multilevel indexing :

- There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.
- Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.
- The multilevel indexing can be represented by following Fig. 4.7.6.

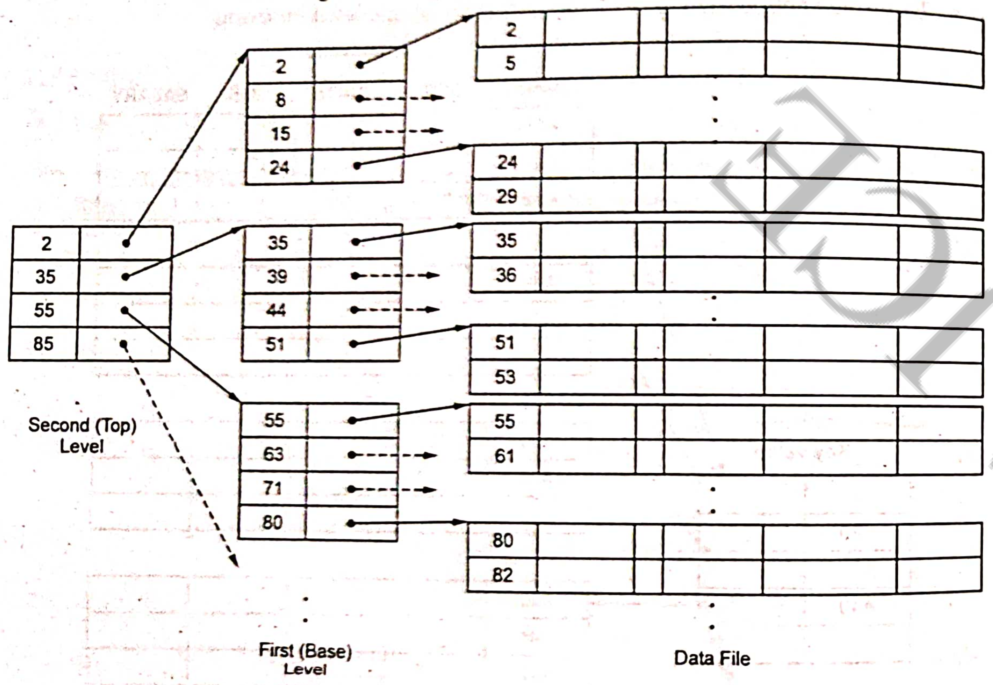


Fig. 4.7.6 : Multilevel indexing

4.7.4 Secondary Indices

- In this technique two levels of indexing are used in order to reduce the mapping size of the first level and in general.
- Initially, for the first level, a large range of numbers is selected so that the mapping size is small. Further, each range is divided into further sub ranges.
- It is used to optimize the query processing and access records in a database with some information other than the usual search key.

- For example -

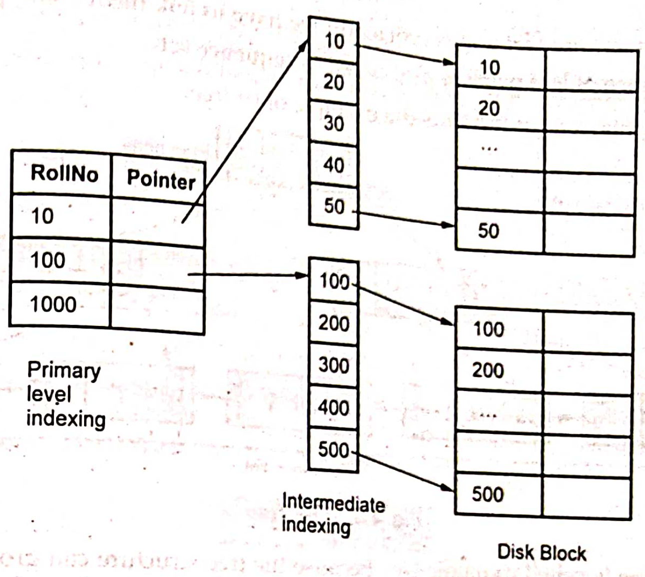


Fig. 4.7.7 : Secondary Indexing

Review Questions

1. *Illustrate indexing and hashing techniques with suitable examples.* **AU : Dec.-15, Marks 8**
2. *What is the use of an index structure and explain the concept of ordered indices.* **AU : Dec 04, Marks 8**
3. *What is the difference between primary index and secondary index ?* **AU : May 06, Marks 8**
4. *Explain the index schemas used in database systems.* **AU : Dec.- 08, Marks 10**

4.8 B+ Tree Index Files

AU : May-03,06,16,19, Dec.-17, Marks 16

- The B+ tree is similar to binary search tree. It is a balanced tree in which the internal nodes direct the search.
- The leaf nodes of B+ trees contain the **data entries.**

Structure of B+ Tree

- The typical node structure of B+ node is as follows -

P_1	K_1	P_2	K_2	...	P_{n-1}	K_{n-1}	P_n

- It contains up to n − 1 search-key values $K_1, K_2, \ldots, K_{n-1}$, and n pointers $P_1, P_2, \ldots, P_n$.

- The search-key values within a node are kept in sorted order; thus, if $i < j$, then $K_i < K_j$.
- To retrieve all the leaf pages efficiently we have to link them using **page pointers.** The sequence of leaf pages is also called as **sequence set.**
- Following Fig. 4.8.1 represents the example of B+ tree.

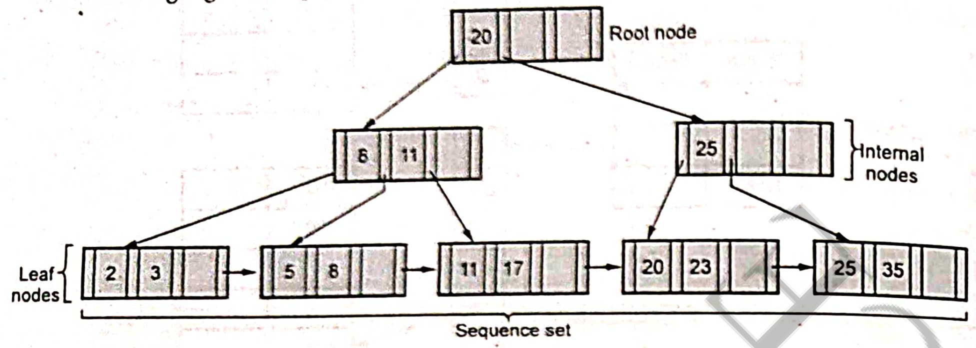


Fig. 4.8.1 : B+ Tree

- The B+ tree is called dynamic tree because the tree structure can grow on insertion of records and shrink on deletion of records.

Characteristics of B+ Tree

Following are the characteristics of B+ tree.

1) The B+ tree is a balanced tree and the operations insertions and deletion keeps the tree balanced.
2) A minimum occupancy of 50 percent is guaranteed for each node except the root.
3) Searching for a record requires just traversal from the root to appropriate leaf.

4.8.1 Insertion Operation

Algorithm for insertion :

Step 1 : Find correct leaf L.

Step 2 : Put data entry onto L.

i) If L has enough space, done!

ii) Else, must split L (into L and a new node L2)

- Allocate new node
- Redistribute entries evenly

- Copy up middle key.
- Insert index entry pointing to L2 into parent of L.

Step 3 : This can happen recursively

i) To split index node, redistribute entries evenly, but push up middle key. (Contrast with leaf splits.)

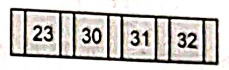
Step 4 : Splits "grow" tree; root split increases height.

i) Tree growth: gets wider or one level taller at top.

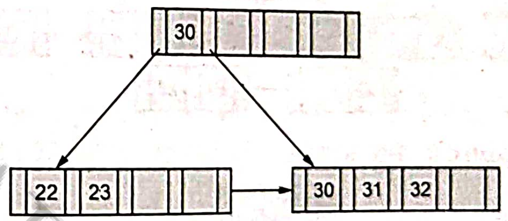
Example 4.8.1 *Construct B+ tree for following data. 30,31,23,32,22,28,24,29, where number of pointers that fit in one node are 5.*

Solution : In B+ tree each node is allowed to have the number of pointers to be 5. That means at the most 4 key values are allowed in each node.

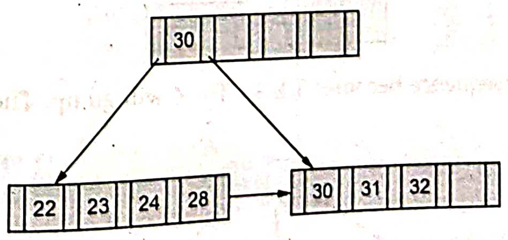
Step 1 : Insert 30,31,23,32. We insert the key values in ascending order.



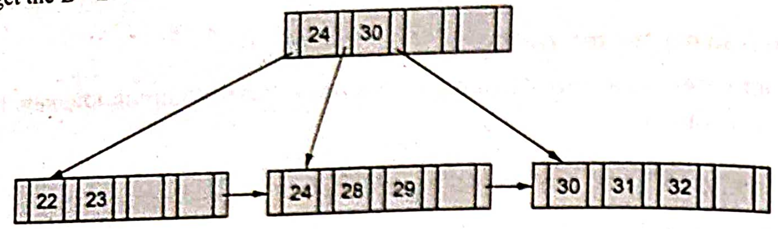
Step 2 : Now if we insert 22, the sequence will be 22 , 23, 30, 31, 32. The middle key 30, will go up.



Step 3 : Insert 28,24. The insertion is in ascending order.



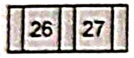
Step 4 : Insert 29. The sequence becomes 22, 23, 24, 28, 29. The middle key 24 will go up. Thus we get the B+ tree.



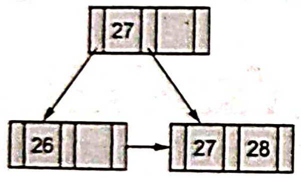
Example 4.8.2 *Construct B+ tree to insert the following (order of the tree is 3)* *26,27,28,3,4,7,9,46,48,51,2,6*

Solution : Order means maximum number of children allowed by each node. Hence order 3 means at the most 2 key values are allowed in each node.

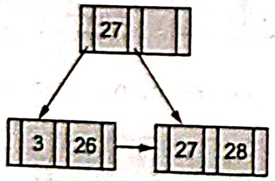
Step 1 : Insert 26, 27 in ascending order



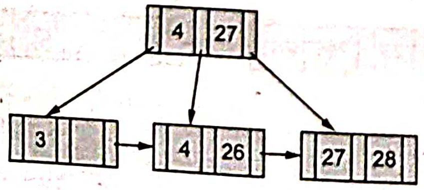
Step 2 : Now insert 28. The sequence becomes 26,27,28. As the capacity of the node is full, 27 will go up. The B+ tree will be,



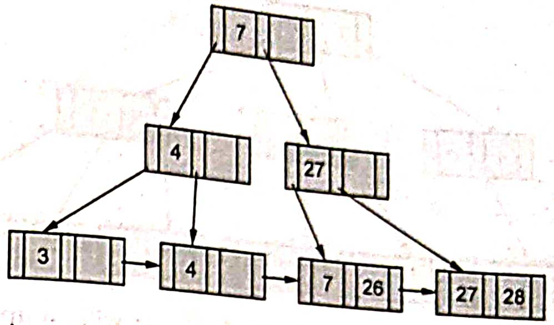
Step 3 : Insert 3. The partial B+ Tree will be,



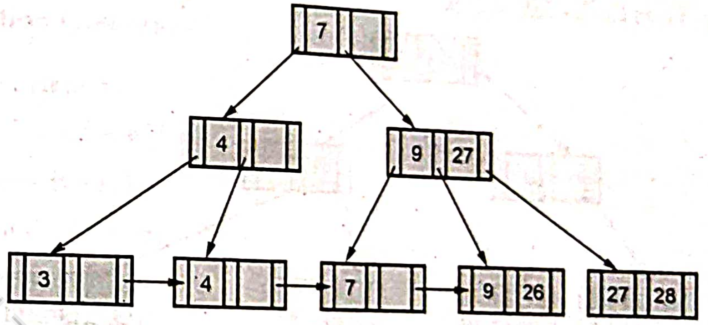
Step 4 : Insert 4. The sequence becomes 3,4, 26. The 4 will go up. The partial B+ tree will be -



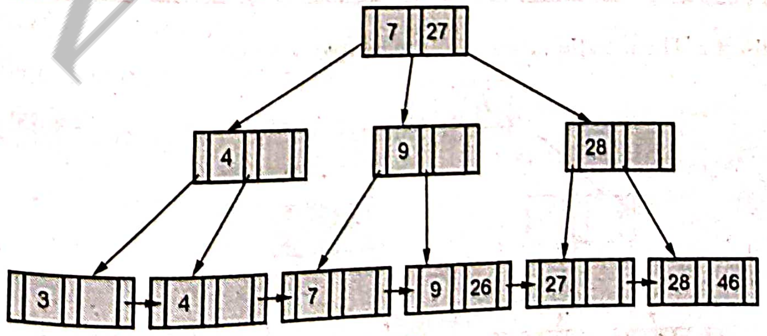
Step 5 : Insert 7. The sequence becomes 4,7,26. The 7 will go up. Again from 4,7,27 the 7 will go up. The partial B+ Tree will be,



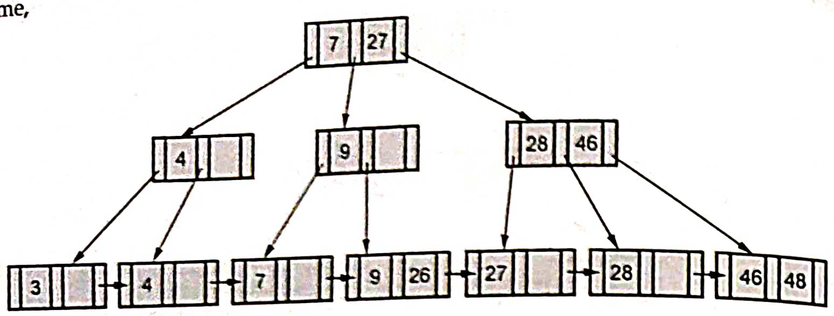
Step 6 : Insert 9. By inserting 7,9, 26 will be the sequence. The 9 will go up. The partial B+ tree will be,



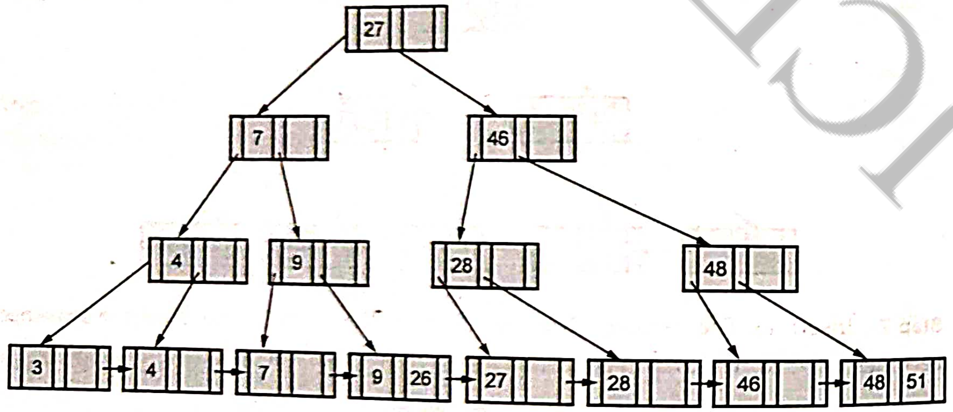
Step 7 : Insert 46. The sequence becomes 27,28,46. The 28 will go up. Now the sequence becomes 9, 27, 28. The 27 will go up and join 7. The B+ Tree will be,



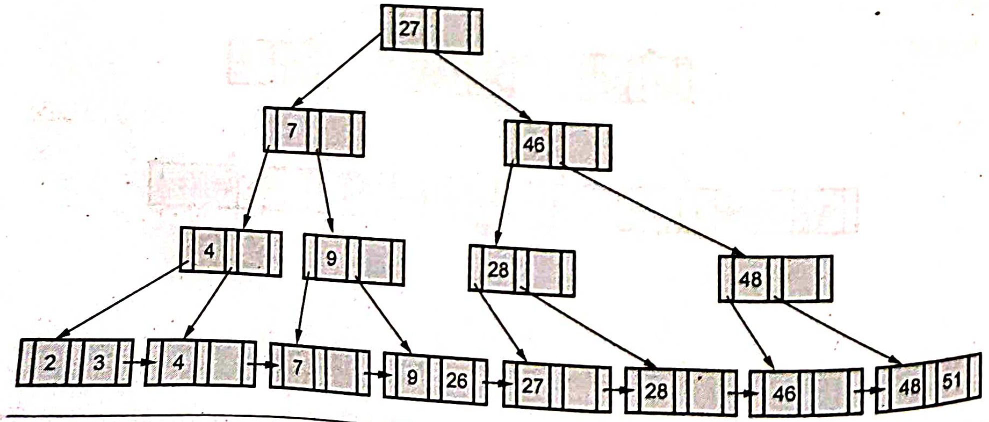
Step 8 : Insert 48. The sequence becomes 28,46,48. The 46 will go up. The B+ Tree will become,



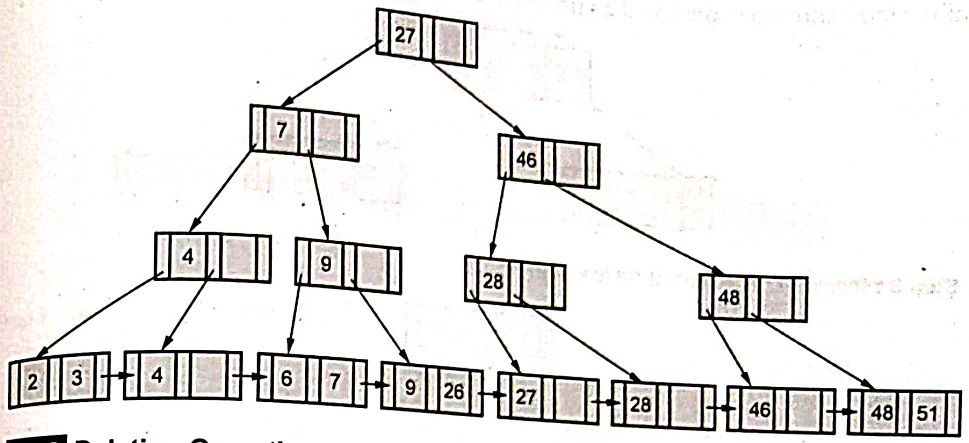
Step 9 : Insert 51. The sequence becomes 46,48,51. The 48 will go up. Then the sequence becomes 28, 46, 48. Again the 46 will go up. Now the sequence becomes 7,27, 46. Now the 27 will go up. Thus the B+ tree will be



Step 10 : Insert 2. The insertion is simple. The B+ tree will be,



Step 11 : Insert 6. The insertion can be made in a vacant node of 7(the leaf node). The final B+ tree will be,



4.8.2 Deletion Operation

Algorithm for deletion :

Step 1 : Start at root, find leaf L with entry, if it exists.

Step 2 : Remove the entry.

i) If L is at least half-full, done!

ii) If L has only d-1 entries,

- Try to **re-distribute,** borrowing keys from sibling. **(adjacent node with same parent as L).**
- If redistribution fails, **merge L and sibling.**

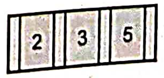
Step 3 : If merge occurred, must delete entry (pointing to L or sibling) from parent of L.

Step 4 : Merge could propagate to root, decreasing height.

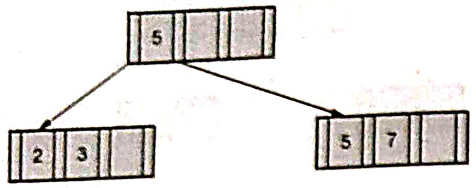
Example 4.8.3 *Construct B+ Tree for the following set of key values (2,3,5,7,11,17,19,23,29,31) Assume that the tree is initially empty and values are added in ascending order. Construct B+ tree for the cases where the number of pointers that fit one node is four. After creation of B+ tree perform following series of operations :*
(a) Insert 9. (b) Insert 10. (c) Insert 8. (d) Delete 23. (e) Delete 19.

Solution : The number of pointers fitting in one node is four. That means each node contains at the most three key values.

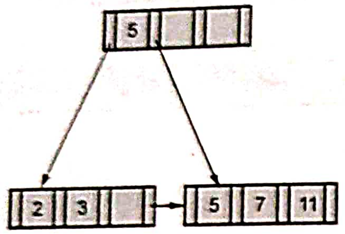
Step 1 : Insert 2, 3, 5.



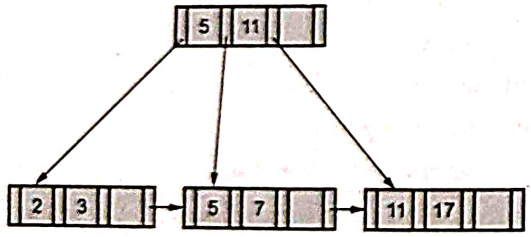
Step 2 : If we insert 7, the sequence becomes 2, 3, 5, 7. Since each node can accommodate at the most three key, the 5 will go up, from the sequence 2, 3, ⑤, 7.



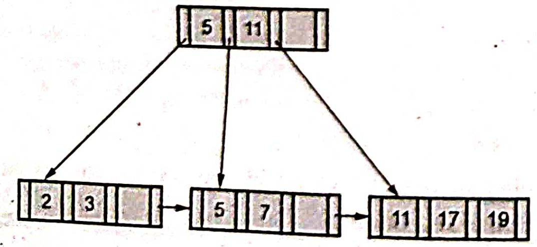
Step 3 : Insert 11. The partial B+ tree will be,



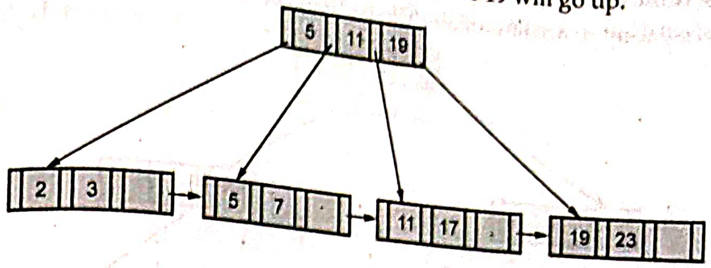
Step 4 : Insert 17. The sequence becomes 5,7, 11,17. The element 11 will go up. Then the partial B+ tree becomes,



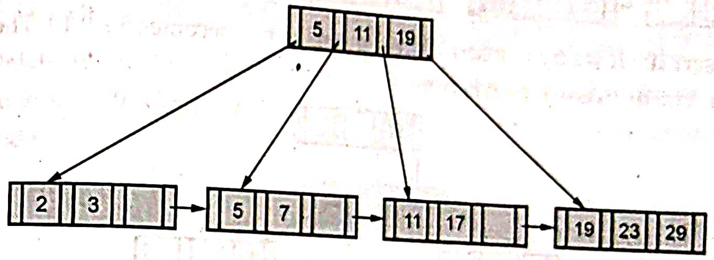
Step 5 : Insert 19.



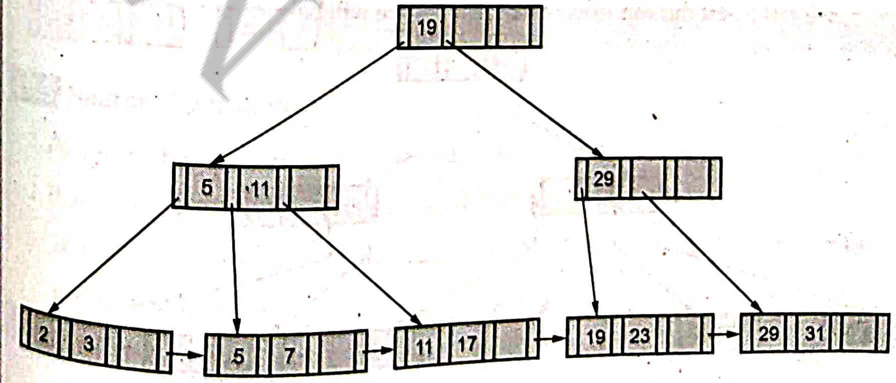
Step 6 : Insert 23. The sequence becomes 11,17,19,23. The 19 will go up.



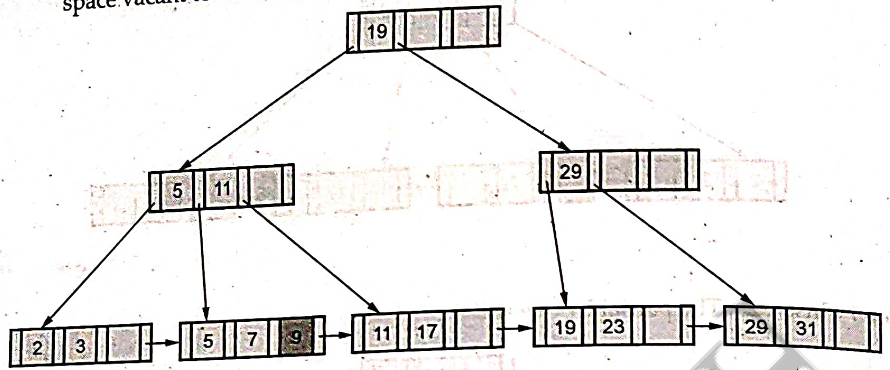
Step 7 : Insert 29. The partial B+ tree will be,



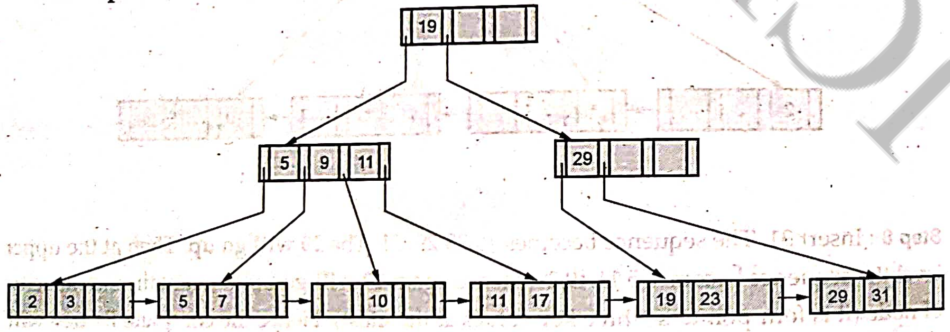
Step 8 : Insert 31. The sequence becomes 19,23,29, 31. The **29 will go up.** Then at the upper level the sequence becomes 5,11,19,29. Hence again **19 will go up** to maintain the capacity of node (it is four pointers= three key values at the most). Hence the complete B+ tree will be,



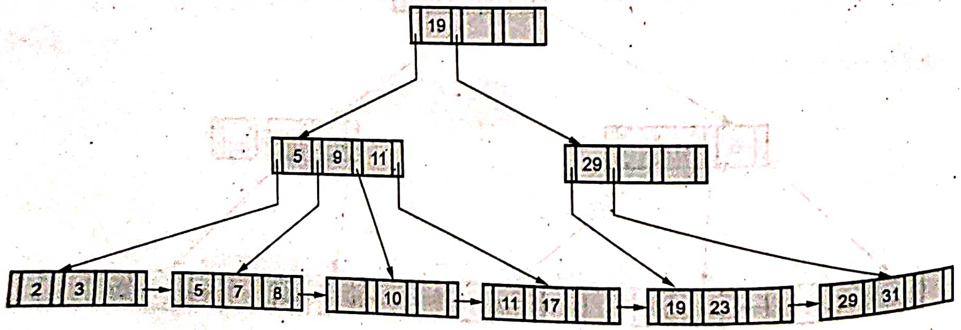
(a) **Insertion of 9 :** It is very simple operation as the node containing 5,7 has one space vacant to accommodate. The B+ tree will be,



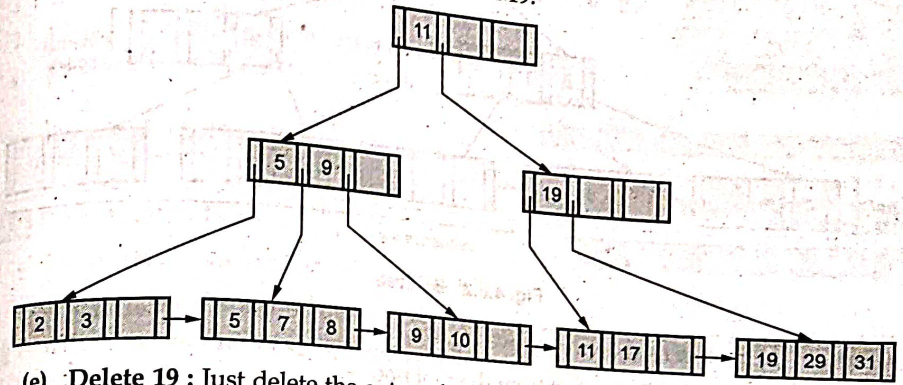
(b) **Insert 10 :** If we try to insert 10 then the sequence becomes 5,7,9,10. The **9 will go up.** The B+ tree will then become -



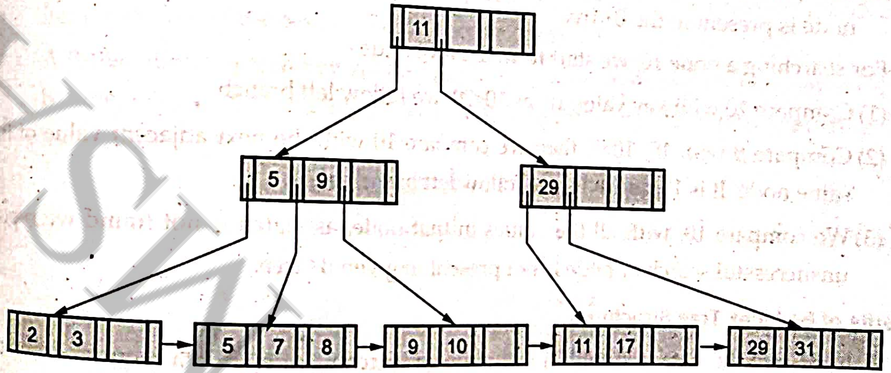
(c) **Insert 8 :** Again insertion of 8 is simple. We have a vacant space at node 5,7. So we just insert the value over there. The B+ tree will be-



(d) **Delete 23 :** Just remove the key entry of 23 from the node 19,23. Then **merge the sibling node** to form a node 19,29,31. Get down the entry of 11 to the leaf node. Attach the node of 11,17 as a left child of 19.



(e) **Delete 19 :** Just delete the entry of 19 from the node 19,29,31. Delete the internal node key 19. Copy the 29 up as an internal node as it is an inorder successor node.



4.8.3 Search Operation

1. Perform a binary search on the records in the current node.
2. If a record with the search key is found, then return that record.
3. If the current node is a leaf node and the key is not found, then report an unsuccessful search.
4. Otherwise, follow the proper branch and repeat the process.

For example -

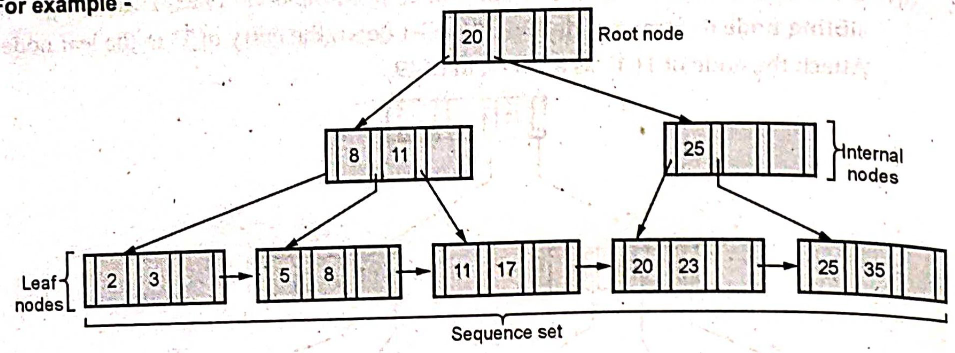


Fig. 4.8.2 : B+ Tree

Consider the B+ tree as shown in above Fig. 4.8.2.

For searching a node 25, we start from the root node -

(1) Compare 20 with key value 25. As 25>20, move on to right branch.

(2) Compare 25 with key value 25. As the match is found we declare, that the given node is present in the B+ tree.

For searching a node 10, we start form the root node -

(1) Compare 20 with key value 10, as 10<20, we follow left branch

(2) Compare 8 with 10, 10>8, then we compare 10 with the next adjacent value of the same node. It is 11, as 10<11, we follow left branch of 11.

(3) We compare 10, with all the values in that node, as match is not found we report unsuccessful search or node is not present in given B+ tree.

Merits of B+ Index Tree Structure

1. In B+ tree the data is stored in leaf node so searching of any data requires scanning only of leaf node alone.
2. Data is ordered in linked list.
3. Any record can be fetched in equal number of disk accesses.
4. Range queries can be performed easily as leaves are linked up.
5. Height of the tree is less as only keys are used for indexing.
6. Supports both random and sequential access.

Demerits of B+ Index Tree Structure

1. Extra insertion of non leaf nodes.
2. There is space overhead.

Review Questions

1. *Explain the B+ tree indexes on multiple keys with suitable example.* **AU : Dec.-17, Marks 7**
2. *Briefly explain about B+ index file with example.* **AU : May-16, Marks 16**
3. *What are the merits and demerits of B+ tree index structures.* **AU : May-03, Marks 4**
4. *Describe structure of B+ tree and list the characteristics of B+ tree.* **AU : May-03, Marks 6, May-06, Marks 8**
5. *Describe the structure of B+ tree and give the algorithm for search in the B+ tree with example.* **AU : May-19, Marks 13**

4.9 B Tree Index Files

AU : Dec.-12,14, May-08, Marks 16

- B-tree indices are similar to B+-tree indices.
- The primary distinction between the two approaches is that a B-tree eliminates the redundant storage of search-key values.
- B-tree is a specialized multiway tree used to store the records in a disk.
- There are number of subtrees to each node. So that the height of the tree is relatively small. So that only small number of nodes must be read from disk to retrieve an item. The goal of B-trees is to get fast access of the data.
- A B-tree allows search-key values to appear only once (if they are unique), unlike a B+-tree, where a value may appear in a nonleaf node, in addition to appearing in a leaf node.

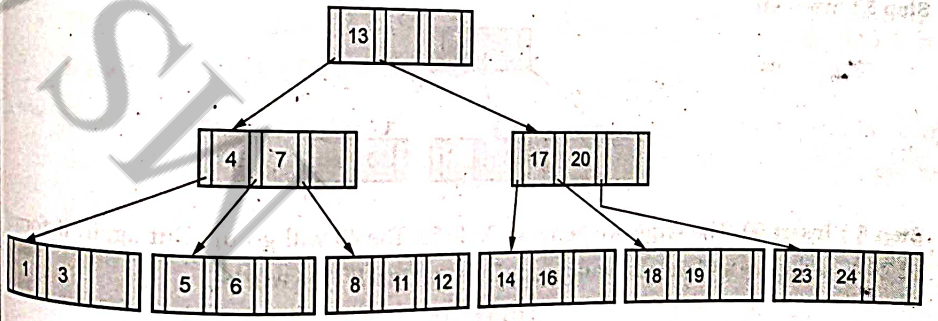
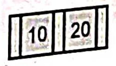


Fig. 4.9.1 : B-Tree

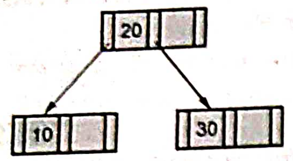
Example 4.9.1 *Create B tree of order 3 for following data: 20,10,30,15,12,40,50.*

Solution : The B tree of order 3 means at the most two key values are allowed in each node of B-Tree.

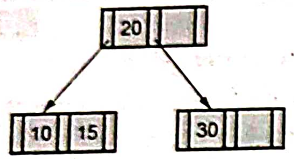
Step 1 : Insert 20,10 in ascending order



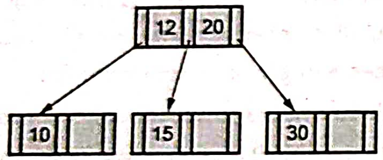
Step 2 : If we insert the value 30. The sequence becomes 10,20,30. As only two key values are allowed in each node (being order 3), the 20 will go up.



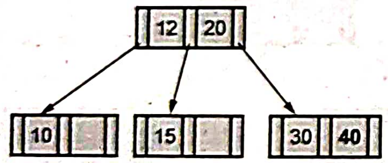
Step 3 : Now insert 15.



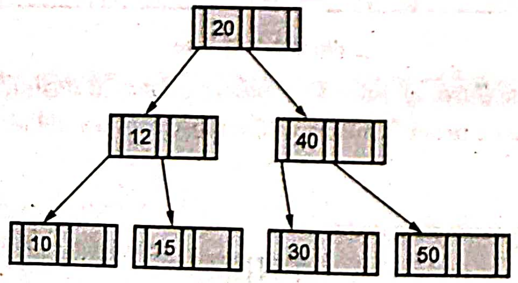
Step 4 : Insert 12. The sequence becomes 10, 12, 15. The middle element 12 will go up.



Step 5 : Insert 40



Step 6 : Insert 50. The sequence becomes 30,40,50. The 40 will go up. But again it forms 12,20,40 sequence and then 20 will go up. Thus the final B Tree will be,



This is the final B-Tree

Difference between B Tree and B+ Tree

B Trees	B+ Trees
In a B-tree you can store both keys and data in the **internal and leaf nodes.**	In a B+ tree you have to store the **data in the leaf nodes** only.
It wastes space.	**It does not waste space.**
The leaf node **cannot store using linked list.**	The leaf nodes are connected using linked list.
Searching becomes **difficult** in B-tree as data cannot be found in the leaf node.	**Searching** of any data in a B+ tree is **very easy** because all data is found in leaf nodes.
The B-tree **does not store redundant search key.**	The B-tree stores **redundant search key.**

Review Questions

1. *Explain detail about i) B+ tree index ii) B tree index files* **AU : Dec.-14, Marks 16, May-08, Marks 10**
2. *Write down detailed notes on ordered indices and B-tree index files.* **AU : Dec.-12, Marks 16**

4.10 Concept of Hashing

AU : Dec.-04,05, May-05,14, Marks 8

- Hash file organization method is the one where data is stored at the data blocks whose address is generated by using **hash function.**
- The memory location where these records are stored is called as **data block** or **bucket.** This bucket is capable of storing one or more records.
- The hash function can use any of the column value to generate the address. Most of the time, hash function uses **primary key** to generate the hash index - address of the data block.
- Hash function can be **simple mathematical function** to any complex mathematical function.
- For example - Following figure represents the records of student can be searched using hash based indexing. In this example the hash function is based on the **age** field of the record. Here the index is made up of data entry k* which is **actual data record.** For a hash function the **age** is converted to binary number format and the last two digits are considered to locate the student record.

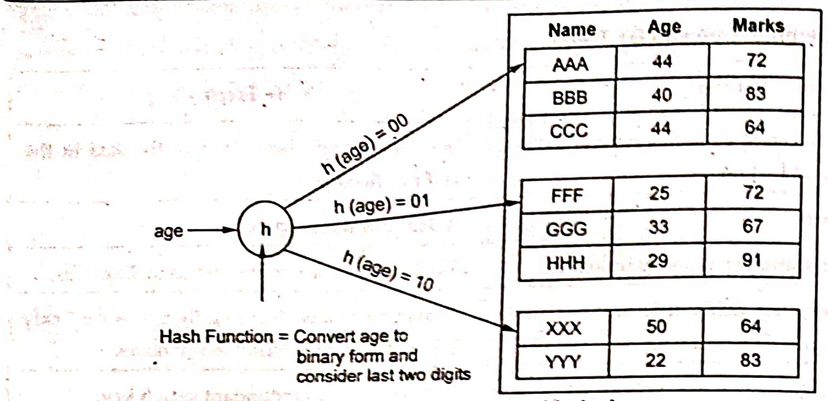


Fig. 4.10.1 : Hash based indexing

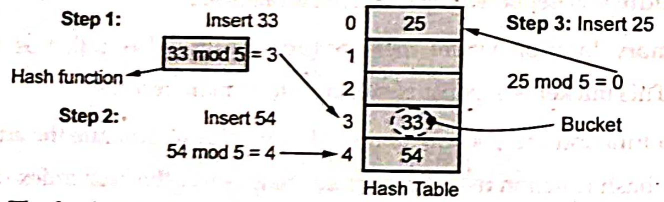
4.10.1 Basic Terms used in Hashing

1) **Hash Table :** Hash table is a data structure used for storing and retrieving data quickly. Every entry in the hash table is made using Hash function.

2) **Hash function :**

- Hash function is a function used to place data in hash table.
- Similarly hash function is used to retrieve data from hash table.
- Thus the use of hash function is to implement hash table.

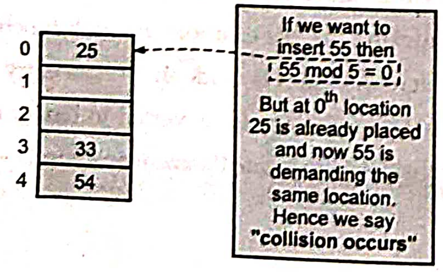
For example : Consider hash function as key mod 5. The hash table of size 5.



3) **Bucket :** The hash function H(key) is used to map several dictionary entries in the hash table. Each position of the hash table is called bucket.

4) **Collision :** Collision is situation in which hash function returns the same address for more than one record.

For example :



0 25
1
2
3 33
4 54
If we want to insert 55 then 55 mod 5 = 0
But at 0th location 25 is already placed and now 55 is demanding the same location. Hence we say "collision occurs"

5) **Probe :** Each calculation of an address and test for success is known as a probe.

6) **Synonym :** The set of keys that has to the same location are called synonyms. For example - In above given hash table computation 25 and 55 are synonyms.

7) **Overflow :** When hash table becomes full and new record needs to be inserted then it is called overflow.

For example -

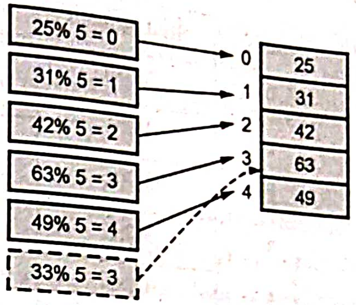


Fig. 4.10.2 : Overflow situation

Review Questions

1 *Write a note on hashing.* **AU : May-14, Dec.- 05, Marks 8**

2 *Describe hash file organization.* **AU : Dec.-04, May-05, Marks 8**

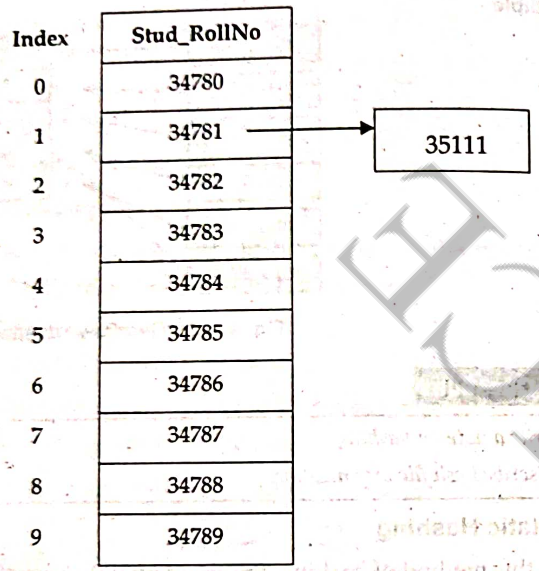
4.11 Static Hashing

AU : May-06, Marks 8

- In this method of hashing, the resultant data bucket address will be always same.

Index	Stud_RollNo
0	34780
1	34781
2	34782
3	34783
4	34784
5	34785
6	34786
7	34787
8	34788
9	34789

- That means, if we want to generate address for **Stud_RollNo = 34789**. Here if we use mod 10 hash function, it always result in the same bucket address 9. There will not be any changes to the bucket address here.
- Hence number of data buckets in the memory for this static hashing remains constant throughout. In our example, we will have ten data buckets in the memory used to store the data.



Index	Stud_RollNo
0	34780
1	34781
2	34782
3	34783
4	34784
5	34785
6	34786
7	34787
8	34788
9	34789

Table 4.11.1

- If there is **no space** for some data entry then we can allocate **new overflow page**, put the data record onto that page and add the page to **overflow chain** of the bucket. For example if we want to add the Stud_RollNo= 35111 in above hash table then as there is no space for this entry and the hash address indicate to place this record at index 1, we create overflow chain as shown in Table 4.11.1.

Example 4.11.1 *Why is hash structure not the best choice for a search key on which range of queries are likely ?* **AU : May-06, Marks 8**

Solution :

- A range query cannot be answered efficiently using a hash index, we will have to read all the buckets.
- This is because key values in the range do not occupy consecutive locations in the buckets, they are distributed uniformly and randomly throughout all the buckets.

Advantages of Static Hashing

(1) It is simple to implement.

(2) It allows speedy data storage.

Disadvantages of Static Hashing

There are two major disadvantages of static hashing :

1) In static hashing, there are fixed number of buckets. This will create a problematic situation if the number of records grow or shrink.

2) The ordered access on hash key makes it inefficient.

4.11.1 Open Hashing

The **open hashing** is a form of **static hashing** technique. When the collision occurs, that means if the hash key returns the same address which is already allocated by some data record, then the **next available data block** is used to enter new record instead of overwriting the old record. This technique is also called as **linear probing**. For example

Consider insertion of record 105 in the hash table below with the hash function h (key) mod 10.

Index	Stud_RollNo
0	10
1	1
2	22
3	
4	
5	55
6	106
7	
8	88
9	19

The 105 is probed at next empty data block as follows -

Index	Stud_RollNo
0	10
1	1
2	22
3	
4	
5	55
6	106
7	105
8	88
9	19

Advantages :

1) It is **faster** technique.
2) It is **simple** to implement.

Disadvantages :

1) It forms clustering, as the record is just inserted to next free available slot.
2) If the hash table gets full then the next subsequent records can not be accommodated.

4.12 Dynamic Hashing

AU : May-04,07,18, Dec.-08,17, Marks 13

- The problem with static hashing is that it does not expand or shrink dynamically as the size of the database grows or shrinks.
- Dynamic hashing provides a mechanism in which data buckets are added and removed dynamically and on-demand.
- The most commonly used technique of dynamic hashing is extendible hashing.

4.12.1 Extendible Hashing

The extendible hashing is a dynamic hashing technique in which, if the bucket is overflow, then the number of buckets are doubled and data entries in buckets are re-distributed.

Example of extendible hashing :

In extendible hashing technique the **directory** of pointers to bucket is used. Refer following Fig. 4.12.1

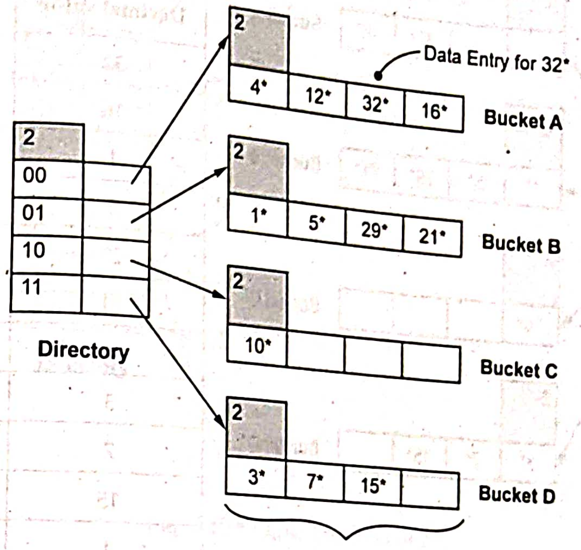
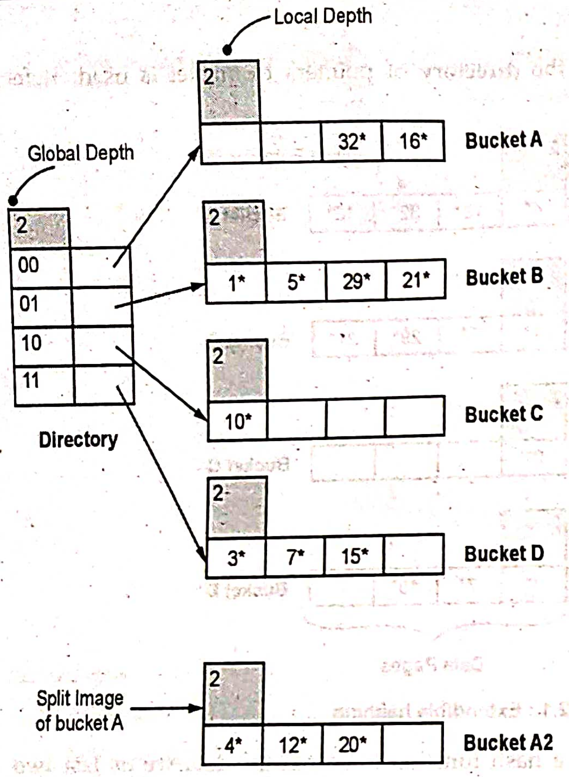


Fig. 4.12.1 : Extendible hashing

To locate a data entry, we apply a hash function to search the data we us **last two digits of binary representation of number.** For instance binary representation of 32* = 10000000. The last two bits are 00. Hence we store 32* accordingly.

Insertion operation :

- Suppose we want to insert 20* (binary 10100). But with 00, the bucket A is full. So we must **split the bucket** by allocating new bucket and redistributing the contents, across the old bucket and its split image.
- For splitting, we consider last three bits of h(r).
- The redistribution while insertion of 20* is as shown in following Fig. 4.12.2.



Decimal value	Binary value
32	100000
16	10000
1	0001
5	0101
29	011101
21	010101
10	01010
3	011
7	0111
15	01111
4	100
12	01100
20	010100

Fig. 4.12.2 : During insertion process

The split image of bucket A i.e. A2 and old bucket A are based on last two bits i.e. 00. Here we need two data pages, to adjacent additional data record. Therefore here it is necessary to **double the directory** using three bits instead of two bits. Hence,

- There will be binary versions for buckets A and A2 as 000 and 100.
- In extendible hashing, last bits d is called **global depth** for directory and d is called **local depth** for data pages or buckets. After insetion of 20*, the global depth becomes 3 as we consider last three bits and local depth of A and A2 buckets become 3 as we are considering last three bits for placing the data records. Refer Fig. 4.12.3.

(**Note :** *Student should refer binary values given in Fig. 4.12.2, for understanding insertion operation*)

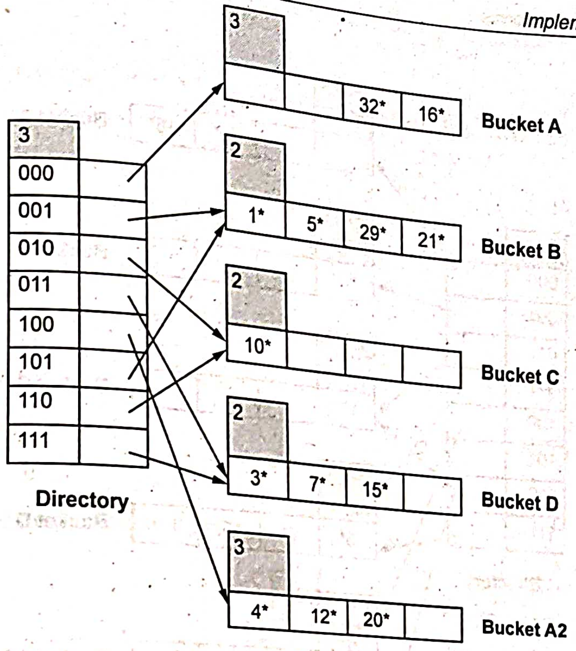


Fig. 4.12.3 : After insertion of 20*

- Suppose if we want to insert 11*, it belongs to bucket B, which is already full. Hence let us split bucket B into old bucket B and split image of B as B2.
- The local depth of B and B2 now becomes 3.
- Now for bucket B, we get and 1 = 001
  11 = 10001
- For bucket B2, we get
  5 = 101
  29 = 11101
  and 21 =**10101**

After insertion of 11* we get the scenario as follows,

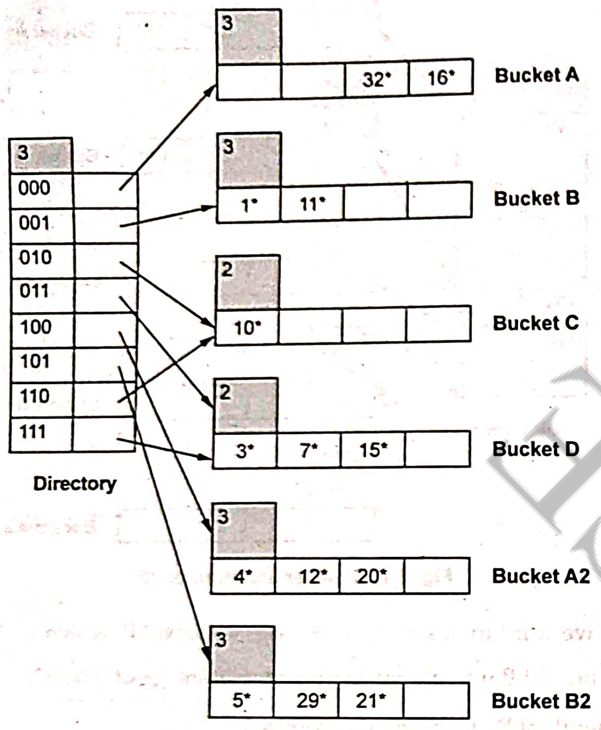


Fig. 4.12.4 : After insertion of 11*

Example 4.12.1 *The following key values are organized in an extendible hashing technique.*

1 3 5 8 9 12 17 28. Show the extendible hash structure for this file if the hash function is h(x) =

x mod 8 and buckets can hold three records. Show how extendable hash structure changes as the result of each of the following steps :

Insert 2

Insert 24

Delete 5

Delete 12

Solution :

Step 1 : Initially we assume the hash function based on last two bits, of result of hash function.

1 mod 8 = 1 = 001

3 mod 8 = 3 = 011

5 mod 8 = 5 = 101

8 mod 8 = 0 = 000

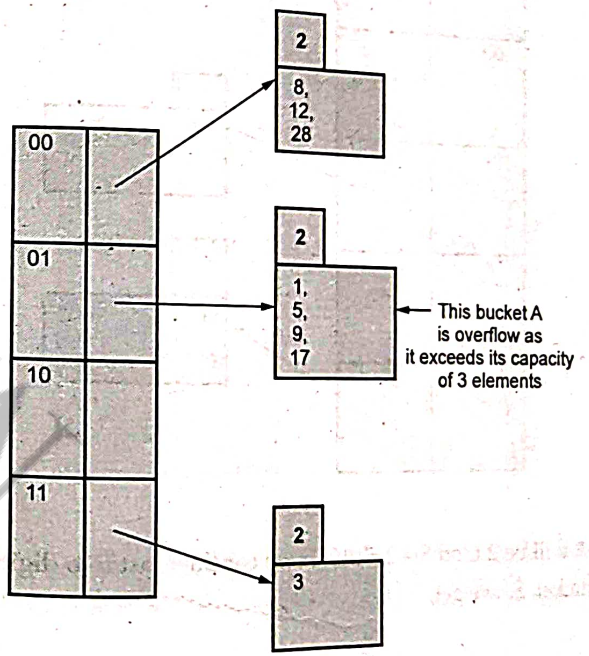
9 mod 8 = 1 = 001

12 mod 8 = 4 = 100

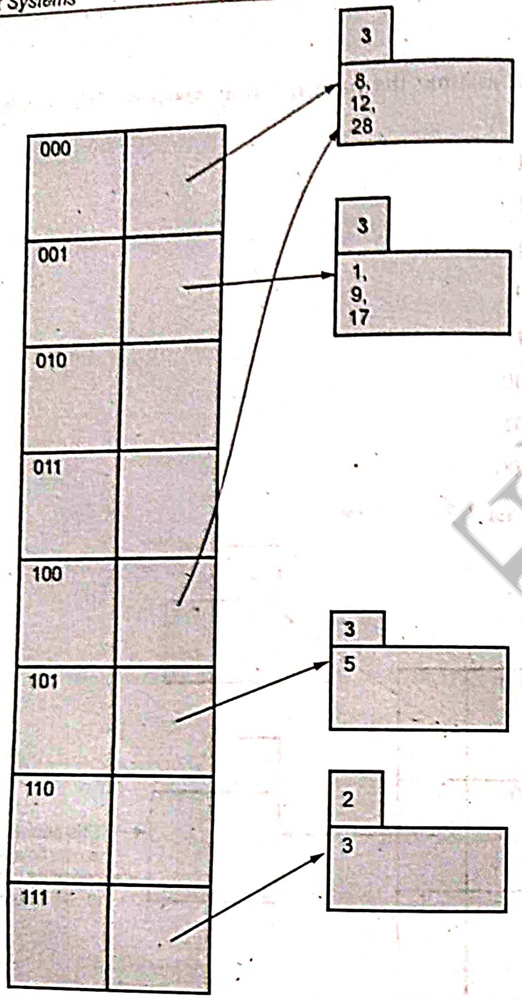
17 mod 8 = 1 = 001

28 mod 8 = 4 = 100

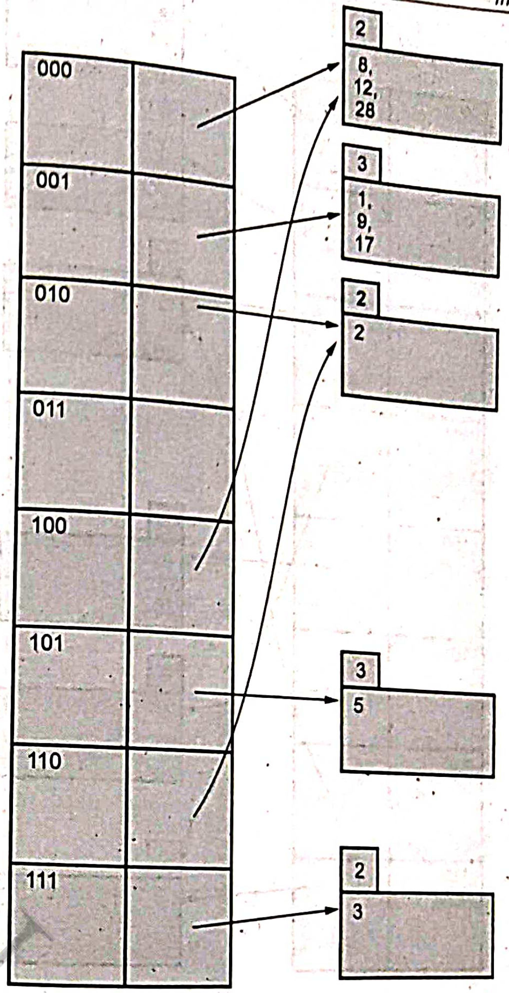
The extendible hash table will be,


00
01
10
11
2
8,
12,
28
2
1,
5,
9,
17
This bucket A is overflow as it exceeds its capacity of 3 elements
2
3

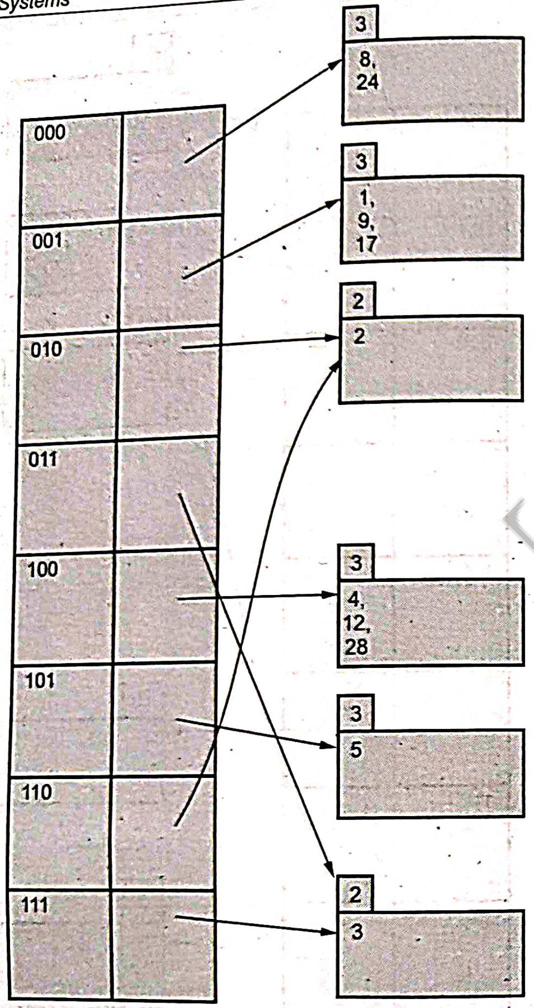
Hence we will extend the table by assuming the bit size as 3. The above indicated bucket A will split based on 001 and 101.



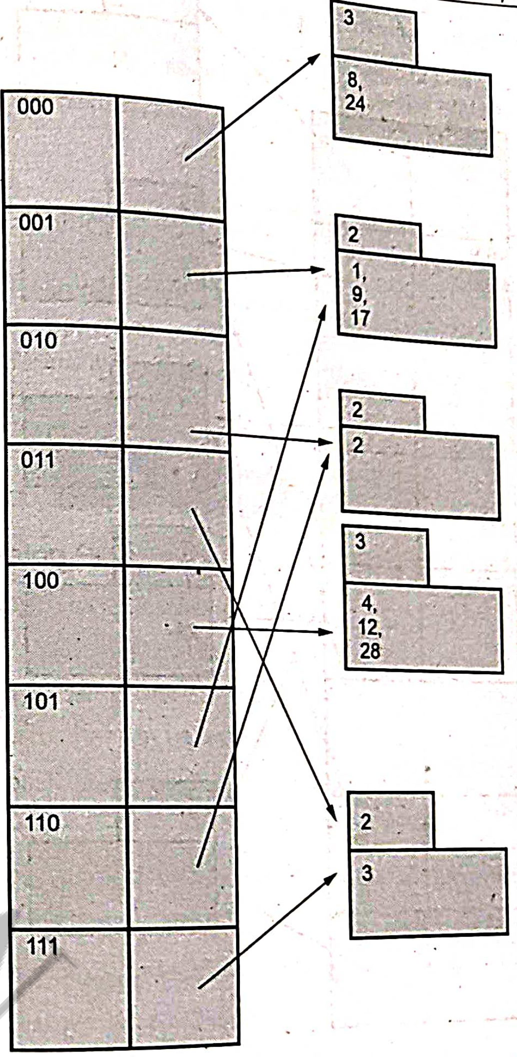
a) **Insert 2** will be 2 mod 8 = 2 = 010. If we consider last two digits i.e. 10 then there is no bucket. So we get,



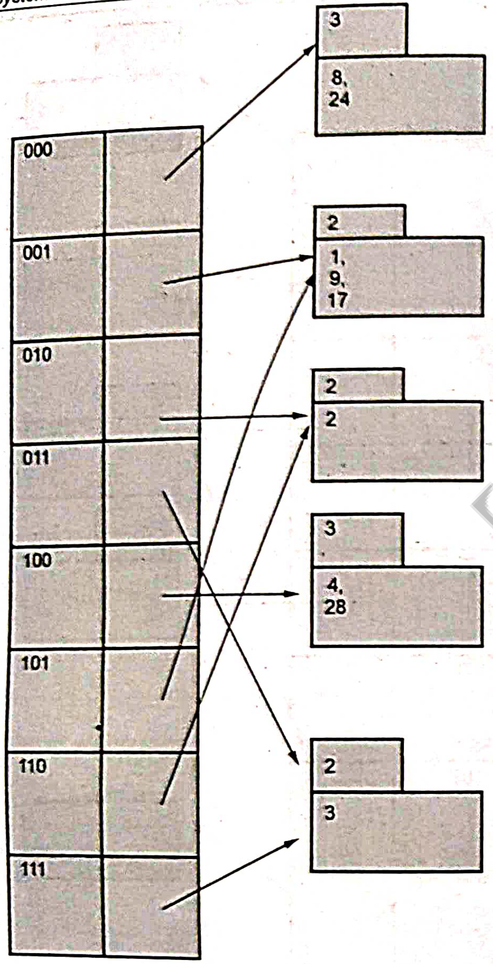
b) **Insert 24 :** 24 mod 8 = 0 = 000. The bucket in which 24 can be inserted is 8, 12, 28. But as this bucket is full we split it in two buckets based on digits 000 100.



c) **Delete 5 :** On deleting 5, we get one bucket pointed by 101 as empty. This will also result in reducing the local depth of the bucket pointed by 001. Hence we get,



(d) **Delete 12 :** We will simply delete 12 from the corresponding bucket there can not be any merging of buckets on deletion. The result of deletion is as given below -



Difference between Static and Dynamic Hashing

Sr. No.	Static Hashing	Dynamic Hashing
1.	The number of buckets are **fixed.**	The number of buckets are **not fixed.**
2.	**Chaining** is used	There is **no need of chaining.**
3.	Open hashing and Closed hashing are forms of static hashing.	Extendible hashing and linear hashing are forms of dynamic hashing.
4.	Space overhead is more.	**Minimum space** overhead due to dynamic nature.
5.	As **file grows** the **performance** of static hash function **decreases.**	There is **no degradation in performance** when the file grows.
6.	The bucket address table is not required.	**The bucket address table is required.**
7.	**The bucket is directly accessed.**	The bucket address table is used to access the bucket.

Review Questions

1. *What is hashing? Explain static hashing and dynamic hashing with an example.* **AU : May-18, Marks 13**
2. *Explain the distinction between static and dynamic hashing. Discuss the relative merits of each technique in database applications.* **AU : Dec.-17, Marks 13**
3. *Describe briefly static and dynamic hashing.* **AU : May-04, Marks 8, Dec.-08, Marks 6**
4. *Explain various hashing techniques.* **AU : May-07, Marks 8**

Part II : Query Processing

4.13 Query Processing Overview

AU : May-14,16,18, Dec.-19, Marks 16

- Query processing is a collection of activities that are involved in extracting data from database.
- During query processing there is translation high level database language queries into the expressions that can be used at the physical level of filesystem.
- There are three basic steps involved in query processing and those are -

1. **Parsing and Translation**
   - o In this step the query is translated into its internal form and then into **relational algebra.**
   - o Parser checks syntax and verifies relations.
   - o For instance - If we submit the query as,

```
SELECT RollNo, name
FROM Student
HAVING RollNo=10
```

Then it will issue a syntactical error-message as the correct query should be

```
SELECT RollNo, name
FROM Student
HAVING RollNo=10
```

Thus during this step the syntax of the query is checked so that only correct and verified query can be submitted for further processing.

2. **Optimization**
   - o During this process the query evaluation plan is prepared from all the relational algebraic expressions.
   - o The query cost for all the evaluation plans is calculated.
   - o Amongst all equivalent evaluation plans the one with lowest cost is chosen.
   - o Cost is estimated using statistical information from the database catalog, such as the number of tuples in each relation, size of tuples, etc.
3. **Evaluation**
   - o The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query.

The above describe steps are represented by following Fig. 4.13.1.

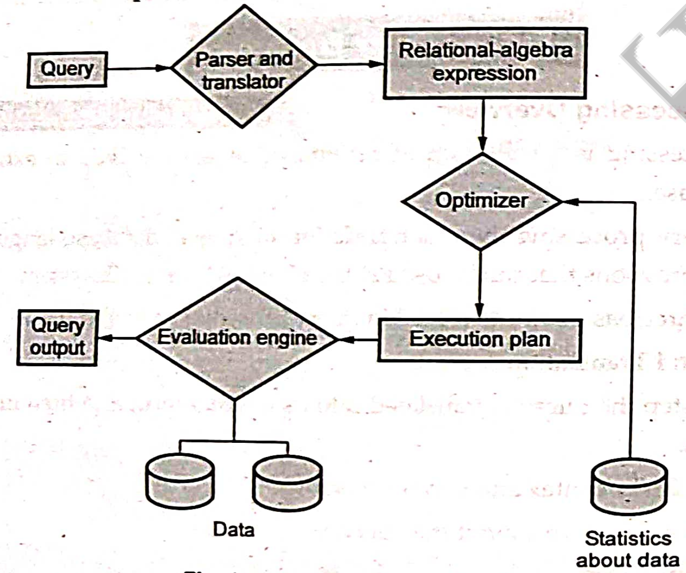


Fig. 4.13.1 : Query processing

For example - If the SQL query is,

```
SELECT balance
FROM account
WHERE balance<1000
```

Step 1 : This query is first verified by the **parser and translator** unit for correct syntax. If so then the relational algebra expressions can be obtained. For the above given queries there are two possible relational algebra

(1) $\sigma_{balance<1000}(\pi_{balance}(account))$

(2) $\pi_{balance}(\sigma_{balance<1000}(account))$

Step 2 : Query Evaluation Plan : To specify fully how to evaluate a query, we need not only to provide the relational-algebra expression, but also to annotate it with instructions specifying how to evaluate each operation. For that purpose, using the order of evaluation of queries, two query evaluation plans are prepared. These are as follows

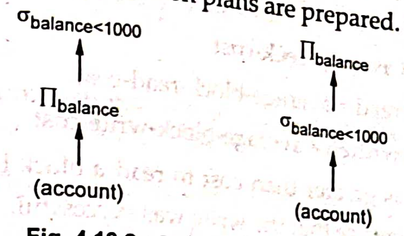


Fig. 4.13.2 : Query evaluation plans

Associated with each query evaluation plan there is a **query cost**. The query optimization selects the query evaluation plan having minimum query cost.

Once the query plan is chosen, the **query is evaluated** with that plan and the **result of the query is output**.

Review Questions

1. *Briefly explain about query processing.* **AU : May-14, 16, Marks 16**
2. *What is query optimization? Outline the steps in query optimization.* **AU : May-18, Marks 13**
3. *Sketch and concise the basic steps in query processing.* **AU : Dec.-19, Marks 3**

4.14 Measure of Query Cost

- There are multiple possible evaluation plans for a query, and it is important to be able to compare the alternatives in terms of their estimated cost and choose the best plan.
- There are many factors that contribute to query cost are -
  - o Disk access
  - o CPU time to execute the query
  - o Cost of communication in distributed and parallel database system.
- The cost of access data from disk is an important cost. Normally disk access is relatively slow as compared to in-memory operations. Moreover, the CPU speed is much faster than the disk speed. Hence the time spent in disk is relatively dominant factor in query execution.
- Computing CPU access time is comparatively harder, hence we will not consider the CPU time to execute the query.

- Similarly the cost of communication does not matter for simple large databases present in the centralized database system.
- Typically disk access is the predominant cost and is also relatively easy to estimate, taking into account :
  - o Number of seeks × average-seek-cost
  - o Number of blocks read × average-block-read-cost
  - o Number of blocks written × average-block-write-cost
- Cost to write a block is greater than cost to read a block because data is read back after being written to ensure that the write was successful.
- We use **number of block transfers** from disk and **number of disk seeks** to estimate the Query cost.
- Let,
  - o b be the number to blocks
  - o S be the number of Seeks
  - o t_T is average time required to transfer a block of data, in seconds
  - o t_S is average block access time in seconds.

Then query cost can be computed using following formula **b*tr+S*ts**

4.15 Algorithms for Selection, Sorting and join Operations **AU : Dec.-19, Marks 3**

4.15.1 Algorithm for Selection Operation

For selection operation, the file scan is an important activity. Basically file scan is a based on searching algorithms. These searching algorithms locate and retrieve the records that fulfills a selection condition.

Let us discuss various algorithms used for SELECT Operation based in file scan

Algorithm A1 : Linear Search

- Scan each file block and test all records to see whether they satisfy the selection condition

  Cost = b_r block transfers + 1 seek

Where,

b_r denotes number of blocks containing records from relation r

- If selection is on a key attribute, can stop on finding record

  Cost = (b_r/2) block transfers + 1 seek

- **Advantages of Linear Search**
  - o Linear search works even-if there is no selection condition specified.
  - o For linear search, there is no need to have records in the file in ordered form.
  - o Linear search works regardless of indices.

Algorithm A2 : Binary Search

- Applicable if selection is an equality comparison on the attribute on which file is ordered.
- Assume that the blocks of a relation are stored contiguously.
- Cost estimate is nothing but the number of disk blocks to be scanned.
- Cost of locating the first tuple by a binary search on the blocks = $\lceil \log_2(b_r) \rceil \times (t_r + t_s)$

Where,

b_r denotes number of blocks containing records from relation r

t_r is average time required to transfer a block of data, in seconds

t_s is average block access time, in seconds

- If there are multiple records satisfying the selection add transfer cost of the number of blocks containing records that satisfy selection condition.

4.15.2 Algorithm for Sorting Operation

External sorting

- In external sorting, the **data stored on secondary memory** is part by part loaded into main memory, sorting can be done over there.
- The sorted data can be then stored in the intermediate files. Finally these intermediate files can be merged repeatedly to get sorted data.
- Thus **huge amount** of data can be sorted using this technique.
- The external merge sort is a technique in which the data is loaded in intermediate files. Each intermediate file is sorted independently and then combined or merged to get the sorted data. **For example :** Consider that there are 10,000 records that has to be sorted. Clearly we need to apply external sorting method. Suppose main memory has a capacity to store 500 records in blocks, with each block size of 100 records.

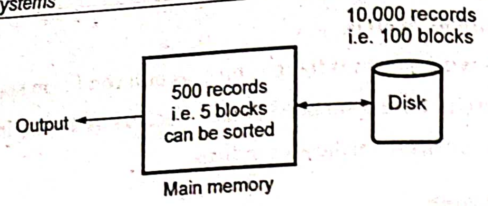


Fig. 4.15.1

The sorted 5 blocks (i.e. 500 records) are stored in intermediate file. This process will be repeated 20 times to get all the records sorted in chunks.

In the second step, we start merging a pair of intermediate files in the main memory to get output file.

Multiway merge

- Multiway merge sort is a technique of merging 'm' sorted lists into single sorted list.

  The **two-way** merge is a special case of multiway merge sort.

Let us understand it with the help of an example.

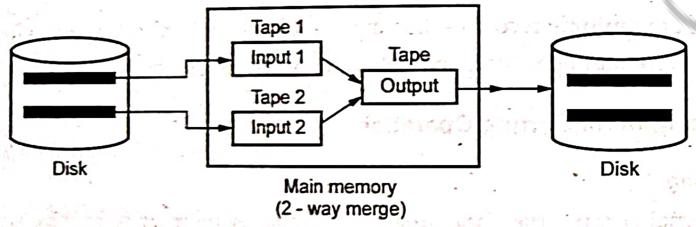


Fig. 4.15.2 : Two-way merge

Example 4.15.1 *Sort the following list of elements using two way merge sort with M = 3. 20, 47, 15, 8, 9, 4, 40, 30, 12, 17, 11, 56, 28, 35.*

Solution : As M = 3, we will break the records in the group of 3 and sort them. Then we will store them on tape. We will store data on alternate tapes.

Stage I : Sorting phase

1) 20, 47, 15. We arrange in sorted order → 15, 20, 47

Tb1 :	15	20	47

2) Read next three records sort them and store them on Tape Tb2

8, 9, 4 → 4, 8, 9

Tb2 :	4	8	9

3) Read next three records, sort them and store on tape Tb1.

40, 30, 12 → 12, 30, 40

Tb1 :	15	20	47	12	30	40

4) Read next three records, sort them and store on tape Tb2.

17, 11, 56 → 11, 17, 56

Tb2 :	4	8	9	11	17	56

5) Read next two remaining records, sort them and store on Tape Tb1

28, 35 → 28, 35

Tb1 :	15	20	47	12	30	40	28	35

At the end of this process we get

Tb1 :	15	20	47	12	30	40	28	35

Tb2 :	4	8	9	11	17	56

Stage II : Merging of runs

The input tapes Tb1 and Tb2 will use two more output tapes Ta1 and Ta2, for sorting. Finally the sorted data will be on tape Ta1.

Tb1 :	15	20	47	12	30	40	28	35

Tb2 :	4	8	9	11	17	56

We will read the elements from both the tapes Tb1 and Tb2, compare them and store on Ta1 in sorted order.

Ta1 :	4	8	9	15	20	47

Now we will read second blocks from Tb1 and Tb2. Sort the elements and store on Ta2.

Ta2 :	11	12	17	30	40	56

Finally read the third block from Tb1 and store in sorted manner on Ta1. We will not compare this block with Ta2 as there is no third block. Hence we will get

Ta1 :	4	8	9	15	20	47	28	35

Ta2 :	11	12	17	30	40	56

Now compare first blocks of Ta1 and Ta2 and store sorted elements on Tb1.

Tb1 :	4	8	9	11	12	15	17	20	30	40	47	56	

Tb2 :	28	35

Now both Tb1 and Tb2 contains only single block each. Sort the elements from both the blocks and store the result on Ta1.

Ta1 :	4	8	9	11	12	15	17	20	28	30	35	40	47	56

Thus we get **sorted list.**

Algorithm

Step 1 : Divide the elements into the blocks of size M. Sort each block and write runs on disk.

Step 2 : Merge two runs

i) Read first value on each of two runs

ii) Compare and sort

iii) Write on output tape.

Step 3 : Repeat the step 2 and get longer and longer runs on alternate tapes. Finally we will get single sorted list.

Analysis : The algorithm requires **log(N/M)** Passes plus initial run construction pass.

- Initially

  1st tape contains N records = M records * N/M runs.

  After storing the runs on two tapes, each contains half of the runs

Two tapes * M_records per run $*\frac{1}{2}$(N/M) = N records

- After merge 1st pass - double the length of runs, halve the number of runs

  Two tapes * 2M_records run $*\frac{1}{2}*\frac{1}{2}*$ (N,M) per runs = N records

- After merge 2nd pass

  Two tapes 4 M_records per run $*\frac{1}{4}*\frac{1}{2}*$ (N,M) runs = N records

- After merge S-th pass

  Two tapes * 2^S M records per run $*(1/2^S)$ (1/2) (N/M) runs = N records

- After the last merge, there will be only one run equal to whole file.

$$2^S M = N$$
$$2^S = N/M$$
$$S = \log (N/M)$$

Hence at each pass the N records are processed and we get the time complexity as O (N log (N/M)).

- The two way merge sort makes use of two input tapes and two output tapes for sorting the records.
- It works in two stages -

Stage 1 : Break the records into block. Sort individual record with the help of two input tapes.

Stage 2 : Merge the sorted blocks and create a single sorted file with the help of two output tapes.

4.15.3 Algorithm for Join Operation

- JOIN operation is the most time consuming operation to process.
- There are Several different algorithms to implement joins

1. Nested-loop join
2. Block nested-loop join
3. Indexed nested-loop join
4. Merge-join
5. Hash-join

Choice of a particular algorithm is based on cost estimate.

Algorithm For Nested Loop Join

This algorithm is for computing r $\bowtie_\theta$ s

Let, r is called the outer relation and s the inner relation of the join.

```
for each tuple $t_r$ in r do begin
    for each tuple $t_s$ in s do begin
    test pair $(t_r, t_s)$ to see if they satisfy the join condition $\theta$
    if $\theta$ is satisfied, then, add $(t_r, t_s)$ to the result.
    end
end
```

- This algorithm requires no indices and can be used with any kind of join condition.
- It is expensive since it examines every pair of tuples in the two relations.
- In the **worst case,** if there is enough memory only to hold one block of each relation, the estimated cost is
- $n_r \times b_s + b_r$ block transfers, plus $n_r + b_r$ seeks
- If the smaller relation fits entirely in memory, use that as the inner relation
  - o Reduces cost to $b_r + b_s$ block transfers and 2 seeks
- **For example -** Assume the query CUSTOMERS $\bowtie$ ORDERS (with join attribute only being CName)

Number of records of customer : 10000 order : 5000

Number of blocks of customer : 400 order : 100

Formula Used :

(1) $n_r \times b_s + b_r$ block transfers,

(2) $n_r + b_r$ seeks

r is outer relation and s is inner relation.

With order as outer relation :

$n_r = 5000$ $b_s = 400$, $b_r = 100$

$5000 \times 400 + 100 = 2000100$ block transfers and

$5000 + 100 = 5100$ seeks

With customer as the outer relation :

n_r=10000, b_s=100, b_r=400

$10000 \times 100 + 400 = 1000400$ block transfers and

10000+400 =10400 seeks

If smaller relation (order) fits entirely in memory, the cost estimate will be :

$b_r + b_s = 500$ block transfers

Algorithm For Block Nested Loop Join

Variant of nested-loop join in which every block of inner relation is paired with every block of outer relation.

This algorithm is for computing r $\bowtie\theta$ s

Let, r is called the outer relation and s the inner relation of the join.

```
for each block $B_r$ of r do
for each block $B_s$ of s do
for each tuple $t_r$ in $B_r$ do begin
        for each tuple $t_s$ in Bs do begin
        test pair $(t_r, t_s)$ to see if they satisfy the join condition $\theta$
        if $\theta$ is satisfied, then, add $(t_r, t_s)$ to the result.
        end
    end
  end
end
```

Worst case estimate: $b_r \times b_s + b_r$ block transfers + $2 \times b_r$ seeks

Each block in the inner relation s is read once for each block in the outer relation.

Best case : $b_r + b_s$ block transfers + 2 seeks

(3) Merge Join

- In this operation, both the relations are sorted on their join attributes. Then merged these sorted relations to join them.
- Only equijoin and natural joins are used.
- The cost of merge join is,

$b_r + b_s$ block transfers + $\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil$ seeks + the cost of sorting, if relations are unsorted.

(4) Hash Join

- In this operation, the hash function h is used to partition tuples of both the relations.
- h maps A values to {0, 1, ..., n}, where A denotes the attributes of r and s used in the join.
- Cost of hash join is :

$3(b_r + b_s) + 4 \times n$ block transfers $+ 2(\lceil b_r / b_b \rceil + \lceil b_s / b_b \rceil)$ seeks

If the entire build input can be kept in main memory no partitioning is required Cost estimate goes down to $b_r + b_s$.

Review Question

1. *With simple example explain the computing of Nested loop-join and block nested loop join.* **AU : Dec.-19, Marks 3**

4.16 Query Optimization using Heuristics - Cost Estimation

AU : Dec.-13,16, May-15, Marks 16

4.16.1 Heuristic Estimation

- Heuristic is a rule that leads to least cost in most of cases.
- Systems may use heuristics to reduce the number of choices that must be made in a cost-based fashion.
- Heuristic optimization transforms the query-tree by using a set of rules that typically t improve execution performance. These rules are
1. Perform selection early (reduces the number of tuples)
2. Perform projection early (reduces the number of attributes)
3. Perform most restrictive selection and join operations before other similar operations (such as cartesian product).
- Some systems use only heuristics, others combine heuristics with partial cost-based optimization.

Steps in Heuristic Estimation

Step 1 : Scanner and parser generate initial query representation

Step 2 : Representation is optimized according to heuristic rules

Step 3 : Query execution plan is developed

For example : Suppose there are two relational algebra -

(1) $\sigma_{city="Pune"}$ (π_{cname} Branch) $\bowtie$ Account $\bowtie$ Customer)

(2) $\pi_{cname}(\sigma_{city="Pune"}(\text{Branch} \bowtie \text{Account} \bowtie \text{Customer}))$

The query evaluation plan can be drawn using the query trees as follows -

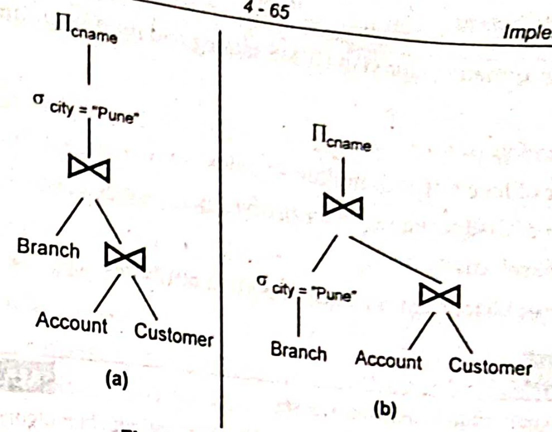


Fig. 4.16.1 : Query evaluation plan

Out of the above given query evaluation plans, the Fig. 4.16.1 (b) is much faster than Fig. 4.16.1 (a) because – in Fig. 4.16.1 (a) the join operation is among Branch, Account and Customer, whereas in Fig. 4.16.1 (b) the join of (Account and Customer) is made with the selected tuple for City="Pune" . Thus the output of entire table for join operation is much more than the join for some selected tuples. Thus we get choose the optimized query.

4.16.2 Cost based Estimation

- A cost based optimizer will look at all of the possible ways or scenarios in which a query can be executed.
- Each scenario will be assigned a 'cost', which indicates how efficiently that query can be run.
- Then, the cost based optimizer will pick the scenario that has the least cost and execute the query using that scenario, because that is the most efficient way to run the query.
- Scope of query optimization is a query block. Global query optimization involves multiple query blocks.
- Cost components for query execution
  - o Access cost to secondary storage
  - o Disk storage cost
  - o Computation cost
  - o Memory usage cost
  - o Communication cost

- Following information stored in DBMS catalog and used by optimizer
  - o File size
  - o Organization
  - o Number of levels of each multilevel index
  - o Number of distinct values of an attribute
  - o Attribute selectivity
- RDBMS stores histograms for most important attributes

Review Questions

1. *Explain query optimization with an example.* **AU : Dec,-16, Marks 8**
2. *Discuss about the Join order optimization and Heuristic optimization algorithms.* **AU : May-15, Marks 16**
3. *Give a detailed description about query processing and optimization. Explain the cost estimation of query optimization.* **AU : Dec.-13, Marks 16**

4.17 Two Marks Questions with Answers

Q.1 What is the need for RAID ? **AU : May-13**

Ans. : Refer section 4.1

Q.2 Define Software and hardware RAID systems. **AU : May-16**

Ans. : Hardware RAID : The hardware-based array manages the RAID subsystem independently from the host. It presents a single disk per RAID array to the host.
Software RAID : Software RAID implements the various RAID levels in the kernel disk code. It offers the cheapest possible solution, as expensive disk controller cards.

Q.3 What are ordered indices ? **AU : June-09,Dec. -11,17, May-14**

Ans. : This is type of indexing which is based on sorted ordering values. Various ordered indices are primary indexing, secondary indexing.

Q.4 What are the two types of ordered indices ? **AU : Dec.-06**

Ans. : Two types of ordered indices are - Primary indexing and secondary indexing. The primary indexing can be further classified into dense indexing and sparse indexing and single level indexing and multilevel indexing.

Q.5 Give the comparison between ordered indices and hashing. **AU : Dec.-06**

Ans. :

(1) If range of queries are common, ordered indices are to be used.

(2) The buckets containing records can be chained in sorted order in case of ordered indices.

(3) Hashing is generally better at retrieving records having a specified value of the key.

(4) Hash function assigns values randomly to buckets. Thus, there is no simple notion of "next bucket in sorted order."

Q.6 What are the causes of bucket overflow in a hash file organization ?

Ans. : Bucket overflow can occur for following reasons -

(1) **Insufficient buckets :** For the total number of buckets there are insufficient number of buckets to occupy.

(2) **Skew :** Some buckets are assigned more records than are others, so a bucket might overflow even while other buckets still have space. This situation is known as bucket skew.

Q.7 What can be done to reduce the occurrences of bucket overflows in a hash file organization ? **AU : May-07, June-09, Dce.-12**

Ans. :

(1) A bucket is a unit of storage containing one or more records (a bucket is typically a disk block).

(2) The file blocks are divided into M equal-sized buckets, numbered bucket0, bucket1... bucketM-1. Typically, a bucket corresponds to one (or a fixed number of) disk block.

(3) In a hash file organization we obtain the bucket of a record directly from its search-key value using a hash function, h (K).

(4) To reduce overflow records, a hash file is typically kept 70-80% full.

(5) The hash function h should distribute the records uniformly among the buckets; otherwise, search time will be increased because many overflow records will exist.

Q.8 Distinguish between dense and sparse indices. **AU : May-08, June-09**

Ans. : Refer section 4.7.

Q.9 When is it preferable to use a dense index rather than a sparse index ? Explain your answer. **AU : Dec. -11**

Ans. : 1. It is preferable to use a dense index instead of a sparse index when the file is not sorted on the indexed field.

2. Or when the index file is small compared to the size of memory.

Q.10 How does B-tree differs from a B+ tree ? Why is a B+ tree usually preferred as an access structure to a data file ? **AU : Dec.-08**

Ans. : Refer section 4.9.

Q.11 What are the disadvantages of B tree over B+ tree ? **AU : Dec.-16**

Ans. :

(1) Searching of a key value becomes difficult in B-tree as data can not be found in the leaf node.

(2) The leaf node can not store linked list and thus wastes the space.

Q.12 Mention different hashing techniques. **AU : May-12**

Ans. : Two types of hashing techniques are - i) Static hashing ii) Dynamic hashing.

Q.13 List the mechanisms to avoid collision during hashing. **AU : Dec.-16**

Ans. : Collision Resolution techniques are : (1) Separate chaining

(2) Open addressing techniques : (i) Linear probing (ii) Quadratic probing

Q.14 What is the basic difference between static hashing and dynamic hashing ? **AU : May-13, 15, Dec.-14, 15**

Ans. : Refer section 4.12.

Q.15 What is the need for query optimization ? **AU : May - 15**

Ans. : Query optimization is required for fast execution of long running complex queries.

Q.16 Which cost component are used most commonly as the basis for cost function. **AU : May-17, Dec.-19**

Ans. : Disk access or secondary storage access is considered most commonly as a basis for cost function.

Q.17 What is query execution plan ? **AU : May-17**

Ans. : To specify fully how to evaluate a query, we need not only to provide the relational-algebra expression, but also to annotate it with instructions specifying how to evaluate each operation. This annotated structure is called query execution plan.

Q.18 Mention all the operations of files. **AU : May-19**

Ans. : Various file operations are - (1) Creation of file (2) Insertion of data (3) Deletion of data (4) Searching desired data from the file.

Q.19 Define dense index. **AU : May-19**

Ans. : Refer section 4.7.

Q.20 How do you represent leaf node of a B+ tree of order p ? **AU : Dec.-19**

Ans. : To retrieve all the leaf pages efficiently we have to link them using page pointers. The sequence of leaf pages is also called as sequence set. Refer Fig. 4.8.1.

UNIT V

5 Advanced Topics

Syllabus

Distributed Databases : Architecture, Data Storage, Transaction Processing, Query processing and optimization - NOSQL Databases : Introduction - CAP Theorem - Document Based systems - Key value Stores - Column Based Systems - Graph Databases. Database Security : Security issues - Access control based on privileges. - Role Based access control - SQL Injection - Statistical Database security - Flow control - Encryption and Public Key infrastructures - Challenges.

Contents

5.1 Distributed Databases

AU : Dec.-04,07,16,17,19, May-03,06,14,16,17,19, Marks 16

Definition of distributed databases :

- A distributed database system consists of loosely coupled sites (computer) that share no physical components and each site is associated a database system.
- The software that maintains and manages the working of distributed databases is called distributed database management system.
- The database system that runs on each site is independent of each other. Refer Fig. 5.1.1.

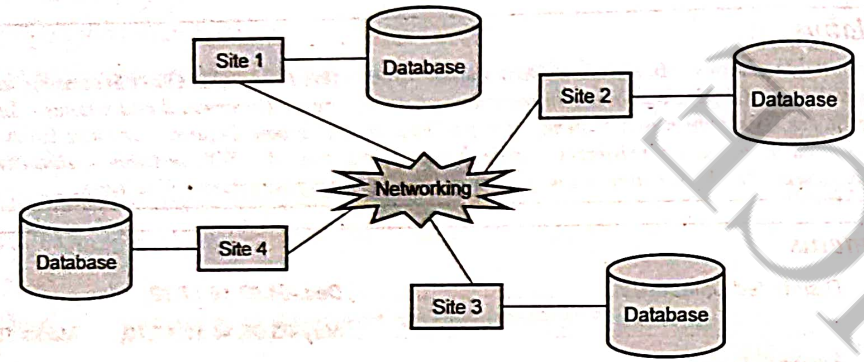


Fig. 5.1.1 : Distributed database system

The transactions can access data at one or more sites.

Advantages of distributed database system

(1) There is **fast data processing** as several sites participate in request processing.

(2) **Reliability and availability** of this system is high.

(3) It possess **reduced operating cost.**

(4) It is **easier to expand** the system by adding more sites.

(5) It has improved **sharing ability and local autonomy.**

Disadvantages of distributed database system

(1) The system becomes **complex** to manage and control.

(2) The **security issues** must be carefully managed.

(3) The system require **deadlock handling** during the transaction processing otherwise the entire system may be in inconsistent state.

(4) There is need of some **standardization** for processing of distributed database system.

Difference between distributed DBMS and centralized DBMS

Sr. No.	Distributed DBMS	Centralized DBMS
1.	The database files are stored at geographically different locations across the network.	The database is stored at centralized location.
2.	As data is distributed over the network, it requires time to synchronize data and thus difficult to maintain.	A centralized database is easier to maintain and keep updated since all the data are stored in a single location.
3.	If one database fails, user can have access to other database files.	If the centralized database fails, then there is no access to a database.
4.	It can have data replication as database is distributed. Hence there can be some data inconsistency.	It have single database system, hence there is no data replication. Therefore there is no data inconsistency.

Uses of distributed system :

(1) Often distributed databases are used by organizations that have numerous offices in different geographical locations. Typically an individual branch is interacting primarily with the data that pertain to its own operations, with a much less frequent need for general company data. In such a situation, distributed systems are useful.

(2) Using distributed system, one can give permissions to single sections of the overall database, for better internal and external protection.

(3) If we need to add a new location to a business, it is simple to create an additional node within the database, making distribution highly scalable.

5.1.1 Types of Distributed Databases

There are two types of distributed databases -

(1) Homogeneous databases

- The homogeneous databases are kind of database systems in which all sites have identical software running on them. Refer Fig. 5.1.2.

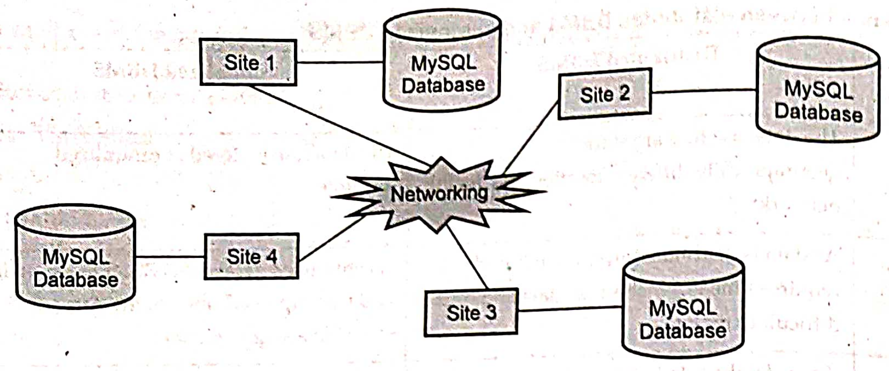


Fig. 5.1.2 : Homogeneous databases

- In this system, all the sites are aware of the other sites present in the system and they all cooperate in processing user's request.
- Each site present in the system, surrenders part of its autonomy in terms of right to change schemas or software.
- The homogeneous database system appears as a single system to the user.

(2) Heterogeneous databases

- The heterogeneous databases are kind of database systems in which different sites have different schema or software. Refer Fig. 5.1.3.

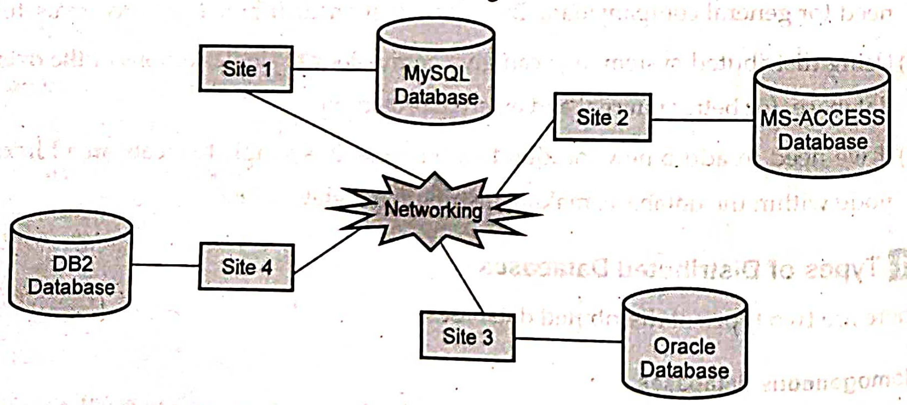


Fig. 5.1.3 : Heterogeneous databases

- The participating sites are not aware of other sites present in the system.
- These sites provide limited facilities for cooperation in transaction processing.

5.1.2 Architecture

- Following is an architecture of distributed databases. In this architecture the local database is maintained by each site.
- Each site is interconnected by communication network.

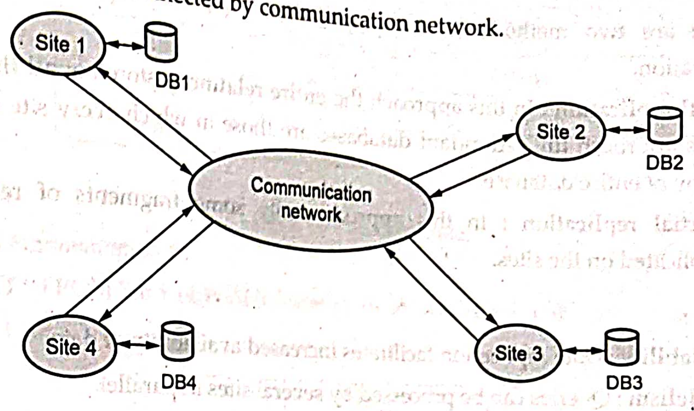
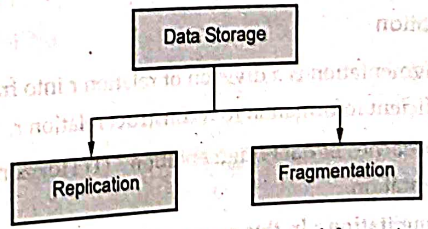


Fig. 5.1.4 : Distributed database architecture

When user makes a request for particular data at site Si then it is first searched at the local database. If the data is not present in the local database then the request for that data is passed to all the other sites via communication network. Each site then searches for that data at its local database. When data is found at particular site say Sj then it is transmitted to site Si via communication network.

5.1.3 Data Storage

There are two approaches of storing relation r in distributed database -



(1) Replication : System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.

(2) Fragmentation : Relation is partitioned into several fragments stored in distinct sites.

5.1.3.1 Data Replication

- **Concept :** Data replication means storing a copy or replica of a relation fragments in two or more sites.
- There are two methods of data replication – (1) Full replication (2) Partial replication.
  - o **Full replication :** In this approach the **entire relation** is stored at **all the sites.** In this approach full redundant databases are those in which every site contains a copy of entire database.
  - o **Partial replication :** In this approach only some fragments of relation are replicated on the sites.

Advantages :

(1) Availability : Data replication facilitates increased availability of data.

(2) Parallelism : Queries can be processed by several sites in parallel.

(3) Faster accessing : The relation r is locally available at each site, hence data accessing becomes faster.

Disadvantages :

(1) Increased cost of update : The major disadvantage of data replication is increased cost of updated. That means each replica of relation r must be updated from all the sites if user makes a request for some updates in relation.

(2) Increased complexity in concurrency control : It becomes complex to implement the concurrency control mechanism for all the sites containing replica.

5.1.3.2 Data Fragmentation

- **Concept :** Data fragmentation is a division of relation r into fragments r1,r2, r3,...,rn which contain sufficient information to reconstruct relation r.
- There are two approaches of data fragmentation - (1) Horizontal fragmentation and (2) Vertical fragmentation.
  - o **Horizontal fragmentation :** In this approach, each tuple of r is assigned to one or more fragments. If relation R is fragmented in r1 and r2 fragments, then to bring these fragments back to R we must use union operation. That means $R = r1 \cup r2$
  - o **Vertical fragmentation :** In this approach, the relation r is fragmented based on one or more columns. If relation R is fragmented into r1 and r2 fragments using vertical fragmentation then to bring these fragments back to original relation R we must use join operation. That means $R = r1 \bowtie r2$

- For example - Consider following relation r

  Student(RollNo, Marks,City)

  The values in this schema are inserted as

RollNo	Marks	City
R101	55	Pune
R102	66	Chennai
R103	77	Mumbai

Fig. 5.1.5 : Student table

Horizontal Fragmentation 1 :

SELECT * FROM Student WHERE Marks >50 AND City='Pune'

We will get

R101	55	Pune

Horizontal Fragmentation 2 :

SELECT * FROM Student WHERE Marks >50 AND City='Mumbai'

We will get

R103	77	Mumbai

Vertical Fragmentation 1 :

SELECT * FROM RollNo

R101
R102
R103

Vertical Fragmentation 2 :

SELECT * FROM city

Pune
Chennai
Mumbai

5.1.4 Transaction Processing

5.1.4.1 Basic Concepts

In distributed system transaction initiated at one site can access or update data at other sites. Let us discuss various basic concepts used during transaction processing in distributed systems -

- **Local and global transactions :**

Local transaction Ti is said to be local if it is initiated at site Si and can access or update data at site Si only.

Global transaction Ti initiated by site Si is said to be global if it can access or update data at site Si, Sj,Sk and so on.

- **Coordinating and participating sites :**

The site at which the transaction is initiated is called **coordinating** site. The participating sites are those sites at which the sub-transactions are executing. For example - If site S1 initiates the transaction T1 then it is called **coordinating site.** Now assume that transaction T1(initiated at S1) can access site S2 and S3. Then sites S2 and S3 are called **participating sites.**

To access the data on site S2, the transaction T1 needs another transaction T12 on site S2 similarly to access the data on site S3,the transaction T2 needs some transaction say T13 on site S3. Then transactions T12 and T13 are called sub-transactions. The above described scenario can be represented by following Fig. 5.1.6.

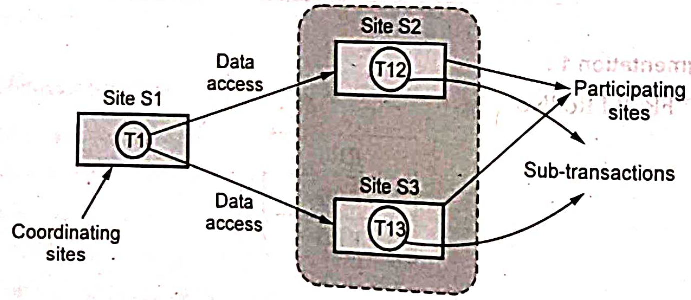


Fig. 5.1.6 : Distributed transaction scenario

- **Transaction manager :**

The transaction manager manages the execution of those transactions (or subtransactions) that access data stored in a local site.

The tasks of transaction manager are -

(1) To maintain the log for recovery purpose.

(2) Participating in coordinating the concurrent execution of the transactions executing at that site.

- **Transaction coordinator :**

The transaction coordinator coordinates the execution of the various transactions (both local and global) initiated at that site.

The tasks of Transaction coordinator are -

(1) Starting the execution of transactions that originate at the site.

(2) Distributing subtransactions at appropriate sites for execution

Let TC denotes the transaction coordinator and TM denotes the transaction manager, then the system architecture can be represented as,

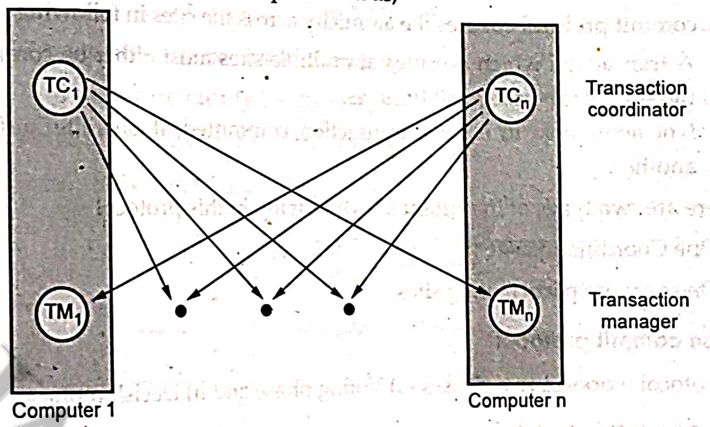


Fig. 5.1.7 : System architecture

5.1.4.2 Failure Modes

There are four types of failure modes,

1. Failure of site	2. Loss of messages
3. Failure of communication link	4. Network partition

The most common type of failure in distributed system is loss or corruption of messages. The system uses Transmission Control Protocol(TCP) to handle such error. This is a standard **connection oriented** protocol in which message is transmitted from one end to another using wired connection.

- If two nodes are not directly connected, messages from one to another must be routed through sequence of **communication links.** If the communication link fails, the messages are rerouted by alternative links.
- A system is partitioned if it has been split into two subsystems. This is called partitions. Lack of connection between the subsystems also cause failure in distributed system.

5.1.4.3 Commit Protocols

Two Phase Commit Protocol

- The atomicity is an important property of any transaction processing. What is this atomicity property ? This property means either the transaction will execute completely or it won't execute at all.
- The commit protocol ensures the atomicity across the sites in following ways -
  - i) A transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - ii) Not acceptable to have a transaction committed at one site and aborted at another.
- There are two types of important sites involving in this protocol -
  - o One Coordinating site
  - o One or more participating sites.

Two phase commit protocol

This protocol works in two phases - i) Voting phase and ii) Decision phase.

Phase 1 : Obtaining decision or voting phase

Step 1 : Coordinator site Ci asks all participants to **prepare** to commit transaction Ti.

- o Ci adds the records **<prepare T>** to the log and writes the log to stable storage.
- o It then sends **prepare T** messages to all participating sites at which T will get executed.

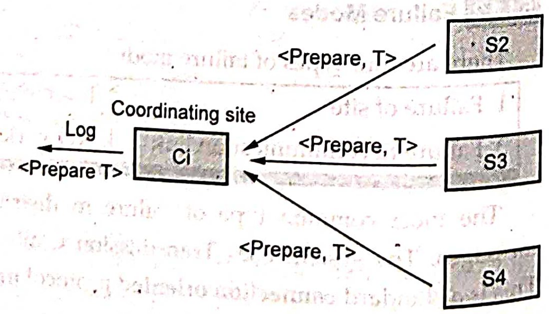


Fig. 5.1.8

Step 2 : Upon receiving message, transaction manager at participating site determines if it can commit the transaction

- o If not, add a record **<no T>** to the log and send **abort** T message to coordinating site Ci.

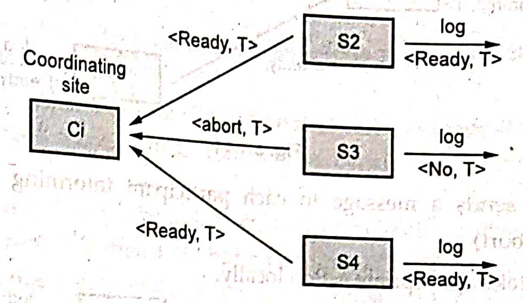


Fig. 5.1.9

- o If the transaction can be committed, then :
  - Add the record **<ready T>** to the log
  - Force all records for T to stable storage
  - Send **ready** T message to Ci

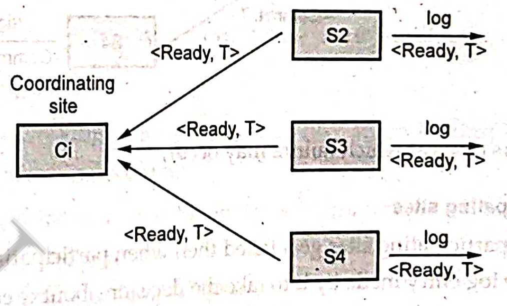


Fig. 5.1.10

Phase 2 : Recoding decision phase

- T can be committed of Ci received a ready T message from all the participating sites : otherwise T must be aborted.
- Coordinator adds a decision record, <commit T> or <abort T>, to the log and forces record onto stable storage. Once the record stable storage it is irrevocable (even if failures occur)

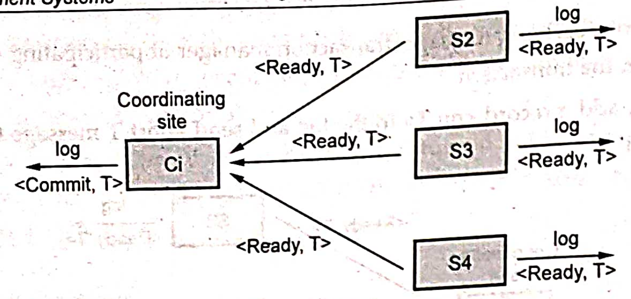
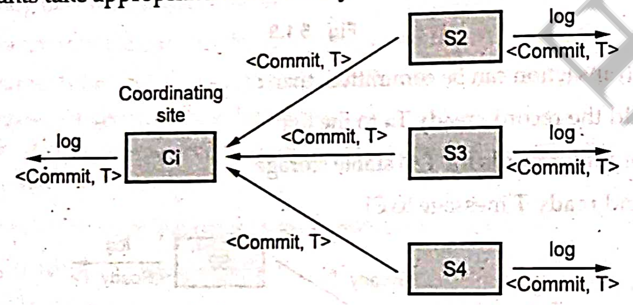


Fig. 5.1.11

- Coordinator sends a message to each participant informing it of the decision (commit or abort)
- Participants take appropriate action locally.



Failure of site

There are various cases at which failure may occur,

(1) Failure of participating sites

- If any of the participating sites gets failed then when participating site Si recovers, it examines the log entry made by it to take the decision about executing transaction.
  - o If the log contains **<commit T>** record : participating site executes **redo** (T)
  - o If the log contains **<abort T>** record : participating site executes **undo** (T)
  - o If the log contains **<ready T>** record : participating site must consult *Coordinating site* to take decision about execution of transaction T.
    - ▪ If T committed, **redo** (T)
    - ▪ If T aborted, **undo** (T)
- If the log of participating site contains no record then that means Si gets failed before responding to **Prepare T** message from coordinating site. In this case it must **abort T**

(2) Failure of coordinator

- If coordinator fails while the commit protocol for T is executing then participating sites must take decision about execution of transaction T :
  i) If an active participating site contains a **<commit T>** record in its **log**, then **T must be committed.**
  ii) If an active participating site contains an **<abort T>** record in its **log**, then **T must be aborted.**
  iii) If some active participating site does not contain a **<ready T>** record in its log, then the failed coordinator Ci cannot have decided to commit T. Can therefore abort T.
  iv) If none of the above cases holds, then all participating active sites must have a **<ready T>** record in their logs, but no additional control records (such as <abort T> of <commit T>). In this case active sites **must wait** for coordinator site Ci to recover, to find decision.

Two phase locking protocol has blocking problem.

What is blocking problem ?

It is a stage at which active participating sites may have to wait for failed coordinator site to recover.

The solution to this problem is to use three phase locking protocol.

Three Phase Commit Protocol

- The three phase locking is an extension of two phase locking protocol **in which eliminates the blocking problem.**
- Various **assumptions** that are made for three phase commit protocol are -
  - o No network partitioning.
  - o At any point at least one site must be up.
  - o At the most k sites(participating as well as coordinating) can fail.
- **Phase 1 :** This phase is similar to phase 1 of two phase protocol. That means Coordinator site Ci asks all participants to **prepare** to commit transaction Ti. The coordinator then makes the decision about commit or abort based on the response from all the participating sites.
- **Phase 2 :** In phase 2 coordinator makes a decision as in 2Phase Commit which is called the **pre-commit** decision **<Pre-commit, T>**, and records it in multiple (at least K) participating sites.

- **Phase 3 :** In phase 3, coordinator sends commit/abort message to all participating sites.
- Under three phase protocol, the knowledge of pre-commit decision can be used to commit despite coordinator site failure. That means if the coordinating site in case gets failed then one of the participating site becomes the coordinating site and consults other participating sites to know the **Pre-commit** message which they possess. Thus using this pre-commit message the decision about commit/abort is taken by this new coordinating site.
- This protocol avoids blocking problem as long as less than k sites fail.

Advantage of three phase commit protocol

(1) It avoid blocking problem.

Disadvantage of three phase commit protocol

(1) The overhead is increased.

5.1.5 Query Processing and Optimization

Distributed database query is processed using following steps –

(1) Query Mapping :

- The input query on distributed data is specified using **query language**.
- This query language is then translated into **algebraic query**.
- During this translation the **global conceptual schema** is referred.
- During the translation some important actions such as normalization, analysis for semantic errors, simplification are carried out then input query is restructured into algebraic query.

(2) Localization :

- In this step, the replication of information is handled.
- The distributed query is mapped on the global schema to separate queries on individual fragments.

(3) Global Query Optimization : optimization means selecting a strategy from list of candidate queries which is closest to optimal. For optimization, the cost is computed. The total cost is combination of CPU cost, I/O cost and communication costs.

(4) Local Query Optimization : This step is common to all sites in distributed database. The techniques of local query optimization are similar to those used in centralized systems.

Review Questions

1. *What are the various features of distributed database versus centralized database system?* **AU : Dec.-17, Marks 6, May-17, Marks 8**
2. *Explain the architecture of a distributed database.* **AU : Dec.-16, Marks 7**
3. *Explain about distributed databases and their characteristics, functions and advantages and disadvantages.* **AU : Dec.-07, May-14, Marks 8, May-16, Marks 16**
4. *Explain design of distributed database.* **AU : Dec.-04, Marks 8**
5. *Discuss homogeneous and heterogeneous databases reference to distributed databases.* **AU : May-03, Marks 8**
6. *Discuss in detail about the distributed databases.* **AU : May-19, Marks 13**
7. *What are data fragmentations ? Explain various approaches for fragmenting a relation with example.* **AU : May-06, Marks 6**
8. *Explain in detail various approaches used for storing a relation in distributed databases.* **AU : Dec.-04, Marks 8, Dec.-19, Marks 9**

5.2 NOSQL Databases

5.2.1 Introduction

- NoSQL stands for not only SQL.
- It is nontabular database system that store data differently than relational tables. There are various types of NoSQL databases such as document, key-value, wide column and graph.
- Using NoSQL we can maintain flexible schemas and these schemas can be scaled easily with large amount of data.

5.2.2 Need

The NoSQL database technology is usually adopted for following reasons -

1) The NoSQL databases are often used for handling big data as a part of fundamental architecture.
2) The NoSQL databases are used for storing and modelling structured, semi-structured and unstructured data.
3) For the efficient execution of database with high availability, NoSQL is used.
4) The NoSQL database is non-relational, so it scales out better than relational databases and these can be designed with web applications.
5) For easy scalability, the NoSQL is used.

5.2.3 Features

1) The NoSQL does not follow any relational model.
2) It is either schema free or have relaxed schema. That means it does not require specific definition of schema.
3) Multiple NoSQL databases can be executed in distributed fashion.
4) It can process both unstructured and semi-structured data.
5) The NoSQL have higher scalability.
6) It is cost effective.
7) It supports the data in the form of key-value pair, wide columns and graphs.

5.2.4 Comparison between RDBMS and NoSQL

Sr. No.	RDBMS	NoSQL
1.	The relational database system is based on relationships among the tables.	It is non-relational database system. It can be used in distributed environment.
2.	It is vertically scalable.	It is horizontally scalable.
3.	It has predefined schema.	It does not have schema or it may have relaxed schema.
4.	It uses SQL to query the database.	It uses unstructured query language.
5.	It is a table based database.	It is document based, graph based or key-value pair.
6.	It emphasizes on ACID properties (Automicity, consistency, isolation and durability)	It follows Brewers CAP theorem (Consistency, availability and partition tolerance)
7.	Schema is fixed or rigid.	Schema is dynamic.
8.	Pessimistic.	Optimistic.
9.	Examples : MySQL,Oracle, PostgreSQL.	Examples : MangoDB, BigTable, Redis.

Review Question

1. *What is NoSQL ? What is the need for it. Enlist various feature of NoSQL.*

5.3 CAP Theorem

- Cap theorem is also called as **brewer's theorem.**
- The CAP theorem is comprised of three components (hence its name) as they relate to distributed data stores :
  - o **Consistency :** All reads receive the most recent write or an error.
  - o **Availability :** All reads contain data, but it might not be the most recent.
  - o **Partition tolerance :** The system continues to operate despite network failures (i.e.; dropped partitions, slow network connections, or unavailable network connections between nodes.)
- The CAP theorem states that it is not possible to guarantee all three of the desirable properties - Consistency, availability and partition tolerance at the same time in a distributed system with data replication.

Review Question

1. *Write a short note on CAP theorem.*

5.4 Types of NoSQL Database

There are four types of NoSQL databases and those are -

1. **Key-value store**
2. **Document store**
3. **Graph based**
4. **Wide column store**

Let us discuss them in detail.

5.4.1 Key-Value Store

- Key-value pair is the simplest type of NoSQL database.
- It is designed in such a way to handle lots of data and heavy load.
- In the key-value storage the key is unique and the value can be JSON, string or binary objects.
- For example -

```
{Customer:
   [
     {"id":1, "name":"Ankita"},
     {"id":2,"name":"Kavita"}
   ]
}
```

Here **id, name** are the keys and 1,2, "Ankita", "Prajkta" are the values corresponding to those keys.

Key-value stores help the developer to store schema-less data . They work best for Shopping cart contents.

The DynamoDB, Riak, Redis are some famous examples of key-value store.

5.4.2 Document Based Systems

- The document store make use of key-value pair to store and retrieve data.
- The document is stored in the form of XML and JSON.
- The document stores appear the most natural among NoSQL database types.
- It is most commonly used due to flexibility and ability to query on any field.
- For example -

```
{
"id" : 101,
"Name" : "AAA",
"City" : "Pune"
}
```

MongoDB and CouchDB are two popular document oriented NoSQL database.

5.4.3 Column Based Systems

- The column store model is similar to traditional relational database. In this model, the columns are created for each row rather than having predefined by the table structure.
- In this model number of columns are not fixed for each record.
- Columns databases can quickly aggregate the value of a given column.
- For example -

Row ID	Columns...		
1	**Name**	**City**	
	Ankita	Pune	
2	**Name**	**City**	**email**
	Kavita	Mumbai	kavita123@gmail.com

The column store databases are widely used to manage data warehouses, business **intelligence**, HBase, Cassandra are examples of column based databases.

5.4.4 Graph Databases

The graph database is typically used in the applications where the relationships among the data elements is an important aspect.

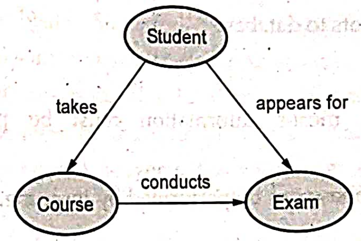
The connections between elements are called **links or relationships.** In a graph database, connections are first-class elements of the database, stored directly. In relational databases, links are implied, using data to express the relationships.

The graph database has two components -

1) **Node :** The entities itself. For example - People, student.

2) **Edge :** The relationships among the entities.

For example -



Graph base database is mostly used for social networks, logistics, spatial data. The graph databases are - Neo4J, Infinite Graph,OrientDB.

5.5 Database Security

- **Definition :** Database security is a technique that deals with protection of data against unauthorized access and protection.
- Database security is an important aspect for any database management system as it deals with sensitivity of data and information of enterprise.
- Database security allows or disallows users from performing actions on the database objects.

5.6 Security Issues

5.6.1 Types of Security

Database security addresses following issues -

(1) Legal Issues : There are many legal or ethical issues with respect to **right to access information.** For example - If some sensitive information is present in the database, then it must not be accessed by unauthorized person.

(2) **Policy Issues :** There are some government or organizational policies that tells us what kind of information should be made available to access publicly.

(3) **System Issues :** Under this issue, it is decided whether security function should be handled at **hardware level** or at **operating system level** or at **database level.**

(4) **Data and User Level Issues :** In many organizations, multiple security levels are identified to categorize data and users based on these classifications. The security policy of organization must understand these levels for permitting access to different levels of users.

5.6.2 Threats to Database

Threats to database will result in loss or degradation of data. There are three kinds of loss that occur due to threats to database

(1) Loss of Integrity :

- Database integrity means information must be protected from improper modification.
- Modification to database can be performed by inserting, deleting or modifying the data.
- Integrity is lost if unauthorized changes are made to data intentionally or accidently.
- If data integrity is not corrected and work is continued then it results in inaccuracy, fraud, or erroneous decision.

(2) Loss of Availability :

- Database availability means making the database objects available to authorized users.

(3) Loss of Confidentiality :

- Confidentiality means protection of data from unauthorized disclosure of information.
- The loss of confidentiality results in loss of public confidence, or embarrassment or some legal action against organization.

5.6.3 Control Measures

- There are **four major control measures** used to provide security on data in database

1. **Access control**
2. **Interface control**
3. **Flow control**
4. **Data encryption**

- **Access Control :** The most common security problem is unauthorized access to computer system. Generally this access is for obtaining the information or to make malicious changes in the database. The security mechanism of a DBMS must include provisions for restricting access to the database system as a whole. This function, called **access control.**
- **Inference Control :** This method is used to provide the security to statistical database security problems. Statistical databases are used to provide statistical information based on some criteria. These databases may contain information about particular age group, income-level, education criteria and so on. Access to some sensitive information must be avoided while using the statistical databases. The corresponding measure that prevents the user from completing any inference channel.
- **Flow Control :** It is a kind of control measure which prevents information from flowing in such a way that it reaches unauthorized users. Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization are called **covert channels.**
- **Data Encryption :** The data encryption is a control measure used to secure the sensitive data. In this technique, the data is **encoded** using some **coding algorithm.** An unauthorized user who accesses encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher the data.

5.6.4 Database Security and DBA

- DBA stands for **Database Administrator,** who is the central authority for managing the database system.
- DBA is responsible for **granting privileges** to users who want to use the database system.
- The DBA has a DBA account in the DBMS which is sometimes called as **system or superuser account.** It provides powerful capabilities that are not made available for regular database accounts and users.

- DBA makes use of some special commands that perform following type of actions -

1. **Account Creation :** This command helps in creating a new account and password for a single user or for group of users.
2. **Privilege Granting :** This command allows the DBA to grant privileges to certain accounts.
3. **Privilege Revocation :** This command allows the DBA to cancel the privileges to certain accounts.
4. **Security Level Assignment :** This action assigns user account to the appropriate security clearance level.

Thus the DBA is responsible for the overall security of the database system.

5.7 Access Control Based on Privileges or Discretionary Access Control

- Discretionary Access Control (DAC) is a access control mechanism based on privileges.
- **Types of discretionary privileges :** The DBMS must provide selective access to each relation in the database on specific accounts. This selective access is known as **privileges.** There are two levels for assigning privileges for using Database systems and these are -
  - o **The account level :** At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
  - o **Relation(or table) level :** At this level, the DBA can control the privilege to access each individual relation or view in the database.
- For granting the privileges, the access control mechanism follows an authorization model for discretionary privileges known as **the access matrix model.**
- The access matrix is a table with rows and columns. It defines the access permissions.
  - o The rows of a matrix M represent subjects (users, accounts, programs)
  - o The columns represent objects (relations, records, columns, views, operations).
  - o Each position M(i, j) in the matrix represents the types of privileges (read, write, update) that subject i holds on object j.
  - o For example -

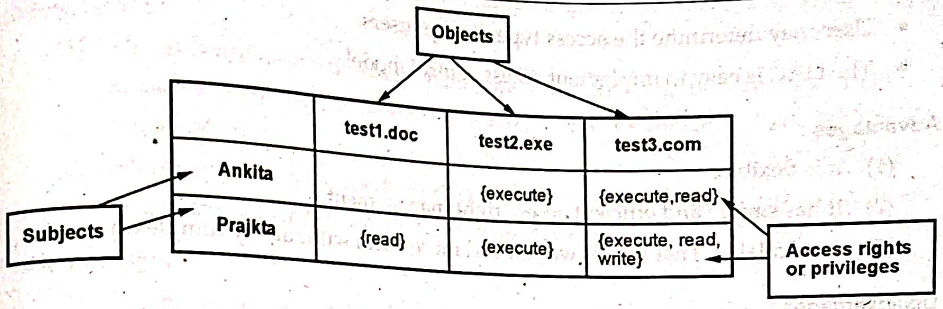


Fig. 5.7.1 : Access matrix model

- Discretionary Access Control allows each user or subject to control access to their own data.
- In DAC, owner of resource restricts access to the resources based on the identity of users.
- DAC is typically the default access control mechanism for most desktop operating systems.
- Each resource object on DAC based system has Account Control List (ACL) associated with it.
- An ACL contains a list of users and groups to which the user has permitted access together with the level of access for each user or group.
- For example - The ACL is an object centered description of access rights as follows -

test1.doc: {Prajka: read}
test2.exe: {Ankita: execute}, {Prajkta: execute}
test3.com: {Ankita: execute, read}, {Prajkta: execute, read,write}

- Object access is determined during Access Control List (ACL) authorization and based on user identification and/or group membership.
- Under DAC a user can only set access permissions for resources which they already own.
- Similarly a hypothetical user A cannot change the access control for a file that is owned by user B. User A can, however, set access permissions on a file that he/she owns.
- User may transfer object ownership to another user(s).

- User may determine the access type of other users.
- The DAC is easy to implement access control model.

Advantages :

(1) It is flexible.

(2) It has simple and efficient access right management.

(3) It is scalable. That means we can add more users without any complexity.

Disadvantages :

(1) It increases the risk that data will be made accessible to users that should not necessarily be given access.

(2) There is no control over information flow as one user can transfer ownership to another user.

5.8 Role Based Access Control

- It is based on the concept that privileges and other permissions are associated with organizational roles, rather than individual users. Individual users are then assigned to appropriate roles.
- For example, an accountant in a company will be assigned to the **Accountant role,** gaining access to all the resources permitted for all accountants on the system. Similarly, a software engineer might be assigned to the **Developer role.**
- In an RBAC system, the roles are centrally managed by the administrator. The administrators determine what roles exist within their companies and then map these roles to job functions and tasks.
- Roles can effectively be implemented using security groups. The security groups are created representing each role. Then permissions and rights are assigned to these groups. Next, simply add the appropriate users to the appropriate security groups, depending on their roles or job functions.
- A user can have more than one role. And more than one user can have the same role.
- Role hierarchies can be used to match natural relations between roles. For example - A lecturer can create a role student and give it a privilege "read course material".
- Role Based Access Control (RBAC), also known as **non discretionary access control.**
- RBAC security strategy is widely used by most organizations for deployment of commercial and off-the-shelf products.

Advantages :

(1) The security is more easily maintained by limiting unnecessary access to sensitive information based on each user's established role within the organization.

(2) All the roles can be aligned with the organizational structure of the business and users can do their jobs more efficiently and autonomously.

Disadvantages :

(1) It is necessary to understand each user's functionality in depth so that roles can be properly assigned.

(2) If roles are not assigned properly then inappropriate access right creates security severe problems for database system.

5.9 SQL Injection

- SQL injection is a **type of code injection technique** that might destroy the databases.
- In this technique the malicious code in SQL statement is placed via web page input. These statements control a database server behind a web application.
- Attackers can use SQL injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL injection to add, modify and delete records in the database.
- An SQL injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server or others.

How SQL Injection Works ?

- To make an SQL injection attack, an attacker must first find vulnerable user inputs within the web page or web application. A web page or web application that has an SQL injection vulnerability uses such user input directly in an SQL query. The attacker can create input content. Such content is often called a **malicious payload** and is the key part of the attack. After the attacker sends this content, malicious SQL commands are executed in the database.
- SQL is a query language that was designed to manage data stored in relational databases. You can use it to access, modify and delete data. Many web applications and websites store all the data in SQL databases. In some cases, you can also use SQL commands to run operating system commands. Therefore, a successful SQL Injection attack can have very serious consequences.

- **Example of SQL Injection**
  - o Following is an example of SQL injection vulnerability works around a simple web application having two input fields - One for user name and another for password.
  - o This example has a table named users with the columns username and password

```
uname=request.POST['username']
passwd=request.POST['password']
query="SELECT id FROM users WHERE username=' "+ uname +" ' AND
                                      password =' "+ passwd +" '
database.execute(query)
```

  - o Here the two input fields - One for user name and another for password is vulnerable to SQL injection.
  - o The attacker can attack using these fields and alter the SQL query to get the access to the database.
  - o They could use a trick on password field. They could add

    OR 1 = 1

    Statement to the password field.
  - o As a result the query would becomes (assuming username as 'user1'and password ='password') -
  - o SELECT id FROM users WHERE username='user1' AND password='password' OR 1 = 1
  - o Because of **OR 1 = 1** statement, the WHERE clause returns the first id from the users table no matter what the username and password are. That means even-if we enter any wrong username or password still the query will get executed because of OR 1 = 1 part which comes out to be true.
  - o The first id is returned by the above query for **users** table and we know that the first id is normally administrator. In this way, the attacker not only bypasses authentication but also gains administrator privileges.

How to prevent SQL injection ?

- The only way to prevent SQL injection is to validate every input field.
- Another method is to make use of parameterized query. This parameterized query **is called prepared statement.** By this ways, application code never use the input directly.
- The Web Application Firewalls (WAF) are also used to filter out the SQL.

Review Question

1. *Write short note on SQL injection.*

5.10 Statistical Database Security

- Statistical databases contain statistical data about various populations.
- A **population** is a set of tuples of a table that satisfy some selection criteria.
- The statistical database may contain the **confidential data about individuals**. For example - The database system about government agencies is a statistical database. The statistical database security helps in protecting the sensitive information present in the statistical database from user's access.
- Users are permitted to use some portion of statistical database. He/she can not have access over the complete database system. For instance - In an employees database, any user is not permitted to access the information about employee's salary.
- **Statistical aggregate functions** such as COUNT, MAX,MIN, AVERAGE and STANDARD DEVIATION are used in the queries which are called as **statistical queries.**
- The possibility of accessing individual information from statistical queries is reduced by following ways -
  - o **No statistical query is permitted** whenever the number of tuples in the population specified by the selection condition falls below some threshold.
  - o **Prohibit sequence of queries** that refer repeatedly to same population of tuples.
  - o **Partition the database.** That means records are stored in groups of some minimum size. The query can refer to any complete group but never to subsets of records within a group.

5.11 Flow Control

- **Flow control** is a mechanism that regulates the flow of information among accessible objects.
- **A flow** between two objects **obj1** and **obj2** occurs when program reads values from **obj1** and writes values to the object **obj2.**
- The flow control checks that the information contained in one object should not get transferred to the less protected object.
- The **flow policy** specifies the **channels** along which the information is allowed to move.

- The simple flow policy specifies two classes of information - **Confidential(C)** and **non confidential(N).** According to flow policy only the information flow from confidential to non confidential class is not allowed.

5.11.1 Convert Channel

- A covert channel is a type of attack that creates a capability to transfer information objects between processes that are not supposed to be allowed to communicate.
- .This convert channel **violates the security** or the policy.
- The convert channel allows information to pass from higher classification level to lower classification level through **improper means.**
- The security experts believe that one way to avoid convert channels is for programmers to not gain the access to sensitive data.

5.12 Encryption and Public Key Infrastructures

Cryptology is a technique of **encoding and decoding** messages, so that they cannot be understood by anybody except the sender and the intended recipient.

There are various encoding and decoding schemes which are called as **encryption schemes.** The sender and recipient of the message decide on an encoding and decoding scheme and use it for communication.

The process of encoding messages is known as **encryption.** The sender sends the original text. The original text called **plaintext,** The encrypted form of plaintext it is called as **ciphertext.** This encrypted text travel through the network. When it reaches at the receiving computer, the recipient understands the meaning and decodes the message to extract the correct meaning out of it. This process is called as **decryption.**

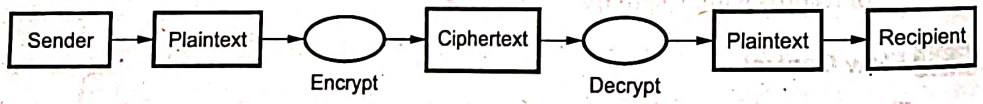


Fig. 5.12.1 : Encryption and decryption process

The sender applies the **encryption algorithm** and recipient applies the decryption algorithm. Both the sender and the receiver must agree on this algorithm for any meaningful communication. The algorithm basically takes one text as input and produces another as the output. Therefore, the algorithm contains the **intelligence** for transforming message.

For example : If we want to send some message through an e-mail and we wish that nobody except the friend should be able to understand it. Then the message can be encoded using some intelligence. For example if the alphabets A to Z are encoded as follows –

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Z	Y	X	A	B	C	W	V	U	D	E	F	T	S	R	G	H	I	Q	P	O	J	K	L	N	M

That means last three letters are placed in reverse order and then first three letters are in straight manner. Continuing this logic the A to Z letters are encoded. Now if I write the message

"SEND SOME MONEY"

it will be

QBSA QRTB TRSBN

This coded message is called **cipher text.**

There are variety of coding methods that can be used.

5.12.1 Types of Cryptography

There are two types encryption schemes based in key used for encryption and decryption.

1. Symmetric key encryption : It is also known as **secret key encryption.** In this method, only one key is used. The same key is shared by sender and receiver for encryption and decryption of messages. Hence both parties must agree upon the key before any transmission begins and nobody else should know about it. At the sender's end, the key is used to change the original message into an encoded form. At the receiver's end using the same key the encoded message is decrypted and original message is obtained. Data Encryption Standard (DES) uses this approach. The problem with this approach is that of key agreement and distribution.

2. Asymmetric key encryption : It is also known as public key encryption. In this method, different keys are used. One key is used for encryption and other key must be used for decryption. No other key can decrypt the message-not even the original key used for encryption.

One of the two keys is known as **public key** and the other is the **private key.** Suppose there are two users X and Y. The X wants to send a message to Y. Then

- X will convey its public key to Y but the private key of X will be known to X only.
- Y should know the private key of Y and X should know the Y's public key.

When X and Y wants to communicate :

1. If X wants to send a message to Y, then first of all X encrypts the message using Y's public key. For that purpose it is necessary that X knows the Y's public key.
2. X then sends this encrypted to Y.
3. Now using Y's private key, Y decrypts X's message. Note that only Y knows his private key. It is not possible for Y to decrypt the message using X's public key.
4. When Y wants to send a message to X then using X's public key Y will encrypt the message and will send the encrypted message to X. On the other hand, X will use its own private key to decrypt this message. Here again Y will not know the private key of X.

5.12.2 Digital Signature

A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message or document. If the recipient gets a message with digital signature then he believes that the message was created by a known sender.

Digital signatures are commonly used for software distribution, financial transactions, and in other cases where it is important to detect forgery or tampering.

When X and Y wants to communicate with each other

1. X encrypts the original plaintext message into **ciphertext** by using Y's public key.
2. Then X executes an algorithm on the original plaintext to calculate a **Message Digest**, also known as **hash**. This algorithm takes the original plaintext in the binary format, apply the hashing algorithm. As an output a small string of **binary digits** gets created. This hashing algorithm is public and anyone can use it. The most popular message digest algorithms are **MD5** and **SHA-1**. X encrypts the message digest. For this, it uses its own private key.
3. X now combines the **ciphertext** and its **digital signature** (i.e encrypted message digest) and it is sent over the network to Y.
4. Y receives the ciphertext and X's digital signature. Y has to decrypt both of these. Y first decrypts ciphertext back to plaintext. For this, it uses its own private key. Thus, Y gets the message itself in a secure manner.
5. Now to ensure that the message has come from the intended sender Y takes X's digital signature and decrypts it. This gives Y the message digest as was generated by X. The X had encrypted the message digest to form a digital signature using its **own private key**. Therefore, Y uses X's public key for decrypting the digital **signature**.

6. Hash algorithm to generate the message digest is **public**. Therefore, Y can also use it.
7. Now there are two message digests one created by X and other by Y. The Y now simply compares the two message digests. If the two match, Y can be sure that the message came indeed from X and not from someone else.

Thus with digital signature **confidentiality, authenticity** as well as **message integrity** is assured.

The other important feature supported by digital signature is **non-repudiation**. That is, a sender cannot refuse having sent a message. Since the digital signature requires the private key of the sender, once a message is digitally signed, it can be legally proven that the sender had indeed sent the message.

Review Question

1. *Explain in brief the concept of digital signature.*

5.13 Challenges

Following are the challenges faced by the database security system -

(1) Data Quality

- The database community need the solution to assess the quality of data. The quality of data can be assessed by a simple mechanism such as quality stamps that are posted on web sites.
- The database community may need more effective technique of integrity semantic verification for accessing the quality of data.
- Application level recovery techniques are also used to repair incorrect data.

(2) Intellectual Property Rights

- Everywhere there is increasing use of internet and intranet. Due to which there are chances of making un-authorized duplication and distribution of the contents. Hence **digital watermarking** technique is used to protect the contents from unauthorized access or ownership.
- However, research is needed to develop the techniques for preventing intellectual property right violation.

(3) Database Survivability

- It is desired that the database systems must continue to work even after information warfare attacks.
- The goal of information warfare attacker is to damage the organization's operation.
- Following are the corrective actions for handling this situation -
  - o **Confinement :** Take immediate action to eliminate attacker's access to the system. Isolate the affected components to avoid further spread.
  - o **Damage Assessment :** Determine the extent of problem.
  - o **Reconfiguration :** Re-configuration allows the system to be in operation in degraded mode while recovery is going on.
  - o **Repair :** Recover the corrupted or lost data by repairing or reinstalling the system.
  - o **Fault treatment :** Identify the weakness exploited in the attack and take steps to prevent a recurrence.

Review Questions

1. *Explain various challenges faced by database security system.*

5.14 Two Marks Questions with Answers

Q.1 Define distributed database management system. **AU : May-08,18, Dec.-16**

Ans. : A distributed database system consists of loosely coupled sites (computer) that share no physical components and each site is associated a database system.

Q.2 What are two approaches to store a relation in the distributed database ? **AU : May-04**

Ans. : (1) **Replication :** System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.

(2) **Fragmentation :** Relation is partitioned into several fragments stored in distinct sites.

Q.3 What are various fragmentations ? State various fragmentations with example. **AU : Dec.-17**

Ans. : There are two types of fragmentations - Horizontal fragmentation and vertical fragmentation.

Example - Refer section 5.1.3.2.

Q.4 What are the advantages of distributed databases ? **AU : Dec.-04, May-08**

Ans. :

(1) There is **fast data processing** as several sites participate in request processing.

(2) **Reliability and availability** of this system is high.

(3) It possess reduced operating cost.

(4) It is **easier to expand** the system by adding more sites.

(5) It has improved sharing ability and local autonomy.

Q.4 List out the reasons for development of distributed databases. **AU : May-06**

Ans. : Following are the reasons for development of distributed databases -

(1) To control the data present at geographically different sites.

(2) To obtain highly available and reliable data processing systems.

Q.5 Difference between homogeneous and heterogeneous schema.

Ans. :

Sr. No.	Homogeneous Schema	Heterogeneous Schema
1.	It contains identical software.	It contains different software.
2.	It shares global schema.	It has different schemas.
3.	Each site provides part of its autonomy in terms of right to change or software.	Each site maintains its own right to change the schema or software.
4.	Due to same schema there is no problems in query processing.	Due to different schemas, there are lot of problems in query processing.

Q.6 What are the advantages of fragmentation ?

Ans. : (1) It allows parallel processing on fragments of a relation.

(2) It allows a relation to be split so that tuples are located where they are most frequently accessed.

Q.7 Give an example of two phase commit protocol. **AU : Dec.-15**

Ans. : Refer section 5.1.4.3.

☐☐☐

Notes

SOLVED MODEL QUESTION PAPER

[As Per New Syllabus]

Database Management Systems

Semester - IV (CSE / IT / CS&BS)

Time : Three Hours] [Maximum Marks : 100

Answers ALL Questions

PART - A (10 × 2 = 20 Marks)

Q.1 *What is database management system ? Why do we need a DBMS ?* **(Refer Two Marks Q.1 of Chapter - 1)**

Q.2 *Define data independence.* **(Refer Two Marks Q.27 of Chapter - 1)**

Q.3 *What is an entity ?* **(Refer Two Marks Q.4 of Chapter - 2)**

Q.4 *Explain atleast two desirable properties of decomposition.* **(Refer Two Marks Q.18 of Chapter - 2)**

Q.5 *What are ACID properties ?* **(Refer Two Marks Q.4 of Chapter - 3)**

Q.6 *What is the difference between shared lock and exclusive lock ?* **(Refer Two Marks Q.15 of Chapter - 3)**

Q.7 *What can be done to reduce the occurrences of bucket overflows in a hash file organization ?* **(Refer Two Marks Q.7 of Chapter - 4)**

Q.8 *What is query execution plan ?* **(Refer Two Marks Q.17 of Chapter - 4)**

Q.9 *Give an example of two phase commit protocol.* **(Refer Two Marks Q.7 of Chapter - 5)**

Q.10 *What are the advantages of fragmentation ?* **(Refer Two Marks Q.6 of Chapter - 5)**

PART - B (5 × 13 = 65 Marks)

Q.11 a) *i) Explain referential integrity rule.* **(Refer section 1.11.2)** [7]

ii) Write short note on - Data model. **(Refer section 1.4)** [6]

OR

b) *i) Consider the following Schema :*

Suppliers(sid:integer, sname:string, address: string)

Parts(pid:integer, pname:string, color:string)

Catalog(sid:integer, pid: integer, cost:real)

The key fields are underlined and the domain of each field is listed after the field name.

Therefore sid is the key for Suppliers, pid is the key for Parts and sid and pid together form the key for Catalog. The Catalog relation lists the prices charged for parts by suppliers.
Write the following queries in relational algebra :
Find the sids of suppliers who supply some red or green part.
Find the sids of suppliers who supply some red part or are at 221 Packer Street.
Find the pids of parts supplied by at least two different suppliers **(Refer example 1.13.12)** **[7]**
ii) *Explain group by and having clause.* **(Refer section 1.14.8)** **[6]**

Q.12 a) i) *Explain with suitable example, the constraints of specialization and generalization in ER modeling.* **(Refer section 2.4)** **[7]**
ii) *What is aggregation in ER model ? Develop an ER diagram using aggregation that captures following information : Employees work for projects. An employee working for particular project uses various machinery. Assume necessary attributes. State any assumptions you make. Also discuss about the ER diagram you have designed.* **(Refer example 2.5.7)** **[6]**

OR

b) i) *Explain mapping weak entity sets of relational mapping.* **(Refer section 2.6.4)** **[7]**
ii) *Compute the closure of the following set of functional dependencies for a relation scheme R(A,B,C,D,E), F={A->BC, CD->E, B->D, E->A).* **(Refer example 2.8.1)** **[6]**

Q.13 a) i) *Check whether following schedule is conflict serializable or not. If it is not conflict serializable then find the serializability order.* **(Refer example 3.5.2)** **[8]**

T_1	T_2	T_3
R(A)		
	R(B)	
		R(B)
	W(B)	
W(A)		
		W(A)
	R(A)	
	W(A)	

ii) *Write short note on - Shadow paging.* **(Refer section 3.20)** **[5]**

OR

b) i) *Give an example of a scenario where two phase locking leads to deadlock.* **(Refer example 3.16.1)** **[5]**
ii) *Explain two phase locking in detail.* **(Refer section 3.10)** **[8]**

Q.14 a) *Briefly explain RAID and RAID levels.* **(Refer section 4.1)** **[13]**

OR

b) i) *Explain data dictionary storage in detail.* **(Refer section 4.4)** **[5]**
ii) *Construct B+ tree for following data. 30,31,23,32,22,28,24,29, where number of pointers that fit in one node are 5.* **(Refer example 4.8.1)** **[8]**

Q.15 a) i) *Explain the architecture of distributed database.* **(Refer section 5.1.2)** **[7]**
ii) *Explain role based access control.* **(Refer section 5.8)** **[6]**

OR

b) i) *Explain in detail the two phase commit protocol.* **(Refer section 5.1.4.3)** **[13]**

PART - C (1 × 15 = 15 Marks)

Q.16 a) *Write the DDL, DML, DCL for the students database. Which contains student details : name, id,DOB, branch, DOJ.*
Course details : Course name, Course id, Stud.id,Faculty name, id, mark.
(Refer example 1.14.1) **[15]**

OR

b) *What is normalization ? Normalize below given relation upto 3NF STUDENT.* **(Refer example 2.15.3)** **[15]**

StudID	StudName	City	Pincode	ProjectID	ProjectName	Course	Content
S101	Ajay	Surat	326201	P101	Health	Programming	C++, Java, C
S102	Vijay	Pune	325456	P102	Social	WEB	HTML, PHP, ASP

❑❑❑