CLASS NOTES
ON

DATABASE MANAGEMENT SYSTEMS
[BAI 3301]

B. TECH II YR – III SEM

(2022-23)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(BBDU)

DEPARTMENT OF COMPUTER SCIENCE & ENGG.

INDEX

S. No	Unit	Topic	Page no
1	I	Introduction To Database Management System	6
2	I	View Of Data	9
3	I	DBMS Architecture	11
4	I	Database Models	13
5	I	Database Languages	15
6	I	Database Users And Administrator	22
7	I	Keys	27
8	I	ER Model	29
9	II	Relational Algebra And Calculus	40
10	II	Sql	62
11	II	Aggregate Functions	69
12	II	Nested Queries	70
13	II	Views	80
14	III	Dependency Preservation Decomposition	97
15	III	Normal Forms	100
16	IV	Transaction	126
17	IV	Serializability	132

18	IV	Lock-Based Protocols	143
19	IV	Multiple Granualarity	149
20	V	Log-Based Recovery	152
21	V	Buffer Management	154
22	V	Recovery	156

UNIT -I

INTRODUCTION TO DBMS:

What is data?

- Data is nothing but facts and statistics stored or free flowing over a network, generally it's raw and unprocessed.
- Data becomes information when it is processed, turning it into something meaningful.
- What is database: The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently.
- It is also used to organize the data in the form of a table, schema, views, and reports, etc.
- Using the database, you can easily retrieve, insert, and delete the information.
- For example: The college Database organizes the data about the admin, staff, students and faculty etc.

What is dbms?

DBMS	File System
DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	File system is a collection of data. In this system, the user has to write the procedures for managing the database.
Searching data is easy in Dbms	Searching is difficult in File System
Dbms is structured data	Files are unstructured data
No data redundancy in Dbms	Data redundancy is there in file system
Memory utilisation well in dbms	Memory utilisation poor in file system
No data inconsistency in dbms	Inconsistency in file system

DBMS gives an abstract view of data that hides the details.	File system provides the detail of the data representation and storage of data.
DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure.	File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost.
DBMS provides a good protection mechanism.	It is very difficult to protect a file under the file system.
DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	File system can't efficiently store and retrieve the data.
DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information.

- A DBMS is software that allows creation, definition and manipulation of database, allowing users to store, process and analyse data easily.
- DBMS provides us with an interface or a tool, to perform various operations like creating database, storing data in it, updating data, creating tables in the database and a lot more.
- DBMS also provides protection and security to the databases.
- It also maintains data consistency in case of multiple users.

Here are some examples of popular DBMS used these days:

- MySql
- Oracle
- SQL Server
- IBM DB2

DATABASE APPLICATIONS – DBMS:

- Applications where we use Database Management Systems are:
- Telecom: There is a database to keeps track of the information regarding calls made, network usage, customer details etc.

- Industry: Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs
- Banking System: For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc.
- Sales: To store customer information, production information and invoice details.
- Airlines: To travel though airlines, we make early reservations; this reservation information along with flight schedule is stored in database.
- Education sector: Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc.

PURPOSE OF DATABASE SYSTEMS

- The main purpose of database systems is to manage the data. Consider a university that keeps the data of students, teachers, courses, books etc. To manage this data we need to store this data somewhere where we can add new data, delete unused data, update outdated data, retrieve data, to perform these operations on data we need a Database management system that allows us to store the data in such a way so that all these operations can be performed on the data efficiently.

Characteristics of DBMS

- Data stored into Tables: Data is never directly stored into the database. Data is stored into tables, created inside the database.
- Reduced Redundancy: In the modern world hard drives are very cheap, but earlier when hard drives were too expensive, unnecessary repetition of data in database was a big problem. But DBMS follows Normalisation which divides the data in such a way that repetition is minimum.
- Data Consistency: On Live data, i.e. data that is being continuosly updated and added, maintaining the consistency of data can become a challenge. But DBMS handles it all by itself.
- Support Multiple user and Concurrent Access: DBMS allows multiple users to work on it(update, insert, delete data) at the same time and still manages to maintain the data consistency.

- Query Language: DBMS provides users with a simple Query language, using which data can be easily fetched, inserted, deleted and updated in a database.

Advantages of DBMS

- Controls database redundancy: It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- Data sharing: In DBMS, the authorized users of an organization can share the data among multiple users.
- Easily Maintenance: It can be easily maintainable due to the centralized nature of the database system.
- Reduce time: It reduces development time and maintenance need.
- Backup: It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.
- multiple user interface: It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS

- Cost of Hardware and Software: It requires a high speed of data processor and large memory size to run DBMS software.
- Size: It occupies a large space of disks and large memory to run them efficiently.
- Complexity: Database system creates additional complexity and requirements.
- Higher impact of failure: Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

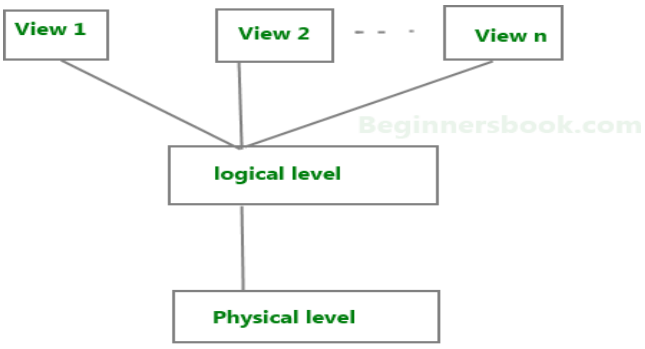
View of Data in DBMS

- Abstraction is one of the main features of database systems.
- Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient user-database interaction.
- the three level of DBMS architecture, The top level of that architecture is “view level”. The view level provides the “view of data” to the users and hides the irrelevant

details such as data relationship, database schema, constraints, security etc from the user.

Data Abstraction in DBMS

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.



Three Levels of data abstraction

We have three levels of abstraction:

Physical level: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level: Highest level of data abstraction. This level describes the user interaction with database system.

Instance and schema in DBMS

- Definition of schema: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.

- The design of a database at physical level is called physical schema, how the data stored in blocks of storage is described at this level.
- Design of database at logical level is called logical schema, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).
- Design of database at view level is called view schema. This generally describes end user interaction with database systems.

Definition of instance:

The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

DBMS ARCHITECTURE:

- Database management systems architecture will help us understand the components of database system and the relation among them.
- The architecture of DBMS depends on the computer system on which it runs.
- the basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.

TYPES OF DBMS ARCHITECTURE

There are three types of DBMS architecture:

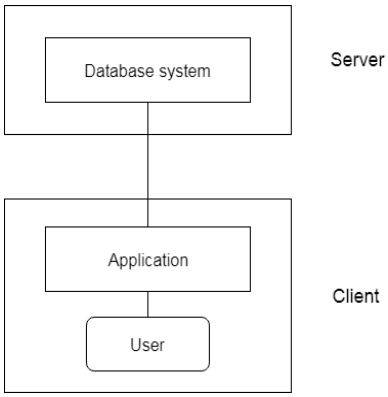
1. Single tier architecture
2. Two tier architecture
3. Three tier architecture

1-Tier Architecture

- In this type of architecture, the database is readily available on the client machine, any request made by client doesn't require a network connection to perform the action on the database.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

2. Two tier architecture

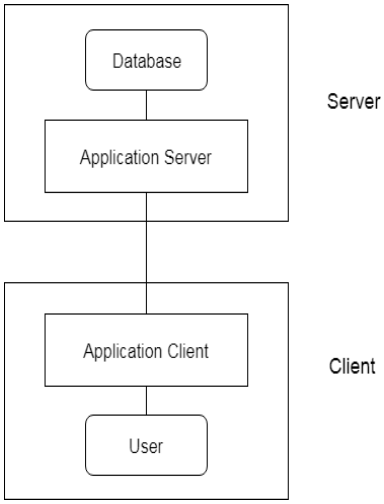
- In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network.
- Whenever client machine makes a request to access the database present at server using a query language like sql, the server perform the request on the database and returns the result back to the client.
- The application connection interface such as JDBC, ODBC are used for the interaction between server and client.



3-Tier Architecture

- In three-tier architecture, another layer is present between the client machine and server machine.
- In this architecture, the client application doesn't communicate directly with the database systems present at the server machine, rather the client application

communicates with server application and the server application internally communicates with the database system present at the server.



DATA MODELS:

- Data Model is the modeling of the data description, data semantics, and consistency constraints of the data.
- It provides the conceptual tools for describing the design of a database at each level of data abstraction.
- Therefore, there are following four data models used for understanding the structure of the database:

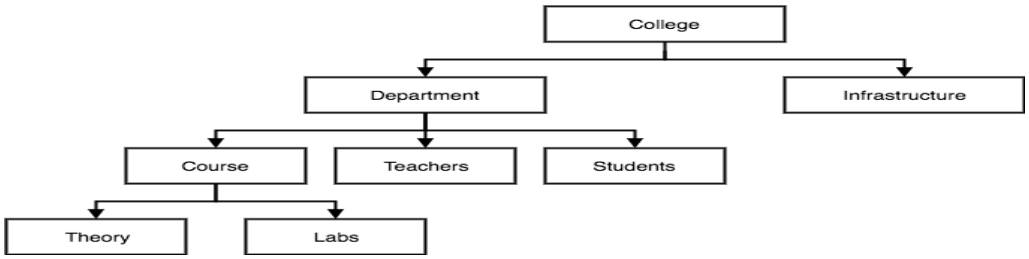
Four Types of DBMS systems are:

- Hierarchical database
- Network database
- Relational database
- ER model database

Hierarchical DBMS

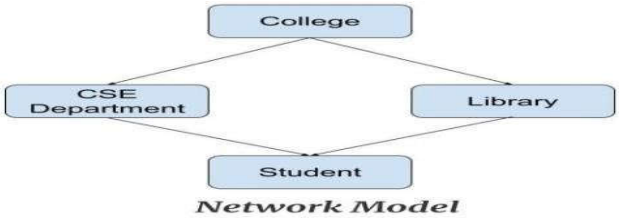
In a Hierarchical database, model data is organized in a tree-like structure. Data is Stored Hierarchically (top down or bottom up) format. Data is represented using a parent-child

relationship. In Hierarchical DBMS parent may have many children, but children have only one parent.



Network Model

The network database model allows each child to have multiple parents. It helps you to address the need to model more complex relationships like as the orders/parts many-to-many relationship. In this model, entities are organized in a graph which can be accessed through several paths.



Network Model

Relational model

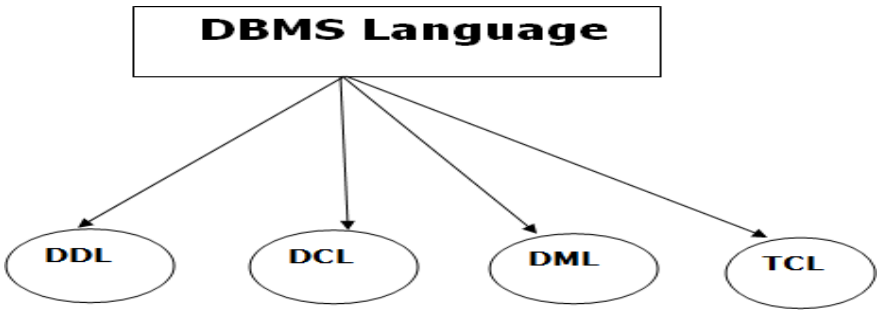
Relational DBMS is the most widely used DBMS model because it is one of the easiest. This model is based on normalizing data in the rows and columns of the tables. Relational model stored in fixed structures and manipulated using SQL.

Entity-Relationship Model

Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

DBMS languages

Database languages are used to read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).



- DDL – Data Definition Language: (CREATE,DROP,ALTER,TRUNCATE,COMMENT,RENAME)
- DML – Data Manipulation Language: (INSERT, UPDATE,DELETE)
- DCL – Data Control Language: (GRANT,REVOKE)
- TCL-Transaction Control Language: (COMMIT,ROLLBACK)

1. DDL(Data Definition Language) : DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

CREATE – it is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

There are two CREATE statements available in SQL:

- CREATE DATABASE
- CREATE TABLE

CREATE DATABASE

A Database is defined as a structured set of data. So, in SQL the very first step to store the data in a well structured manner is to create a database. The CREATE DATABASE statement is used to create a new database in SQL.

Syntax:

CREATE DATABASE database_name;

Example:

SQL> CREATE DATABASE Employee;

In order to get the list of all the databases, you can use SHOW DATABASES statement.

Example –

SQL> SHOW DATABASES;

CREATE TABLE:

The CREATE TABLE statement is used to create a table in SQL. We know that a table comprises of rows and columns. So while creating tables we have to provide all the information to SQL about the names of the columns, type of data to be stored in columns, size of the data etc. Let us now dive into details on how to use CREATE TABLE statement to create tables in SQL.

Syntax:

```
CREATE TABLE table_name
(
column1 data_type(size),
column2 data_type(size),
column3 data_type(size),
....
);
```

Example Query:

This query will create a table named Students with three columns, ROLL_NO, NAME and SUBJECT.

```
CREATE TABLE Students
(
ROLL_NO int(3),
NAME varchar(20),
SUBJECT varchar(20),
);
```

DROP:

DROP is used to delete a whole database or just a table.The DROP statement destroys the objects like an existing database, table, index, or view.

A DROP statement in SQL removes a component from a relational database management system (RDBMS).

Syntax:

DROP object object_name;

Examples:

DROP TABLE table_name;

table_name: Name of the table to be deleted.

DROP DATABASE database_name;

database_name: Name of the database to be deleted.

TRUNCATE

It is used to remove all records from a table, including all spaces

The TRUNCATE TABLE mytable statement is logically (though not physically) equivalent to the DELETE FROM mytable statement (without a WHERE clause).

Syntax:

TRUNCATE TABLE table_name;

DROP vs TRUNCATE

- Truncate is normally ultra-fast and its ideal for deleting data from a temporary table.
- Truncate preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.
- Table or Database deletion using DROP statement cannot be rolled back, so it must be used wisely.

To delete the whole database

DROP DATABASE student_data;

After running the above query whole database will be deleted.

To truncate Student_details table from student_data database.

TRUNCATE TABLE Student_details;

ALTER (ADD, DROP, MODIFY)

ALTER TABLE is used to add, delete/drop or modify columns in the existing table. It is also used to add and drop various constraints on the existing table.

ALTER TABLE – ADD:

ADD is used to add columns into the existing table. Sometimes we may require to add additional information, in that case we do not require to create the whole database again, ADD comes to our rescue.

Syntax:

```
ALTER TABLE table_name
        ADD (Columnname_1 datatype,
        Columnname_2 datatype,
        …
        Columnname_n  datatype);
```

ALTER TABLE – DROP

DROP COLUMN is used to drop column in a table. Deleting the unwanted columns from the table.

Syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

ALTER TABLE-MODIFY

It is used to modify the existing columns in a table. Multiple columns can also be modified at once.

```
ALTER TABLE table_name
MODIFY column_name column_type;
```

QUERY:

To ADD 2 columns AGE and COURSE to table Student.

```
ALTER TABLE Student ADD (AGE number(3),COURSE varchar(40));
```

MODIFY column COURSE in table Student

```
ALTER TABLE Student MODIFY COURSE varchar(20);
```

Comments

As is any programming languages comments matter a lot in SQL also. In this set we will learn about writing comments in any SQL snippet.

Comments can be written in the following three formats:

- Single line comments.
- Multi line comments
- In line comments

DML(Data Manipulation Language) : The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

SELECT Statement

select statement is used to fetch data from relational database. A relational database is organized collection of data. As we know that data is stored inside tables in a database. SQL select statement or SQL select query is used to fetch data from one or more than one tables.

SELECT Syntax

One column:

Here column_name is the name of the column for which we need to fetch data and table_name is the name of the table in the database.

SELECT column_name FROM table_name;

More than one columns:

SELECT column_name_1, column_name_2, ... FROM table_name;

To fetch the entire table or all the fields in the table:

SELECT * FROM table_name;

Example:

SELECT EMP_NAME FROM EMPLOYEES;

To fetch the entire EMPLOYEES table:

SELECT * FROM EMPLOYEES;

Query to fetch the fields ROLL_NO, NAME, AGE from the table Student:

SELECT ROLL_NO, NAME, AGE FROM Student;

INSERT INTO Statement

The INSERT INTO statement of SQL is used to insert a new row in a table. There are two ways of using INSERT INTO statement for inserting rows:

Only values: First method is to specify only the value of data to be inserted without the column names.

INSERT INTO table_name VALUES (value1, value2, value3,…);

Column names and values both: In the second method we will specify both the columns which we want to fill and their corresponding values as shown below:

INSERT INTO table_name (column1, column2, column3,..) VALUES (value1, value2, value3,..);

Example:

Method 1 (Inserting only values) :

INSERT INTO Student VALUES (‘5′,’HARSH’,’WEST BENGAL’,’XXXXXXXXXX’,’19’);

Method 2 (Inserting values in only specified columns):

INSERT INTO Student (ROLL_NO, NAME, Age) VALUES (‘5′,’PRATIK’,’19’);

UPDATE Statement

The UPDATE statement in SQL is used to update the data of an existing table in database. We can update single columns as well as multiple columns using UPDATE statement as per our requirement.

Basic Syntax:

```
UPDATE TableName
SET column_name1 = value, column_name2 = value....
WHERE condition;
```

EX1:

```
SQL> UPDATE EMPLOYEES
SET EMP_SALARY = 10000
WHERE EMP_AGE > 25;
```

EX2;

```
SQL> UPDATE EMPLOYEES
SET EMP_SALARY = 120000
WHERE EMP_NAME = 'Apoorv';
```

DELETE Statement

The DELETE Statement in SQL is used to delete existing records from a table. We can delete a single record or multiple records depending on the condition we specify in the WHERE clause.

Basic Syntax:

```
DELETE FROM table_name WHERE some_condition;
```

Deleting single record: Delete the rows where NAME = ‘Ram’. This will delete only the first row.

```
DELETE FROM Student WHERE NAME = 'Ram';
```

Deleting multiple records: Delete the rows from the table Student where Age is 20. This will delete 2 rows(third row and fifth row).

DELETE FROM Student WHERE Age = 20;

Delete all of the records: There are two queries to do this as shown below,

query1: "DELETE FROM Student";

query2: "DELETE * FROM Student";

DCL(Data Control Language) : DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples of DCL commands:

GRANT-gives user's access privileges to database.

REVOKE-withdraw user's access privileges given by using the GRANT command.

TCL(transaction Control Language) : TCL commands deals with the transaction within the database.

Examples of TCL commands:

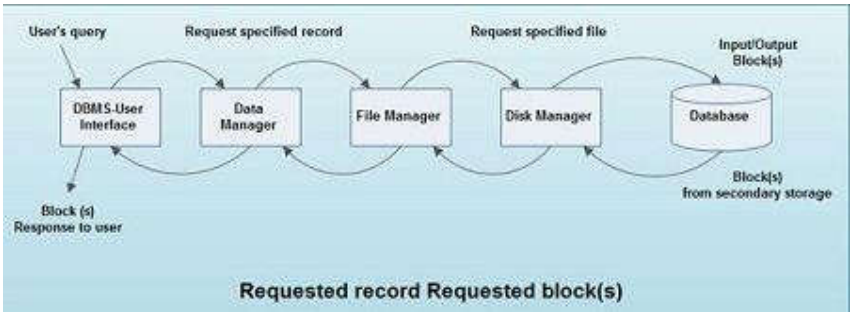
COMMIT– commits a Transaction.

ROLLBACK– rollbacks a transaction in case of any error occurs. SAVEPOINT– sets a savepoint within a transaction.

SET TRANSACTION–specify characteristics for the transaction.

What is the Procedure for Database Access?

- Any access to the stored data is done by the data manager. A user's request for data is-received by the data manager, which detern1ines the physical record required. The decision as 10 which physical record is needed may require some preliminary consultation of the database and/or the data dictionary prior to the access of the actual data itself.
- The data manager sends the request for a specific physical record to the file manager. The file manager decides which physical block of secondary storage devices contains the required record and sends the request for the appropriate block to the disk manager. A block is a unit of physical input/output operations between primary and secondary storage. The disk manager retrieves the block and sends it to the file manager, which sends the required record to the data manager.



DATA BASE USERS AND ADMINISTRATORS:

Database users are the persons who interact with the database and take the benefits of database.

They are differentiated into different types based on the way they expect to interact with the system.

Naive users: They are the unsophisticated users who interact with the system by using permanent applications that already exist. Example: Online Library Management System, ATMs (Automated Teller Machine), etc.

Application programmers: They are the computer professionals who interact with system through DML. They write application programs.

Sophisticated users: They interact with the system by writing SQL queries directly through the query processor without writing application programs.

Specialized users: They are also sophisticated users who write specialized database applications that do not fit into the traditional data processing framework. Example: Expert System, Knowledge Based System, etc.

Database Administrators

The life cycle of database starts from designing, implementing to administration of it. A database for any kind of requirement needs to be designed perfectly so that it should work without any issues. Once all the design is complete, it needs to be installed. Once this step is complete, users start using the database. The database grows as the data grows in the database. When the database becomes huge, its performance comes down. Also accessing the data from the database becomes challenge. There will be unused memory in database, making the memory inevitably huge. These administration and maintenance of database is taken care

by database Administrator – DBA.

A DBA has many responsibilities. A good performing database is in the hands of DBA. Database Administrators coordinate all the activities of the database system. They have all the permissions.

Tasks of DBA

- Creatingtheschema
- Specifying integrity constraints
- Storage structure and access method definition
- Granting permission to other users.
- Monitoring performance
- Routine Maintenance

Transaction Management?

- A Database Transaction is a logical unit of processing in a DBMS which entails one or more database access operation. In a nutshell, database transactions represent real-world events of any enterprise.
- All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction in DBMS. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.

What are ACID Properties?

ACID Properties are used for maintaining the integrity of database during transaction processing. ACID in DBMS stands for Atomicity, Consistency, Isolation, and Durability.

- **Atomicity**: A transaction is a single unit of operation. You either execute it entirely or do not execute it at all. There cannot be partial execution.
- **Consistency**: Once the transaction is executed, it should move from one consistent state to another.

- **Isolation:** Transaction should be executed in isolation from other transactions (no Locks). During concurrent transaction execution, intermediate transaction results from simultaneously executed transactions should not be made available to each other. (Level 0,1,2,3)
- **Durability:** · After successful completion of a transaction, the changes in the database should persist. Even in the case of system failures.

Storage Manager In DBMS

- A storage manager is a program module that provides the interface between the lowlevel data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for the interaction with the file manager.
- The raw data are stored on the disk using the file system, which is usually provided by a conventional operating system.
- The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

The storage manager components include:

1. Authorization and integrity manager, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

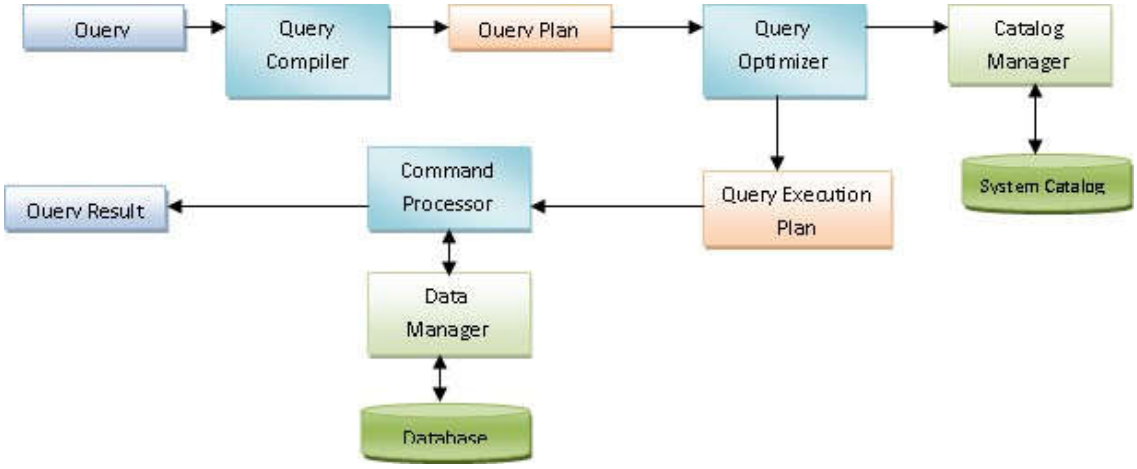
2. Transaction manager, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

3. File manager, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

4. Buffer manager, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.The storage manager implements several data structures as part of the physical system implementation:

Query Processing in DBMS

A query processor is one of the major components of a relational database or an electronic database in which data is stored in tables of rows and columns. It complements the storage engine, which writes and reads data to and from storage media.



Parsing and Translation

As query processing includes certain activities for data retrieval. Initially, the given user queries get translated in high-level database languages such as SQL. It gets translated into expressions that can be further used at the physical level of the file system. After this, the actual evaluation of the queries and a variety of query -optimizing transformations and takes place.

Query Evaluation Plan

- In order to fully evaluate a query, the system needs to construct a query evaluation plan.
- The annotations in the evaluation plan may refer to the algorithms to be used for the particular index or the specific operations.
- Such relational algebra with annotations is referred to as **Evaluation Primitives**. The evaluation primitives carry the instructions needed for the evaluation of the operation.

- Thus, a query evaluation plan defines a sequence of primitive operations used for evaluating a query. The query evaluation plan is also referred to as **the query execution plan**.
- A **query execution engine** is responsible for generating the output of the given query. It takes the query execution plan, executes it, and finally makes the output for the user query.

Optimization

- The cost of the query evaluation can vary for different types of queries. Although the system is responsible for constructing the evaluation plan, the user does need not to write their query efficiently.
- Usually, a database system generates an efficient query evaluation plan, which minimizes its cost. This type of task performed by the database system and is known as Query Optimization.
- For optimizing a query, the query optimizer should have an estimated cost analysis of each operation. It is because the overall operation cost depends on the memory allocations to several operations, execution costs, and so on.

What is Relational Model?

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.

2. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain

Keys in DBMS

KEYS in DBMS is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table. Key is also helpful for finding unique record or row from the table. Database key is also helpful for finding unique record or row from the table.

Why we need a Key?

Here are some reasons for using sql key in the DBMS system.

- Keys help you to identify any row of data in a table. In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that you can uniquely identify a table record despite these challenges.
- Allows you to establish a relationship between and identify the relation between tables
- Help you to enforce identity and integrity in the relationship.

Types of Keys in Database Management System

There are mainly seven different types of Keys in DBMS and each key has its different functionality:

- **Super Key -** A super key is a group of single or multiple keys which identifies rows in a table.
- **Primary Key -** is a column or group of columns in a table that uniquely identify every row in that table.
- **Candidate Key -** is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes.
- **Alternate Key -** is a column or group of columns in a table that uniquely identify every row in that table.
- **Foreign Key -** is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity.
- **Compound Key -** has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database.
- **Composite Key -** An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key.
- **Surrogate Key -** An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key.

Primary key example:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

Create Primary Key (ALTER TABLE statement)

Syntax

The syntax to create a primary key using the ALTER TABLE statement in SQL is:

```
ALTER TABLE table_name
  ADD CONSTRAINT constraint_name
    PRIMARY KEY (column1, column2, ... column_n);
```

FOREIGN KEY on CREATE TABLE

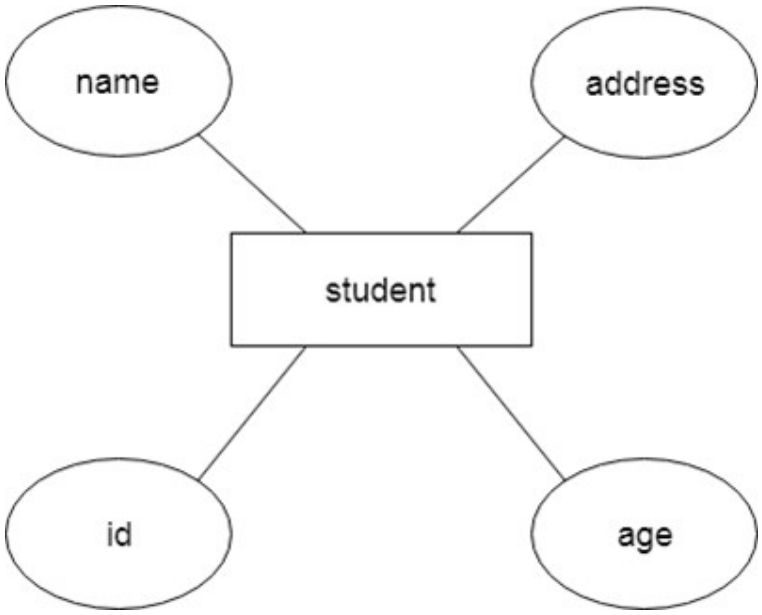
The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

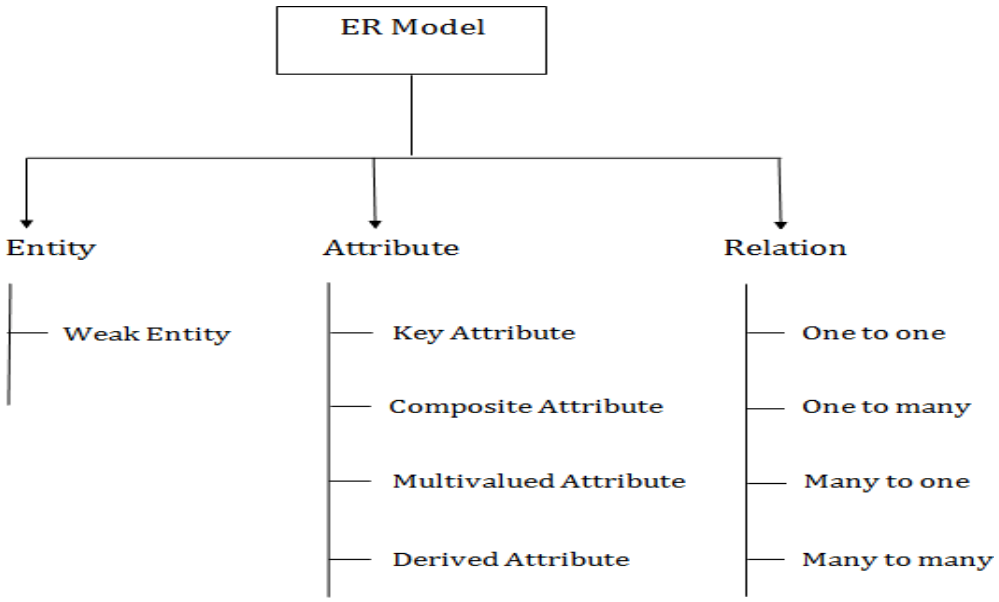
ER model

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



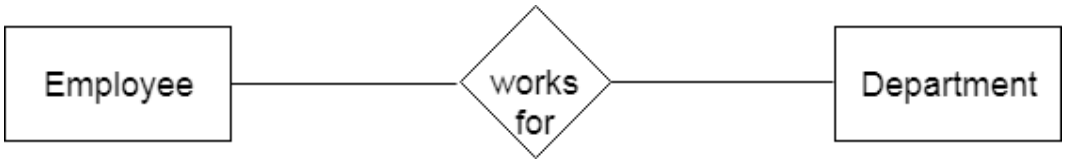
Component of ER Diagram



1. Entity:

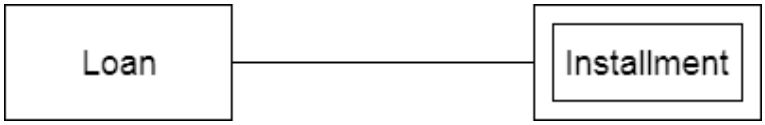
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



a. Weak Entity

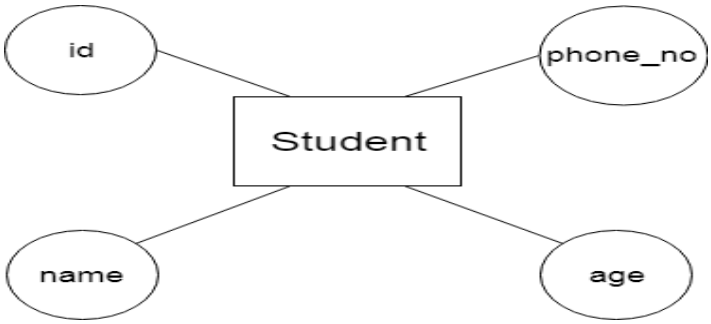
An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



2. Attribute

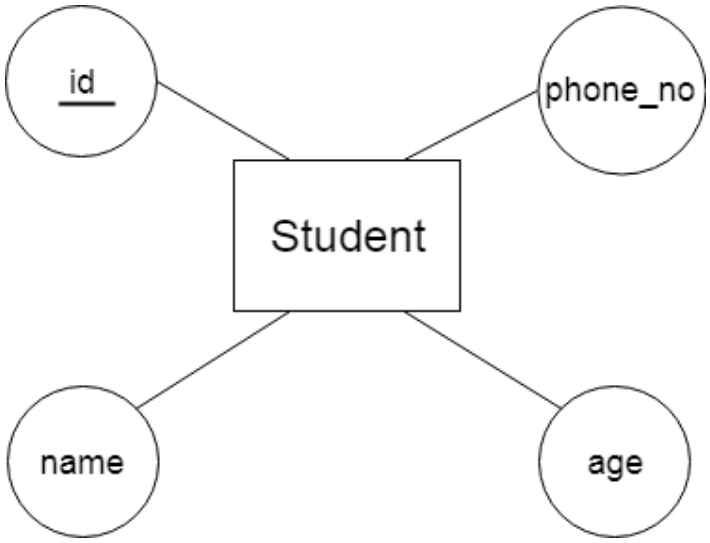
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



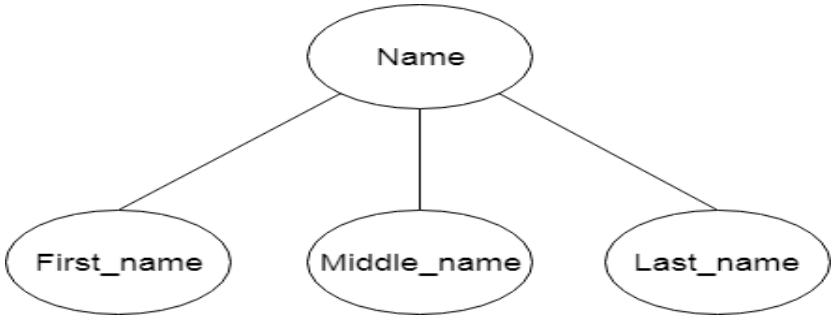
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

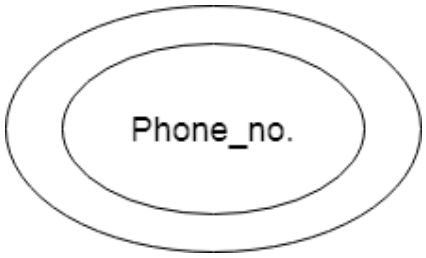
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

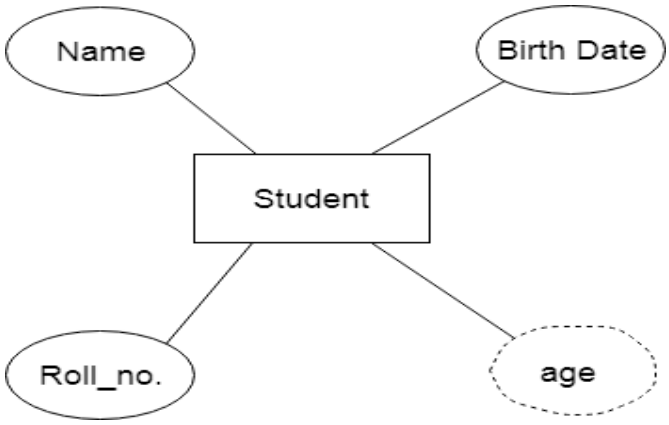
For example, a student can have more than one phone number.



d. Derived Attribute

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship

Types of relationship are as follows:

a. One-to-One Relationship

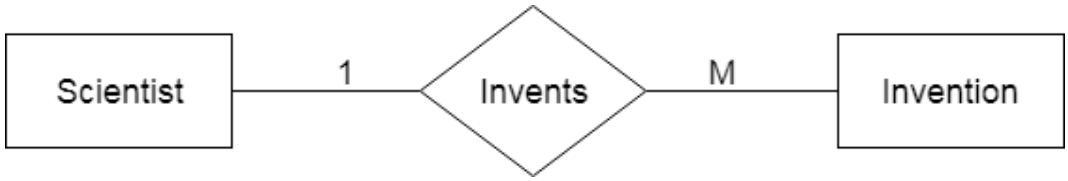
When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

For example, A female can marry to one male, and a male can marry to one female.

b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

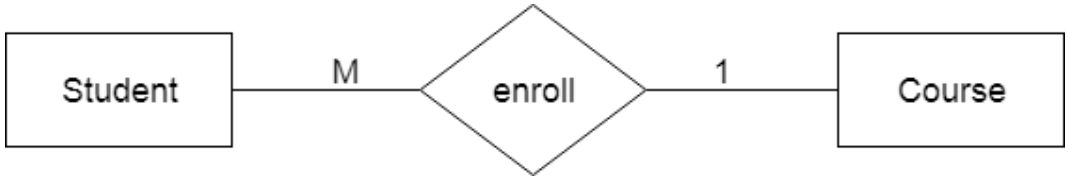
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only one course, but a course can have many students.



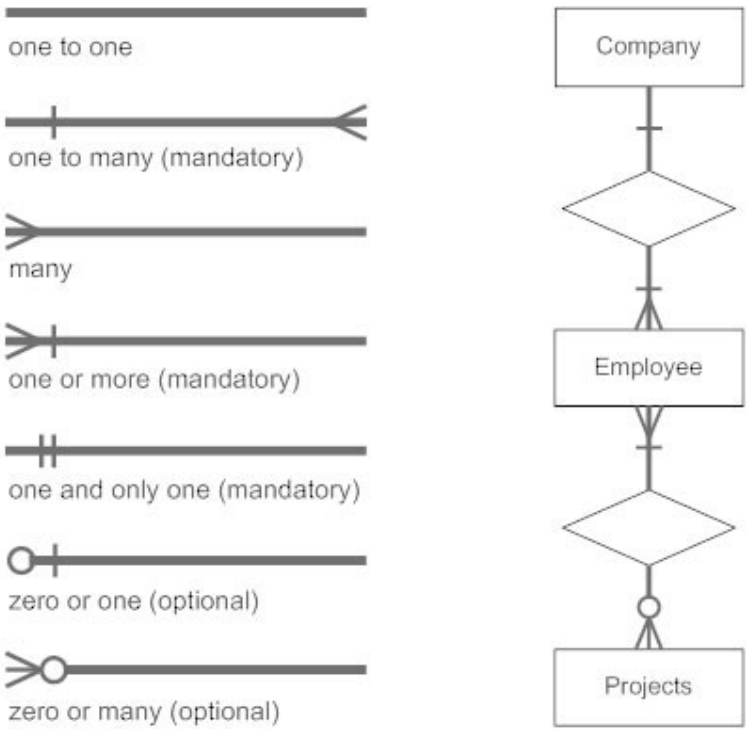
d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have manyemployees.

Notation of ER diagram

Database can be represented using the notations. In ER diagram, many notations are used to express the cardinality. These notations are as follows:

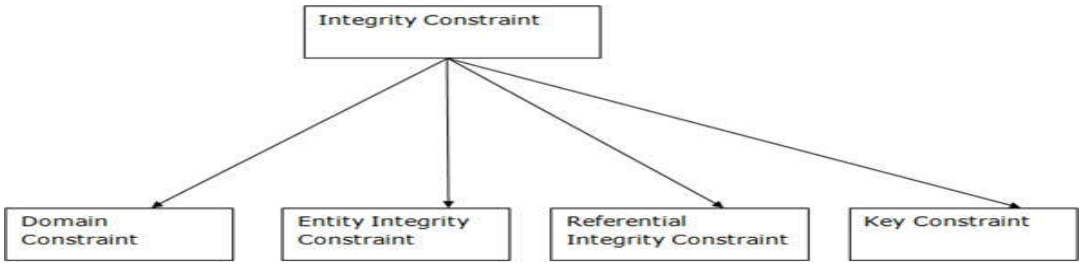


Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.

- Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraint



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

EMPLOYEE

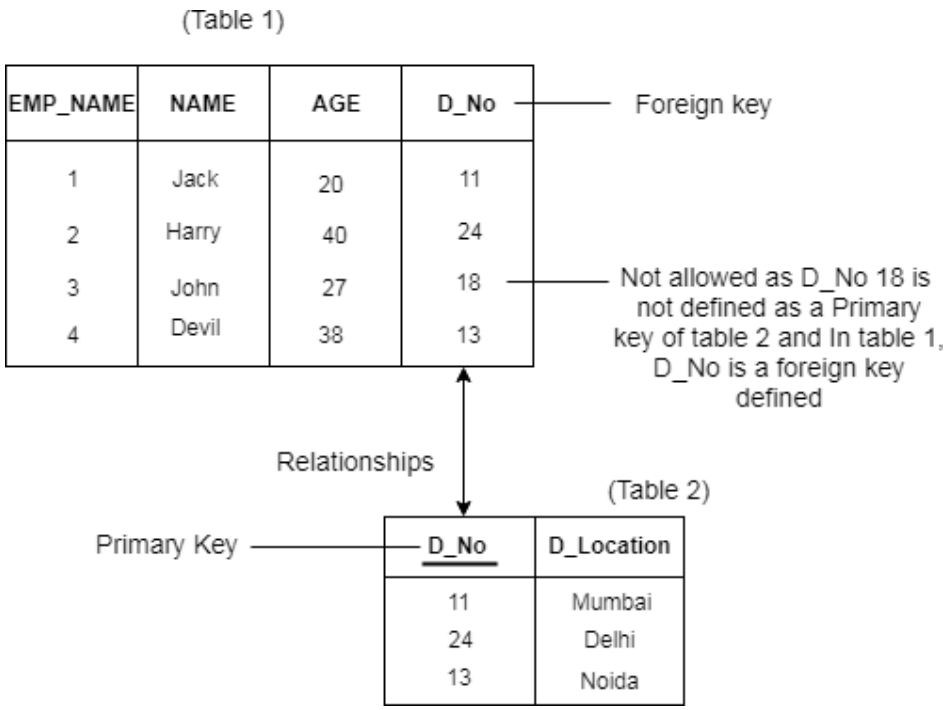
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential

Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.



4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

ER Design Issues

- ER design issues need to be discussed for better ER- design
- users often mislead the concept of the elements and the design process of the ER diagram. Thus, it leads to a complex structure of the ER diagram and certain issues that does not meet the characteristics of the real-world enterprise model.
- Here, we will discuss the basic design issues of an ER database schema in the following points:

1) Use of Entity Set vs Attributes

The use of an entity set or attribute depends on the structure of the real-world enterprise that is being modelled and the semantics associated with its attributes. It leads to a mistake when the user use the primary key of an entity set as an attribute of another entity set. Instead, he should use the relationship to do so. Also, the primary key attributes are implicit in the relationship set, but we designate it in the relationship sets.

2) Use of Entity Set vs. Relationship Sets

It is difficult to examine if an object can be best expressed by an entity set or relationship set.

3) Use of Binary vs n-ary Relationship Sets

Generally, the relationships described in the databases are binary relationships. However, non-binary relationships can be represented by several binary relationships.

4) Placing Relationship Attributes

The cardinality ratios can become an affective measure in the placement of the relationship attributes. So, it is better to associate the attributes of one-to-one or one-to-many relationship sets with any participating entity sets, instead of any relationship set.

Conceptual design

Conceptual design is the first stage in the database design process. The goal at this stage is to design a database that is independent of database software and physical details. The output of this process is a conceptual data model that describes the main data entities, attributes, relationships, and constraints of a given problem domain.

Keep in mind the following minimal data rule:

"All that is needed is there, and all that is there is needed".

1. Data analysis and requirements

2. Entity relationship modeling and normalization

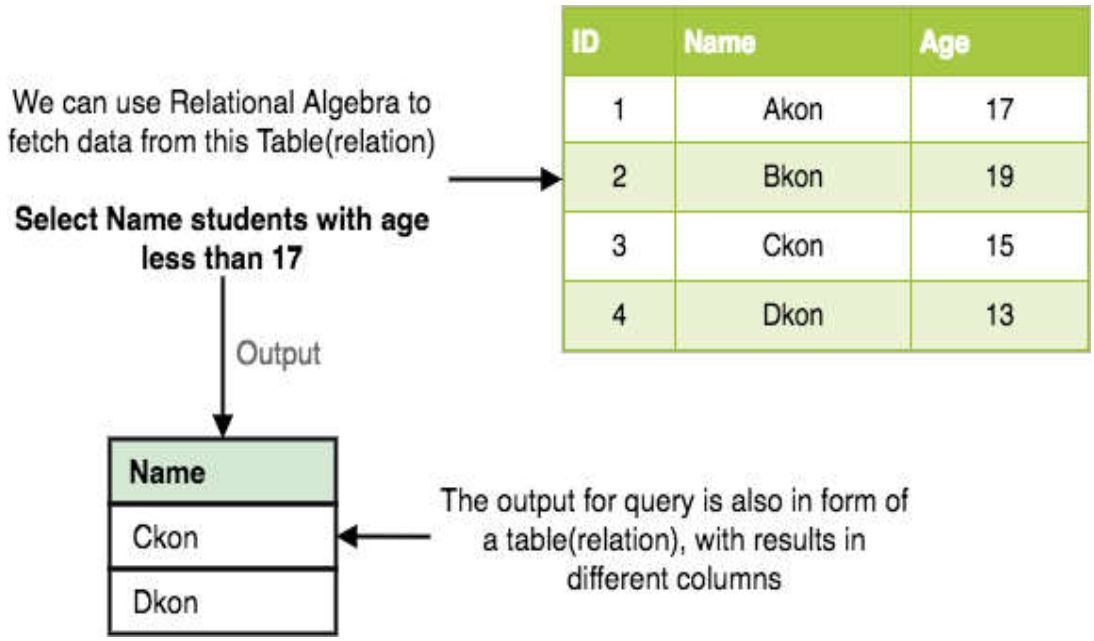
3. Data model verification

4. Distributed database design

UNIT-2

Relational Algebra

- Relational Algebra is procedural query language, which takes Relation as input and generates relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.
- Relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.
- Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows (tuples) of data one by one.
- All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.



Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

1. Select Operation (σ) : This is used to fetch rows (tuples) from table(relation) which satisfies a given condition.

Syntax: $\sigma_p(r)$

- σ is the predicate
- r stands for relation which is the name of the table
- p is prepositional logic

ex: $\sigma_{age > 17}$ (Student)

This will fetch the tuples(rows) from table **Student**, for which **age** will be greater than **17**.

$\sigma_{age > 17 \text{ and } gender = 'Male'}$ (Student)

This will return tuples(rows) from table **Student** with information of male students, of age more than 17.

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

Project Operation (∏):

- Project operation is used to project only a certain set of attributes of a relation. In simple words, If you want to see only the **names** all of the students in the **Student** table, then you can use Project Operation.
- It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.

Syntax of Project Operator (∏)

∏ column_name1, column_name2, ..., column_nameN(table_name)

Example:

$\prod_{Name, Age}(Student)$

Above statement will show us only the **Name** and **Age** columns for all the rows of data in **Student** table.

Example: CUSTOMER RELATION

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison

Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input:

∏ NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

Union Operation (∪):

- This operation is used to fetch data from two relations(tables) or temporary relation(result of another operation).
- For this operation to work, the relations(tables) specified should have same number of attributes(columns) and same attribute domain. Also the duplicate tuples are autamatically eliminated from the result.

Syntax: A ∪ B

$\prod_{Student}(RegularClass) \cup \prod_{Student}(ExtraClass)$

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11

Williams	L-17

Input:

∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

Set Difference (-):

This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation.

Syntax: A - B

where A and B are relations.

For example, if we want to find name of students who attend the regular class but not the extra class, then, we can use the below operation:

$\prod_{Student}$(RegularClass) - $\prod_{Student}$(ExtraClass)

Input: ∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

CUSTOMER_NAME
Smith
Jones

Cartesian Product (X):

This is used to combine data from two different relations(tables) into one and fetch data from the combined relation.

Syntax: A X B

For example, if we want to find the information for Regular Class and Extra Class which are conducted during morning, then, we can use the following operation:

$\sigma_{time = 'morning'}$ (RegularClass X ExtraClass)

For the above query to work, both **RegularClass** and **ExtraClass** should have the attribute **time**.

Notation: E X D

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:

EMPLOYEE **X** DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales

3	John	B	C	Legal

Rename Operation (ρ):

This operation is used to rename the output relation for any query operation which returns result like Select, Project etc. Or to simply rename a relation(table)
Syntax: ρ(RelationNew, RelationOld)

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.
ρ(STUDENT1, STUDENT)

Join in DBMS:

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- **Join in DBMS** is a binary operation which allows you to combine join product and selection in one single statement.
- The goal of creating a join condition is that it helps you to combine the data from two or more DBMS tables.
- The tables in DBMS are associated using the primary key and foreign keys.

Types of SQL JOIN

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN

Table name: EMPLOYEE

EMP_ID	EMP_NAME	CITY	SALARY	AGE
1	Angelina	Chicago	200000	30
2	Robert	Austin	300000	26
3	Christian	Denver	100000	42
4	Kristen	Washington	500000	29
5	Russell	Los angels	200000	36
6	Marry	Canada	600000	48

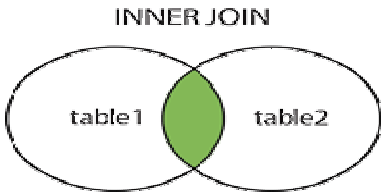
PROJECT

PROJECT_NO	EMP_ID	DEPARTMENT
101	1	Testing
102	2	Development
103	3	Designing
104	4	Development

1. INNER JOIN

In SQL, INNER JOIN selects records that have matching values in both tables as long as the condition is satisfied.

It returns the combination of all rows from both the tables where the condition satisfies.



Syntax

```
SELECT table1.column1, table1.column2
FROM table1    INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

Query

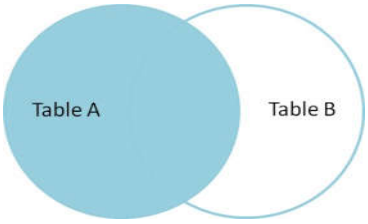
```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE INNER JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

2. LEFT JOIN

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.



Syntax

SELECT table1.column1, table1.column2 FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;

Query

SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT

FROM EMPLOYEE LEFT JOIN PROJECT

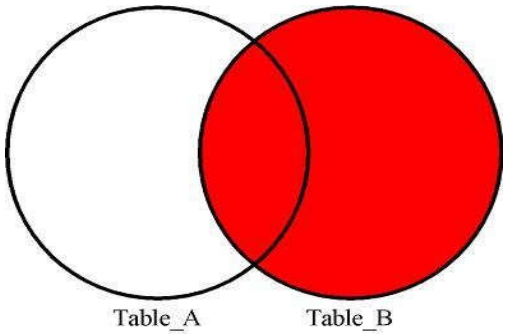
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL
Marry	NULL

3. RIGHT JOIN

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.



Syntax

SELECT table1.column1, table1.column2
FROM table1 RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;

Query

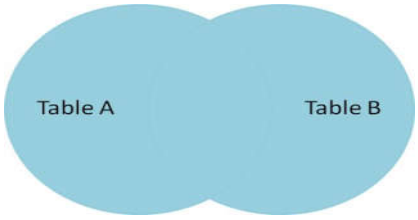
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE RIGHT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development

4. FULL JOIN

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.



Syntax

SELECT table1.column1, table1.column2
FROM table1 FULL JOIN table2
ON table1.matching_column = table2.matching_column;

Query

SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
FULL JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;

Output

EMP_NAME	DEPARTMENT
Angelina	Testing
Robert	Development
Christian	Designing
Kristen	Development
Russell	NULL

Marry	NULL

Division Operator in SQL

Division Operator (÷): Division operator A÷B can be applied if and only if:

- Attributes of B is proper subset of Attributes of A.
- The relation returned by division operator will have attributes = (All attributes of A – All Attributes of B)
- The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.

The division operator is used when we have to evaluate queries which contain the keyword ALL.

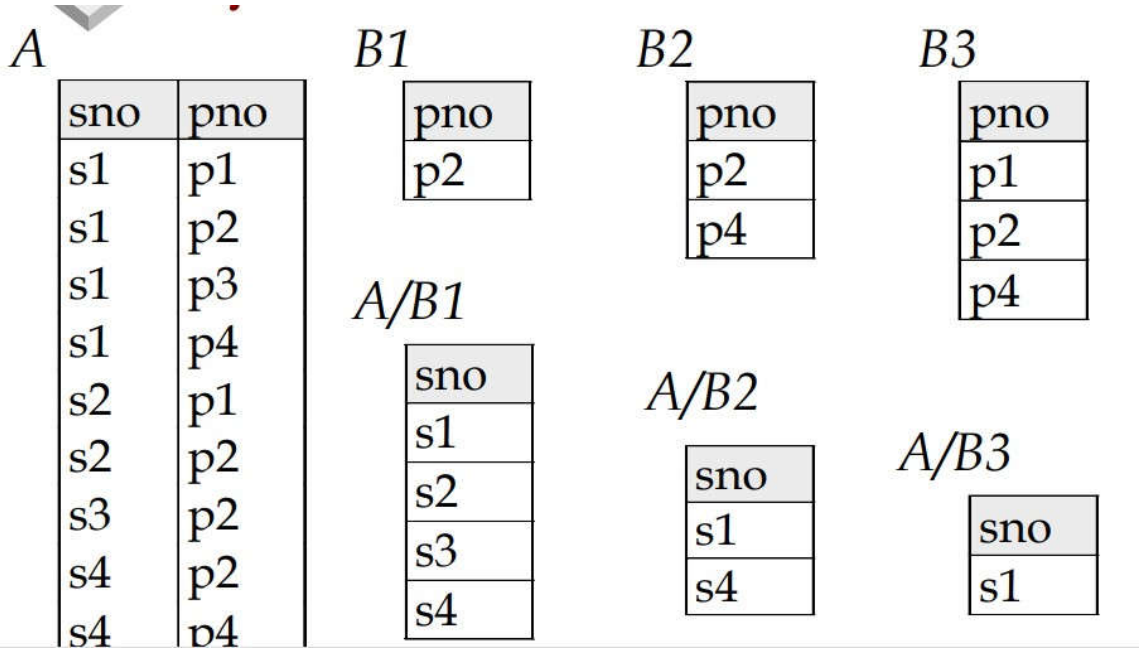


Table 1: Course_Taken → It consists of the names of Students against the courses that they have taken.

Student_Name	**Course**
Robert	Databases

Robert	Programming Languages
David	Databases
David	Operating Systems
Hannah	Programming Languages
Hannah	Machine Learning
Tom	Operating Systems

Table 2: Course_Required → It consists of the courses that one is required to take in order to graduate.

Course
Databases
Programming Languages

1. Find all the students

Create a set of all students that have taken courses. This can be done easily using the following command.

CREATE TABLE AllStudents AS SELECT DISTINCT Student_Name FROM Course_Taken

This command will return the table **AllStudents**, as the resultset:

Student_name
Robert
David
Hannah
Tom

2. Find all the students and the courses required to graduate

Next, we will create a set of students and the courses they need to graduate. We can express this in the form of Cartesian Product of **AllStudents** and **Course_Required** using the following command.

CREATE table StudentsAndRequired AS
SELECT AllStudents.Student_Name, Course_Required.Course
FROM AllStudents, Course_Required

Now the new resultset - table **StudentsAndRequired** will be:

Student_Name	Course
Robert	Databases
Robert	Programming Languages
David	Databases
David	Programming Languages
Hannah	Databases
Hannah	Programming Languages
Tom	Databases
Tom	Programming Languages

Relational Calculus:

Relational calculus is a non-procedural query language that tells the system what data to be retrieved but doesn't tell how to retrieve it. Relational Calculus exists in two forms:

1. Tuple Relational Calculus (TRC)
2. Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC)

Tuple relational calculus is used for selecting those tuples that satisfy the given condition.

Table: Student

```
First_Name     Last_Name     Age
-----------
Ajeet          Singh         30
Chaitanya      Singh         31
Rajeev         Bhatia        27
Carl           Pratap        28
```

Lets write relational calculus queries.

Query to display the last name of those students where age is greater than 30

```
{ t.Last_Name | Student(t) AND t.age > 30 }
```

In the above query you can see two parts separated by | symbol. The second part is where we define the condition and in the first part we specify the fields which we want to display for the selected tuples.

The result of the above query would be:

```
Last_Name
----------
Singh
```

Query to display all the details of students where Last name is ‘Singh’

```
{ t | Student(t) AND t.Last_Name = 'Singh' }
```

Output:

```
First_Name     Last_Name     Age
------------   -----------   -----
Ajeet          Singh         30
Chaitanya      Singh         31
```

Ex:

Table-1: Customer

Customer name	Street	City
Saurabh	A7	Patiala
Mehak	B6	Jalandhar
Sumiti	D9	Ludhiana
Ria	A5	Patiala

Table-2: Branch

Branch name	Branch city
ABC	Patiala
DEF	Ludhiana
GHI	Jalandhar

Table-3: Account

Account number	Branch name	Balance
1111	ABC	50000
1112	DEF	10000
1113	GHI	9000

Account number	Branch name	Balance
1114	ABC	7000

Table-4: Loan

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L49	GHI	9000
L98	DEF	65000

Table-5: Borrower

Customer name	Loan number
Saurabh	L33
Mehak	L49
Ria	L98

Table-6: Depositor

Customer name	Account number
Saurabh	1111

Customer name	Account number
Mehak	1113
Sumiti	1114

Queries-1: Find the loan number, branch, amount of loans of greater than or equal to 10000 amount.

$\{t \mid t \in loan \wedge t[amount] >= 10000\}$

Resulting relation:

Loan number	Branch name	Amount
L33	ABC	10000
L35	DEF	15000
L98	DEF	65000

Domain Relational Calculus (DRC)

In domain relational calculus the records are filtered based on the domains.

Again we take the same table to understand how DRC works.

Table: Student

```
First_Name     Last_Name     Age
-----------
Ajeet          Singh         30
Chaitanya      Singh         31
Rajeev         Bhatia        27
Carl           Pratap        28
```

Query to find the first name and age of students where student age is greater than 27

$\{< First\_Name, Age > \mid \in Student \wedge Age > 27\}$

Note:

The symbols used for logical operators are: ∧ for AND, ∨ for OR and ¬ for NOT.

Output:

```
First_Name     Age
------------   -----
Ajeet          30
Chaitanya      31
Carl           28
```

SQL Basic Structure

1. Basic structure of an SQL expression consists of **select, from** and **where** clauses.
    - **select** clause lists attributes to be copied - corresponds to relational algebra **project**.
    - **from** clause corresponds to Cartesian product - lists relations to be used.
    - **where** clause corresponds to selection predicate in relational algebra.

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

To fetch the entire table or all the fields in the table:

SELECT * FROM table_name;

To fetch individual column data

SELECT column1,column2 FROM table_name

WHERE SQL clause

WHERE clause is used to specify/apply any condition while retrieving, updating or deleting data from a table. This clause is used mostly with SELECT, UPDATE and DELETEquery.

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

SELECT column1, column2, columnN

FROM table_name

WHERE [condition]

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY |
+----+----------+-----+-----------+----------+
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    | 6500.00 |
|  5 | Hardik   |  27 | Bhopal    | 8500.00 |
|  6 | Komal    |  22 | MP        | 4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 −

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

This would produce the following result −

```
+----+----------+----------+
| ID | NAME     | SALARY |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

From clause:

From clause can be used to specify a sub-query expression in SQL. The relation produced by the sub-query is then used as a new relation on which the outer query is applied.

- Sub queries in the from clause are supported by most of the SQL implementations.
- The correlation variables from the relations in from clause cannot be used in the sub-queries in the from clause.

Syntax:

```
SELECT column1, column2 FROM
(SELECT column_x as C1, column_y FROM table WHERE PREDICATE_X)
as table2
WHERE PREDICATE;
```

SET Operations

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.
In this tutorial, we will cover 4 different types of SET operations, along with example:

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

1. Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax

SELECT column_name FROM table1

UNION

SELECT column_name FROM table2;

The First table

ID	NAME
1	Jack
2	Harry
3	Jackson

The Second table

ID	NAME
3	Jackson
4	Stephan
5	David

Union SQL query will be:

SELECT * FROM First

UNION

SELECT * FROM Second;

The resultset table will look like:

ID	NAME
1	Jack
2	Harry
3	Jackson
4	Stephan
5	David

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

Example: Using the above First and Second table.

Union All query will be like:

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

The resultset table will look like:

-ID	NAME
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephan
5	David

3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Syntax

SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;

Example:

Using the above First and Second table.

Intersect query will be:

SELECT * FROM First
INTERSECT
SELECT * FROM Second;

The resultset table will look like:

ID	NAME
3	Jackson

4. Minus

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;

Example

Using the above First and Second table.

Minus query will be:
SELECT * FROM First
MINUS
SELECT * FROM Second;

The resultset table will look like:

ID	NAME
1	Jack
2	Harry

Aggregate functions in SQL

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Aggregate Functions

1) Count()
2) Sum()
3) Avg()
4) Min()
5) Max()

1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Count(*): Returns total number of records

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

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

Output:

```
10
```

Example: COUNT with WHERE

```
SELECT COUNT(*)
FROM PRODUCT_MAST;
WHERE RATE>=20;
```

Output:7

Example: COUNT() with DISTINCT

```
SELECT COUNT(DISTINCT COMPANY)
FROM PRODUCT_MAST;
```

Output:

```
3
```

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

```
SUM()
or
SUM( [ALL|DISTINCT] expression )
```

Example: SUM()

```
SELECT SUM(COST)
FROM PRODUCT_MAST;
```

Output:

```
670
```

Example: SUM() with WHERE

```
SELECT SUM(COST)
FROM PRODUCT_MAST
WHERE QTY>3;
```

Output:

```
320
```

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

AVG()

Example:

SELECT AVG(COST)
FROM PRODUCT_MAST;

Output:

67.00

4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax: MAX()

Example:

SELECT MAX(RATE)
FROM PRODUCT_MAST;

30

5. MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax:MIN())

Example: SELECT MIN(RATE)
FROM PRODUCT_MAST;

Output:10

GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
ORDER BY *column_name(s);*

SI NO	NAME	SALARY	AGE
1	Harsh	2000	19
2	Dhanraj	3000	20
3	Ashish	1500	19
4	Harsh	3500	19
5	Ashish	1500	19

SUBJECT	YEAR	NAME
English	1	Harsh
English	1	Pratik
English	1	Ramesh
English	2	Ashish
English	2	Suresh
Mathematics	1	Deepak
Mathematics	1	Sayan

Example:

- **Group By single column**: Group By single column means, to place all the rows with same value of only that particular column in one group. Consider the query as shown below:
- SELECT NAME, SUM(SALARY) FROM Employee
- GROUP BY NAME;

  The above query will produce the below output:

NAME	SALARY
Ashish	3000
Dhanraj	3000
Harsh	5500

Group By multiple columns: Group by multiple column is say for example, **GROUP BY column1, column2**. This means to place all the rows with same values of both the columns **column1** and **column2** in one group. Consider the below query:

SELECT SUBJECT, YEAR, Count(*)
FROM Student
GROUP BY SUBJECT, YEAR;

SUBJECT	YEAR	Count
English	1	3
English	2	2
Mathematics	1	2

HAVING Clause:

We know that WHERE clause is used to place conditions on columns but what if we want to place conditions on groups?

This is where HAVING clause comes into use. We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause if we want to use any of these functions in the conditions.

Syntax:

SELECT column1, function_name(column2)
FROM table_name
WHERE condition
GROUP BY column1, column2
HAVING condition
ORDER BY column1, column2;

function_name: Name of the function used for example, SUM() , AVG().

table_name: Name of the table.

condition: Condition used.

Example:

```
SELECT NAME, SUM(SALARY) FROM Employee
GROUP BY NAME
HAVING SUM(SALARY)>3000;
```

Example

Consider the CUSTOMERS table having the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY |
+----+----------+-----+-----------+----------+
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    | 6500.00 |
|  5 | Hardik   |  27 | Bhopal    | 8500.00 |
|  6 | Komal    |  22 | MP        | 4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would display a record for a similar age count that would be more than or equal to 2.

```
SQL > SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
GROUP BY age
HAVING COUNT(age) >= 2;
```

This would produce the following result −

```
+----+--------+-----+---------+---------+
| ID | NAME | AGE | ADDRESS | SALARY |
+----+--------+-----+---------+---------+
| 2 | Khilan | 25 | Delhi | 1500.00 |
```

+----+---------+----+---------+---------+

Nested Queries

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query. We will use **STUDENT, COURSE, STUDENT_COURSE** tables for understanding nested queries.

STUDENT

S_ID	S_NAME	S_ADDRESS	S_PHONE	S_AGE
S1	RAM	DELHI	9455123451	18
S2	RAMESH	GURGAON	9652431543	18
S3	SUJIT	ROHTAK	9156253131	20
S4	SURESH	DELHI	9156768971	18

COURSE

C_ID	C_NAME
C1	DSA
C2	Programming
C3	DBMS

STUDENT_COURSE

S_ID	C_ID

S1	C1
S1	C3
S2	C1
S3	C2
S4	C2
S4	C3

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY |
+----+----------+-----+-----------+----------+
| 1 | Ramesh | 35 | Ahmedabad | 2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    | 6500.00 |
|  5 | Hardik   |  27 | Bhopal    | 8500.00 |
|  6 | Komal    |  22 | MP        | 4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Now, let us check the following subquery with a SELECT statement.

```
SQL> SELECT *
  FROM CUSTOMERS
  WHERE ID IN (SELECT ID
```

```
FROM CUSTOMERS
WHERE SALARY > 4500) ;
```

This would produce the following result.

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

Students

id	name	class_id	GPA
1	Jack Black	3	3.45
2	Daniel White	1	3.15
3	Kathrine Star	1	3.85
4	Helen Bright	2	3.10
5	Steve May	2	2.40

Teachers

id	name	subject	class_id	monthly_salary
1	Elisabeth Grey	History	3	2,500
2	Robert Sun	Literature	[NULL]	2,000
3	John Churchill	English	1	2,350
4	Sara Parker	Math	2	3,000

Classes

id	grade	teacher_id	number_of_students
1	10	3	21
2	11	4	25
3	12	1	28

```
SELECT *
FROM students
WHERE GPA > (
   SELECT AVG(GPA)
   FROM students);
```

result:

id	name	class_id	GPA
1	Jack Black	3	3.45
3	Kathrine Star	1	3.85

```
SELECT AVG(number_of_students)
FROM classes
WHERE teacher_id IN (
   SELECT id
   FROM teachers
   WHERE subject = 'English' OR subject = 'History');
```

Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Sample table:

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi

2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18
4	Alina	90	20
5	John	96	18

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

2. Creating View from a single table

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

Just like table query, we can query the view to view the data.

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

3. Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
CREATE VIEW MarksView AS
SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS
FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME;
```

To display data of View MarksView:

SELECT * FROM MarksView;

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

1. DROP VIEW view_name;

Example:

If we want to delete the View **MarksView**, we can do this as:

1. DROP VIEW MarksView;

Uses of a View :

A good database should contain views due to the given reasons:

1. **Restricting data access –**
   Views provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table.
2. **Hiding data complexity –**
   A view can hide the complexity that exists in a multiple table join.

3. **Simplify commands for the user –**
   Views allows the user to select information from multiple tables without requiring the users to actually know how to perform a join.
4. **Store complex queries –**
   Views can be used to store complex queries.
5. **Rename Columns –**
   Views can also be used to rename the columns without affecting the base tables provided the number of columns in view must match the number of columns specified in select statement. Thus, renaming helps to to hide the names of the columns of the base tables.
6. **Multiple view facility –**
   Different views can be created on the same table for different users.

Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.

AFTER triggers run the trigger action after the triggering statement is run.

Example:

Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

Suppose the database Schema –

```
mysql> desc Student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00  sec)
```

SQL Trigger to problem statement.

```
create trigger stud_marks
before INSERT
on
Student
for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3, Student.per =
Student.total * 60 / 100;
```

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);
```

Query OK, 1 row affected (0.09 sec)

mysql> select * from Student;

```
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 | 36   |
+-----+-------+-------+-------+-------+-------+------+
```

1 row in set (0.00 sec)

In this way trigger can be creates and executed in the databases.

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger:

Syntax for creating trigger:

CREATE [OR REPLACE] **TRIGGER** trigger_name
{BEFORE | **AFTER** | **INSTEAD OF** }
{**INSERT** [OR] | **UPDATE** [OR] | **DELETE**}
[**OF** col_name]
ON table_name
[REFERENCING OLD **AS** o NEW **AS** n]

[**FOR** EACH ROW]

WHEN (condition)

DECLARE

Declaration-statements

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

END;

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create trigger:

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

After the execution of the above code at SQL Prompt, it produces the following result.

```
Trigger created.
```

Check the salary difference by procedure:

Use the following code to get the old salary, new salary and salary difference after the trigger created.

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/ Output:
Old salary: 20000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
Old salary: 26000
New salary: 31000
Salary difference: 5000
Old salary: 28000
New salary: 33000
Salary difference: 5000
```

```
Old salary: 30000
New salary: 35000
Salary difference: 5000
6 customers updated
```

Note: As many times you executed this code, the old and new both salary is incremented by 5000 and hence the salary difference is always 5000.

After the execution of above code again, you will get the following result.

```
Old salary: 25000
New salary: 30000
Salary difference: 5000
Old salary: 27000
New salary: 32000
Salary difference: 5000
Old salary: 29000
New salary: 34000
Salary difference: 5000
Old salary: 31000
New salary: 36000
Salary difference: 5000
Old salary: 33000
New salary: 38000
Salary difference: 5000
Old salary: 35000
New salary: 40000
Salary difference: 5000
6 customers updated
```

Important Points

Following are the two very important point and should be noted carefully.

- OLD and NEW references are used for record level triggers these are not avialable for table level triggers.

- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

- **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
- **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

How to pass parameters in procedure:

When you want to create a procedure or function, you have to define parameters .There is three ways to pass parameters in procedure:

1. **IN parameters:** The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. **OUT parameters:** The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **INOUT parameters:** The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

A procedure may or may not return any value.

PL/SQL Create Procedure

Syntax for creating procedure:

CREATE [OR REPLACE] **PROCEDURE** procedure_name
[(parameter [,parameter])]
IS

[declaration_section]
BEGIN
executable_section
[EXCEPTION
exception_section]
END [procedure_name];

Create procedure example

In this example, we are going to insert record in user table. So you need to create user table first.

Table creation:

create table user(id number(10) **primary key**,**name** varchar2(100)); Now write the procedure code to insert record in user table.

Procedure Code:

```
create or replace procedure "INSERTUSER"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user values(id,name);
end;
/
```

Output:

```
Procedure created.
```

PL/SQL program to call procedure

Let's see the code to call above created procedure.

```
BEGIN
  insertuser(101,'Rahul');
  dbms_output.put_line('record inserted successfully');
END;
/
```

Now, see the "USER" table, you will see one record is inserted.

ID	Name
101	Rahul

PL/SQL Drop Procedure

Syntax for drop procedure

DROP PROCEDURE procedure name;

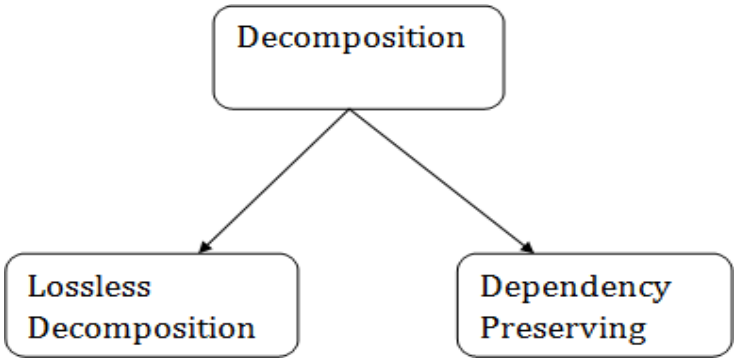
Example of drop procedure

DROP PROCEDURE pro1;

Normalization – Introduction, Non loss decomposition and functional dependencies, First, Second, and third normal forms – dependency preservation, Boyce/Codd normal form. Higher Normal Forms - Introduction, Multi-valued dependencies and Fourth normal form, Join dependencies and Fifth normal form

Decomposition: the process of breaking up or dividing a single relation into two or more sub relations is called as decomposition of a relation.

Decomposition in DBMS removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables.



Lossless Decomposition

- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

Example:

- **EMPLOYEE_DEPARTMENT table:**

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

- The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT
- **EMPLOYEE table:**

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY
22	Denim	28	Mumbai
33	Alina	25	Delhi
46	Stephan	30	Bangalore
52	Katherine	36	Mumbai
60	Jack	40	Noida

- **DEPARTMENT table**
- Now, when these two relations are joined on the common column "EMP_ID", then the resultant relation will look like:

  Employee Department

EMP_ID	EMP_NAME	EMP_AGE	EMP_CITY	DEPT_ID	DEPT_NAME
22	Denim	28	Mumbai	827	Sales
33	Alina	25	Delhi	438	Marketing
46	Stephan	30	Bangalore	869	Finance
52	Katherine	36	Mumbai	575	Production
60	Jack	40	Noida	678	Testing

- Hence, the decomposition is Lossless join decomposition.

Lossy Decomposition

As the name suggests, when a relation is decomposed into two or more relational schemas, the loss of information is unavoidable when the original relation is retrieved.

Let us see an example −

<EmpInfo>

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables −

<EmpDetails>

Emp_ID	**Emp_Name**	**Emp_Age**	**Emp_Location**
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

<DeptDetails>

Dept_ID	**Dept_Name**
Dpt1	Operations
Dpt2	HR
Dpt3	Finance

Now, you won't be able to join the above tables, since **Emp_ID** isn't part of the **DeptDetails** relation.

Therefore, the above relation has lossy decomposition.

Dependency Preserving

- It is an important constraint of the database.
- In the dependency preservation, at least one decomposed table must satisfy every dependency.
- If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.

- For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

Multivalued Dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

BIKE_MODEL → → MANUF_YEAR
BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

Normalization: **Normalization** is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

Update anomaly: we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Ex2:First normal form (1NF)

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212

104	Lester	Bangalore	9990000123 8123450987

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF** as the rule says "each attribute of a table must have atomic (single) values", the emp_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123

104	Lester	Bangalore	8123450987

Example 3 –

ID Name Courses

- - - - - - - - - - - -

1 A c1, c2

2 E c3

3 M C2, c3

In the above table Course is a multi valued attribute so it is not in 1NF.

Below Table is in 1NF as there is no multi valued attribute

ID Name Course

- - - - - - - - - - - -

1 A c1

1 A c2

2 E c3

3 M c2

3 M c3

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".

To make the table complies with 2NF we can break it in two tables like this:
teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics

333	Chemistry

Now the tables comply with Second normal form (2NF).

Second Normal Form –

- a relation must be in first normal form and relation must not contain any partial dependency.
- A relation is in 2NF if it has **No Partial Dependency,** i.e.**,** no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

- **Partial Dependency –** If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Example 1 – Consider table-3 as following below.

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

Note that, there are many courses having the same course fee. }

Here,
COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;

COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;

COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;

Hence,
COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;

But, COURSE_NO -> COURSE_FEE , i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,

we need to split the table into two tables such as :

Table 1: STUD_NO, COURSE_NO

Table 2: COURSE_NO, COURSE_FEE

Table 1

STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1

Table 2

COURSE_NO	COURSE_FEE
C1	1000
C2	1500
C3	1000
C4	2000
C5	2000

Example 2 – Consider following functional dependencies in relation R (A, B , C, D)

AB -> C [A and B together determine C]

BC -> D [B and C together determine D]

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago

555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}... so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys: {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Third Normal Form –

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes as well as it is in second normal form.

A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency X –> Y

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

- **Example 1 –** In relation STUDENT given in Table 4,
  FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE}
  Candidate Key: {STUD_NO}

  For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:
  STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
  STATE_COUNTRY (STATE, COUNTRY)

- **Example 2 –** Consider relation R(A, B, C, D, E)
  A -> BC,
  CD -> E,
  B -> D,
  E -> A
  All possible candidate keys in above relation are {A, E, CD, BC} All attribute are on right sides of all functional dependencies are prime.

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer

21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer

21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Example – Consider the database table of a class whaich has two relations R1 contains student ID(SID) and student name (SNAME) and R2 contains course id(CID) and course name (CNAME).

Table – R1(SID, SNAME)

SID	SNAME
S1	A
S2	B
CID	CNAME
C1	C
C2	D

When there cross product is done it resulted in multivalued dependencies:

Table – R1 X R2

SID	SNAME	CID	CNAME
S1	A	C1	C
S1	A	C2	D
S2	B	C1	C
S2	B	C2	D

Multivalued dependencies (MVD) are:

SID->->CID; SID->->CNAME; SNAME->->CNAME

Multivalued Dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL → → MANUF_YEAR
2. BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

Join Dependency

- Join decomposition is a further generalization of Multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD □ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2, ... , Rn is a lossless-join decomposition.
- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1

Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

UNIT-4

TRANSACTION MANAGEMENT IN DBMS:

- A **transaction** is a set of logically related operations.
- Now that we understand what is transaction, we should understand what are the problems associated with it.
- The main problem that can happen during a transaction is that the transaction can fail before finishing the all the operations in the set. This can happen due to power failure, system crash etc.
- This is a serious problem that can leave database in an inconsistent state. Assume that transaction fail after third operation (see the example above) then the amount would be deducted from your account but your friend will not receive it.

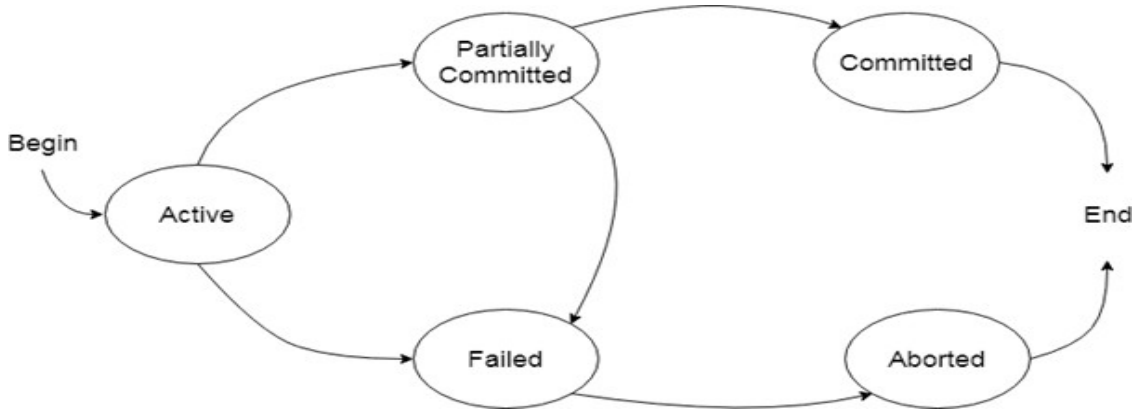
To solve this problem, we have the following two operations

Commit: If all the operations in a transaction are completed successfully then commit those changes to the database permanently.

Rollback: If any of the operation fails then rollback all the changes done by previous operations.

STATES OF TRANSACTION

Transactions can be implemented using SQL queries and Server. In the below-given diagram, you can see how transaction states works.



Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction

TRANSACTION PROPERTY

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

IMPLEMENTATION OF ATOMICITY AND DURABILITY

The recovery-management component of a database system can support atomicity and durability by a variety of schemes.

E.g. the shadow-database scheme:

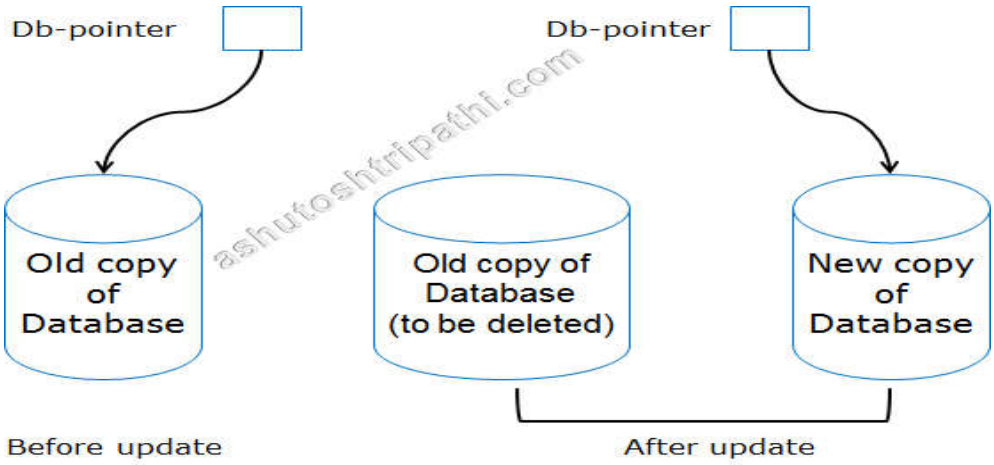
Shadow copy:

- 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.
- This scheme is based on making copies of the database, called shadow copies, assumes that only one transaction is active at a time.
- The scheme also assumes that the database is simply a file on disk. A pointer called db-pointer is maintained on disk; it points to the current copy of the database.

If the transaction completes, it is committed as follows:

- 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. (Unix systems use the flush command for this purpose.)
- 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;
- the new copy then becomes the current copy of the database. The old copy of the database is then deleted.

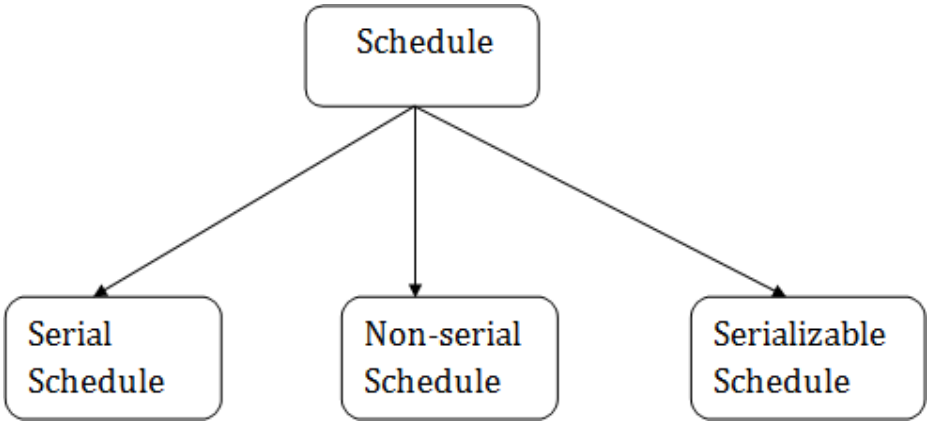
Figure below depicts the scheme, showing the database state before and after the update.



Shadow-copy technique for atomicity and durability

SCHEDULE

A series of operation from one transaction to another transaction is known as schedule. It is used to preserve the order of the operation in each of the individual transaction.



1. SERIAL SCHEDULE

The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction. In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.

For example: Suppose there are two transactions T1 and T2 which have some operations. If it has no interleaving of operations, then there are the following two possible outcomes:

1. Execute all the operations of T1 which was followed by all the operations of T2.
2. Execute all the operations of T1 which was followed by all the operations of T2.

- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.
- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

2. NON-SERIAL SCHEDULE

- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules. It has interleaving of operations.

3. SERIALIZABLE SCHEDULE

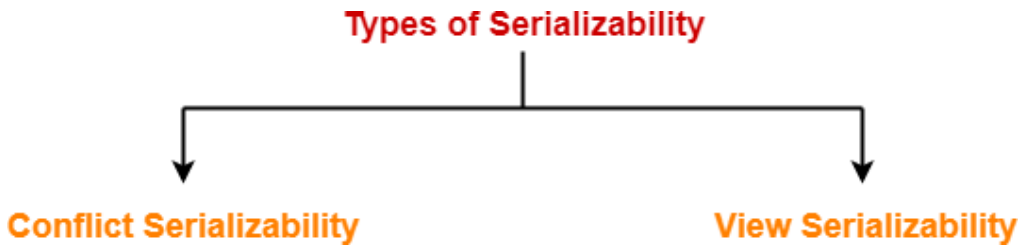
- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

SERIALIZABILITY IN DBMS

- Some non-serial schedules may lead to inconsistency of the database.
- Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.

Types of Serializability

Serializability is mainly of two types-



1. Conflict Serializability
2. View Serializability

Conflict Serializability

If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a **conflict serializable schedule**.

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 the same data item
- At least one of the two operations is a write operation

Example-

Consider the following schedule-

Transaction T1	Transaction T2
R1 (A)	
W1 (A)	
	R2 (A)
R1 (B)	

In this schedule,

- W1 (A) and R2 (A) are called as conflicting operations.
- This is because all the above conditions hold true for them.

Checking Whether a Schedule is Conflict Serializable Or Not-

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Follow the following steps to check whether a given non-serial schedule is conflict serializable or not-

Step-01:

Find and list all the conflicting operations.

Step-02:

Start creating a precedence graph by drawing one node for each transaction.

Step-03:

Draw an edge for each conflict pair such that if $X_i(V)$ and $Y_j(V)$ forms a conflict pair then draw an edge from T_i to T_j.

- This ensures that T_i gets executed before T_j.

Step-04:

Check if there is any cycle formed in the graph.

- If there is no cycle found, then the schedule is conflict serializable otherwise not.

VIEW SERIALIZABILITY?

View Serializability is a process to find out that a given schedule is view serializable or not.

To check whether a given schedule is view serializable, we need to check whether the given schedule is **View Equivalent** to its serial schedule. Lets take an example to understand what I mean by that.

View Serializability

- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read:

An initial read of both schedules must be the same. Suppose two schedule S1 and S2. In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	Write(A)

Schedule S1

T1	T2
Read(A)	Write(A)

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule S1

T1	T2	T3
Write(A)	Write(A)	Read(A)

Schedule S2

3. Final Write

A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S1

T1	T2	T3
Write(A)	Read(A)	Write(A)

Schedule S2

Above two schedules is view equal because Final write operation in S1 is done by T3 and in S2, the final write operation is also done by T3.

T1	T2	T3
Read(A) Write(A)	Write(A)	Write(A)

Recoverability of Schedule

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But

TRANSACTION ISOLATION LEVELS IN DBMS

some other transaction may also have used value produced by the failed transaction. So we also have to rollback those transactions.

The SQL standard defines four isolation levels :

1. **Read Uncommitted –** Read Uncommitted is the lowest isolation level. In this level, one transaction may read not yet committed changes made by other transaction, thereby allowing dirty reads. In this level, transactions are not isolated from each other.
2. **Read Committed –** This isolation level guarantees that any data read is committed at the moment it is read. Thus it does not allows dirty read. The transaction holds a read or write lock on the current row, and thus prevent other transactions from reading, updating or deleting it.
3. **Repeatable Read –** This is the most restrictive isolation level. The transaction holds read locks on all rows it references and writes locks on all rows it inserts, updates, or

deletes. Since other transaction cannot read, update or delete these rows, consequently it avoids non-repeatable read.

4. **Serializable –** This is the Highest isolation level. A *serializable* execution is guaranteed to be serializable. Serializable execution is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

FAILURE CLASSIFICATION

To find that where the problem has occurred, we generalize a failure into the following categories:

1. Transaction failure
2. System crash
3. Disk failure

1. Transaction failure

The transaction failure occurs when it fails to execute or when it reaches a point from where it can't go any further. If a few transaction or process is hurt, then this is called as transaction failure.

Reasons for a transaction failure could be -

1. **Logical errors:** If a transaction cannot complete due to some code error or an internal error condition, then the logical error occurs.
2. **Syntax error:** It occurs where the DBMS itself terminates an active transaction because the database system is not able to execute it. **For example,** The system aborts an active transaction, in case of deadlock or resource unavailability.

2. System Crash

- System failure can occur due to power failure or other hardware or software failure. **Example:** Operating system error.

Fail-stop assumption: In the system crash, non-volatile storage is assumed not to be corrupted.

3. Disk Failure

- It occurs where hard-disk drives or storage drives used to fail frequently. It was a common problem in the early days of technology evolution.
- Disk failure occurs due to the formation of bad sectors, disk head crash, and unreachability to the disk or any other failure, which destroy all or part of disk storage.

CONCURRENT EXECUTION OF TRANSACTION

In the transaction process, a system usually allows executing more than one transaction simultaneously. This process is called a concurrent execution.

Advantages of concurrent execution of a transaction

1. Decrease waiting time or turnaround time.
2. Improve response time
3. Increased throughput or resource utilization.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

1: Lost Update Problems (W - W Conflict)

2. Dirty Read Problems (W-R Conflict)

3. Unrepeatable Read Problem (W-R Conflict)

1. Lost update problem (Write – Write conflict)

This type of problem occurs when two transactions in database access the same data item and have their operations in an interleaved manner that makes the value of some database item incorrect.

If there are two transactions T1 and T2 accessing the same data item value and then update it, then the second record overwrites the first record.

Example: Let's take the value of A is 100

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2	A=A-50	
t3		Read(A)
t4		A=A+50
t5	Write(A)	
t6		Write(A)

Here,

- At t1 time, T1 transaction reads the value of A i.e., 100.
- At t2 time, T1 transaction deducts the value of A by 50.
- At t3 time, T2 transactions read the value of A i.e., 100.
- At t4 time, T2 transaction adds the value of A by 150.
- At t5 time, T1 transaction writes the value of A data item on the basis of value seen at time t2 i.e., 50.

- At t6 time, T2 transaction writes the value of A based on value seen at time t4 i.e., 150.
- So at time T6, the update of Transaction T1 is lost because Transaction T2 overwrites the value of A without looking at its current value.
- Such type of problem is known as the Lost Update Problem.

Dirty read problem (W-R conflict)

This type of problem occurs when one transaction T1 updates a data item of the database, and then that transaction fails due to some reason, but its updates are accessed by some other transaction.

Example: Let's take the value of A is 100

Time	Transaction T1	Transaction T2
t1	Read(A)	
t2	A=A+20	
t3	Write(A)	
t4		Read(A)
t5		A=A+30
t6		Write(A)
t7	Write(B)	

Here,

- At t1 time, T1 transaction reads the value of A i.e., 100.

- At t2 time, T1 transaction adds the value of A by 20.
- At t3 time, T1transaction writes the value of A (120) in the database.
- At t4 time, T2 transactions read the value of A data item i.e., 120.
- At t5 time, T2 transaction adds the value of A data item by 30.
- At t6 time, T2transaction writes the value of A (150) in the database.
- At t7 time, a T1 transaction fails due to power failure then it is rollback according to atomicity property of transaction (either all or none).
- So, transaction T2 at t4 time contains a value which has not been committed in the database. The value read by the transaction T2 is known as a dirty read.

Unrepeatable read (R-W Conflict)

It is also known as an inconsistent retrieval problem. If a transaction T_1 reads a value of data item twice and the data item is changed by another transaction T_2 in between the two read operation. Hence T_1 access two different values for its two read operation of the same data item.

Example: Let's take the value of A is 100

Time	**Transaction T1**	**Transaction T2**
t1	Read(A)	
t2		Read(A)
t3		A=A+30
t4		Write(A)
t5	Read(A)	

Here,

- At t1 time, T1 transaction reads the value of A i.e., 100.

- At t2 time, T2transaction reads the value of A i.e., 100.
- At t3 time, T2 transaction adds the value of A data item by 30.
- At t4 time, T2 transaction writes the value of A (130) in the database.
- Transaction T2 updates the value of A. Thus, when another read statement is performed by transaction T1, it accesses the new value of A, which was updated by T2. Such type of conflict is known as R-W conflict.

CONCURRENCY CONTROL

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

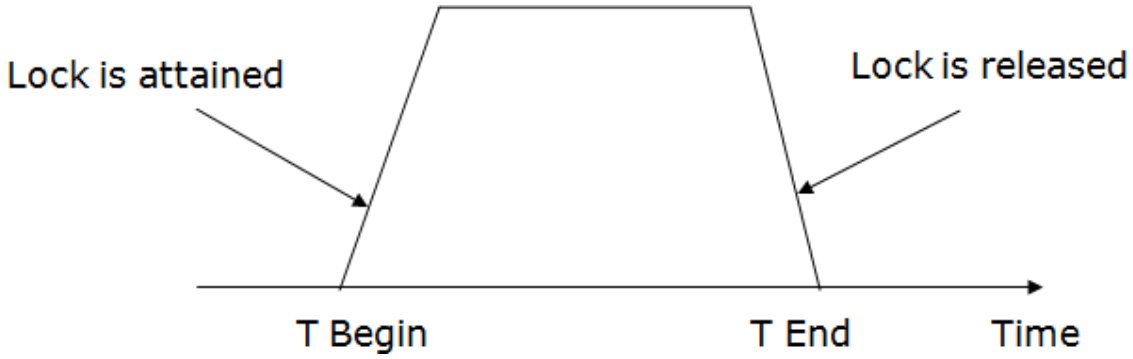
- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

TWO-PHASE LOCKING (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Example:

	T1	T2
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

The following way shows how unlocking and locking work with 2-PL.

Transaction T1:

- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7

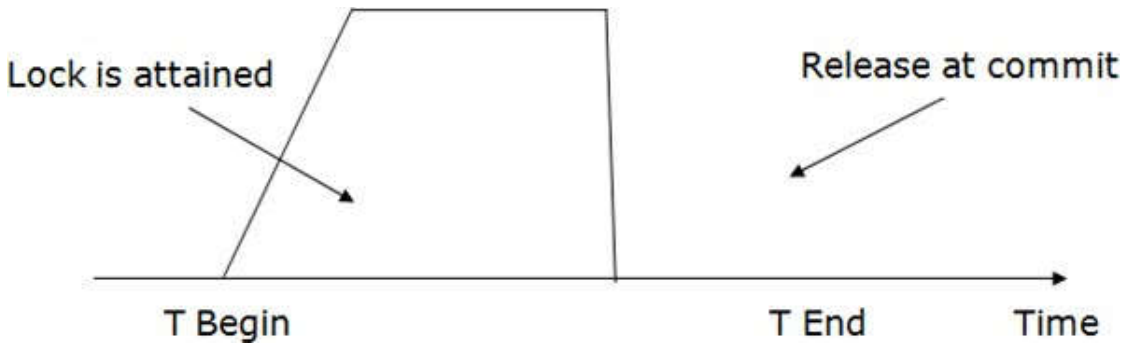
- **Lock point:** at 3

Transaction T2:

- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

4. Strict Two-phase locking (Strict-2PL)

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release.



TIMESTAMP ORDERING PROTOCOL

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:

- If W_TS(X) >TS(Ti) then the operation is rejected.
- If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:

- If TS(Ti) < R_TS(X) then the operation is rejected.
- If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

1. **Read phase:** In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
2. **Validation phase:** In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.
3. **Write phase:** If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Here each phase has the following different timestamps:

Start(Ti): It contains the time when Ti started its execution.

Validation (T_i): It contains the time when Ti finishes its read phase and starts its validation phase.

Finish(Ti): It contains the time when Ti finishes its write phase.

- This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.
- Hence TS(T) = validation(T).
- The serializability is determined during the validation process. It can't be decided in advance.
- While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.
- Thus it contains transactions which have less number of rollbacks.

THOMAS WRITE RULE

Thomas Write Rule provides the guarantee of serializability order for the protocol. It improves the Basic Timestamp Ordering Algorithm.

The basic Thomas write rules are as follows:

- If TS(T) < R_TS(X) then transaction T is aborted and rolled back, and operation is rejected.
- If TS(T) < W_TS(X) then don't execute the W_item(X) operation of the transaction and continue processing.
- If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction Ti and set W_TS(X) to TS(T).

MULTIPLE GRANULARITY

Let's start by understanding the meaning of granularity.

Granularity: It is the size of data item allowed to lock.

Multiple Granularity:

- It can be defined as hierarchically breaking up the database into blocks which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.
- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
- Hence, the levels of the tree starting from the top level are as follows:
    - Database

- Area
- File
- Record

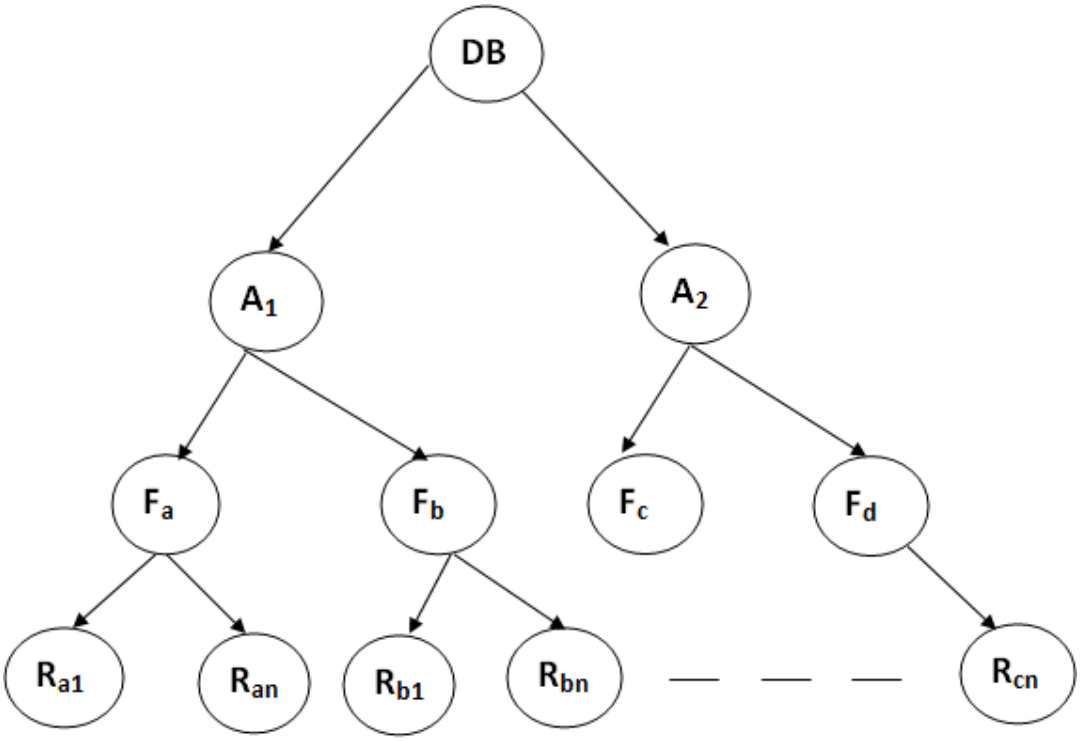


Figure: Multi Granularity tree Hierarchy

Recovery and Atomicity – Log – Based Recovery – Recovery with Concurrent Transactions – Check Points - Buffer Management – Failure with loss of nonvolatile storage-Advance Recovery systems- ARIES Algorithm, Remote Backup systems. File organization – various kinds of indexes - B+ Trees- Query Processing – Relational Query Optimization.

Recovery and Atomicity:

- When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.
- But according to ACID properties of **DBMS**, **atomicity** of transactions as a whole must be maintained, that is, either all the operations are executed or none.
- Database **recovery** means **recovering** the data when it get deleted, hacked or damaged accidentally.
- Atomicity is must whether is transaction is over or not it should reflect in the database permanently or it should not effect the database at all.

When a DBMS recovers from a crash, it should maintain the following −

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Log-Based Recovery

- The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- If any operation is performed on the database, then it will be recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied in the database.

There are two approaches to modify the database:

1. Deferred database modification:

- The deferred modification technique occurs if the transaction does not modify the database until it has committed.
- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. **Immediate database modification**:

- The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery with Concurrent Transactions

Concurrency control means that multiple transactions can be executed at the same time and then the interleaved logs occur. But there may be changes in transaction results so maintain the order of execution of those transactions.
During recovery, it would be very difficult for the recovery system to backtrack all the logs and then start recovering.

Recovery with concurrent transactions can be done in the following four ways.

1. Interaction with concurrency control
2. Transaction rollback
3. Checkpoints

4. Restart recovery

Interaction with concurrency control:

In this scheme, the recovery scheme depends greatly on the concurrency control scheme that is used. So, to rollback a failed transaction, we must undo the updates performed by the transaction.

Transaction rollback :

- In this scheme, we rollback a failed transaction by using the log.
- The system scans the log backward a failed transaction, for every log record found in the log the system restores the data item.

Checkpoints :

- Checkpoints is a process of saving a snapshot of the applications state so that it can restart from that point in case of failure.
- Checkpoint is a point of time at which a record is written onto the database form the buffers.
- Checkpoint shortens the recovery process.
- When it reaches the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till the next checkpoint and so on.
- The checkpoint is used to declare the point before which the DBMS was in the consistent state, and all the transactions were committed.

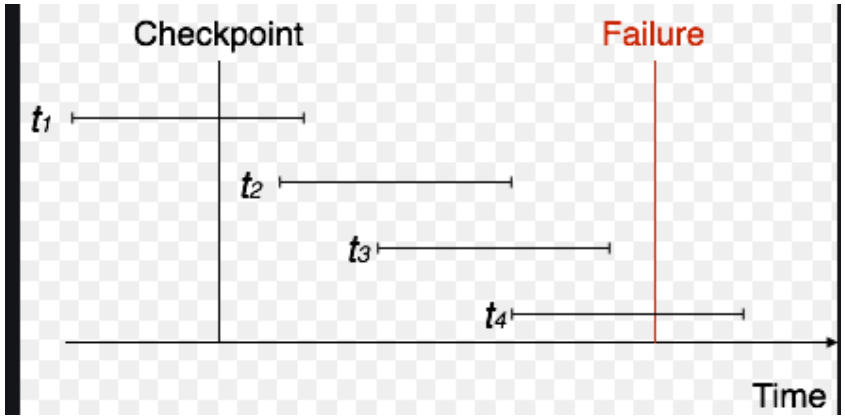
Restart recovery:

- When the system recovers from a crash, it constructs two lists.
- The undo-list consists of transactions to be undone, and the redo-list consists of transaction to be redone.
- The system constructs the two lists as follows: Initially, they are both empty. The system scans the log backward, examining each record, until it finds the first <checkpoint> record.

Check Points:

- Checkpoints are a process of saving a snapshot of the applications state so that it can restart from that point in case of failure.
- Checkpoint is a point of time at which a record is written onto the database form the buffers.
- Checkpoint shortens the recovery process.
- When it reaches the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till the next checkpoint and so on.

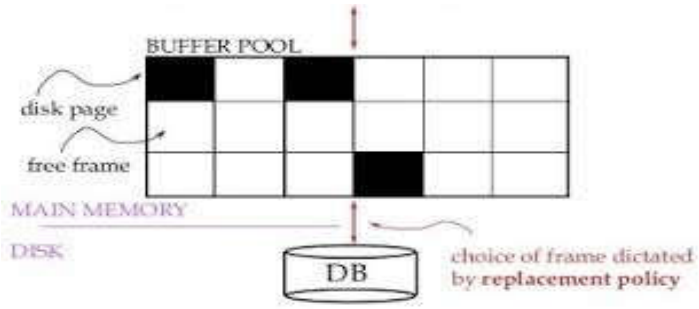
The checkpoint is used to declare the point before which the DBMS was in the consistent state, and all the transactions were committed.



BUFFER MANAGEMENT

The **buffer manager** is the software layer that is responsible for bringing pages from physical disk to main memory as needed. The buffer manages the available main memory by dividing the main memory into a collection of pages, which we called as **buffer pool.** The main memory pages in the buffer pool are called **frames.**

- **Data must be in RAM for DBMS to operate on it!**
- **Buffer manager hides the fact that not all data is in RAM.**



Buffer Manager

- A Buffer Manager is responsible for allocating space to the buffer in order to store data into the buffer.
- If a user request a particular block and the block is available in the buffer, the buffer manager provides the block address in the main memory.
- If the block is not available in the buffer, the buffer manager allocates the block in the buffer.
- If free space is not available, it throws out some existing blocks from the buffer to allocate the required space for the new block.
- The blocks which are thrown are written back to the disk only if they are recently modified when writing on the disk.
- If the user requests such thrown-out blocks, the buffer manager reads the requested block from the disk to the buffer and then passes the address of the requested block to the user in the main memory.
- However, the internal actions of the buffer manager are not visible to the programs that may create any problem in disk-block requests. The buffer manager is just like a virtual machine

Failure with Loss of Nonvolatile Storage

Loss of Volatile Storage

A volatile storage like RAM stores all the active logs, disk buffers, and related data. In addition, it stores all the transactions that are being currently executed. What happens if such a volatile storage crashes abruptly? It would obviously take away all the logs and active

copies of the database. It makes recovery almost impossible, as everything that is required to recover the data is lost.

Following techniques may be adopted in case of loss of volatile storage −

- We can have **checkpoints** at multiple stages so as to save the contents of the database periodically.
- A state of active database in the volatile memory can be periodically **dumped** onto a stable storage, which may also contain logs and active transactions and buffer blocks.
- <dump> can be marked on a log file, whenever the database contents are dumped from a non-volatile memory to a stable one.

Recovery

- When the system recovers from a failure, it can restore the latest dump.
- It can maintain a redo-list and an undo-list as checkpoints.
- It can recover the system by consulting undo-redo lists to restore the state of all transactions up to the last checkpoint.

ARIES Algorithm:

Algorithm for Recovery and Isolation Exploiting Semantics (ARIES) is based on the Write Ahead Log (WAL) protocol. Every update operation writes a log record which is one of the following :

1. **Undo-only log record:**
   Only the before image is logged. Thus, an undo operation can be done to retrieve the old data.
2. **Redo-only log record:**
   Only the after image is logged. Thus, a redo operation can be attempted.
3. **Undo-redo log record:**
   Both before images and after images are logged.

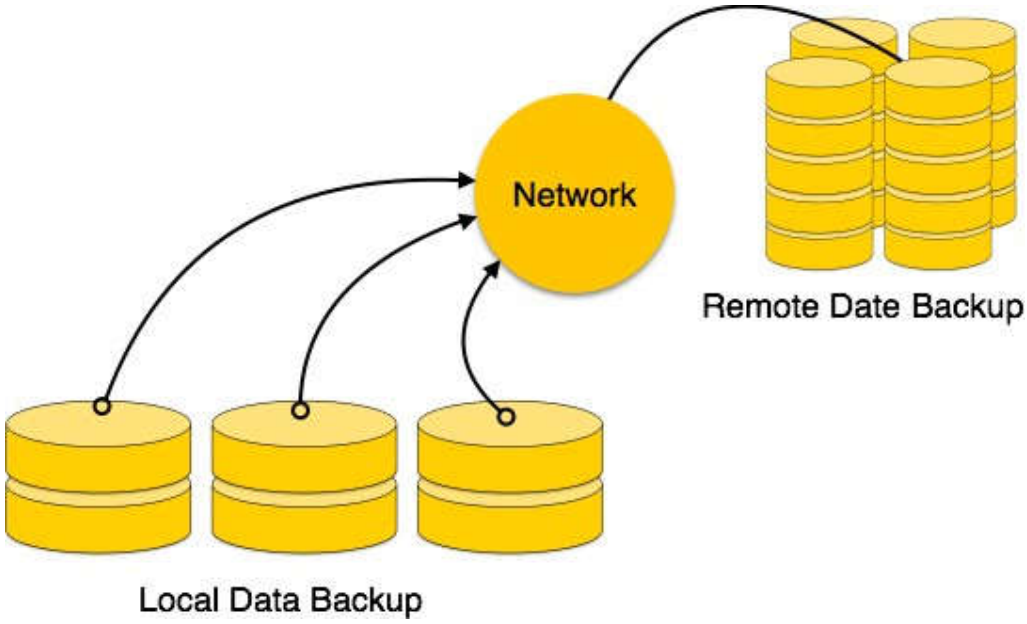
- In it, every log record is assigned a unique and monotonically increasing log sequence number (LSN).
- Every data page has a page LSN field that is set to the LSN of the log record corresponding to the last update on the page.
- WAL requires that the log record corresponding to an update make it to stable storage before the data page corresponding to that update is written to disk.
- For performance reasons, each log write is not immediately forced to disk. A log tail is maintained in main memory to buffer log writes.
- The log tail is flushed to disk when it gets full. A transaction cannot be declared committed until the commit log record makes it to disk.
- Once in a while the recovery subsystem writes a checkpoint record to the log. The checkpoint record contains the transaction table and the dirty page table.
- A master log record is maintained separately, in stable storage, to store the LSN of the latest checkpoint record that made it to disk.
- On restart, the recovery subsystem reads the master log record to find the checkpoint's LSN, reads the checkpoint record, and starts recovery from there on.

The recovery process actually consists of 3 phases:

1. **Analysis:**
   The recovery subsystem determines the earliest log record from which the next pass must start. It also scans the log forward from the checkpoint record to construct a snapshot of what the system looked like at the instant of the crash.
2. **Redo:**
   Starting at the earliest LSN, the log is read forward and each update redone.
3. **Undo:**
   The log is scanned backward and updates corresponding to loser transactions are undone.

Remote Backup

Remote backup provides a sense of security in case the primary location where the database is located gets destroyed. Remote backup can be offline or real-time or online. In case it is offline, it is maintained manually.



Online backup systems are more real-time and lifesavers for database administrators and investors. An online backup system is a mechanism where every bit of the real-time data is backed up simultaneously at two distant places. One of them is directly connected to the system and the other one is kept at a remote place as backup.

As soon as the primary database storage fails, the backup system senses the failure and switches the user system to the remote storage. Sometimes this is so instant that the users can't even realize a failure.

File – A file is named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tables and optical disks.