CMP206 Principles of Database Management Systems Syllabus

• Introduction to Database Management Systems: Database – An Introduction, The Database Management System, Advantages of Using a Database, Features of Data in a Database, Components of a DBMS, The Three Level Architecture for a Database System, Different Levels of Abstraction, Database Design

• Data Modelling: Types of Data Models, Record Based Logical Models, Object Based Data Models, Other Data Models

• Entity Relationship Modelling: Entity – Relationship Model, Components of an E–R model, E – R Diagram, Relationships

• Introduction to Relational Database Management Systems: RDBMS Terminology, The Relational Data Structure, Keys, Relational Data Manipulation

• Normalization: Relational Database Design, Functional Dependencies, Closures of a Set of Functional Dependencies, Closures of a Attribute Sets, Canonical Cover, Normalization, Normalization Using Functional Dependencies, Boyce – Codd Normal Form (BCNF), Normalization Using Multivalued Dependencies, Normalization Using Join Dependencies

• Structured Query Language: Introduction, History, Advantages, SQL Commands, SQL Data Types and Literals, Literals, SQL Operators, Embedded SQL

• Queries and Sub queries: Basic Queries in SQL, Aggregate Functions, Grouping While Selecting, Joins, Set Operations, Sub queries, Join Vs Subqueries

• DML and DDL Commands: Data Manipulation Language Commands, Data Definition Language Commands, Tables, Views

• Transaction Processing: The Concept of Transaction, States of Transaction, Concurrent Execution of Multiple transactions, Serializability

• Concurrency Control and Deadlock Recovery: Introduction, Lock – Based Protocols, Protocols, Timestamp – Based Protocols, Thomas' Write Rule, Validation – Based Protocols, Deadlock Handling

# Chapter 1 : Introduction to Database Management Systems

Database – An Introduction, The Database Management System, Advantages of Using a Database, Features of Data in a Database, Components of a DBMS, The Three Level Architecture for a Database System, Different Levels of Abstraction, Database Design

Database – An Introduction:-

Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information. Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks. A database management system stores data in such a way that it becomes easier to retrieve, manipulate, and produce information.

**Characteristics**

Traditionally, data was organized in file formats. DBMS was a new concept then, and all the research was done to make it overcome the deficiencies in traditional style of data management. A modern DBMS has the following characteristics:

**Real-world entity**: A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.

**Relation-based tables**: DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names. Isolation of data and application: A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.

**Less redundancy**: DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Normalization is a mathematically rich and scientific process that reduces data redundancy.

**Consistency:** Consistency is a state where every relation in a database remains consistent. There exist methods and techniques, which can detect attempt of leaving database in inconsistent state. A DBMS can provide greater consistency as compared to earlier forms of data storing applications like file-processing systems.

**Query Language**: DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.

ACID Properties: DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.

**Multiuser and Concurrent Access**: DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.

**Multiple views:** DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.

**Security:** Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break

**WHAT IS DATABASE MANAGEMENT SYSTEM:**

A primary aim of the database system is to provide a convenient A database is a computer generated software program which and efficient way to store and retrieve data stored in a database. can be used to access the data stored in database in an The term database is a structured collection of data stored organised manner. which can be stored in digital form. Before the actual data is stored in the database, we should clearly specify the schema of the database and different techniques used to manipulate the Database shouldn't only care about the insertion and data stored in a database. modification of the data in the database. At times, it should also focus on how to protect the data stored in the database from DBMS must provide efficient techniques in order to protect the unauthorised access. data from accidental system crashes. If the data has to be shared among number of users there are highly chances that the data might not remain consistent because too many users might try to access it at same time and may try to change the value.

**ADVANTAGES AND DISADVANTAGES OF DATABASE SYSTEMS**

**Advantages of Database Systems -**

**Controlling Data Redundancy** In the conventional file processing system, every user group maintains its own files for handling its data files. This may lead to Duplication of same data in different files. Wastage of storage space, since duplicated data is stored. Errors may be generated due to updation of the same data in different files. Time in entering data again and again is wasted. Computer Resources are needlessly used. It is very difficult to combine information.

**Elimination of Inconsistency** In the file processing system information is duplicated throughout the system. So changes made in one file may be necessary be carried over to another file. This may lead to inconsistent data. So we need to remove this duplication of data in multiple file to eliminate inconsistency. Let us consider the following example of student. Imagine that a particular student has opted for Embedded system as one of the elective subject in Sem –V for TYBScIT Sem V examination while filling up the examination form. If, after getting the hall ticket the student realize that rather than expecting Embedded system as the choice of elective subject in the hall ticket, if some other subject is highlighted, it means that the data for that student has not correctly inserted in the database. To avoid the above problem, there is a need to have a centralize database in order to have this conflicting information. On centralizing the data base the duplication will be controlled and hence inconsistency will be removed.

**Integrity can be improved:** Since data of the organization using database approach is centralized and would be used by a number of users at a time, it is essential to enforce integrity-constraints. o In the conventional systems because the data is duplicated in multiple files so updating or changes may sometimes lead to entry of incorrect data in some files wherever it is applicable.

*For example*: - The example of Hall Ticket Generation system that we have already discussed, since multiple files are to maintained, as sometimes you may enter a value for subject which may not exist. Suppose Elective Subjects can have values (Embedded Systems, Advanced Java, Web Designing etc) but we enter a value 'Mathematics -I' for it, it may lead to database inconsistency. o Even if we centralized the database it may still contain incorrect data. For example: - Salary of full time clerk may be entered as Rs. 1500 rather than Rs. 4500. A student may be shown to have borrowed library books but has no enrollment. The above problems can be avoided by defining the validation procedures whenever any update operation is attempted

**Provides backup and Recovery:** Centralizing a database provides the schemes such as recovery and backups from the failures including disk crash, power failures, software errors which may help the database to recover from the inconsistent state to the state that existed prior to the occurrence of the failure, though methods are very complex

**Disadvantages of Database Systems:**

*1 Database Complexity* The design of the database system is complex, difficult and is very time consuming task to perform.

*2. Substantial hardware and software start-up costs:* Huge amount of investment is needed to setup the required hardware and the softwares needed to run those applications.

*3. Damage to database affects virtually all applications programs* If one part of the database is corrupted or damaged because of the hardware or software failure, since we don't have many versions of the file, all the application programs which are dependent on this database are implicitly affected.

*4. Extensive conversion costs in moving form a file-based system to a database system* If you are currently working on file based system and need to upgrade it to database system, then large amount of cost is incurred in purchasing different tools, adopting different techniques as per the requirement.
*5. Initial training required for all programmers and user*. Large amount of human efforts, the time and cost is needed to train the end users and application programmers in order to get used to the database systems.

**DATABASE STRUCTURE**

In a database structure, the DBMS acts as an interface between the user and the database. o o The user requests the DBMS to perform various operations such as insert, delete, update and retrieval on the database. o o The components of DBMS perform these requested operations on the database and provide necessary data to the users

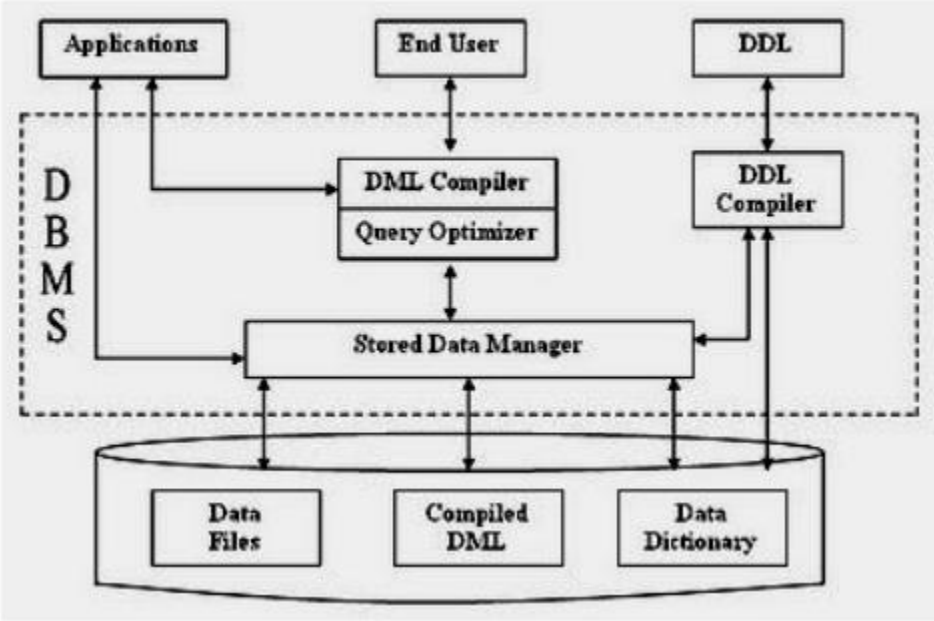



**Fig. 3.3 Structure Of DBMS**

1. **DDL Compiler**
   Data Description Language compiler processes schema definitions specified in the DDL. It includes metadata information such as the name of the files, data items, storage details of each file, mapping information and constraints etc.

2. **DML Compiler and Query optimizer**
   The DML commands such as insert, update, delete, retrieve from the application program are sent to the DML compiler for compilation into object code for database access. The object code is then optimized in the best way to execute a query by the query optimizer and then send to the data manager.

3. **Data Manager**

The Data Manager is the central software component of the DBMS also knows as Database Control System.

**The Main Functions Of Data Manager are**

- It convert operations in user's Queries coming from the application programs or combination of DML Compiler and Query optimizer which is known as Query Processor from user's logical view to physical file system.
- It controls DBMS information access that is stored on disk.
- It also controls handling buffers in main memory.

4. **Data Dictionary**

Data Dictionary is a repository of description of data in the database. It contains information about

- *Data* - names of the tables, names of attributes of each table, length of attributes, and number of rows in each table.
- *Relationships* between database transactions and data items referenced by them which are useful in determining which transactions are affected when certain data definitions are changed.
- *Constraints* on data i.e. range of values permitted.
- Detailed information on physical database design such as storage structure, access paths, files and record sizes.
- *Access Authorization* which is the description of database users their responsibilities and their access rights.
- *Usage statistics* such as frequency of query and transactions.

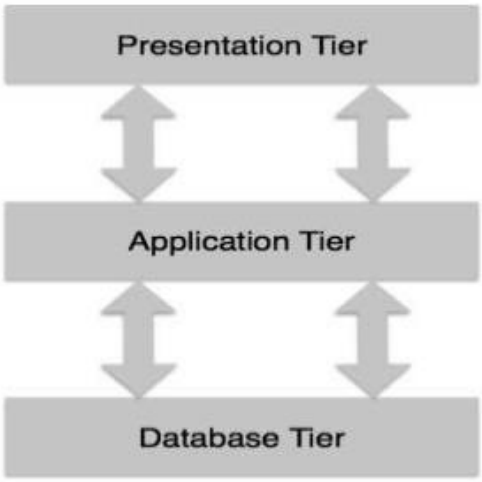
5. **Data Files -** It contains the data portion of the database.

6. **Compiled DML -** The DML complier converts the high level Queries into low level file access commands known as compiled DML.

7. **End Users –** They are the users of the system who is going to use the system for their day to day activities.

## 3-tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.




[*Image: 3-tier DBMS architecture*]

- **Database (Data) Tier**: At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

- **Application (Middle) Tier**: At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
- **User (Presentation) Tier**: End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

**Database design**

The database design process consists of a number of steps listed below. We will focus mainly on step 2, the conceptual database design, and the models used during this step.

**Step 1: Requirements Collection and Analysis**

- Prospective users are interviewed to understand and document data requirements
- This step results in a concise set of user requirements, which should be detailed and complete.
- The functional requirements should be specified, as well as the data requirements. Functional requirements consist of user operations that will be applied to the database, including retrievals and updates.

- Functional requirements can be documented using diagrams such as sequence diagrams, data flow diagrams, scenarios, etc.

**Step 2: Conceptual Design**

- Once the requirements are collected and analyzed, the designers go about creating the conceptual schema.
- Conceptual schema: concise description of data requirements of the users, and includes a detailed description of the entity types, relationships and constraints.
- The concepts do not include implementation details; therefore the end users easily understand them, and they can be used as a communication tool.
- The conceptual schema is used to ensure all user requirements are met, and they do not conflict.

**Step 3: Database Implementation**

- Many DBMS systems use an implementation data model, so the conceptual schema is transformed from the high-level data model into the implementation data model.
- This step is called logical design or data model mapping, which results in the implementation data model of the DBMS.

**Step 4: Physical Design**

- Internal storage structures, indexes, access paths and file organizations are specified.
  Application programs are designed and implemented

# Chapter 2 : Data Models

Types of Data Models, Record Based Logical Models, Object Based Data Models, Other Data Models

- A *data model* is a picture or description which shows how the data is to be arranged to achieve a given task.
- It is a clear model which specifies how the data items are arranged in a given model.
- Some data models which gives a clear picture which shows the manner in which the data records are connected or related within a file structure. These are called structural data models.
- DBMS organize and structure data so that it can be retrieved and manipulated by different users and application programs.

---

- The data structures and access techniques provided by a particular DBMS are called its data model.
- A data model determined both the personality of a DBMS and the applications for which it is particularly well suited.

---

## 4.3 TYPES OF DATA MODEL

There are four different types of data models

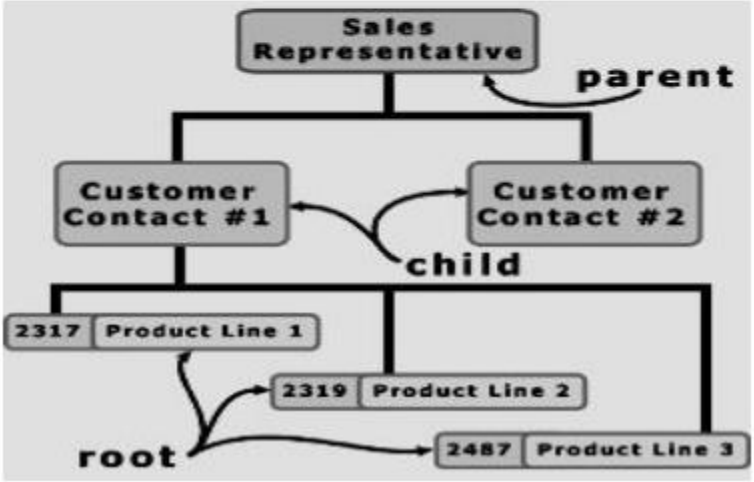
4.3.1 Hierarchical databases

4.3.2 Network databases

4.3.3 Relational databases

4.3.4 Object oriented databases

### *4.3.1 Hierarchical databases*




---

Hierarchical Databases is most commonly used with mainframe systems. o It is one of the oldest methods of organizing and storing data and it is still used by some organizations for making travel reservations. o A hierarchical database is organized in pyramid fashion, like the branches of a tree extending downwards

### 4.3.2Network databases


- **Network databases** are similar to hierarchical databases by also having a hierarchical structure. There are a few key differences, however.

---

Instead of looking like an upside-down tree, a network database looks more like a cobweb or interconnected network of records. In network databases, children are called members and parents are called owners

**3 Relational databases** : Pre-relational models depended upon being able to determine explicitly where and how individual records were stored.

o Early relational proponents argued that the relational data model viewed information logically rather than physically, but this is not quite correct.

o Earlier data models associated the logical and physical aspects of information together; logically-related information was stored in physical proximity within a data file. The relational data model first separated the logical from the physical aspects.

**4 Object oriented databases** o A data model is a logic organization of the real world objects (entities), constraints on them, and the relationships among objects. A DB language is a concrete syntax for a data model. A DB system implements a data model.

A core object-oriented data model consists of the following basic object-oriented concepts:

(1) object and object identifier: Any real world entity is uniformly modeled as an object (associated with a unique id: used to pinpoint an object to retrieve).

(2) attributes and methods: Here every object has a state (the set of values for the attributes of the object) and a behavior (the set of methods - program code - which operate on the state of the object). The state and behavior encapsulated in an object are accessed or invoked from outside the object only through explicit message passing. An attribute is an instance variable, whose domain may be any class: user-defined or primitive. A class composition hierarchy (aggregation relationship) is orthogonal to the concept of a class hierarchy. The link in a class composition hierarchy may form cycles.

(3) class: a means of grouping all the objects which share the same set of attributes and methods. An object must belong to only one class as an instance of that class (instance-of relationship). A class is similar to an abstract data type. A class may also be primitive (no attributes), e.g., integer, string, Boolean.

(4) Class hierarchy and inheritance: derive a new class (subclass) from an existing class (superclass). The subclass inherits all the attributes and methods of the existing class and may have additional attributes and methods. single inheritance (class hierarchy) vs. multiple inheritance (class lattice).

# Chapter 3 : Entity Relationship Modelling

Entity – Relationship Model, Components of an E–R model, E – R Diagram, Relationships

Data models define how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected to each other and how they are processed and stored inside the system.

The very first data model could be flat data-models, where all the data used are to be kept in the same plane. Earlier data models were not so scientific, hence they were prone to introduce lots of duplication and update anomalies.

## Entity-Relationship Model

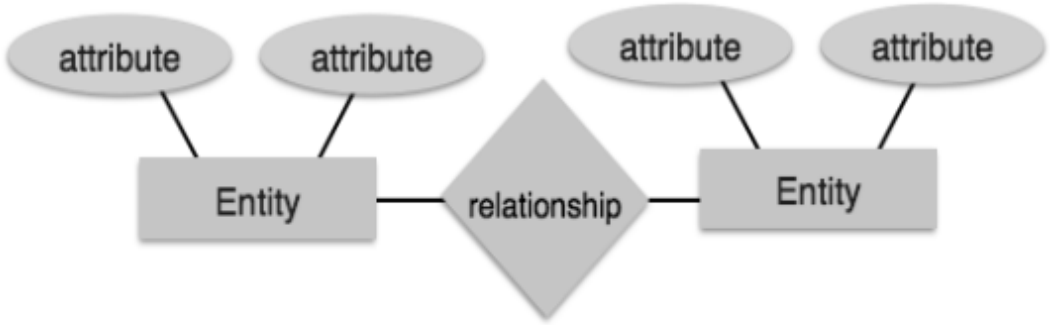
Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes, and constraints.

ER Model is best used for the conceptual design of a database.

ER Model is based on:

- **Entities** and their attributes.
- **Relationships** among entities.

These concepts are explained below.




*[Image: ER Model]*

**Entity** An entity in an ER Model is a real-world entity having properties called attributes. Every attribute is defined by its set of values called domain. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.

**Relationship** The logical association among entities is called relationship. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities. Mapping cardinalities: o one to one o one to many o many to one o many to many

**ER Diagram**

- A database can be modeled as

  A collection of entities

  Relationship among the entities
- An entity is an real world object that exist and it is distinguishable from other entities

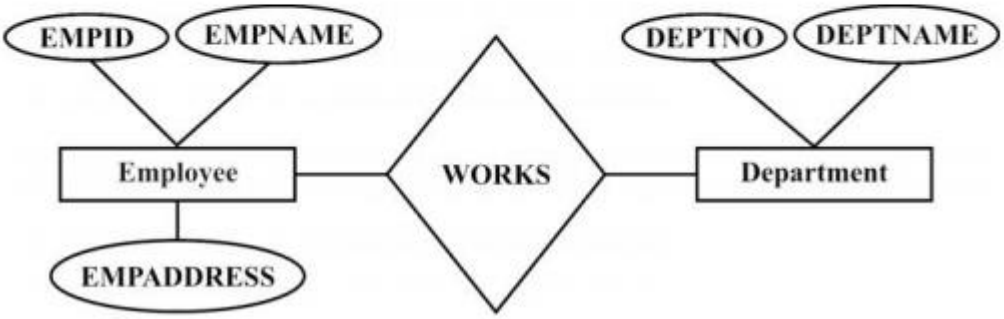
  Example Person, company, event, plant
- All the entities in the data model have attributes as known as properties of an entities

  Example: people have names and addresses

An Entity set is a set of an entities of all same type that share the same properties.

Example: set of all persons ,companies,trees, holidays

**ER Diagram**

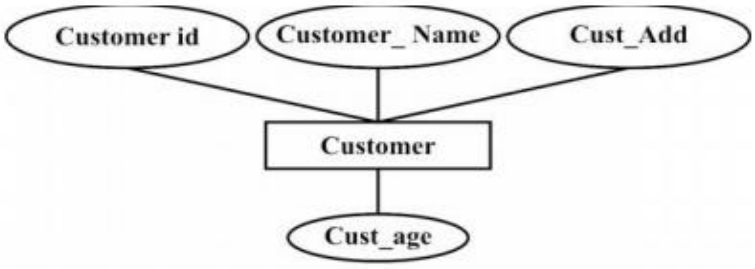



- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Lines link attributes to entity sets and entity sets to relationship sets.
- Underline indicates primary key attributes
- Ellipses represent an attributes
- Double Lines represent total participation of an entity in a relationship set
- Double rectangle represent a weak entity sets

**Strong Entity type** An entity type which has own distinct primary key that used to identify specific uniquely from another entity type is called as Strong Entity type An Entity type which is independent on some other entity type icalled Strong Entity type

**Example** In the Case of Client entity Client_no is the primary key of Client entity which is used to uniquely identified among the Client 's entity set In the case of Customer Entity , Customer_id is the primary key of Customer Entity which is used to uniquely identified among the Customer's entity set Strong Entity type is represented by rectangle Symbol

Strong Entity type is represented by rectangle Symbol




Customer Entity

**Weak entity Type**

Entity type which is dependent on some other entity type is called as Weak entity type

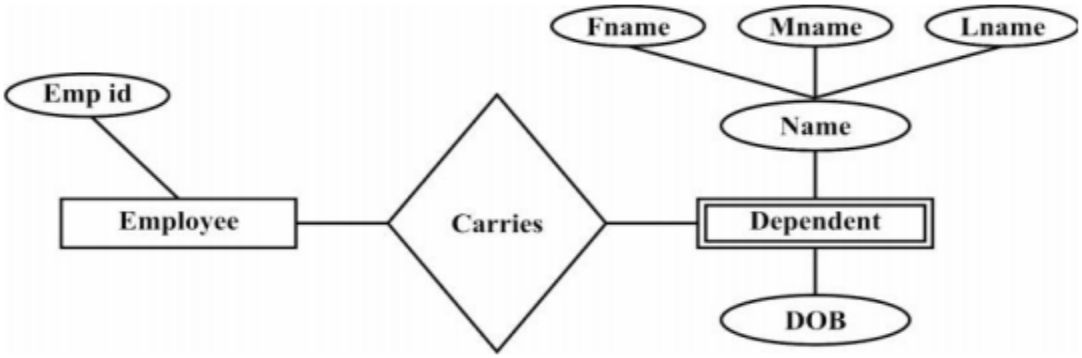
- Weak entity type is dependent on a strong entity and cannot exist on its own
- It does not have a unique identifier that has partial identifier
- Partial identifier is represented by double-line

Some weak entities assign partial identifiers and such partial identifiers of an weak entity called as discriminator

Weak entity type is represented by double rectangle.

Identify relationship

Strong entity type is link with the weak entity type




Dependent entity depend upon Employee entity for primary key

**Attributes** Properties of an entity or relationship type is called as attribute Example Staffno, staffname,staff_designation is describes an entity Staff Value of an attribute play a major role of data strored in database Each entity will have the value which is assigned to its attributes Consider an example Above stated example of Staff Entity which has the attribute named as staffno, the value which is assigned to the staff attribute is '101' and the staffname attribute has the value is 'Mahendra, and staff_desigination attribute has the value is 'Manager'

**Attribute domains** The set of allowable values which is assigned to one or more attribute is knowns as Attribute domains There are types of attributes has been classified Such as simple and Composite type,single valued and multi valued attributes Stored and derived attributes, null attributes and Key attributes

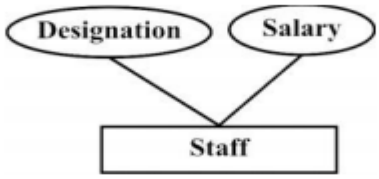
**1) Simple Attributes**

Simple attributes is an attributes which can further divided in to two parts

**Or**

An Attribute composed of single compoenent with an independent existence

For an example: Desgination of an staff and Salary of an staff




**Simple Attributes**

**Composite Attribute**

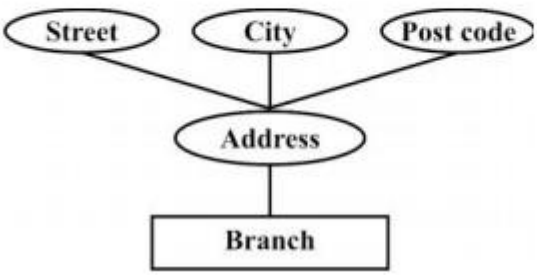
Composite Attribute is an attribute which is futher divided into many parts

**Or**

An attributed composed of multiple component, each component has its own independent existence

**Example**

Address attributes of an Branch entity that can be further divided in to sub parts i.e street, city and postalcode as an attributes



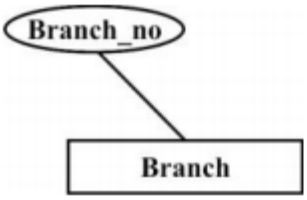
**2) Single valued and Mutli Valued attributes**

Single valued attribute is an attribute which as single value(atomic) for each entity.

**Or**

An attribute that holds a single valuefor each occurrence of an entity type

Example: Each branch has only single valued attributes is known as branch_no




**Single Valued attributes**

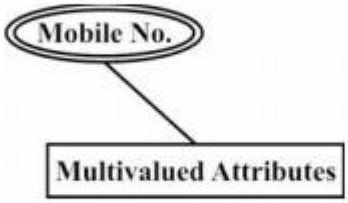
**Mutli valued attributes**

Mutli valued attribute is an attributes which as many values for each entity

**Or**

An attribute that holds multiple values for each occurrence of an entity type.

**Example :** Each staff member has multiple mobile numbers




**Multivalued Attributes**

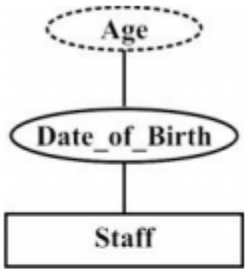
**3) Stored and Derived attributes**

Stored attributes is an attribute which is used supplied a value to the related attreibute

Example Date_of_Birth of an staff is a stored attributes

**Derived attributes**

The value from the derived attribute is derived from the stored attribute for an example Date_of_Birth is a stored attribute for an each staff member . The value for an Age can be derived from the Date_of_ Birth attributes I.e by subtracting the Date_of _Birth from the Current date, therefore the Stored attributes is used supplied a value to the related attributes




**Null attribute**

The attribute which take NULL value when entity does not have the value to it.

The Null attribute is an attribute their value is unknown, unassigned and missing information

**Key Attributes**

This attribute has the unique value for an entity which is used to identified given row in the table is called as key attribute of an entity

**Example :** Staff_ no is an key attribute which has an unique value which is used to identifies given row in the table

Let us now learn how the ER Model is represented by means of an ER diagram. Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

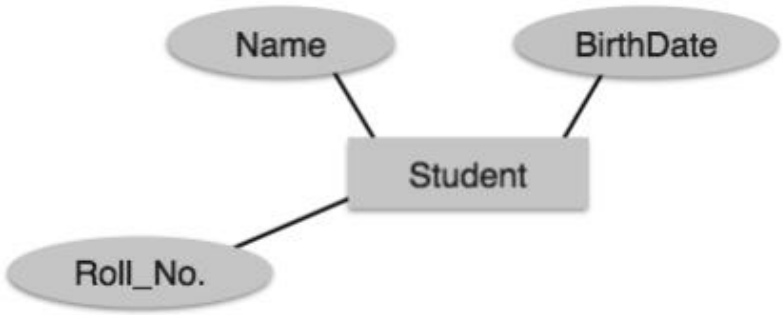
## Entity

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.


[*Image: Entities in a school database*]

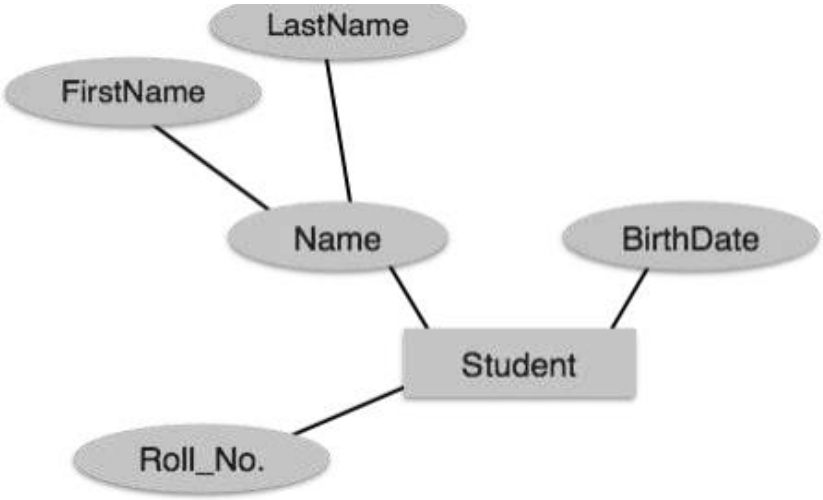
## Attributes

Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



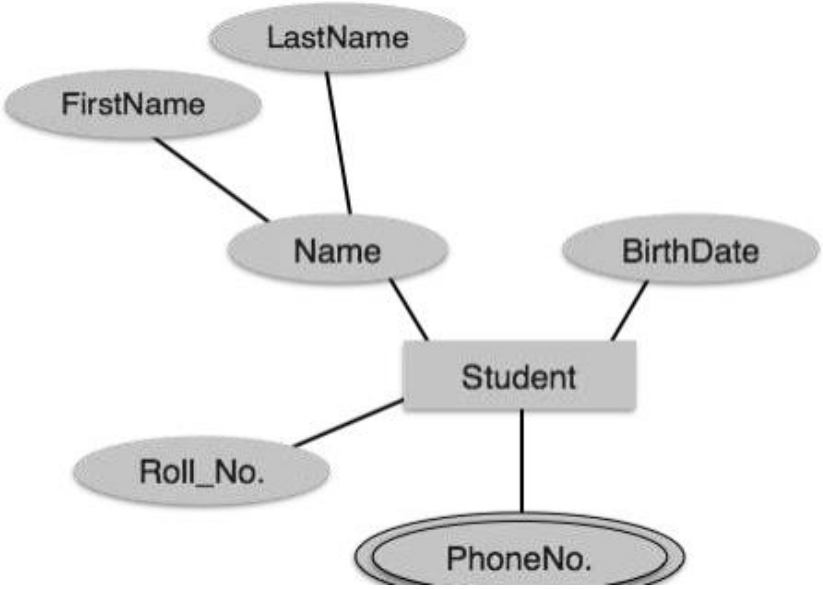

[*Image: Simple Attributes*]

If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.

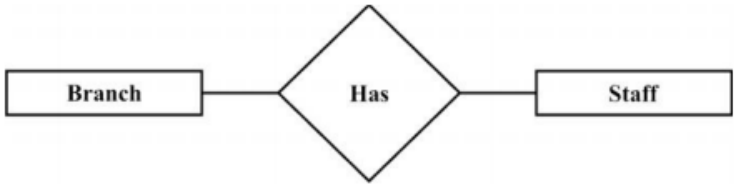



[*Image: Composite Attributes*]

**Multivalued** attributes are depicted by double ellipse.



**Relationships** A set of meaningful relationship among several entities We used to inidicate the diamond symbol for Relationships among the several entities, it could read from left to right
Example : Branch has a staff




**Degree of relationship**

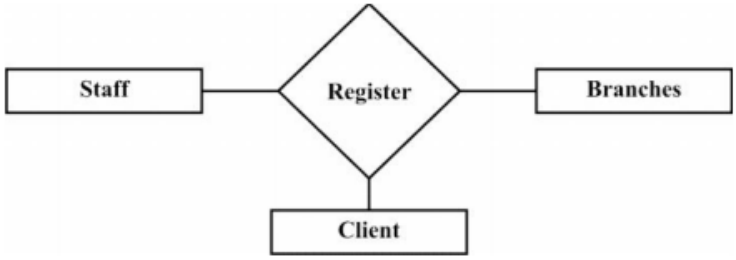
It is the number of entities participated in a particular relational model

There are two type of degree of relationship.

**Binary relationship**: A Relationship of degree two is called as binary relationship

**Ternary Relationship**: A relationship of degree three is called as Ternary relationship.

**Example**




**Constraints on relationship** 1) Mapping Constraints / Cardinalities The number (or range) of possible entity type that is associated to another entity type through a particular entity Cardinalities indicates that a specific number of entity occurrence of related entity .
Type of Mapping Constraints
**1 )One-to-one 2) One-to-many 3)Many- to-one 4)Many-to-many**
**One to One-**

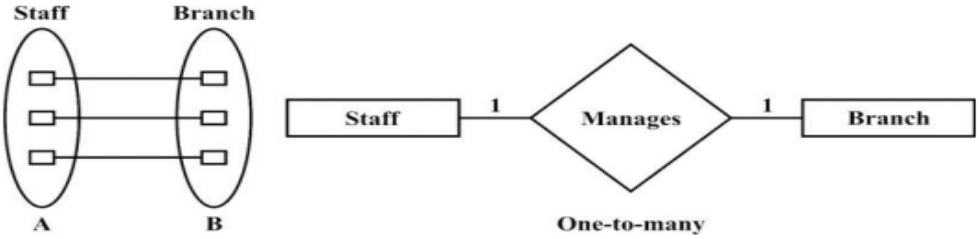
That is one row on an table is related to an one row of another table

i.e A is associated with at most one entity in B and Bis associated with at most one entity in A

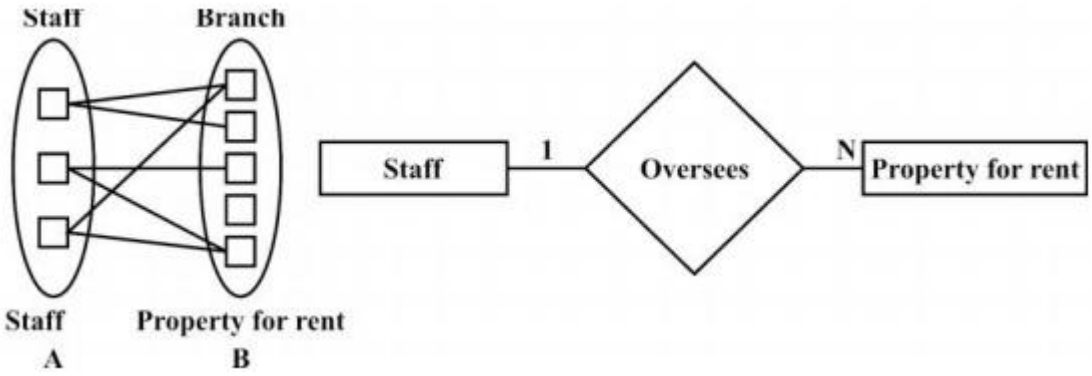
**Example**

Each branch is managed by one member of the staff that's means Branch Manager

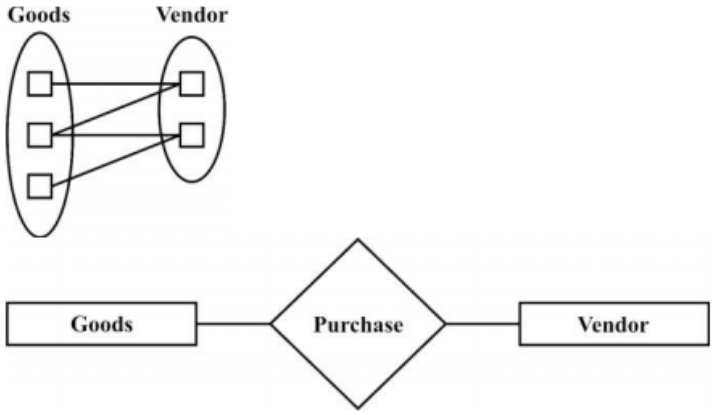
A member of staff can manage zero or one branch



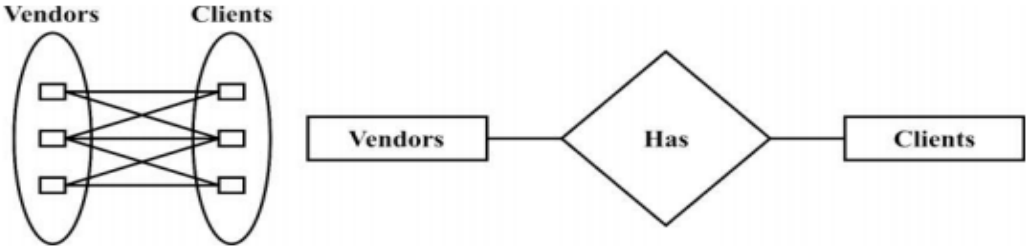
**2) One- to- many In this constraints**, One record in the entity can be related with many record in other entity A is associated with any number of entities in B B is associated with at most one entity in A E.g. each member of staff oversees zero or more prosperity for rent Every row in the Staff table can have relationship with many rows in the properityforRent Table




**3) One To Many** In this type Mapping Constraints , Many records in the one enity is related to the only one records in the other entity An entity in A is associated with at least one entity in B . an entity in B can be associated with any number of entities in A. Example one vendors has many Goods and Many Goods is purchase by one Vendors

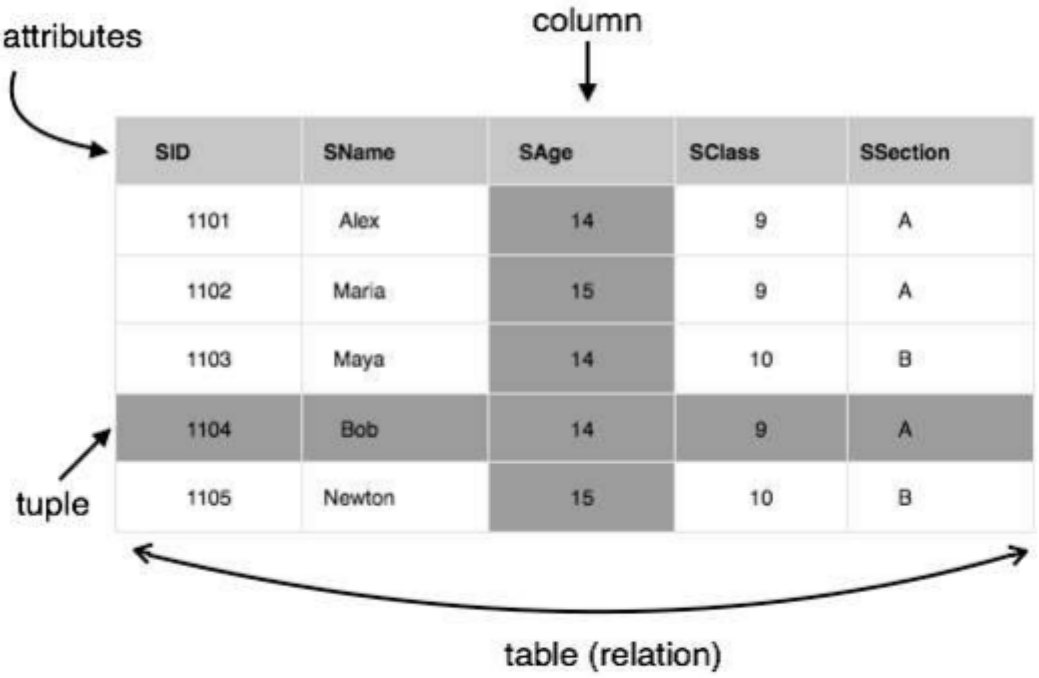



**4) Many to Many** In this Mapping Constraints , Many records in the entity is related Many records in the other entity An entity in A is associated with any number of entities in B. and an entity in B is associated with any number of entities in A. Many Vendors Has Clients and Many Clients has may Vendors



# Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.




| SID | SName | SAge | SClass | SSection |
|---|---|---|---|---|
| 1101 | Alex | 14 | 9 | A |
| 1102 | Maria | 15 | 9 | A |
| 1103 | Maya | 14 | 10 | B |
| 1104 | Bob | 14 | 9 | A |
| 1105 | Newton | 15 | 10 | B |

[*Image: Table in relational Model*]

The main highlights of this model are:

- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain

# Chapter 4 : Introduction to Relational Database Management Systems

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

**Concepts**

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

***Tuple:*** A single row of a table, which contains a single record for that relation is called a tuple. Relation ***instance:*** A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

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

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

***Attribute domain:*** Every attribute has some predefined value scope, known as attribute domain. Constraints Every relation has some conditions that must hold for it to be a valid relation. These conditions are called Relational Integrity

***Constraints***. There are three main integrity constraints:

Key constraints• Domain constraints• Referential integrity constraints•

**Key**

Definition A Column value in the table that uniquely identifies a single record in the table is called key of an table A attribute or the set of attribute in the table that uniquely identifies each record in the entity set is called a key for that entity set

**Types of keys**

Simple Key: A key which has the single attribute is known as a simple key

Composite key: A key which consist two or more attributes is called a Composite Key.

*Example:* Cust_id is a key attribute of Customer Table it is possible to have a single key for one customer i.e is Cust_id ie Cust_id =1 is only for the Cust_name =”Yogesh” please refer to the Customer Table which is mentioned above.

| Types of key | Definition of Key |
|---|---|
| Super Key | A key is called super key which is sufficient to identify the unique record in the table |
| Candidate Key | A minimal super key is called Candidate key .A super key has no proper subset of candidate key |
| Primary Key | A candidate key is chosen as a principal to identify a unique |
| Secondary Key | |
| Foreign Key | an Column (or combination of Columns) in the one tables whose values is match the primary key in the another table |

**Secondary key**

Defination Seconday key of the table consist the column and combination of the some columns which meant for data retrival purpose. The secondary key not always required to primary key, other tah the pimary key there are some attribute which is required to retrieve data from the customer table using the another attribute such as Cust_Name and Cust_Age columns

# Chapter 5: Normalization

Relational Database Design, Functional Dependencies, Closures of a Set of Functional Dependencies, Closures of a Attribute Sets, Canonical Cover, Normalization, Normalization Using Functional Dependencies, Boyce – Codd Normal Form (BCNF), Normalization Using Multivalued Dependencies, Normalization Using Join Dependencies

**RELATIONAL DATABASE DESIGN PROCESS**

While designing relational database model you have consider in the mind that how choose a best model in the real world and how this best model is fitted in the database. While designing the relational model you have o consider that which table you want to create, what column the table will consist, consider the relationship between the tables. While developing the relational model it would be nice you process was totally clear and intuitive it can be even better to automated.

The benefits of a relational Database Design process.

Data entry, updating and deleting would be efficient and The benefits of a relational Database Design process. it can be even better to automated.

Data retrieval, summarization and reporting will be efficient simple in manner

Database must follows a well designed model hence it behave predictably

Large amount of information must stored in the database rather than in the application, the database must somewhat well documented

Change to database structured are easy to make e.r creating database, tables , views.

**Feature of Good Relational Database Design- Normalization**

i) In the Relational Database Design, the process of organizing data to minimizing redundancy is known as Normalization

ii) The main aim of the Normalization is to decompose complex relation into smaller, well-structured relation

iii) Normalization is the process that involves dividing a large table into smaller table(which contain less redundant data) and stating the relationship among the tables.

iv) Data normalization or Database Normalization is also canonical synthesis is mean for preventing the inconsistent in a set of data by using unique values to reference common information

v) The main objective of the normalization is to isolate the data so that user can apply the operation such as addition, deletion and modification of a field in one table and then its propagated to the rest of the database through the well defined relationships

vi) The same set of data is repeated in multiple tables of database so there are chances that data in the database may lead to be inconsistent, so while updating , deleting or inserting the data into the inconsistent database which leads to problem of data integrity

vii) If we can apply the normalization on the table we can reduce the problem of data inconsistency for some extent

## NORMAL FORM:

Normal form are designed for addressing potential problem in the database such that inconsistent and redundant data which is stored in the database

Normal form is based on relation rather than table . The normal form has a set of attribute which table should be satify. The Following attributes are

1) They describe one entity
2) They do not have duplicate rows, hence there must a primary key for each row.
3) The columns are unordered
4) The rows are unordered

**Types of Normal Forms**

1) Edgar F. Codd, the inventor of the relational model, introduced the concept of normalization and what we now know as the First Normal Form (1NF) in 1970.
2) Second Normal Form (2NF) and Third Normal Form (3NF) in 1971,
3) Codd and Raymond F. Boyce defined the Boyce-Codd Normal Form (BCNF) in 1974.
4) Fourth normal form(4NF)
5) Fifth Normal form(5NF)
6) Higher normal forms were defined by other theorists in subsequent years, the most recent being the Sixth Normal Form (6NF) introduced by Chris Date, Hugh Darwen, and Nikos Lorentzos in 2002.

---

**First Normal Form**

This Normal form is introduced by Edgar F. Codd, is Known as First Normal Form(1NF) in 1971 Definition A relational database table which consist first normal form (1NF) is to meets certain minimum set of criteria. These criteria are basically concerned with ensuring that the table is a faithful representation of a relation and that it is free of repeating groups

1. There are no duplicated rows in the table.
2. Each cell is single-valued (i.e., there are no repeating groups or arrays).
3. Entries in a column (attribute, field) are of the same kind.

Let us consider the example Consider a table"Customer_Rental " consisting the attribute such as Customer_NO, Cust_Name Property_no,P_Address, Rent_start, Rent_finish,Rent ,Owner_No,Owner_Name

| Customer_ NO | Cust_Name | Property_ no | P_Address | Rent_start | Rent_finish | Rent | Owner_ No | Owner_Name |
|---|---|---|---|---|---|---|---|---|
| CR78 | Mahesh Lad | PG34<br>PG78 | Nerul,Navi Mumbai<br>Turbhe, Navi mumbai | 1-July-91<br>1-Nov-95 | 30-Oct-95<br>1-Nov-98 | 450<br>500 | C045<br>C093 | Sanjay More<br>Mahavir Jain |
| CR98 | Pramod Patel | PG34<br>PG36<br>PG78 | Nerul,Navi Mumbai<br>Kalyan,Thane<br>Karjat,Raigad | 1-July-95<br>1-Nov-97<br>1-july-96 | 30-Oct-98<br>1-Nov-99<br>1-Nov-97 | 450<br>350<br>450 | C045<br>C093<br>C093 | Sanjay More<br>Mahavir Jain<br>Mahavir Jain |

The above table does not contain the atomaic values in the Property_no, P_Address , Rent_start, Rent_finish,Rent ,Owner_No,Owner_Name Hence it is called un-normalizes table,we cannot Insert ,update and delete the record from the table because it is inconsistent state .The above tabe has to be normimalized

| Customer_ NO | Cust_Name | Property_ no | P_Address | Rent_start | Rent_finish | Rent | Owner_No | Owner_Name |
|---|---|---|---|---|---|---|---|---|
| CR78 | Mahesh Lad | PG34 | Nerul,Navi Mumbai | 1-July-91 | 30-Oct-95 | 450 | C045 | Sanjay More |
| CR78 | Mahesh Lad | PG78 | Nerul,Navi Mumbai | 1-Nov-95 | 1-Nov-98 | 500 | C093 | Mahavir Jain |
| CR98 | Pramod Patel | PG34 | Nerul,Navi Mumbai | 1-July-95 | 30-Oct-98 | 450 | C045 | Sanjay More |
| CR98 | Pramod Patel | PG36 | Kalyan,Thane | 1-Nov-97 | 1-Nov-99 | 350 | C093 | Mahavir Jain |
| CR98 | Pramod Patel | PG78 | Karjat,Raigad | 1-july-96 | 1-Nov-97 | 450 | C093 | Mahavir Jain |

---

The above table show the same set of data as the previous table however we have eliminated the repeated groups.so the table shown in the above table to be in First Normal form(1NF)

**Second Normal Form** Second Normal Form based on the concept of Full Functional Dependency and it tries remove the problem of redundant data in the First normal form.
Defination: A 2NF relation in 1NF and every non-primary key attritube is fully functionally dependent on the primary key Converting from 1NF to 2NF: o Firstly Identify the primary key for the 1NF relation. o Identify whether the functional dependencies in the relation. o If partial dependencies exist on the primary key remove them by placing then in a new relation along with a copy of their determinant.
*Example*
Functional Dependency for Customer_Rental Relation
Step 1 : Primary key: Customer_No + Property_no
Step 2 :Full Functional Dependency: (Customer_No+Property_No)->(Rent-Start, RentFinish)
Step3 Partial Dependency: (Customer_No+Property_No)->Cust_Name
(Customer_No+Property_No)->(P_Address, Rent, Owner_No, Owner_Name)

| Customer_NO | Cust_Name | Property_no | P_Address | Rent_start | Rent_finish | Rent | Owner_No | Owner_Name |
|---|---|---|---|---|---|---|---|---|

---

Customer Relation

| Customer_NO | Cust_Name |
|---|---|
| CR78 | Mahesh Lad |
| CR98 | Pramod Patel |

---

Rental Relation

| Customer_NO | Property_No | Rent_start | Rent_finish |
|---|---|---|---|
| CR78 | PG34 | 1-July-91 | 30-Oct-95 |
| CR78 | PG78 | 1-Nov-95 | 1-Nov-98 |
| CR98 | PG34 | 1-July-95 | 30-Oct-98 |
| CR98 | PG36 | 1-Nov-97 | 1-Nov-99 |
| CR98 | PG78 | 1-july-96 | 1-Nov-97 |

Property_owner Relation

| Property_No | P_Address | Rent | Owner_No | Owner_Name |
|---|---|---|---|---|
| PG34 | Nerul,Navi Mumbai | 450 | C045 | Sanjay More |
| PG78 | Nerul,Navi Mumbai | 500 | C093 | Mahavir Jain |
| PG36 | Kalyan,Thane | 350 | C093 | Mahavir Jain |

Here Customer_no is the only key to identify The Customer name hence Customer_No is the primary key in the Customer Relation Table but Forieng key in the Rental relation table

---

**Third Normal Form** Third Normal form Based on the concept of transitive dependency. A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on the primary key.

*Converting from 2NF to 3NF*: o Identify the primary key in the 2NF relation. o Identify functional dependencies in the relation. o If transitive dependencies exist on the primary key remove them by placing them in a new relation along with a copy of their dominant

Property_Owner to 3NF Relations

Property_owner Relation

| Property_No | P_Address | Rent | Owner_No | Owner_Name |
|---|---|---|---|---|

**Transitive Dependency:**

(Customer_No+Property_No)->Owner_No

Owner_No ->OName

Property_for_Rent

| Property_No | P_Address | Rent | Owner_No |
|---|---|---|---|
| PG34 | Nerul,Navi Mumbai | 450 | C045 |
| PG78 | Nerul,Navi Mumbai | 500 | C093 |
| PG36 | Kalyan,Thane | 350 | C093 |

**Owner**

| Owner_No | Owner_Name |
|---|---|
| C045 | Sanjay More |
| C093 | Mahavir Jain |
| | |

---

**Boyce-Codd Normal Form (BCNF)**

o Based on functional dependencies that takes into account all candidate keys in a relation.
o For a relation with only one candidate key, 3NF and BCNF are equivalent.
o A relation is in BCNF, if and only if every determinant is a candidate key.
o Violation of BCNF may occur in a relation that
– contains 2 (or more) composite keys
– which overlap and share at least 1 attribute

**3NF to BCNF**

o Identify all candidate keys in the relation.
o Identify all functional dependencies in the relation.
o If functional dependencies exists in the relation where their determinants are not candidate keys for the relation, remove the functional dependencies by placing them in a new relation along with a copy of their determinant.

*Example* - 3NF to BCNF Relations
Client_Interview Relation

| Client_No | Interview_Date | Interview_Date | Staff_No | Room_No |
|---|---|---|---|---|
| CR76 | 13/05/98 | 10.30 | SG5 | G101 |
| CR56 | 13/05/98 | 12.30 | SG5 | G101 |
| CR74 | 13/05/98 | 12.30 | SG37 | G102 |
| CR56 | 01/06/08 | 10.30 | SG5 | G102 |

(Client_No, Interview_Date) -> (Interview_Time, Staff_No, Room_No)

(Staff_No, Interview_Date, Interview_Time) -> Client_No

(Room_No, Interview_date, Interview_Time) -> Staff_No, Client_No

(Staff_No, Interview_Date) -> Room_No

| Client_No | Interview_Date | Interview_Date | Staff_No |
|---|---|---|---|
| CR76 | 13/05/98 | 10.30 | SG5 |
| CR56 | 13/05/98 | 12.30 | SG5 |
| CR74 | 13/05/98 | 12.30 | SG37 |
| CR56 | 01/06/08 | 10.30 | SG5 |

| Staff_No | Interview_Date | Room_No |
|---|---|---|
| SG5 | 13/05/98 | G101 |
| SG37 | 13/05/98 | G102 |
| SG5 | 01/06/08 | G102 |

# Chapter 6: Structured Query Language

Introduction, History, Advantages, SQL Commands, SQL Data Types and Literals, Literals, SQL Operators, Embedded SQL

SQL is a programming language for Relational Databases. It is designed over relational algebra and tuple relational calculus. SQL comes as a package with all major distributions of RDBMS.

SQL comprises both data definition and data manipulation languages. Using the data definition properties of SQL, one can design and modify database schema, whereas data manipulation properties allows SQL to store and retrieve data from database.

## Data Definition Language

SQL uses the following set of commands to define database schema:

### CREATE

Creates new databases, tables, and views from RDBMS.

**For example:**

```
Create database tutorialspoint;
Create table article;
Create view for_students;
```

### DROP

Drops commands, views, tables, and databases from RDBMS.

**For example:**

```
Drop object_type object_name;
Drop database tutorialspoint;
Drop table article;
Drop view for_students;
```

### ALTER

Modifies database schema.

```
Alter object_type object_name parameters;
```

**For example:**

```
Alter table article add subject varchar;
```

This command adds an attribute in the relation **article** with the name **subject** of string type.

## Data Manipulation Language

SQL is equipped with data manipulation language (DML). DML modifies the database instance by inserting, updating, and deleting its data. DML is responsible for all forms data modification in a database. SQL contains the following set of commands in its DML section:

- SELECT/FROM/WHERE
- INSERT INTO/VALUES
- UPDATE/SET/WHERE
- DELETE FROM/WHERE

These basic constructs allow database programmers and users to enter data and information into the database and retrieve efficiently using a number of filter options.

### SELECT/FROM/WHERE

- **SELECT**

  This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.

- **FROM**

  This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.

- **WHERE**

  This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.

**For example:**

```
Select author_name
From book_author
Where age > 50;
```

## INSERT INTO/VALUES

This command is used for inserting values into the rows of a table (relation).

**Syntax:**

```
INSERT INTO table (column1 [, column2, column3 ... ]) VALUES (value1 [,
value2, value3 ... ])
```

Or

```
INSERT INTO table VALUES (value1, [value2, ... ])
```

**For example:**

```
INSERT INTO tutorialspoint (Author, Subject) VALUES ("anonymous",
"computers");
```

## UPDATE/SET/WHERE

This command is used for updating or modifying the values of columns in a table (relation).

**Syntax:**

```
UPDATE table_name SET column_name = value [, column_name = value ...]
[WHERE condition]
```

---

**For example:**

```
UPDATE tutorialspoint SET Author="webmaster" WHERE Author="anonymous";
```

## DELETE/FROM/WHERE

This command is used for removing one or more rows from a table (relation).

**Syntax:**

```
DELETE FROM table_name [WHERE condition];
```

---

DELETE FROM tutorialspoint WHERE Author="unknown";

**SQL Data Types and Literals:**

# SQL Data Types

The SQL data type defines a kind of value that a column can contain.

In a database table, every column is required to have a name and a data type.

These are the general data types in SQL.

| Data-type | Syntax | Explanation |
|---|---|---|
| Integer | INTEGER | The integer data type is used to specify an integer value. |
| Smallint | SMALLINT | The smallint data type is used to specify small integer value. |
| Numeric | NUMERIC(P,S) | It specifies a numeric value. Here 'p' is precision value and 's' is scale value. |
| Real | REAL | The real integer is used to specify a single precision floating point number. |
| Decimal | DECIMAL(P,S) | It specifies a decimal value. Here 'p' is precision value and 's' is scale value. |
| Double precision | DOUBLE PRECISION | It specifies double precision floating point number. |
| Float | FLOAT(P) | It specifies floating-point value e.g. 12.3, 4.5 etc. Here, 'p' is precision value. |
| Character | CHAR(X) | Here, 'x' is the character's number to store. |
| Character varying | VARCHAR2(X) | Here, 'x' is the character's number to store |
| Bit | BIT(X) | Here, 'x' is the number of bits to store |
| Bit varying | BIT VARYING(X) | Here, 'x' is the number of bits to store (length can vary up to x). |
| Date | DATE | It stores year, month and days values. |
| Time | TIME | It stores hour, minute and second values |
| Timestamp | TIMESTAMP | The timestamp data type is used to store year, month, day, hour, minute and second values. |
| Time with time zone | TIME WITH TIME ZONE | It is exactly same as time but also store an offset from UTC of the time specified. |
| Timestamp with time zone | TIMESTAMP with TIME ZONE | It is same as timestamp but also stores an offset from UTC of the time specified. |

**A literal** string is a sequence of bytes or characters, enclosed within either two single quotes (' ') or two double quotes (" ").

**Consider the following facts when using literal strings in a SELECT statement:**

1. Literal strings are enclosed in single or double quotation marks.
2. You can use literal strings just like you normally use a column name in the SELECT statement. The literal string will be displayed in very row of the query result.
3. Literal strings can be concatenated with another literal string or another column by using function CONCAT.
4. Special characters (e.g. single or double quotes) in the literal string need to be escaped.

**Using a literal string in SELECT statement.**

```
-- Use single quotes around literal string
SELECT CategoryID, CategoryName, 'Northwind Category' AS Note
FROM categories;

-- Use double quotes around literal string
SELECT CategoryID, CategoryName, "Northwind Category" AS Note
FROM categories;
```

The two queries above produce the same result set. You can use two single quotes or two double quotes but you can't use one single quote and one double quote for a literal string.

Query result set - 8 rows returned:

| CategoryID | CategoryName | Note |
|---|---|---|
| 1 | Beverages | Northwind Category |
| 2 | Condiments | Northwind Category |
| 3 | Confections | Northwind Category |
| 4 | Dairy Products | Northwind Category |
| 5 | Grains/Cereals | Northwind Category |
| 6 | Meat/Poultry | Northwind Category |
| 7 | Produce | Northwind Category |
| 8 | Seafood | Northwind Category |

# SQL Operators

SQL statements generally contain some reserved words or characters that are used to perform operations such as comparison and arithmetical operations etc. These reserved words or characters are known as operators.

Generally there are three types of operators in SQL:

1. SQL Arithmetic Operators
2. SQL Comparison Operators

| Operators | Descriptions | Examples |
|---|---|---|
| + | It is used to add containing values of both operands | a+b will give 150 |
| - | It subtracts right hand operand from left hand operand | a-b will give -50 |
| * | It multiply both operand?s values | a*b will give 5000 |
| / | It divides left hand operand by right hand operand | b/a will give 2 |
| % | It divides left hand operand by right hand operand and returns reminder | b%a will give 0 |

3. SQL Logical Operators

# SQL Arithmetic Operators:

Let's assume two variables "a" and "b". Here "a" is valued 50 and "b" valued 100.

**Example:**

# SQL Comparison Operators:

Let's take two variables "a" and "b" that are valued 50 and 100.

| Operator | Description | Example |
|---|---|---|
| = | Examine both operands value that are equal or not,if yes condition become true. | (a=b) is not true |
| != | This is used to check the value of both operands equal or not,if not condition become true. | (a!=b) is true |
| < > | Examines the operand?s value equal or not, if values are not equal condition is true | (a<>b) is true |
| > | Examine the left operand value is greater than right Operand, if yes condition becomes true | (a>b) is not true |
| < | Examines the left operand value is less than right Operand, if yes condition becomes true | (a<="" td=""> |
| >= | Examines that the value of left operand is greater than or equal to the value of right operand or not,if yes condition become true | (a>=b) is not true |
| <= | Examines that the value of left operand is less than or equal to the value of right operand or not, if yes condition becomes true | (a<=b) is true |
| !< | Examines that the left operand value is not less than the right operand value | (a!<="" td=""> |
| !> | Examines that the value of left operand is not greater than the value of right operand | (a!>b) is true |

# SQL Logical Operators:

This is the list of logical operators used in SQL.

| Operator | Description |
|---|---|
| ALL | this is used to compare a value to all values in another value set. |
| AND | this operator allows the existence of multiple conditions in an SQL statement. |
| ANY | this operator is used to compare the value in list according to the condition. |
| BETWEEN | this operator is used to search for values, that are within a set of values |
| IN | this operator is used to compare a value to that specified list value |
| NOT | the NOT operator reverse the meaning of any logical operator |
| OR | this operator is used to combine multiple conditions in SQL statements |
| EXISTS | the EXISTS operator is used to search for the presence of a row in a specified table |
| LIKE | this operator is used to compare a value to similar values using wildcard operator |

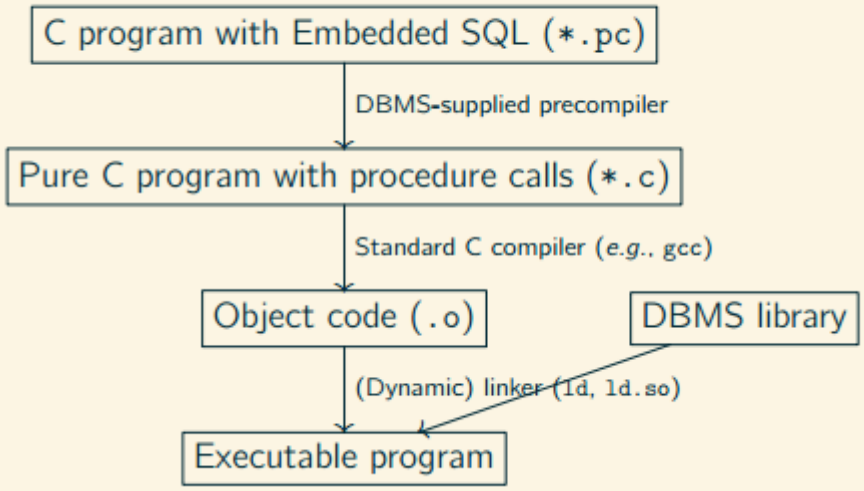
**Embdedded SQL** inserts specially marked SQL statements into program source texts written in C, C++, Cobol, and other PLs.

Inside SQL statements, variables of the PL may be used where SQL allows a constant term only (parameterized queries).

Insert a row into table RESULTS:

EXEC SQL INSERT INTO RESULTS(SID, CAT, ENO, POINTS) VALUES (:sid, :cat, :eno, :points); . Here, sid etc. are C variables and the above may be emdbedded into any C source text

## Compilation/linkage of Embedded SQL programs



# Chapter 7 : Queries and Sub queries

Basic Queries in SQL, Aggregate Functions, Grouping While Selecting, Joins, Set Operations, Sub queries, Join Vs Subqueries

**Basic Queries in SQL**

## SQL CREATE Database

The **SQL CREATE DATABASE** statement is used by a developer to create a database.

Let's see the syntax of SQL CREATE DATABASE:

1. **CREATE DATABASE** database_name;

If you want to add tables in that database, you can use CREATE TABLE statement.

## Create Database in MySQL

In MySQL, same command is used to create a database.

1. **CREATE DATABASE** database_name;

## Create Database in Oracle

You don't need to create database in Oracle. In Oracle database, you can create tables directly.

---

You can also rename, drop and select database that is covered in next pages.

# Aggregate Functions

| | |
|---|---|
| MIN | returns the smallest value in a given column |
| MAX | returns the largest value in a given column |
| SUM | returns the sum of the numeric values in a given column |
| AVG | returns the average value of a given column |
| COUNT | returns the total number of values in a given column |
| COUNT(*) | returns the number of rows in a table |

Aggregate functions are used to compute against a "returned column of numeric data" from your SELECT statement. They basically summarize the results of a particular column of selected data. We are covering these here since they are required by the next topic, "GROUP BY". Although they are required for the "GROUP BY" clause, these functions can be used without the "GROUP BY" clause. For example:

```
SELECT AVG(salary)

FROM employee;
```

This statement will return a single result which contains the average value of everything returned in the salary column from the *employee* table.

Another example:

```
SELECT AVG(salary)


FROM employee

WHERE title = 'Programmer';
```

This statement will return the average salary for all employee whose title is equal to 'Programmer'

Example:

```
SELECT Count(*)

FROM employee;
```

This particular statement is slightly different from the other aggregate functions since there isn't a column supplied to the count function. This statement will return the number of rows in the employees table.

# GROUP BY clause

The GROUP BY clause will gather all of the rows together that contain data in the specified column(s) and will allow aggregate functions to be performed on the one or more columns. This can best be explained by an example:

**GROUP BY** clause syntax:

```
SELECT column1,
SUM(column2)

FROM "list-of-tables"

GROUP BY "column-list";
```

Let's say you would like to retrieve a list of the highest paid salaries in each dept:

```
SELECT max(salary), dept

FROM employee

GROUP BY dept;
```

This statement will select the maximum salary for the people in each unique department. Basically, the salary for the person who makes the most in each department will be displayed. Their, salary and their department will be returned.

# SQL JOIN

As the name shows, JOIN means *to combine something*. In case of SQL, JOIN means **"to combine two or more tables"**.

The SQL JOIN clause takes records from two or more tables in a database and combines it together.

**ANSI standard SQL** defines five types of JOIN :

1. inner join,
2. left outer join,
3. right outer join,
4. full outer join, and
5. cross join.

In the process of joining, rows of both tables are combined in a single table.

## *Why SQL JOIN is used?*

If you want to access more than one table through a select statement.

If you want to combine two or more table then SQL JOIN statement is used .it combines rows of that tables in one table and one can retrieve the information by a SELECT statement.

The joining of two or more tables is based on common field between them.

SQL INNER JOIN also known as simple join is the most common type of join.

## How to use SQL join or SQL Inner Join?

Let an example to deploy SQL JOIN process:

1.Staff table

| ID | Staff_NAME | Staff_AGE | STAFF_ADDRESS | Monthley_Package |
|---|---|---|---|---|
| 1 | ARYAN | 22 | MUMBAI | 18000 |
| 2 | SUSHIL | 32 | DELHI | 20000 |
| 3 | MONTY | 25 | MOHALI | 22000 |
| 4 | AMIT | 20 | ALLAHABAD | 12000 |

2.Payment table

| Payment_ID | DATE | Staff_ID | AMOUNT |
|---|---|---|---|
| 101 | 30/12/2009 | 1 | 3000.00 |
| 102 | 22/02/2010 | 3 | 2500.00 |
| 103 | 23/02/2010 | 4 | 3500.00 |

So if you follow this JOIN statement to join these two tables ?

```
1. SELECT Staff_ID, Staff_NAME, Staff_AGE, AMOUNT
2.   FROM STAFF s, PAYMENT p
3.   WHERE s.ID =p.STAFF_ID;
```

This will produce the result like this:

| STAFF_ID | NAME | Staff_AGE | AMOUNT |
|---|---|---|---|
| 3 | MONTY | 25 | 2500 |
| 1 | ARYAN | 22 | 3000 |
| 4 | AMIT | 25 | 3500 |
| 1 | ARYAN | 22 | 3000 |

# SQL OUTER JOIN

In the SQL outer JOIN all the content of the both tables are integrated together either they are matched or not.

If you take an example of employee table

**Outer join of two types:**

**1.Left outer join** (also known as left join): this join returns all the rows from left table combine with the matching rows of the right table. If you get no matching in the right table it returns NULL values.

**2.Right outer join** (also known as right join): this join returns all the rows from right table are combined with the matching rows of left table .If you get no column matching in the left table .it returns null value.

This diagram shows the different type of joins:

# SQL LEFT JOIN

The SQL left join returns all the values from the left table and it also includes matching values from right table, if there are no matching join value it returns NULL.

**BASIC SYNTAX FOR LEFT JOIN:**

```
1. SELECT table1.column1, table2.column2....
2. FROM table1
3. LEFTJOIN table2
```

4. **ON** table1.column_field = table2.column_field;

let us take two tables in this example to elaborate all the things:

**CUSTOMER TABLE:**

| ID | NAME | AGE | SALARY |
|---|---|---|---|
| 1 | ARYAN | 51 | 56000 |
| 2 | AROHI | 21 | 25000 |
| 3 | VINEET | 24 | 31000 |
| 4 | AJEET | 23 | 32000 |
| 5 | RAVI | 23 | 42000 |

This is second table

**ORDER TABLE:**

| O_ID | DATE | CUSTOMER_ID | AMOUNT |
|---|---|---|---|
| 001 | 20-01-2012 | 2 | 3000 |
| 002 | 12-02-2012 | 2 | 2000 |
| 003 | 22-03-2012 | 3 | 4000 |
| 004 | 11-04-2012 | 4 | 5000 |

**join these two tables with LEFT JOIN:**

1. SQL **SELECT** ID, **NAME**, AMOUNT,**DATE**
2. **FROM** CUSTOMER
3. LEFT JOIN **ORDER**
4. **ON** CUSTOMER.ID = **ORDER**.CUSTOMER_ID;

This will produce the following result:

| ID | NAME | AMOUNT | DATE |
|---|---|---|---|
| 1 | ARYAN | NULL | NULL |
| 2 | AROHI | 3000 | 20-01-2012 |
| 2 | AROHI | 2000 | 12-02-2012 |
| 3 | VINEET | 4000 | 22-03-2012 |
| 4 | AJEET | 5000 | 11-04-2012 |
| 5 | RAVI | NULL | NULL |

# SQL RIGHT JOIN

The SQL right join returns all the values from the rows of right table. It also includes the matched values from left table but if there is no matching in both tables, it returns NULL.

Basic syntax for right join:

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

let us take an example with 2 tables table1 is CUSTOMERS table and table2 is ORDERS table.

**CUSTOMER TABLE:**

| ID | NAME | AGE | SALARY |
|---|---|---|---|
| 1 | ARYAN | 51 | 56000 |
| 2 | AROHI | 21 | 25000 |
| 3 | VINEET | 24 | 31000 |
| 4 | AJEET | 23 | 32000 |
| 5 | RAVI | 23 | 42000 |

and this is the second table:

**ORDER TABLE:**

| DATE | O_ID | CUSTOMER_ID | AMOUNT |
|---|---|---|---|
| 20-01-2012 | 001 | 2 | 3000 |
| 12-02-2012 | 002 | 2 | 2000 |
| 22-03-2012 | 003 | 3 | 4000 |
| 11-04-2012 | 004 | 4 | 5000 |

Here we will join these two tables with SQL RIGHT JOIN:

```
SQL> SELECT ID,NAME,AMOUNT,DATE
FROM CUSTOMER
RIGHT JOIN ORDER
ON CUSTOMER.ID = ORDER.CUSTOMER_ID;
```

| ID | NAME | AMOUNT | DATE |
|---|---|---|---|
| 2 | AROHI | 3000 | 20-01-2012 |
| 2 | AROHI | 2000 | 12-02-2012 |
| 3 | VINEET | 4000 | 22-03-2012 |
| 4 | AJEET | 5000 | 11-04-2012 |

# SQL FULL JOIN

The SQL full join is the result of combination of both left and right outer join and the join tables have all the records from both tables. It puts NULL on the place of matches not found.

SQL full outer join and SQL join are same. generally it is known as SQL FULL JOIN.

# SQL full outer join:

What is SQL full outer join?

SQL full outer join is used to combine the result of both left and right outer join and returns all rows (don?t care its matched or unmatched) from the both participating tables.

**Syntax for full outer join:**

```
1. SELECT *
2. FROM table1
3. FULL OUTER JOIN table2
4. ON table1.column_name = table2.column_name;
```

**Note:**here table1 and table2 are the name of the tables participating in joining and column_name is the column of the participating tables.

Let us take two tables to demonstrate full outer join:

**table_A**

| A | M |
|---|---|
| 1 | m |
| 2 | n |
| 4 | o |

**table_B**

| A | N |
|---|---|
| 2 | p |
| 3 | q |
| 5 | r |

**Resulting table**

| A | M | A | N |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 2 | n | 2 | p |
| 1 | m | - | - |
| 4 | o | - | - |
| - | - | 3 | q |
| - | - | 5 | r |

Because this is a full outer join so all rows (both matching and non-matching) from both tables are included in the output. Here only one row of output displays values in all columns because there is only one match between table_A and table_B.

# SQL Cross Join

When each row of first table is combined with each row from the second table, known as Cartesian join or cross join. In general words we can say that SQL CROSS JOIN returns the Cartesian product of the sets of rows from the joined table.

**We can specify a CROSS JOIN in two ways:**

1. Using the JOIN syntax.
2. the table in the FROM clause without using a WHERE clause.

**SYNTAX of SQL Cross Join**

1. **SELECT** * **FROM** [TABLE1] CROSS JOIN [TABLE2]
2. OR
3. **SELECT** * **FROM** [ TABLE1] , [TABLE2]

Let us take an example of two tables,

**Table1 - MatchScore**

| Player | Department_id | Goals |
|---|---|---|
| Franklin | 1 | 2 |
| Alan | 1 | 3 |
| Priyanka | 2 | 2 |
| Rajesh | 3 | 5 |

**Table2 - Departments**

| Department_id | Department_name |
|---|---|
| 1 | IT |
| 2 | HR |

| | |
|---|---|
| 3 | Marketing |

**SQL Statement:**

1. **SELECT** * **FROM** MatchScore CROSS JOIN Departments

**After executing this query , you will find the following result:**

| Player | Department_id | Goals | Depatment_id | Department_name |
|---|---|---|---|---|
| Franklin | 1 | 2 | 1 | IT |
| Alan | 1 | 3 | 1 | IT |
| Priyanka | 2 | 2 | 1 | IT |
| Rajesh | 3 | 5 | 1 | IT |
| Franklin | 1 | 2 | 2 | HR |
| Alan | 1 | 3 | 2 | HR |
| Priyanka | 2 | 2 | 2 | HR |
| Rajesh | 3 | 5 | 2 | HR |
| Franklin | 1 | 2 | 3 | Marketing |
| Alan | 1 | 3 | 3 | Marketing |
| Priyanka | 2 | 2 | 3 | Marketing |
| Rajesh | 3 | 5 | 3 | Marketing |

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow –

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.

- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

# Subqueries with the SELECT Statement

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows –

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
   (SELECT column_name [, column_name ]
   FROM table1 [, table2 ]
   [WHERE])
```

## 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 |
+----+----------+-----+---------+----------+
```

# Subqueries with the INSERT Statement

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows.

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
   SELECT [ *|column1 [, column2 ]
   FROM table1 [, table2 ]
   [ WHERE VALUE OPERATOR ]
```

## Example

Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

```
SQL> INSERT INTO CUSTOMERS_BKP
   SELECT * FROM CUSTOMERS
   WHERE ID IN (SELECT ID
   FROM CUSTOMERS) ;
```

# Subqueries with the UPDATE Statement

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows.

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

## Example

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
SQL> UPDATE CUSTOMERS

   SET SALARY = SALARY * 0.25

   WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP

      WHERE AGE >= 27 );
```

This would impact two rows and finally CUSTOMERS table would have the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |   125.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  2125.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

# Subqueries with the DELETE Statement

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows.

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

## Example

Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the CUSTOMERS table for all the customers whose AGE is greater than or equal to 27.

```
SQL> DELETE FROM CUSTOMERS

   WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP

      WHERE AGE >= 27 );
```

This would impact two rows and finally the CUSTOMERS table would have the following records.

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   |  1500.00 |
|  3 | kaushik  |  23 | Kota    |  2000.00 |
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  6 | Komal    |  22 | MP      |  4500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

# Subqueries versus Joins

Joins and subqueries are both used to combine data from different tables into a single result. They share many similarities and differences.

Subqueries can be used to return either a scalar (single) value or a row set; whereas, joins are used to return rows.

A common use for a subquery may be to calculate a summary value for use in a query.  For instance we can use a subquery to help us obtain all products have a greater than average product price.

```
SELECT ProductID,

       Name,

       ListPrice,

       (SELECT AVG(ListPrice)
          FROM Production.Product) AS AvgListPrice
  FROM Production.Product
 WHERE ListPrice > (SELECT AVG(ListPrice)
  FROM Production.Product)
```

There are two subqueries in this SELECT statement.  The first's purpose is to display the average list price of all products, the second's purpose is for filtering out products less than or equal to the average list price.

Here the subquery is returning a single value which is then used filter out products.

# Chapter 8 : DML and DDL Commands

## DDL

**Data Definition Language** (DDL) statements are used to define the database structure or schema. Some examples:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

## DML

**Data Manipulation Language** (DML) statements are used for managing data within schema objects. Some examples:

- SELECT - retrieve data from the a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain
- MERGE - UPSERT operation (insert or update)
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - explain access path to data
- LOCK TABLE - control concurrency

## DCL

**Data Control Language** (DCL) statements. Some examples:

- GRANT - gives user's access privileges to database
- REVOKE - withdraw access privileges given with the GRANT command

## TCL

**Transaction Control** (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

- COMMIT - save work done
- SAVEPOINT - identify a point in a transaction to which you can later roll back
- ROLLBACK - restore database to original since the last COMMIT
- SET TRANSACTION - Change transaction options like isolation level and what rollback segment to use

**Table:**

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL **CREATE TABLE** statement is used to create a new table.

# Syntax

The basic syntax of the CREATE TABLE statement is as follows –

```
CREATE TABLE table_name(
   column1 datatype,
   column2 datatype,
   column3 datatype,
   .....
   columnN datatype,
   PRIMARY KEY( one or more columns )
);
```

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with the following example.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check the complete details at Create Table Using another Table.

# Example

The following code block is an example, which creates a CUSTOMERS table with an ID as a primary key and NOT NULL are the constraints showing that these fields cannot be NULL while creating records in this table –

```
SQL> CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

You can verify if your table has been created successfully by looking at the message displayed by the SQL server, otherwise you can use the **DESC** command as follows –

```
SQL> DESC CUSTOMERS;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
```

```
| ID      | int(11)       | NO   | PRI |         |       |
| NAME    | varchar(20)   | NO   |     |         |       |
| AGE     | int(11)       | NO   |     |         |       |
| ADDRESS | char(25)      | YES  |     | NULL    |       |
| SALARY  | decimal(18,2) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Now, you have CUSTOMERS table available in your database which you can use to store the required information related to customers.

# SQL - Using Views

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following –

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

## Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows –

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

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

## 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 to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

This would produce the following result.

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
```

# Chapter 9: Transaction Processing

The Concept of Transaction, States of Transaction, Concurrent Execution of Multiple transactions, Serializability

## DBMS - Transaction

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

### A’s Account

```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

### B’s Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

## ACID Properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability – commonly known as ACID properties – in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** – This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

- **Consistency** – The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

- **Durability** – The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

- **Isolation** – In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

# Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

- **Schedule** – A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.

- **Serial Schedule** – It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

# Equivalence Schedules

An equivalence schedule can be of the following types –

## Result Equivalence

If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

## View Equivalence

Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.

For example –

- If T reads the initial data in S1, then it also reads the initial data in S2.
- If T reads the value written by J in S1, then it also reads the value written by J in S2.
- If T performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

## Conflict Equivalence

Two schedules would be conflicting if they have the following properties –

- Both belong to separate transactions.
- Both accesses the same data item.
- At least one of them is "write" operation.

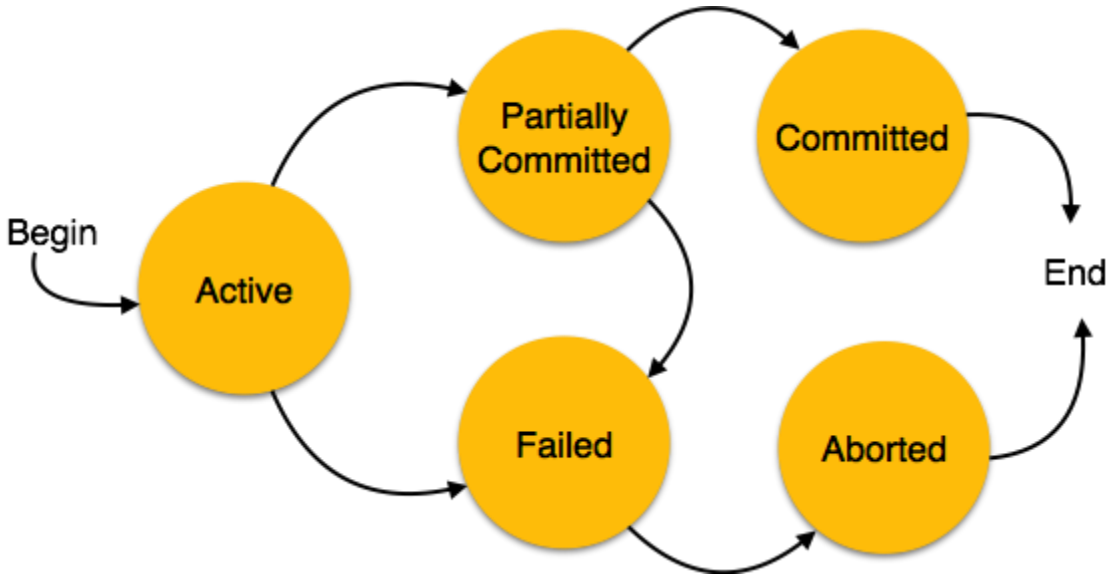
Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if –

- Both the schedules contain the same set of Transactions.
- The order of conflicting pairs of operation is maintained in both the schedules.

**Note** – View equivalent schedules are view serializable and conflict equivalent schedules are conflict serializable. All conflict serializable schedules are view serializable too.

# States of Transactions

A transaction in a database can be in one of the following states –




- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** – When a transaction executes its final operation, it is said to be in a partially committed state.

- **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

- **Aborted** – If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts –

    - Re-start the transaction
    - Kill the transaction

- **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

# Chapter 10: Concurrency Control and Deadlock Recovery

Introduction, Lock – Based Protocols, Protocols, Timestamp – Based Protocols, Thomas' Write Rule, Validation – Based Protocols, Deadlock Handling

## DBMS - Concurrency Control

n a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories –

- Lock based protocols
- Time stamp based protocols

## Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds –

- **Binary Locks** – A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive** – This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

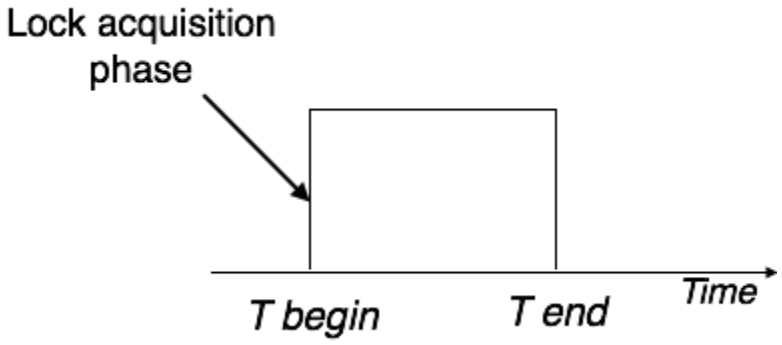
There are four types of lock protocols available –

## Simplistic Lock Protocol

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the 'write' operation.

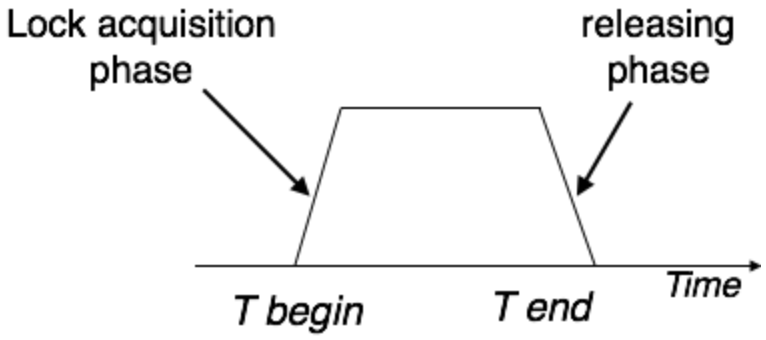
## Pre-claiming Lock Protocol

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.




## Two-Phase Locking 2PL

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

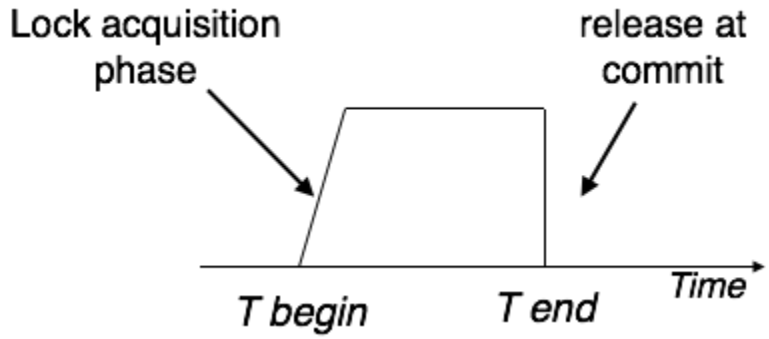



Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

## Strict Two-Phase Locking

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.




Strict-2PL does not have cascading abort as 2PL does.

# Timestamp-based Protocols

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

# Timestamp Ordering Protocol

The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

- The timestamp of transaction $T_i$ is denoted as $TS(T_i)$.
- Read time-stamp of data-item X is denoted by R-timestamp(X).
- Write time-stamp of data-item X is denoted by W-timestamp(X).

Timestamp ordering protocol works as follows −

- **If a transaction Ti issues a read(X) operation −**
    - If TS(Ti) < W-timestamp(X)
        - Operation rejected.

- If TS(Ti) >= W-timestamp(X)
  - Operation executed.
- All data-item timestamps updated.

- **If a transaction Ti issues a write(X) operation –**
  - If TS(Ti) < R-timestamp(X)
    - Operation rejected.
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected and Ti rolled back.
  - Otherwise, operation executed.

### Thomas' Write Rule

This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and $T_i$ is rolled back.

Time-stamp ordering rules can be modified to make the schedule view serializable.

Instead of making $T_i$ rolled back, the 'write' operation itself is ignored.

# DBMS - Deadlock

In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

For example, assume a set of transactions {$T_0$, $T_1$, $T_2$, ...,$T_n$}. $T_0$ needs a resource X to complete its task. Resource X is held by $T_1$, and $T_1$ is waiting for a resource Y, which is held by $T_2$. $T_2$ is waiting for resource Z, which is held by $T_0$. Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock.

Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

## Deadlock Prevention

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

## Wait-Die Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur −

- If $TS(T_i) < TS(T_j)$ − that is $T_i$, which is requesting a conflicting lock, is older than $T_j$ − then $T_i$ is allowed to wait until the data-item is available.

- If $TS(T_i) > TS(t_j)$ − that is $T_i$ is younger than $T_j$ − then $T_i$ dies. $T_i$ is restarted later with a random delay but with the same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

## Wound-Wait Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur −

- If $TS(T_i) < TS(T_j)$, then $T_i$ forces $T_j$ to be rolled back − that is $T_i$ wounds $T_j$. $T_j$ is restarted later with a random delay but with the same timestamp.

- If $TS(T_i) > TS(T_j)$, then $T_i$ is forced to wait until the resource is available.

This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.

In both the cases, the transaction that enters the system at a later stage is aborted.

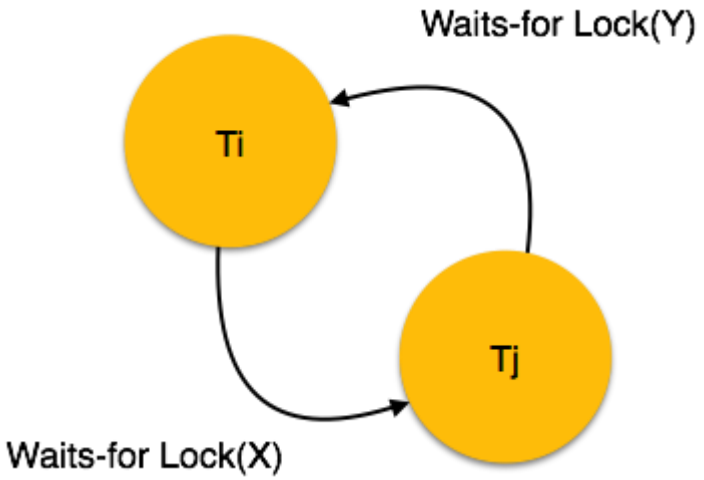
# Deadlock Avoidance

Aborting a transaction is not always a practical approach. Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance. Methods like "wait-for graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource. In a bulky system, deadlock prevention techniques may work well.

## Wait-for Graph

This is a simple method available to track if any deadlock situation may arise. For each transaction entering into the system, a node is created. When a transaction $T_i$ requests for a lock on an item, say X, which is held by some other transaction $T_j$, a directed edge is created from $T_i$ to $T_j$. If $T_j$ releases item X, the edge between them is dropped and $T_i$ locks the data item.

The system maintains this wait-for graph for every transaction waiting for some data items held by others. The system keeps checking if there's any cycle in the graph.




Here, we can use any of the two following approaches –

- First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.

- The second option is to roll back one of the transactions. It is not always feasible to roll back the younger transaction, as it may be important than the older one. With the help of some relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as the **victim** and the process is known as **victim selection**.