GOVT. POLYTECHNIC, KANDHAMAL


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

# DATABASE MANAGEMENT SYSTEM

**BRANCH – CSE**

**NAME OF FACULTY – SMT. KUMARI JYOTI**

**Semester – 4th**

# BASIC CONCEPTS OF DBMS

- **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** consists of interrelated data and a set of programs to access those data in such a way that it becomes easier to retrieve, manipulate, and produce required information efficiently on demand.
- **Purpose of Database Systems**:
- 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 be represented in tables which can be handled easily and efficiently.
- **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 system works and organizes. DBMS also stores metadata,called data dictionary, 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 all the attempts of leaving database in inconsistent state and restricts such updates to be made. A DBMS can provide greater consistency as compared to earlier forms of data storing applications like file-processing systems by providing certain integrity checks before an attempt to update the data in the databse is made.
- **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.
- **Multiuser and Concurrent Access**: DBMS supports multi-user environment and allows more than one user to access and manipulate data in parallel or simultaneously. 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 giving different levels of abstraction. A person working in Sales department will have a different view of the 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.

- **ACID Properties**: DBMS follows the concepts of **A**tomicity, **C**onsistency, **I**solation, and **D**urability (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.

- **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 by imposing security constraints which enables multiple users to have different views with different features with different operation restritions as imposed by the DBA. 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. These should not be accessed by unauthorized persons. DBMS can ensure that proper access procedures are followed including proper authentication schemes for access to DBMS and additional checks before permitting access to sensitive data.

**Shared data:**

- A database allows the sharing of data under its control by any number of application programs or users.

**Data independence:**

- DBMS supports both physical and logical data independence. Data independence is advantageous in the database

environment since it allows for changes at one level of the database without affection other levels.

- **DataBase Abstraction:** For the Database Security aspects, the Database System does not disclose all the details of the data to all the types of users rather it hides certain details of how the data is actually stored and maintained and only discloses that much of data required by the authentic users efficiently.This concept is called as DataBase Abstraction.

- **DataBase Users:**

- A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows:


**A Administrators**: Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance. The functions of DBA include

i. Schema definition

ii. Storage structure and access method definition

iii. Schema and physical organization modification

iv. Granting of authorization for data access

v. Routine maintenance

**B. Designers:** These are made up of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views. They are of two types

1. Application Programmers:-

- ❖ Application programmers are computer professionals who write application programs.
- ❖ Application programmers can choose from many tools like RAD tools, programming languages, fourth generation languages etc. to develop user interfaces.

2. Sophisticated Users:-

- ➢ Sophisticated users interact with the system without writing programs. Instead they form their requests in database query language. They submit each such query to query processor whose function is to break down DML statements into instructions that the storage manager understands.
- ➢ Analysts who submit queries to explore data in the database fall in this category.

➢ Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data processing framework.

- Among these applications are computer aided design systems, knowledge base& expert systems that store data with complex structure(Example:- Graphics and audio data) and environment modeling systems

**C. End Users or Naïve Users**: End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

## Data Definition Language(DDL):

- A database system provides a data definition language to specify the database schemas.
- For example the following SQL statement defines the department table.

- Sql> create table department (deptno number(3), dname varchar2(10));
- Execution of the above DDL statement creates the department table. In addition it updates a special set of tables called data dictionary.
  - DDL is used to define the database. This definition includes all the entity sets and their associated attributes as well as the relationship among the entity sets. It also includes any constraints that have to be maintained.
  - The DDL used at the external schema is called the view definition language (VDL) from where the defining process starts.
  - A data dictionary contains metadata. A database system consults data dictionary before reading or

modifying actual data. The schema of a table is an example of a metadata i.e. data about data.

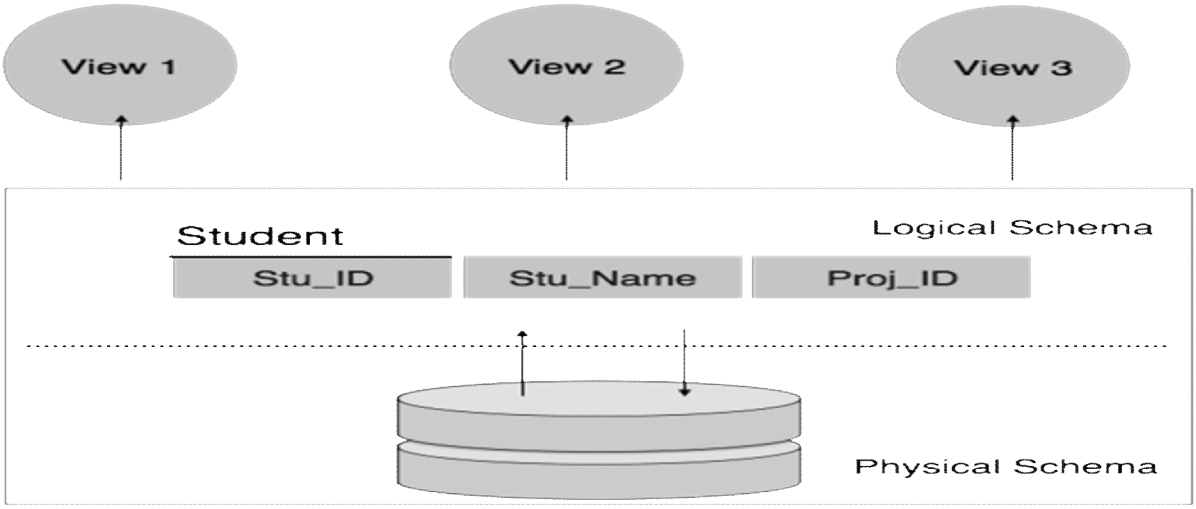
- We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called data storage and definition language (DSDL). These statements define the implementation details of the database schemas which are usually hidden from the users.
- The DDL provides facilities to specify such consistency constraints. The database system checks these constraints every time the database is updated.

**DATA DICTIONARY**:-

- Information regarding the structure and usage of data contained in the database, the metadata maintained in a data dictionary. The term system catalog also describes this metadata. The data dictionary which is a database itself documents the data.
- Each database users can consult the data dictionary to learn what each piece of data and various synonyms of data fields mean.
- In an integrated system (i.e. in system where the data dictionary is a part of the DBMS) the data dictionary stores information concerning the external, conceptual and internal levels of the database. It contains the source of each data field value, the frequency of its use and an audit trail concerning updates, including the who and when of each update.

# DATA MODELS

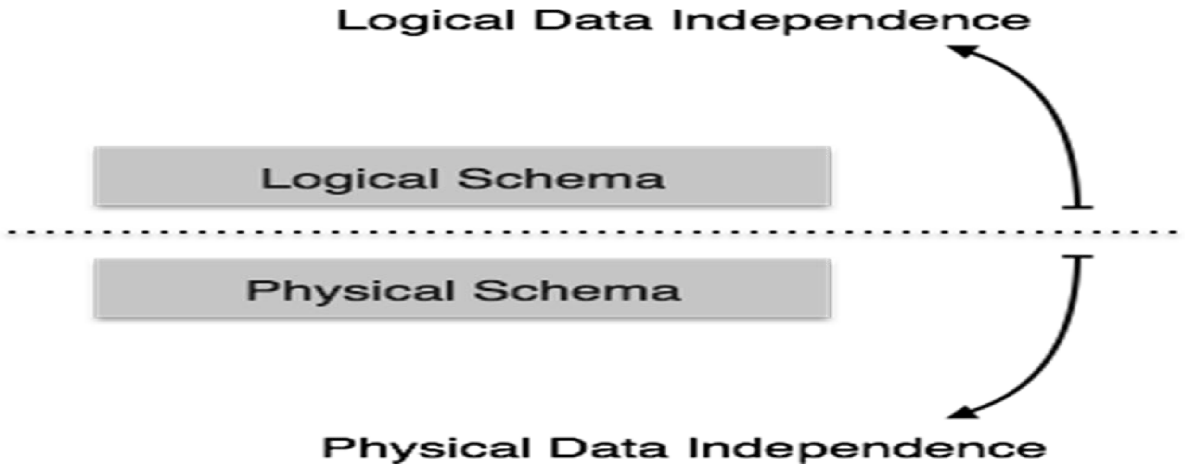
- **Database Schema**
- A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.
- A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.




- **Physical Database Schema**: This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
- **Logical Database Schema**: This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.
- **Data Independence**

- A database system normally contains a lot of data in addition to users' data. For example, it stores data about data, known as metadata, to locate and retrieve data easily. It is rather difficult to modify or update a set of metadata once it is stored in the database. But as a DBMS expands, it needs to change over time to satisfy the requirements of the users. If the entire data is dependent, it would become a tedious and highly complex job. Three levels of abstraction along with the mappings from internal to conceptual and from conceptual to external level provide two distinct levels of data independence: logical data independence and physical data independence.
- Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemas. The change would be absorbed by the mapping between the external and conceptual level.
- Logical data independence also insulates application programs from operations such as combining two records into one or splitting an existing record into two or more records.
- Logical data independence is achieved by providing the external level or user view of the database. The application programs or users see the database as described by their respective external views.
- Physical data independence is achieved by the presence of the internal level of the database and the mapping or transformation from conceptual level of database to internal level.

- The physical data independence criterion requires that the conceptual level does not specify storage structures or the access methods (indexing, hashing etc) used to retrieve the data from the physical storage medium.
- Another aspect of data independence allows different interpretation of the same data. The storage of data is in bits and may change from EBCDIC to ASCII coding.



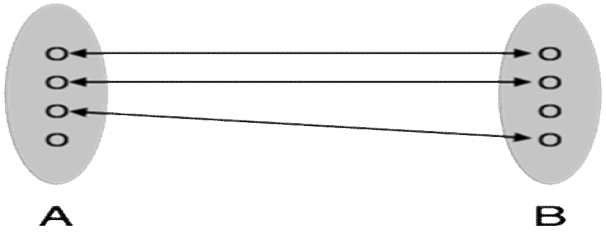

- Metadata itself follows a layered architecture, so that when we change data at one layer, it does not affect the data at another level. This data is independent but mapped to each other.
- **Logical Data Independence**
- Logical data is data about database, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all its constraints applied on that relation.
- Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format, it should not change the data residing on the disk.

- **Physical Data Independence**
- All the schemas are logical, and the actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data.
- For example, in case we want to change or upgrade the storage system itself — suppose we want to replace hard-disks with SSD — it should not have any impact on the logical data or schemas.
- **ER MODEL BASIC CONCEPTS:**
- The ER model defines the conceptual view of a database. It works around real- world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.
- **Entity**
- An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.
- An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.
- **Attributes**

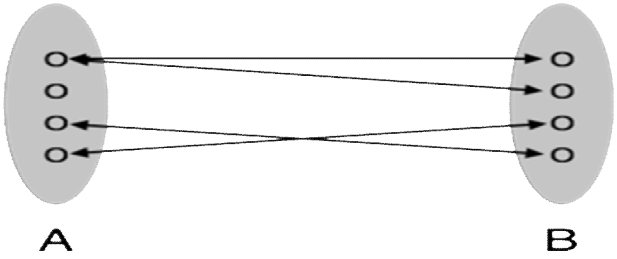
- Entities are represented by means of their properties called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.
- There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.
- **Types of Attributes**
- **Simple attribute:** Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute:** Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- **Derived attribute:** Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- **Single-value attribute:** Single-value attributes contain single value. For example: Social_Security_Number.
- **Multi-value attribute:** Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.
- These attribute types can come together in a way like:
- simple single-valued attributes

- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes
- **Entity-Set and Keys**
- Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.
- For example, the roll_number of a student makes him/her identifiable among students.
- **Super Key**: A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key**: A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key**: A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.
- **Relationship**
- The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.
- **Relationship Set**
- A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.
- **Degree of Relationship**

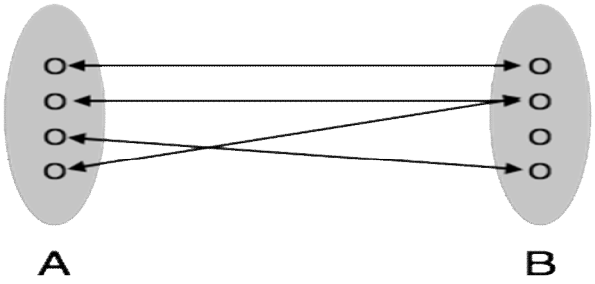
- The number of participating entities in a relationship defines the degree of the relationship.
- Binary = degree 2
- Ternary = degree 3
- n-ary = degree n
- **Mapping Cardinalities:**
- **Cardinality** defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.
- **One-to-one**: One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



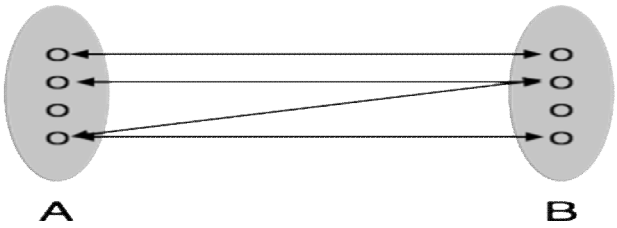

- **One-to-many**: One entity from entity set A can be associated with more than one entities of entity set B, however an entity from entity set B can be associated with at most one entity.




- **Many-to-one**: More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.




- **Many-to-many**: One entity from A can be associated with more than one entity from B and vice versa.



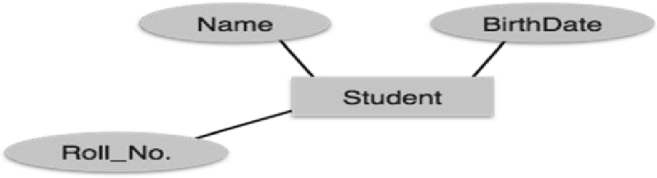

- **ER DIAGRAM :** Let us now learn how the ER Model is represented by means of an ER diagram. Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

- **Entity**

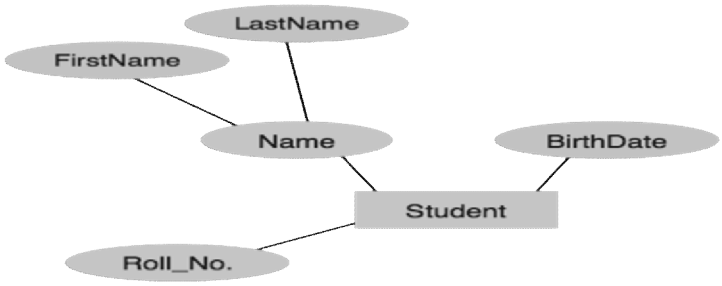
- Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.


- **Attributes**

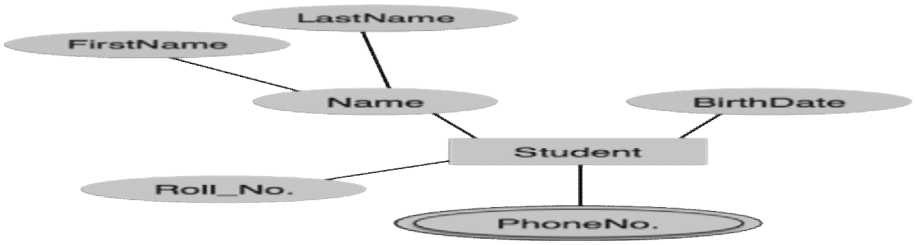
- Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).



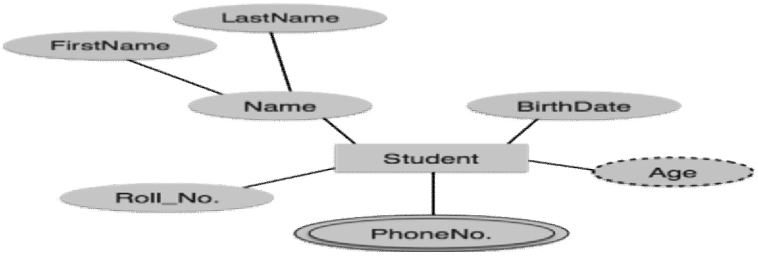
If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.




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



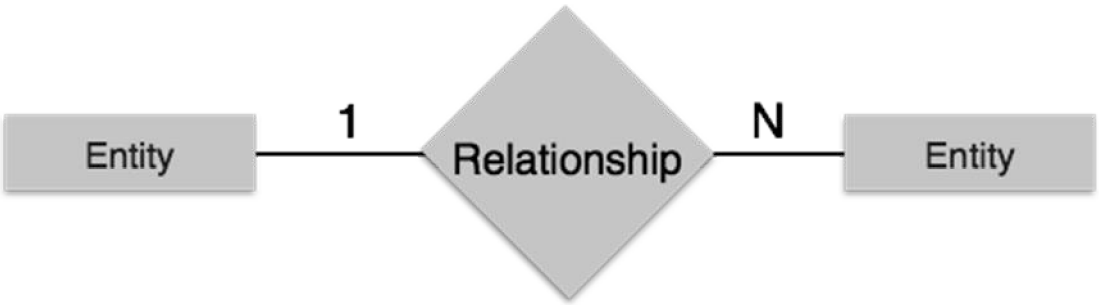

**Derived** attributes are depicted by dashed ellipse




- **<u>Relationship</u>**
- Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship are connected to it by a line.
- **Binary Relationship and Cardinality**

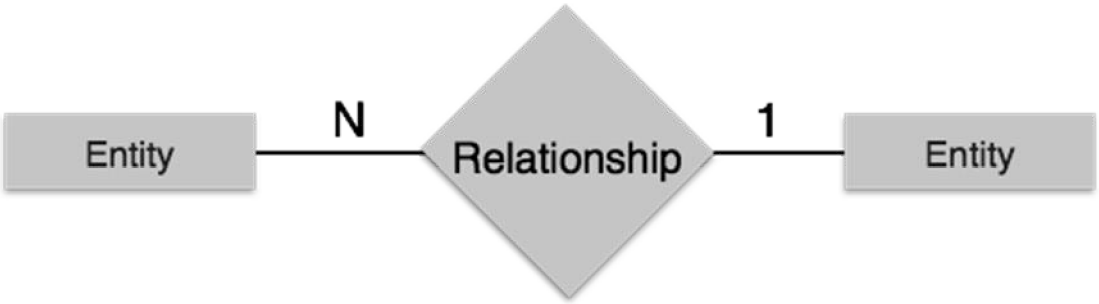
- A relationship where two entities are participating is called a **binary relationship**. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.
- **One-to-one:** When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.


- **One-to-many:** When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one- to-many relationship.

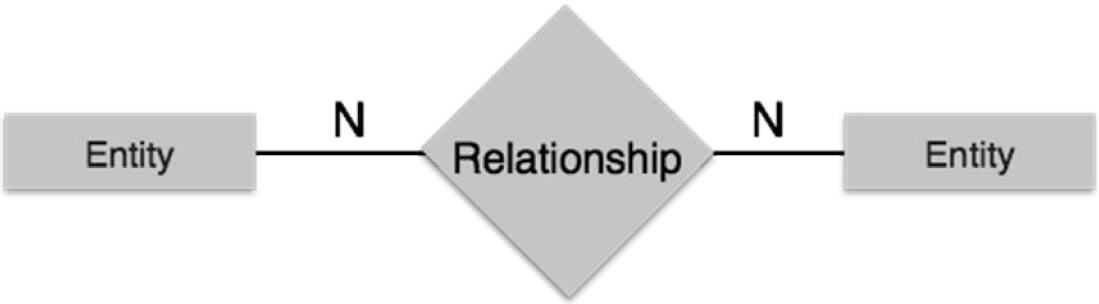



**Many-to-one:** When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an

entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.




**Many-to-many:** The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.




- **<u>NETWORK DATA MODEL</u>**:
  - The network data model was formalized in the late 1960's by the Database Task Group of Conference on data system languages (DBTG/CODASYL). Hence it is also known as DBTG model.
- The network model uses two different data structures to represent the database entities and relationship between the entities named *record type* and *set type*. A record type is used to represent an entity type. It is made up of a number of data items that represents the attributes of an entity.

- A set type is used to represent a directed relationship between two record types, the so called owner record type and the member record type.
- The set type like the record type is named and specifies that there is a one to many relationship (1:M) between the owner and member record types. The set type can have more than one record type as its member, but only one record type is allowed to be the owner in a given set type.
  - A database could have one or more occurrences of each of its record types and set types. An occurrence of a set type consists of an occurrence of the owner record type and any number of occurrences of each of its member record type. A record type can't be a member of two distinct occurrences of the same type.
  - To avoid the confusion inherent in the use of the word 'set' to describe the mechanism for showing relationship in the network model, the other terms suggested are co set, fan set, owner coupled set, CODASYL set, DBTG set etc.
  - Bachman introduced a graphical means called a data structure diagram to denote the logical relationship implied by the set. Here a labeled rectangle represents the corresponding entity or record type. An arrow that connects two labeled rectangles represents a set type. The arrow direction is from owner record type to member record type.
- In the network model, the relationships as well as the navigation through the database are predefined at database creation time.

## HIERARCHICAL DATA MODEL:-

- A tree may be defined as a set of nodes such that there is one specially designated node called root node and the remaining nodes are partitioned into disjoint sets, each of which in turn is a tree, the sub trees of the root. If the relative order of the sub trees is significant the tree is an ordered tree.
- In a hierarchical database the data is organized in a hierarchical or ordered tree structure and the database is a collection of such disjoint trees (sometimes referred to as forests or spanning trees). The nodes of the tree represent record types. Each tree effectively represents a root record type and all its dependent record types. If we define the root record type at level 0, then the level of its dependent record types can be defined at level 1. The dependents of the record types at level 1 are said to be at level 2 and so on.
- An occurrence of a hierarchical tree type consists of one occurrence of the root record type along with zero or more occurrences of its dependent sub tree types. Each dependent sub tree is in turn, hierarchical and consists of a record type as its root node.
- In a hierarchical model no dependent record can occur without its parent record occurrence. Furthermore no dependent record occurrence may be connected to more than one parent record occurrence.
- A hierarchical model can represent a one to many relationships between two entities where the two are respectively parent and child. However to represent many to many relationship requires duplication of one of the record types corresponding to one of the entities involved in this relationship. Note that such

duplications lead to inconsistencies when only one copy of a duplicate record is updated.

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

- **Characteristics:**

- **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 Constraints**
- There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called **candidate keys**.
- Key constraints force that:
- in a relation with a key attribute, no two tuples can have identical values for key attributes.
- a key attribute cannot have NULL values.
- Key constraints are also referred to as **Entity Integrity Constraints**.
- **Domain Constraints**
- Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.
- **Referential Integrity Constraints**
- Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

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

# RELATIONAL DATABASE

## CODD'S RULES:

- Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.
- These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

**Rule 1: Information Rule**

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

**Rule 2: Guaranteed Access Rule**

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

**Rule 3: Systematic Treatment of NULL Values**

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following: data is missing, data is not known, or data is not applicable.

**Rule 4: Active Online Catalog**

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query

language to access the catalog which they use to access the database itself.

**Rule 5: Comprehensive Data Sub-Language Rule**

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

**Rule 6: View Updating Rule**

All the views of a database, which can theoretically be updated, must also be updatable by the system.

**Rule 7: High-Level Insert, Update, and Delete Rule**

The database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

**Rule 8: Physical Data Independence**

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

**Rule 9: Logical Data Independence**

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no

impact or change on the user application. This is one of the most difficult rule to apply.

**Rule 10: Integrity Independence**

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

**Rule 11: Distribution Independence**

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

**Rule 12: Non-Subversion Rule**

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

- **RELATIONAL ALGEBRA:** Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances. There are two kinds of query languages: relational algebra and relational calculus.
- **Relational Algebra**
- Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is

performed recursively on a relation and intermediate results are also considered relations.

- The fundamental operations of relational algebra are as follows:
- Select or restrict
- Project
- Natural join
- Divide
- Union
- Set difference
- Cartesian product
- Rename
- Set Intersection
- **Select Operation (σ)**
- It selects tuples that satisfy the given predicate from a relation.
- **Notation**: $\sigma_{p}(r)$
- Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and**, **or**, and **not**. These terms may use relational operators like: =, ≠, ≥, <, >, ≤.
- **For example**:$\sigma$subject="database"(Books)
- **Output**: Selects tuples from books where subject is 'database'

  $\sigma$subject="database" and price="450"(Books)

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

$\sigma$subject="database" and price < "450" or year > "2010"(Books)

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

- **Project Operation (∏)**
- It projects column(s) that satisfy a given predicate.
- **Notation**: $\prod A1, A2, An\ (r)$
- Where $A_1, A_2, A_n$ are attribute names of relation **r**.
- Duplicate rows are automatically eliminated, as relation is a set. For example:

  $\prod$subject, author (Books)

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

- **Natural join:** Returns a relation consisting of all possible tuples that are a combination of two tuples ,one from each of the two specified relations such that the two tuples contributing to any given relation have a common value for the common attributes of the two relations.
- **Divide:** It takes two relations , one binary and the other unary and returns a relation consisting of all values of one attribute of the binary relation that match all values in the unary relation.

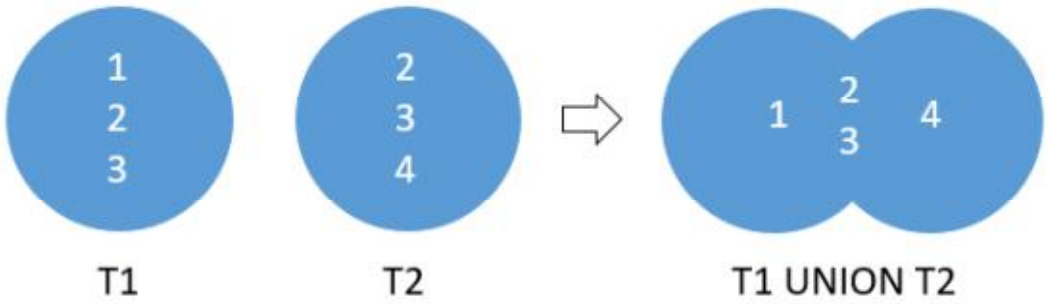
- **Union Operation (∪)**
- It performs binary union between two given relations and is defined as:

  $r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

- Where **r** and **s** are either database relations or relation result set (temporary relation).
- For a union operation to be valid, the following conditions must hold:
- **r** and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

  $\prod_{\text{author}}^{(\text{Books})} \cup \prod_{\text{author}}^{(\text{Articles})}$

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




The following illustrates the syntax of the Oracle UNION operator:

SQL>SELECT column_list FROM T1 UNION SELECT column_list FROM T2;

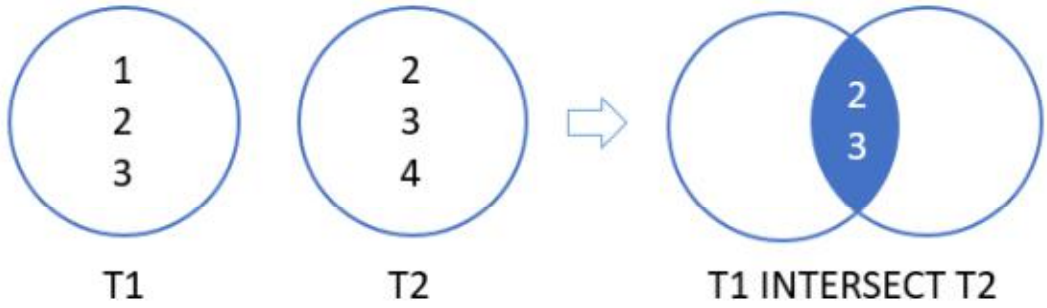
For Example: The following statement uses the UNION operator to build a list of contacts from the employees and contacts tables:

SQL> SELECT first_name,last_name,email FROM contacts UNION SELECT first_name, last_name, email FROM employees;

- **Set intersection**: It performs binary intersection between two given relations and is defined as:

  r n s = { t | t ∈ r and t ∈ s}

- Where **r** and **s** are either database relations or relation result set (temporary relation).
- For a intersection operation to be valid, the following conditions must hold:
- **r** and **s** must have the same number of attributes.
- Attribute domains must be compatible.




The following illustrates the syntax of the Oracle INTERSECT operator:

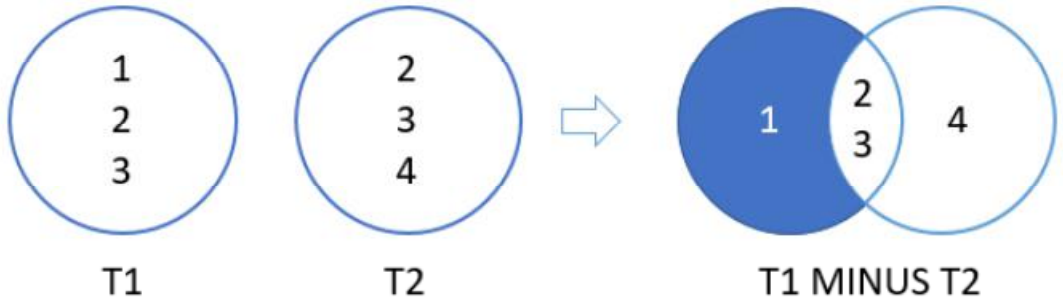
SQL>SELECT column_list FROM T1 INTERSECT SELECT column_list FROM T2;

The following statement uses the INTERSECT operator to get the last names used by people in both contacts and employees tables:

SQL> SELECT last_name FROM contacts INTERSECT SELECT last_name FROM employees ORDER BY last_name;

- **Set Difference (−)**
- The result of set difference operation is tuples, which are present in one relation but are not in the second relation. The two relations must be type compatible i,e they must have identical attributes defined on the same domains.
- **Notation**: **r − s**
- Finds all the tuples that are present in **r** but not in **s**.

  $\Pi_{author}(Books) - \Pi_{author}(Articles)$
- **Output**: Provides the name of authors who have written books but not articles.




The following illustrates the syntax of the Oracle MINUS operator:

SQL>SELECT column_list_1 FROM T1 MINUS
SELECT column_list_2 FROM T2;

For Example: The following statement returns distinct last names from the query to the left of the MINUS operator which are not also found in the right query.

SQL> SELECT last_name FROM contacts MINUS
SELECT last_name FROM employees

ORDER BY last_name;

- **Cartesian Product (X)**
- Combines information of two different relations into one.
- **Notation**: r X s
- Where **r** and **s** are relations and their output will be defined as:

  r X s = { q t | q ∈ r and t ∈ s}

  $\Pi_{author = \text{'ucpes'}}(\text{Books X Articles})$

- **Output**: Yields a relation, which shows all the books and articles written by ucpes.
- **Rename Operation (ρ)**
- The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** $\rho$.
- **Notation**: $\rho_{X}(E)$
- Where the result of expression **E** is saved with name of **x**.

# NORMALISATION IN RELATIONAL SYSTEMS

- **Functional Dependency**
- Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes $A_1, A_2, ..., A_n$, then those two tuples must have to have same values for attributes $B_1, B_2, ..., B_n$.
- Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.
- **Armstrong's Axioms**
- If F is a set of functional dependencies then the closure of F, denoted as $F^+$, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules that, when applied repeatedly, generates a closure of functional dependencies.
- **Reflexive rule**: If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
- **Augmentation rule**: If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule**: Same as transitive rule in algebra, if a → b holds and b
- → c holds, then a → c also holds. a → b is called as a functionally that determines b.
- **Normalization**

- If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.
- **Update anomalies**: If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.
- **Deletion anomalies**: We tried to delete a record, but parts of it was left undeleted because of unawareness, the data is also saved somewhere else.
- **Insert anomalies**: We tried to insert data in a record that does not exist at all.
- Normalization is a method to remove all these anomalies and bring the database to a consistent state.
- **First Normal Form**
- First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.
-

| Course | Content |
|---|---|
| Programming | Java, c++ |
| Web | HTML, PHP, ASP |

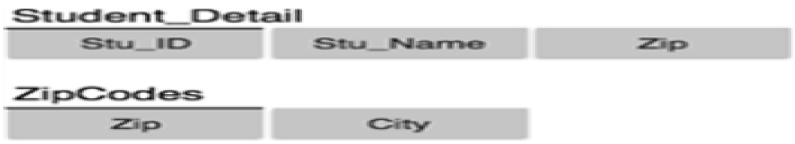
We re-arrange the relation (table) as below, to convert it to First Normal Form.

| Course | Content |
|---|---|
| Programming | Java |
| Programming | c++ |
| Web | HTML |
| Web | PHP |
| Web | ASP |

- Each attribute must contain only a single value from its predefined domain.
- **Second Normal Form**
- Before we learn about the second normal form, we need to understand the following:
- **Prime attribute**: An attribute, which is a part of the prime-key, is known as a prime attribute.
- **Non-prime attribute**: An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.
- A Relation is said to be in second normal form if and only if it is in first normal form and every non-prime attribute should be fully functionally (irreducibly) dependent on prime key attribute. That is, if $X \rightarrow A$ holds, then there should not be any proper subset Y of X for which $Y \rightarrow A$ also holds true.


- We see here in Student Project relation that the prime key attributes are the combination of Stu ID and Proj ID. According to the rule, non key attributes, i.e., Stu Name and Proj Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu Name can be identified by Stu ID and Proj Name can be identified by Proj ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.


- We decomposed the relation in to two relations as depicted in the above picture. So there exists no partial dependency.
- **Third Normal Form**
- For a relation to be in Third Normal Form, it must be in Second Normal form and No non-prime attribute is transitively dependent on the prime key attribute.
- We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally,
- Stu_ID → Zip → City, so there exists **transitive dependency**.
- To bring this relation into third normal form, we decompose the relation into two relations as follows:




- **Boyce-Codd Normal Form**
- Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that -
- For any non-trivial functional dependency, X → A, X must be a candidate key. i,e a relation is in BCNF if and only if the only determinants are the candidate keys.
- In the above image, Stu_ID is the candidate key in the relation Student_Detail and Zip is the candidate key in the relation ZipCodes. So,
- Stu_ID → Stu_Name and Stu_ID → Zip in the relation Student_Detail
- and
- Zip → City in the relation ZipCodes.
- Which confirms that both the relations are in BCNF because in both the cases the only determinants are the candidate keys.

# STRUCTURED QUERY LANGUAGE

## QUERY LANGUAGE:-

- A query language is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language.
- Query languages can be categorized as either procedural or non-procedural. In a procedural language the user instructs the system to perform a sequence of operations on the database to compute the desired result. In a non-procedural language the user describes the desired information without giving a specific procedure for obtaining that information.

## QUERIES IN SQL:-

- Table:-
  - A table is a database object that holds user data. The simplest analogy is to think of a table as a spread sheet
  - The columns of a table are associated with a specific data type.
  - Oracle ensures that only data which is identical to the data type of the column will be stored within the column.
- Data type:

1. Character (size):

- This data type is used to store character string of fixed length. The size in bracket determines the number of characters the shell can hold. Maximum size is 255 characters.
- Varchar(size)/ Varchar2(size):
- This data type is used to store variable length alpha numeric data. Maximum size is 2000 characters.
- Date: Date data type is used to represent date and time. The standard format is DD-MMM-YY.
- Number:
- The number data type is used to store numbers (fixed or floating) number of any magnitude may be stored up to 38 digit of decision. Maximum size is $9.99 * 10^{124}$ . The precision P determines the maximum length of data where as the scale S determines the number of places to right of the decimal.
- Long:
- This data type is used to store variable length character, strings up to 2GB. Long data can be used to store arrays of binary data in ASCII format.
- RAW/ LONG RAW:
- The raw data type is used to store binary data such as picture or image. Raw data type can have maximum 255 byte and maximum size of long raw is up to 2GB.
- 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 ,data manipulation ,transaction control and data control languages. Using the data definition language of SQL, one can design and modify database schema, data manipulation language allows SQL to store and retrieve data from database, transaction control language is used to control the transactions to update the database temporarily or permanently by using rollback and commit and data control languages like grant and revoke to give permission and withdraw permissions to access the database.

- **Data Definition Language(DDL)**

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

- **CREATE**

- Creates new databases, tables, and views in RDBMS.

- **For example:**

  To create a new table in Oracle Database, we use the CREATE TABLE statement. The following illustrates the basic syntax of the CREATE TABLE statement:

```
SQL>CREATE TABLE table_name (
   column_1 data_type column_constraint,
   column_2 data_type column_constraint,
   ...
   table_constraint
 );
```

  In this syntax:

- First, specify the table name in the CREATE TABLE clause.

- Second, list all columns of the table within the parentheses. In case a table has multiple columns, you need to separate them by commas (,). A column definition includes the column name

followed by its data type e.g., NUMBER, VARCHAR2, and a column constraint such as NOT NULL, primary key, check.

- Third, add table constraints if applicable e.g., primary key, foreign key, check.

  Oracle CREATE TABLE statement example

  The following example shows how to create a new table named persons :
  SQL>CREATE TABLE persons( person_id NUMBER, first_name VARCHAR2(50) NOT NULL,last_name VARCHAR2(50) NOT NULL, PRIMARY KEY(person_id));
  In this example, the persons table has three columns: person_id, first_name, and last_name. The data type of the person_id column is NUMBER. The first_name column has data type VARCHAR2 with the maximum length is 50. It means that a first name whose length is greater than 50 can not be inserted into the first_name column. Besides, the NOT NULL column constraint prevents the first_name column to have NULL values. The last_name column has the same characteristics as the first_name column.The PRIMARY KEY clause specifies the person_id column as the primary key column which is used for identifying the unique row in the persons table.

- **ALTER**

  To modify the structure of an existing table, we use the ALTER TABLE statement. The following illustrates the syntax:
  SQL>ALTER TABLE table_name action;
  In this statement:

- First, we must specify the table name which is to be modified.
- Second, the action that we want to perform is to be specified after the table name.

The ALTER TABLE statement allows us to:

- Add one or more columns
- Modify column definition
- Drop one or more columns

Let's see some examples to understand how such actions work.

To add a new column to a table, we use the following syntax:

**SQL>**ALTER TABLE table_name ADD column_name type constraint;

For example, the following statement adds a new column named birthdate to the persons table:

**SQL>**ALTER TABLE persons ADD birthdate DATE NOT NULL;

To view the persons table and see that the birthdate column is appended at the end of the column list:

```
SQL>DESC persons;

Name       Null     Type
---------- -------- ------------
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
BIRTHDATE  NOT NULL DATE
```

To add multiple columns to a table at the same time, we place the new columns inside the parenthesis as follows:

```
SQL>ALTER TABLE table_name
ADD (
   column_name type constraint,
   column_name type constraint,
   ...
);
```

See the following example:

```
SQL>ALTER TABLE persons
ADD (
   phone VARCHAR(20),
   email VARCHAR(100)
);
```

In this example, the statement added two new columns named phone and email to the persons table.

```
DESC persons

Name       Null     Type
---------- -------- -------------
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
BIRTHDATE  NOT NULL DATE
PHONE               VARCHAR2(20)
EMAIL               VARCHAR2(100)
```

To modify the attributes of a column, you use the following syntax:

```
SQL>ALTER TABLE table_name  MODIFY column_name type constraint;
```

For example, the following statement changes the birthdate column to a null-able column:

```
SQL>ALTER TABLE persons MODIFY birthdate DATE NULL;
```

Let's verify the persons table structure again:

```
DESC persons

Name       Null     Type
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
```

```
LAST_NAME  NOT NULL VARCHAR2(50)
BIRTHDATE          DATE
PHONE              VARCHAR2(20)
EMAIL              VARCHAR2(100)
```

As it can be seen, the birthdate became null-able.

To modify multiple columns, you use the following syntax:

```
SQL>ALTER TABLE table_name
  MODIFY ( column_1 type constraint,
         column_2 type constraint,
        ...);
```

For example, the following statement changes the phone and email column to NOT NULLcolumns and extends the length of the email column to 255 characters:

```
SQL>ALTER TABLE persons MODIFY(
   phone VARCHAR2(20) NOT NULL,
email VARCHAR2(255) NOT NULL
);
```

Verify the persons table structure again:

```
DESC persons;

Name       Null     Type
---------- -------- -------------
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
BIRTHDATE          DATE
PHONE      NOT NULL VARCHAR2(20)
EMAIL      NOT NULL VARCHAR2(255)
```

Oracle ALTER TABLE DROP COLUMN example

To remove an existing column from a table, you use the following syntax:

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

This statement deletes the column from the table structure and also the data stored in that column.

The following example removes the birthdate column from the persons table:

```
SQL>ALTER TABLE persons DROP COLUMN birthdate;
```

Viewing the persons table structure again, we will find that the birthdate column has been removed:

```
DESC persons;

Name       Null     Type
---------- -------- -------------
PERSON_ID  NOT NULL NUMBER
FIRST_NAME NOT NULL VARCHAR2(50)
LAST_NAME  NOT NULL VARCHAR2(50)
PHONE      NOT NULL VARCHAR2(20)
EMAIL      NOT NULL VARCHAR2(255)
```

- **DROP** : Drops views, tables, and databases from RDBMS.
- **Syntax**: SQL>Drop table table name;
- **For example:**

  SQL>DROP TABLE persons;

- **<u>Data Manipulation Language(DML)</u>**
- 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 restrict 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. 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.
- **Syntax:**

```
SQL>SELECT
  column_1,
  column_2,
  ...
FROM
```

- table_name WHERE condition;
- **For example:**
- SQL>Select author_name From book_author Where age > 50;

- This command will display the names of the authors from the relation book_author whose age is greater than 50.
- **INSERT INTO/VALUES**
- This command is used for inserting values into the rows of a table (relation).
- Syntax: SQL>INSERT INTO table (column1 [, column2, column3 ... ]) VALUES (value1 [, value2, value3 ... ]);
- **For example:**
- SQL>INSERT INTO ucpes (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:**
- SQL>UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE condition];
- **For example:**
- SQL>UPDATE ucpes SET Author="webmaster" WHERE Author="anonymous";
- **DELETE/FROM/WHERE**
- This command is used for removing one or more rows from a table (relation).
- **Syntax:**
- SQL>DELETE FROM table_name [WHERE condition];

- SQL>DELETE FROM ucpes WHERE Author="unknown";
- **JOINS:**
- It gives us all the possible tuples that are paired together. But it might not be feasible for us in certain cases to take a Cartesian product where we encounter huge relations with thousands of tuples having a considerable large number of attributes.
- **Join** is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.
- We will briefly describe various join types in the following sections.
- **Theta (θ) Join**
- Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol **θ**.
- **Notation:**
- $R1 \bowtie_{\theta} R2$
- R1 and R2 are relations having attributes $(A_1, A_2, .., A_n)$ and $(B_1, B_2, .., B_n)$ such that the attributes don't have anything in common, that is, $R1 \cap R2 = \Phi$.
- Theta join can use all kinds of comparison operators.
- **Equijoin**
- When Theta join uses only **equality** comparison operator, it is said to be equijoin.
- **Natural Join (⋈)**

- Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.
- Natural join acts on those matching attributes where the values of attributes in both the relations are same.
- SQL>SELECT <select_list> FROM TableAA JOIN TableB B ON A.Key = B.Key;

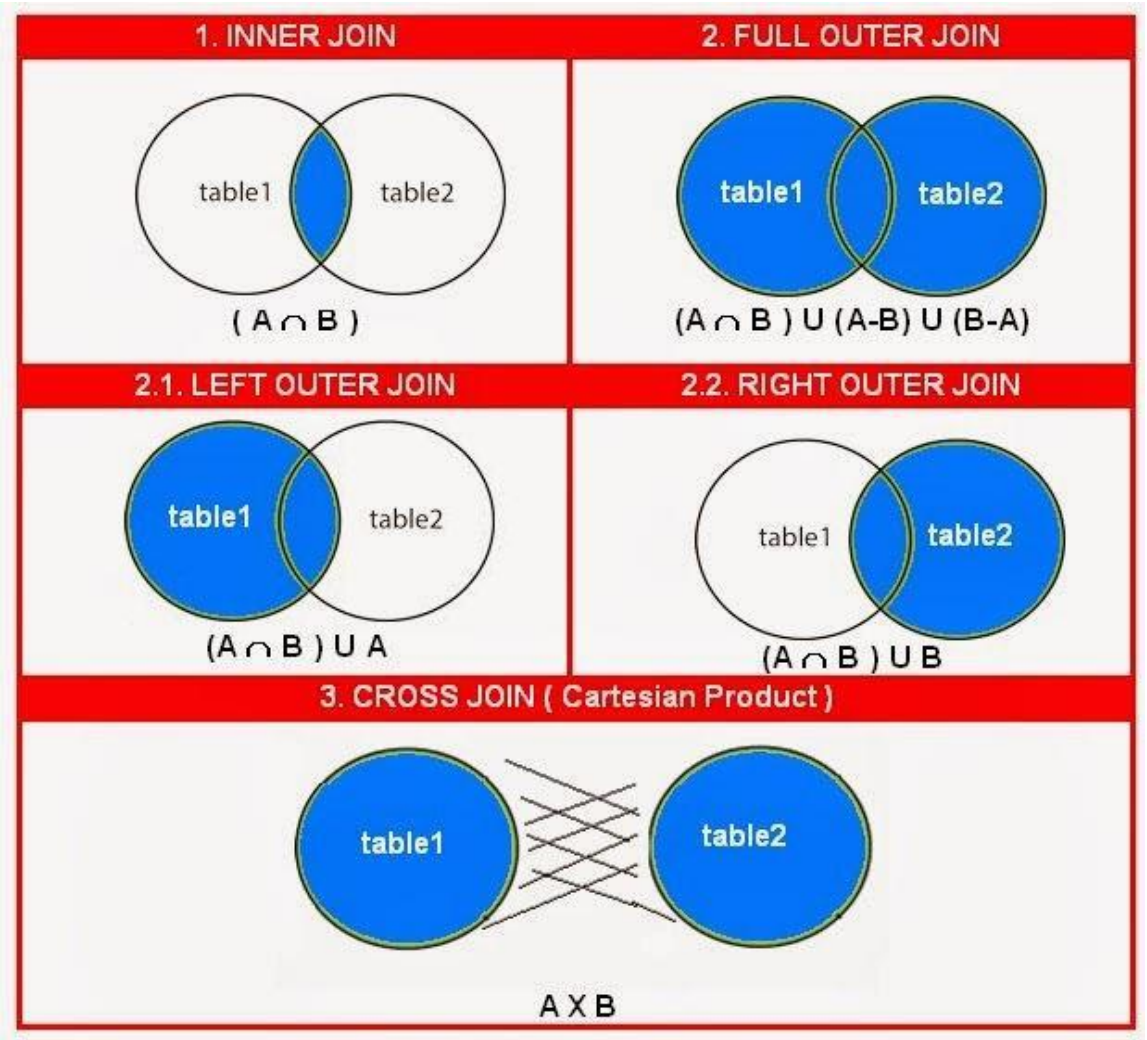
- **Outer Joins**
- Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins: left outer join, right outer join, and full outer join.
- **Left Outer Join :**
- All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.
- SQL>SELECT <selec1_1ist> FROM TableA A LEFT JOIN TableB B ON A.Key = B.Key;

- **Right Outer Join:** All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

- SQL>SELECT <select_list> FROM TableA A RIGHT JOIN TableB B ON A.Key = B.Key;
-

- **Full Outer Join:** All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

  SQL> SELECT <selcctJist> FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key;

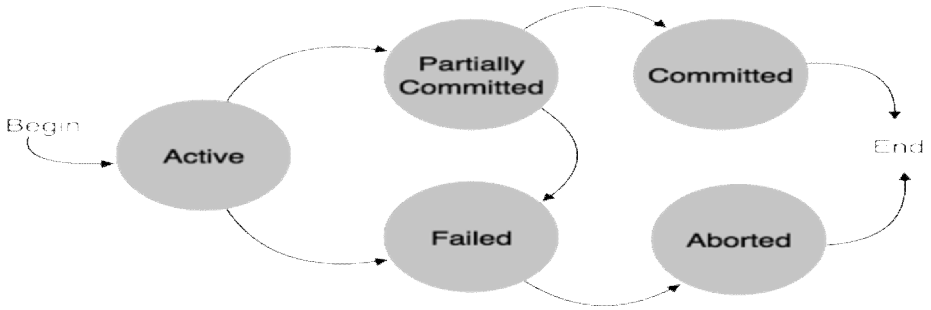


# TRANSACTION PROCESSING CONCEPTS

- TRANSACTION: Transaction can be defined as a a logical unit of work consisting of 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 low- level 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.
- **<u>States of transaction</u>:-**
- A transaction can be considered to be an atomic operation by the user but in reality it goes through a number of states during its life time.




- **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.
- A transaction can end in three possible ways. It can end after a commit operation (a successful termination). It can detect an error during its processing and decide to abort itself by performing a rollback operation (a suicidal termination). The

DBMS or operating system can force it to be aborted for one reason or another (a murderous termination).

- **Serializability**
- When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transaction 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.
- **Data Recovery:**
- **Crash Recovery**
- DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.
- **Failure Classification**
- To see where the problem has occurred, we generalize a failure into various categories, as follows:
- **Transaction Failure**
- A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.
- Reasons for a transaction failure could be:
- **Logical errors**: Where a transaction cannot complete because it has some code error or any internal error condition.
- **System errors**: Where the database system itself terminates an active transaction because the DBMS is not able to execute it,

or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

- **System Crash**
- There are problems – external to the system – that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.
- Examples may include operating system errors.
- **Disk Failure**
- In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.
- Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.
- **Storage Structure**
- We have already described the storage system. In brief, the storage structure can be divided into two categories:
- **Volatile storage**: As the name suggests, a volatile storage cannot survive system crashes. Volatile storage devices are placed very close to the CPU; normally they are embedded onto the chipset itself. For example, main memory and cache memory are examples of volatile storage. They are fast but can store only a small amount of information.
- **Non-volatile storage**: These memories are made to survive system crashes. They are huge in data storage capacity, but slower in accessibility. Examples may include hard-disks,

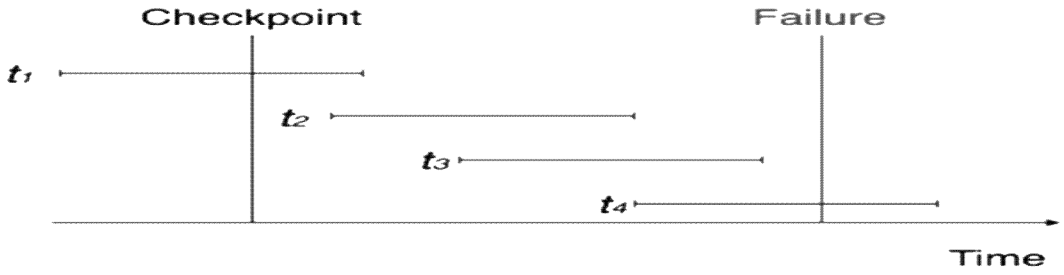
magnetic tapes, flash memory, and non-volatile (battery backed up) RAM.

- **Recovery and Atomicity**
- When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.
- When a DBMS recovers from a crash, it should maintain the following:
- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.
- There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction:
- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.
- **Log-based Recovery**

- Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.
- Log-based recovery works as follows:
- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.
- $<T_n, \text{Start}>$
- When the transaction modifies an item X, it write logs as follows:
- $<T_n, X, V_1, V_2>$
- It reads Tn has changed the value of X, from V1 to V2.
- When the transaction finishes, it logs:
- $<T_n, \text{commit}>$
- The database can be modified using two approaches:
- **Deferred database modification**: All logs are written on to the stable storage and the database is updated when a transaction commits.
- **Immediate database modification**: Each log follows an actual database modification. That is, the database is modified immediately after every operation.
- **<u>Recovery with Concurrent Transactions</u>**
- When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and

then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

- **Checkpoint**
- Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.
- **Recovery**
- When a system with concurrent transactions crashes and recovers, it behaves in the following manner:




- [ *Recovery with concurrent transactions*]
- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with <$T_n$, Start> and <$T_n$, Commit> or just <$T_n$, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <$T_n$, Start> but no commit or abort log found, it puts the transaction in the undo-list.

- All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

# CONCURRENCY CONTROL CONCEPTS

- In 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

- Timestamp-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 Locks** 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.

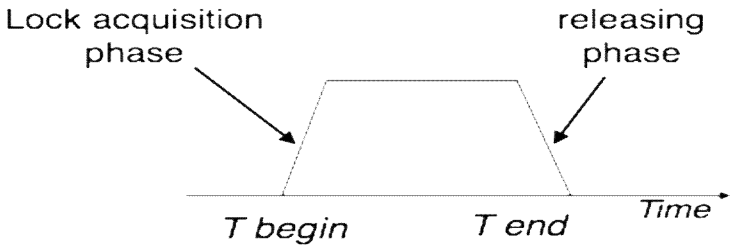
- There are four types of lock protocols available:

- **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.


- **<u>Timestamp-based Protocols</u>**

- 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 timestamp of data-item X is denoted by R-timestamp(X).
- Write timestamp of data-item X is denoted by W-timestamp(X). Timestamp ordering protocol works as follows:
- **If a transaction $T_i$ issues a read(X) operation:**
- If $TS(T_i) < W\text{-timestamp}(X)$
- Operation rejected.
- If $TS(T_i) >= W\text{-timestamp}(X)$
- Operation executed.
- All data-item timestamps updated.
- **If a transaction $T_i$ issues a write(X) operation:**
- If $TS(T_i) < R\text{-timestamp}(X)$
- Operation rejected. If $TS(T_i) < W\text{-timestamp}(X)$
- Operation rejected and $T_i$ 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.
- Timestampordering rules can be modified to make the schedule view serializable.
- Instead of making Ti rolled back, the 'write' operation itself is ignored.
- **LIVE LOCK**:-
- A **live lock** is similar to a deadlock, except that the states of the processes involved in the live lock constantly change with regard to one another, none progressing. This term was defined formally at some time during the 1970s – in Babich's 1979 article on program correctness. Live lock is a special case of resource starvation; the general definition only states that a specific process is not progressing.
- A real-world example of live lock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time.
- Live lock is a risk with some algorithms that detect and recover from deadlock. If more than one process takes action, the deadlock detection algorithm can be repeatedly triggered. This can be avoided by ensuring that only one process (chosen arbitrarily or by priority) takes action.
- **DEAD LOCK:** 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 is a common problem in multiprocessing systems, parallel computing and distributed systems, where software and hardware locks are used to handle shared resources and implement process synchronization
- **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 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.

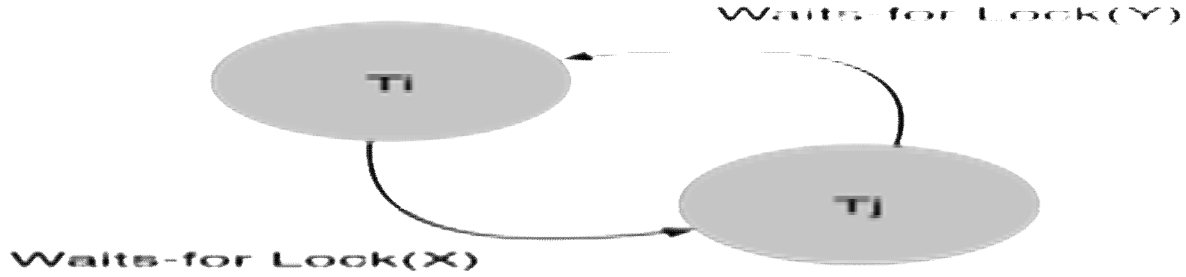
  This scheme allows the younger transaction to wait; but when an older scheme allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.
- In both the cases, the transaction that enters the system at a later stage is aborted.
- **<u>Deadlock Avoidance</u>**

- 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.




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.**

- **SERIALIZABILITY**:-
- **Serializability** is a property of a transaction schedule (history). It relates to the *isolation* property of a database transaction.
- **Serializability** of a schedule means equivalence to a *serial schedule* (i.e., sequential with no transaction overlap in time) with the same transactions. It is the major criterion for the correctness of concurrent transactions' schedule, and thus supported in all general purpose database systems.
- The rationale behind serializability is the following:
- If each transaction is correct by itself i.e. meets certain integrity conditions, then a schedule that comprises any *serial* execution of these transactions is correct (its transactions still meet their conditions): "Serial" means that transactions do not overlap in time and cannot interfere with each other, i.e. complete *isolation* between each other exists. Any order of the

transactions is legitimate, if no dependencies among them exist, which is assumed.

- As a result a schedule that comprises any execution (not necessarily serial) that is equivalent to any serial execution of these transactions is correct.
- Schedules that are not serializable are likely to generate erroneous outcomes. Examples are with transactions that debit and credit accounts with money: If the related schedules are not serializable, then the total sum of money may not be preserved. Money could disappear, or be generated from nowhere. It does not happen if serializability is maintained.
- If any specific order between some transactions is requested by an application, then it is enforced independently of the underlying serializability mechanisms. These mechanisms are typically indifferent to any specific order, and generate some unpredictable partial order that is typically compatible with multiple serial orders of these transactions.
- Two major types of serializability exist: *view-serializability*, and *conflict-serializability*. View-serializability matches the general definition of serializability given above. Conflict-serializability is a broad special case, i.e., any schedule that is conflict-serializable is also view-serializable, but not necessarily the opposite. Conflict-serializability is widely utilized because it is easier to determine and covers a substantial portion of the view-serializable schedules.
- **View-serializability** of a schedule is defined by equivalence to a serial schedule (no overlapping transactions) with the same transactions, such that respective transactions in the two schedules read and write the same data values ("view" the same data values).

- **Conflict-serializability** is defined by equivalence to a serial schedule (no overlapping transactions) with the same transactions, such that both schedules have the same sets of respective chronologically ordered pairs of conflicting operations (same precedence relations of respective conflicting operations).

# SECURITY AND INTEGRITY

- **AUTHORIZATION:**
- Authorization is the culmination of the administrative policies of the organization, expressed as a set of rules that can be used to determine which user has what type of access of which portion of database. The person who is in charge of specifying the authorization is usually called the authorizer. The authorizer is distinct from DBA and usually the person who owns the data.
- The authorization is usually maintained in the form of a table called an ***access matrix***. The access matrix contains rows called ***subject*** and columns called ***objects***. The entry in the matrix at the position corresponding to the intersection of a row and column indicate the ***type of access*** that the subject has with respect to the object.
- **Object**:
- An object is something that needs protection and one of the first steps in the authorization process is to select the objects to be used for security enforcement. Example: - a unit of data, views etc.
- The objects in the access matrix represent content independent access control. However to enforce content dependent access control, some structure for conditions or access predicates are incorporated in the access matrix.
- **Views as objects**:
- Views or sub schemes can be used to enforce security. A user is allowed to access only that portion of the database defined by the user's view. A number of users may share a view. However the user may create new views based on the views allowed. The

advantage of this approach is that the number of objects accessible to a class of users and the entry for it in the authorization table is reduced to one per view. This reduces the size of authorization matrix. The disadvantage is that the entire classes of users have the same access rights.

- **Granularity**:
- This is used for security enforcement. This could be a file, a record or a data item. The smaller the protected object, the finer the degree of specifying protection. However the finer granularity increases the size of the authorization matrix and overhead in enforcing security.
- **Subject**:
- A subject is an active element in the security mechanism. It operates on objects. A subject is a user who is given some rights to access a data object. We can also treat a class of users or an application program as a subject.
- **Access Types**:
- The access allowed to a user could be for data manipulation or control. The manipulation operations are read, insert, delete, and update. The control operations are add, drop, alter and propagate.
- **Read**: Allows reading only the object.
- **Insert**: Allows inserting new occurrences of the object type. Insert access type requires that the subject has the read access as well. However it may not allow the modification of the existing data.
- **Delete**: Allows deleting an existing occurrence of the object type.

- **Update**: Allows the subject to change the value of the occurrence of the object. An update authorization may not include a delete authorization as well.
- **Add**: Allows the subject to add new object types such as new relations, record and set types or record types and hierarchies.
- **Drop**: Allows the subject to drop or delete existing object types from the database.
- **Alter**: Allows the subject to add new data items or attributes to an existing record type or relation. It also allows the subject to drop existing data items or attributes from existing record types or relations.
- **Propagate access control**: This is an additional right that if this subject is allowed to propagate the right over the object to other subjects.
- **VIEWS**:
- Sometimes for security and other concerns, it is undesirable to have all users to see the entire relation. It would also be beneficial if we would create useful relations for different groups of users, rather than have them all manipulate the base relations. Any relation that is not part of the physical database, i.e., a virtual relation is made available to the users is known as a ***view***.
- It is possible to create views in SQL. A relation view is virtual since no corresponding physical relation exists. A view represents a different perspective of a base relation or relations.
- The result of a query operation on one or more base relations is a relation. Therefore if a user needs a particular view based on the base relations, it can be defined using a query expression.

To be useful, we assign the view a name and relate it to the query expression.

- SQL>Create view <view name> as <query expression>;

For Example: SQL> CREATE VIEW salesman AS
SELECT * FROM employees WHERE
job_title = 'Sales Representative';
The view returns only employees whose job titles are Sales Representative.

To drop a view, we use the following statement:

SQL> DROP VIEW view name;

To drop the salesman view, we use the following statement:

SQL> DROP VIEW salesman;

A view is a relation (virtual rather than base) and can be used in query an expression that is queries can be written using views as a relation.

- Views generally are not stored, since the data in the base relations may change.
- The definition of a view in a create view statement is stored in the system catalog. Having been defined, it can be used as if the view really represents a real relation. However such a virtual relation defined by a view is recomputed whenever a query refers to it.
- Views or sub schemes are used to enforce security. A user is allowed access to only that portion of the database defined by the user's view.

- A number of users may share a view. However, the users may create new views based on the views allowed.
- The advantage of this approach is that the number of objects accessible to a class of users and the entry for it in the authorization matrix is reduced one per view. This reduces the size of authorization matrix. The disadvantage is that the entire class of users has the same access rights.
- We can customize all aspects of a view, including:
- The name of the view
- The fields that appear in the view
- The column title for each field in the view
- The order of the fields in the view
- The width of columns in the view, as well as the overall width of the view
- The set of records that appear in the view (Filtering)
- The order in which records are displayed in the view (Sorting & Grouping)

**SECURITY CONSTRAINTS**:

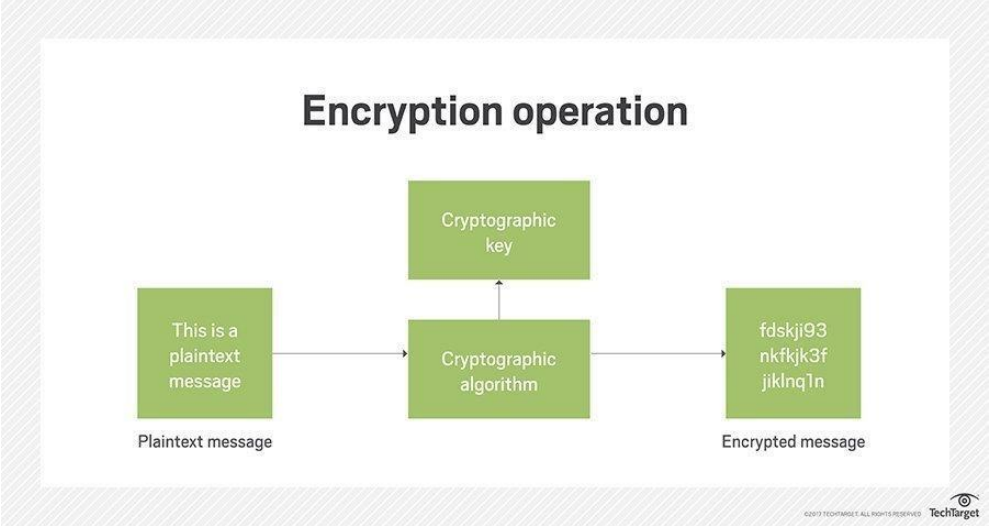
- Security in a database involves both policies and mechanisms to protect the data and ensure that it is not accessed, altered or deleted without proper authorization.
- There are four levels of defense or security constraints are generally recognized for database security: human factors, physical security, administrative control, and security and integrity mechanisms built into operating system and DBMS.
- **Human Factors**:

- At the outermost level are the human factors, which encompass the ethical, legal and social environments. An organization depends on these to provide a certain degree of protection. Thus it is unethical for a person to obtain something by stealth and it is illegal to forcibly enter the premises of an organization and hence the computing facility containing the database.
- Many countries have enacted legislation that makes it a crime to obtain unauthorized dial in access into computing system of an organization. Privacy laws also make it illegal to use information for purposes other than that for which it was collected.
- An organization usually performs some type of clearance procedure for personnel who are going to be dealing with sensitive information, including that contained in a database. This clearance procedure can be a very informal one, in the form of the reliability and trust that an employee has earned in the eyes of management or the clearance procedure could be a formal one.
- The authorizer is responsible for granting proper database access authorization to the user community. Assignment of authorization to a wrong class of users can result in possibly security violations.
- **Physical Security**:
- Physical security mechanisms include appropriate locks and keys and entry logs to computing facility and terminals.
- Security and physical storage devices (magnetic disk packs etc.) within the organization and when being transmitted from one location to another must be maintained. Access to the computing facility must be guarded, since an unauthorized

person can make copies of files by bypassing the normal security mechanism built into the DBMS and the operating system.

- Authorized terminals from which database access is allowed to have to be physically secure, otherwise unauthorized persons may be able to glean information from the database using these terminals.
- User identification and passwords have to be kept confidential; otherwise unauthorized users can borrow the identification and password of a more privileged user and compromise the database.
- **Administrative Controls**:-
- Administrative controls are the security and access control policies that determine what information will be accessible to what class of users and the type of access that will be allowed to this class.
- **DBMS and Operating System Mechanisms**:
- The proper mechanisms for the identification and verification of users. Each user is assigned an account number and a password. The operating system ensures that access to the system is denied unless the number and password are valid. In addition to the DBMS could also require a number and password before allowing the user to perform any database operations.
- The protection of data and programs, both in primary and secondary memories. This is usually done by the operating system to avoid direct access to the data in primary memory or to online files.

- The DBMS has the following features for providing security and integrity mechanisms to support concurrency, transaction management, audit and recovery data logging. In addition the DBMS provides mechanisms for defining the authorization of the user community and specifying semantic integrity constraints and checking.
- **CRYPTOGRAPHY and ENCRYPTION**:
- Consider the secure transmission of this message: "AS KINGFISHERS CATCH FIRE"
- One method of transmitting this message is to substitute a different character of the alphabet for each character in the message using an encryption key "ELIOT" and encryption algorithm as follows:
- Divide the plain text into blocks of length equal to the length of the encryption key.
- Replace each character of the plain text by an integer 00 to 26 as blank=00,A=01 …..Z=26.
- Repeat step 2 for the encryption key.
- For each block of the plain text and encryption key,replace each character by the sum modulo 27.
- Replace each integer coding to its character equivalent to get the the cipher text.
- Using the above encryption algorithm and encryption key , we get our cipher text as : "FDIZB SSOXL MQTGT HMBRA ERRFY"




- To be effective, a cipher includes a variable as part of the algorithm. The variable, which is called a *key,* that makes a cipher's output unique.
- **How does Encryption work?**
- At the beginning of the encryption process, the sender must decide what cipher will best disguise the meaning of the message and what variable to use as a key to make the encoded message unique. The most widely used types of ciphers fall into two categories: symmetric and asymmetric.
- Symmetric ciphers: also referred to as *secret key encryption*, use a single key. The key is sometimes referred to as a *shared secret* because the sender or computing system doing the encryption must share the secret key with all entities authorized to decrypt the message. Symmetric key encryption is usually much faster than asymmetric encryption. The most widely used symmetric key cipher is the Advanced Encryption Standard (AES), which was designed to protect government-classified information.
- Data Encryption Standard(DES) works by using the same key to encrypt and decrypt a message, so both the sender and the

receiver must know and use the same private key. DES has been superseded by the more secure AES algorithm.

- Asymmetric ciphers: also known as *public key encryption*, use two different -- but logically linked -- keys. This type of cryptography often uses prime numbers to create keys since it is computationally difficult to factor large prime numbers and reverse-engineer the encryption. The Rivest-Shamir-Adleman (RSA) encryption algorithm is currently the most widely used public key algorithm. With RSA, the public or the private key can be used to encrypt a message; whichever key is not used for encryption becomes the decryption key.
- Today, many cryptographic processes use a symmetric algorithm to encrypt data and an asymmetric algorithm to securely exchange the secret key.
- **Importance of Encryption:**
- Encryption plays an important role in securing many different types of information technology (IT) assets. It provides the following:
- **Confidentiality** encodes the message's content.
- **Authentication** verifies the origin of a message.
- **Integrity** proves the contents of a message have not been changed since it was sent.
- **Nonrepudiation** prevents senders from denying they sent the encrypted message.
- **Encryption vs. Decryption**
- Encryption, which encodes and disguises the message's content, is performed by the message sender. Decryption, which is the

process of decoding an obscured message, is carried out by the message receiver.

## INTEGRITY CONSTRAINTS:

Integrity constraints ensure that any properly authorized access, alteration, deletion or insertion of the data in the database does not change the consistency and validity of the data. Database integrity involves the correctness of data; this correctness has to be preserved in the presence of concurrent operations, errors in the users operations and application programmes and failures in hardware and software. Constraints are restrictions or rules applied to a database to maintain its integrity.

## TYPES OF INTEGRITY CONSTRAINTS:

1. Check constraint.
2. Entity Integrity Constraint.
3. Referential integrity constraint

1. **Check Constraint:** The value that each attribute or data item can be assigned is expressed in the form of data type, a range of values or a value from a specified set called as Check constraint. Example: In the relation EMPLOYEE the domain of the attribute Salary may be in the range of 12000 to 300000 or Mark secured by a student in STUDENT relation must be less than or equal to the Total mark.

**Creating Check constraint syntax**

```
SQL>CREATE TABLE table_name (
   ...
   column_name data_type CHECK (expression),
   ...
);
```

In this syntax, a check constraint consists of the keyword CHECK followed by an expression in parentheses. The expression should always involve the column thus constrained. Otherwise, the check constraint does not make any sense.

If we want to assign the check constraint an explicit name, we use the CONSTRAINT clause below:

```
CONSTRAINT check_constraint_name
CHECK (expression);
```

**Add Check constraint to a table**

To add a check constraint to an existing table, we use the ALTER TABLE ADD CONSTRAINT statement as follows:

**SQL>**ALTER TABLE table_name ADD CONSTRAINT check_constraint_name CHECK(expression);

**To drop a check constraint**, we use the ALTER TABLE DROP CONSTRAINT statement as follows:

**SQL>** ALTER TABLE table_name DROP CONSTRAINT check_constraint_name;

2. **Entity Integrity Constraint**:The domain values for any attribute that forms a primary key of a relation are validated against the domain constraint, called Entity Integrity Constraint.

**It does not to allow null values and redundant values against a primary key.**

**Adding a primary key to a table in ORACLE:**

To add a primary key constraint to an existing table:

SQL>ALTER TABLE table_name ADD CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ...);

**Example:** SQL>ALTER TABLE vendors ADD CONSTRAINT pk_vendors PRIMARY KEY (vendor_id);

**Dropping an Oracle PRIMARY KEY constraint**

To drop a PRIMARY KEY constraint from a table:

SQL>ALTER TABLE table_name DROP CONSTRAINT primary_key_constraint_name;

To drop the primary key constraint of the vendors table as follows:

SQL>ALTER TABLE vendors DROP CONSTRAINT pk_vendors;

**3.Referential integrity constraint:**

- The constraint that the relation R2 must not contain any unmatched foreign key values and it must contain foreign key values matching to the corresponding(Having the same Domain) primary key of another relation R1 to which it refers to, is called as Referentital Integrity Constraint.
- Two constraints while an attempt to update the relations are made:
- (a) We can not delete the records from relation R1 having the matching foreign key values in the relation R2.
- (b) We can not insert records into the relation R2 which is not having a corresponding primary key in the Relation R1.
- For Ex: In the two relations Shipment(SID,PID,QTY) and Supplier(SID, City,Status), the domain of the SID in Supplier and SID in Shipment are same and SID in Shipment is the foreign key referencing to the SID in Supplier .
- Syntax in ORACLE:

```
SQL>CREATE TABLE child_table (
   ...
   CONSTRAINT fk_name
   FOREIGN KEY(col1, col2,...) REFERENCES parent_table(col1,col2)
```

);

First, to explicitly assign the foreign key constraint a name, we use the CONSTRAINT clause followed by the name. The CONSTRAINT clause is optional. If we omit it, Oracle will assign a system-generated name to the foreign key constraint.

Second, we specify the FOREIGN KEY clause to define one or more column as a foreign key and parent table with columns to which the foreign key columns reference.

Unlike the primary key constraint, a table may have more than one foreign key constraint.

# MODEL QUESTIONS

## Questions Carrying Two Marks Each:

1. Define Data Abstraction.
2. Define Data Independence.
3. Mention the Role of DBA.
4. Define Data Base Management System.
5. What is an entity?
6. What do you mean by an attribute?
7. What is an E-R Diagram?
8. How entities and attributes are represented in an E-R Diagram?
9. What do you mean by a Relation?
10. Point out the different types of relations and how are they represented in ER Diagrams?
11. Define Selection and Projection Operations in relational algebra.
12. Define Cartesian product Operation.
13. Write down the Importance of Normalization.
14. Define Primary Key.
15. Define Foreign Key.
16. Define Third Normal Form.
17. Define Transaction.
18. Mention the different states of Transactions .

19. Mention the various properties of Transactions.

20. Define BCNF.

21. Define Data Dictionary.

22. Define Deadlock.

23.How is Two Phase Locking algorithm used to prevent Deadlock?

24. Mention the different types of failures?

25. Define Entity Integrity Constraint.

26. Define Encryption.

27. Mention the advantages and disadvantages of Hierarchical Data Model.

28. Define view.

29. Point out the various types of integrity constraints.

30. Differentiate between Rollback and Commit.

31. Define serializability.

32. Define Referential integrity constraint.

33. Define Data Integrity.

34. Mention the different types of Access Types.

35. Write down four DML commands with proper syntax.

## Questions Carrying Five Marks Each:

1. Discuss the advantages of Database Management Systems.
2. Who are the Various Database users? Explain.
3. Explain different types of Database Languages.
4. Explain briefly the Importance of Data Dictionary.
5. Define Data Independence? Discuss the various types of Data Independence.
6. What is an Attribute? Describe how are the Various Types of Attributes represented in an ER-Model.
7. What do you mean by Mapping Constraint? Explain it with Suitable Examples.
8. What do you mean by Relational Algebra? Explain different types of Operations used in database with suitable examples.
9. Differentiate between non loss decomposition and lossy decomposition with proper example.
10. Distinguish between DDL and DML.
11. Discuss the ACID properties of a Transaction.
12. Explain briefly the Different States of a Transaction?
13. Explain the two Phase Locking Concept with suitable example.
14. How Recovery can be done in case of a Database System failure?
15. Discuss the Idea behind Encryption Technique?
16. Discuss the various types of Joins.

## Questions Carrying Ten Marks Each:-

1. Distinguish between Hierarchical Data Model and Network Data Model.
2. Explain E-R Modelling Technique with suitable examples.
3. What is a Relational data Model? Explain it briefly.
4. Describe the various features of Hierarchical data Model.
5. Explain the Network data model with Example.
6. Discuss the various relational algebra operations with suitable Examples.
7. Discuss the Second Normal Form with proper example.
8. What is SQL? Write down the various DDL & DML Commands used with suitable examples.
9. Define Deadlock. How can it be prevented?
10. How can the Database be recovered in case of failures?
11. Discuss the various concurrency control protocols in DBMS.
12. How is the security maintained in a Database?
13. Discuss the advantages and disadvantages of DBMS.
14. Who is DBA? Discuss the various responsibilities of DBA.
15. Explain briefly the different Deadlock prevention and Avoidance schemes.