CHAPTER

Three

RELATIONAL DATA MODEL NORMALIZATION FOR RELATIONAL DATABASES, DATA BASE TRANSACTION PROCESSING

CHAPTER OUTLINE

3.1 RELATIONAL MODEL CONCEPTS

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.

The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

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

Some Popular Relational Database Management Systems are :

- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB - Oracle
- SQL Server and Access - Microsoft

After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDMBS languages like Oracle SQL, MySQL etc.

Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). Consider a relation Student with attributes Roll_NO, Name, Address, Phone and Age shown in Table 1.

Student

Roll No	Name	Address	Phone	Age
1	Ram	Hyd	9455123451	18
2	Ramesh	Nalgonda	9652431543	18
3	Sujit	Warangal	9156253131	20
4	Suresh	Gadwal		18

RELATIONAL MODEL CONCEPTS

1. **Attribute :** Each column in a Table. Attributes are the properties which define a relation.
   Eg : ROLL_NO, NAME

2. **Tables :** In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

3. **Tuple :** Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as :

1	Ram	Delhi	9455123451	18

4. **Relation Schema :** A relation schema represents the name of the relation with its attributes. **Eg :** Student (Roll– No, Name, Address, Phone and Age) is relation schema for Student. If a schema has more than 1 relation, it is called ***Relational Schema.***

5. **Degree :** The number of attributes in the relation is known as degree of the relation. The Student relation defined above has degree 5.

6. **Cardinality :** The number of rows present in the table is known as cardinality of the relation. The student relation defined above has cardinality 4.

7. **Column :** Column represents the set of values for a particular attribute. The column Roll_No is extracted from relation Student.

Roll_No
1
2
3
4

8. **Relation Instance :** The set of tuples of a relation at a particular instance of time is called as ***relation instance.*** Table 1 shows the relation instance of Student at a particular time. It can change whenever there is insertion, deletion or updation in the database.

9. **Relation Key :** Every row has one, two or multiple attributes, which is called ***relation key.***

10. **Attribute Domain :** Every attribute has some pre-defined value and scope which is known as ***attribute domain***.

11. **NULL Values :** The value which is not known or unavailable is called Null value. It is represented by blank space.

    Eg : Phone of Student having Roll_No 4 is Null.

Properties of Relations :

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value.
- Each attribute contains a distinct name.
- Attribute domain has no significance.
- tuple has no duplicate value.
- Order of tuple can have a different sequence.

3.2 RELATIONAL MODEL CONSTRAINTS

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called Relational Integrity or Relational Model Constraints.

1. Entity Integrity Constraint (Integrity Rule 1)

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.

Employee

EID	Name	Salary	Department	
1	Amit	6,000	Accounts	
2	Sumit	10,000	Computer	
3	Lalit	15,000	Accounts →	This is not allowed because EID is a primary key
4	Deepak	9,000	Electrical	
5	Sandeep	4,000	Civil	

2. Referential Integrity Rule (Integrity Rule 2)

A foreign key can be either null or it can have only those values which are present in the primary key with which it is related.

Suppose A be the attribute in relation R1, which is also the primary key in relation R2, then value of A in R1 is either null or same as in relation R2.

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.

These are the rules or constraints applied to the database to keep data stable, accurate or consistent. To keep database consistent we have to follow some rules known as integrity rules or integrity constraints.

Employee :

EID	Name	Salary	Dept-ID
1	Amit	6,000	1A
2	Sumit	10,000	2C
3	Lalit	15,000	4F
4	Deepak	9,000	3F
5	Sandeep	4,000	-

Null value is allowed

This is not allowed because Dept-ID is a foreign key and the value 4F is not present in attribute Dept-ID of relation Department.

Department :

Dept-ID	Dept-Name
1A	Accounts
2C	Computer
3E	Electrical
4C	Civil

3. Domain Constraints :

The restrictions which we applied on domain are known as domain constraints. These restrictions are applied to every value of attribute. By following these constraints you can keep consistency in database. These restrictions include data types (integer, varchar, char, time format, date format etc.), size of variable, checks (like value not null etc.) etc.

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.

Ex : Create table employee

(Eid char (4),

Name char (20),

Age integer (2),

Salary integer,

Primary key (Eid),

Check (age>18))

Employee :

EID	Name	Age	Salary
1	Aditya	22	10,000
2	Dinesh	– 18	5,600
3	Sumit	25	8,000
4	Lalit	20	ABC
5	Gaurav	23	11,000

Not allowed because age must be greater than 18

Not allowed because salary has integer data type

4. **Key Constraints :**

In any relation R, if attribute A is primary key then A must have unique value or you can say that primary key attribute must have unique value.

Duplicate values in primary key are invalid.

Primary key or a part of it in any relation cannot be null. Suppose A be the attribute in relation R which is taken as primary key then A must not be null.

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.

Employee :

EID	Name	Age
1	Aditya	22
2	Sumit	19
3	Deepak	25
2	Manoj	24
4	Dheeraj	28

Invalid ← (EID = 2, Manoj)

5. **Tuple Uniqueness Constraints :** In any relation R, all tuples in relation R must have distinct values. In other words Duplicate tuples within a single relation are not allowed.

Employee :

EID	Name	Age
1	Aditya	22
2	Sumit	19
3	Deepak	25
2	Sumit	19

Not allowed (Duplicate tuple) ← (EID = 2, Sumit)

3.3 RELATIONAL DATABASE SCHEMA

A set S of relation schemas that belong to the same database is called relational database schema. S is the name of the whole database schema – S = {R1, R2... Rn} – R1, R2... Rn are the names of the individual relation schemas within the database S.

Example : Company database with 6 relation database schemas.

Employee :

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

Department :

Dname	Dnumber	Mgr_ssn	Mgr_str_date

Dept_Locations

Dnumber	Dlocation

Project

Pname	Pnumber	Plocation	Dnum

Works_On

Essn	Pno	Hours

Dependent

Essn	Dependent_name	Sex	Bdate	Relationship

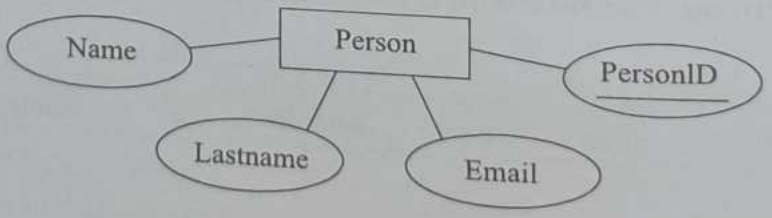
Convert ER Diagram to Relational Database schema/table

The guidelines or rules to design the relational database schema provide a step-by-step procedure. This procedure converts the ER design into a Relational Data model design in order to form the desired database. Following are the steps to convert the ER design into a Relational Data model design :

The ER Model is intended as a description of real-world entities. Although it is constructed in such a way as to allow easy translation to the relational schema model, this is not an entirely trivial process. The ER diagram represents the conceptual level of database design meanwhile the relational schema is the logical level for the database design. We will be following the simple rules :

1. **Entities and Simple Attributes :** An entity type within ER diagram is turned into a table. You may preferably keep the same name for the entity or give it a sensible name but avoid DBMS reserved words as well as avoid the use of special characters. Each attribute turns into a column (attribute) in the table. The key attribute of the entity is the primary key of the table which is usually underlined. It can be composite if required but can never be null.

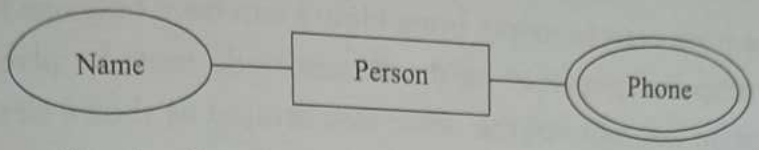
Taking the following simple ER diagram :



The initial relational schema is expressed in the following format writing the table names with the attributes list inside a parentheses as shown below for

Person(personid , name, lastname, email)

2. **Multi-Valued Attributes :** A multi-valued attribute is usually represented with a double-line oval.



If you have a multi-valued attribute, take the attribute and turn it into a new entity or table of its own. Then make a 1:N relationship between the new entity and the existing one. In simple words.

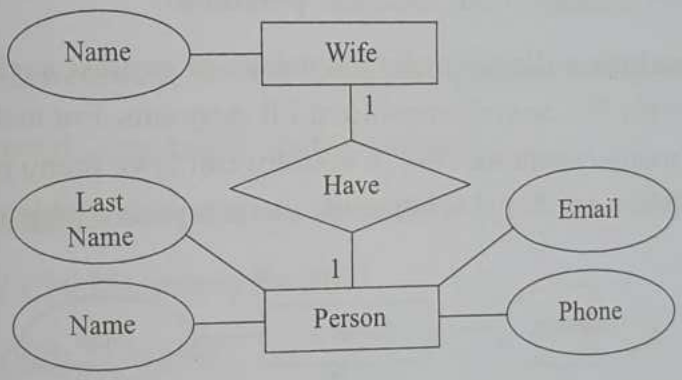
(a) Create a table for the attribute.

(b) Add the primary (id) column of the parent entity as a foreign key within the new table as shown below :

Person(personid , name, lastname, email)

Phone(phoneid , ***personid***, phone)

3. **1 : 1 Relationships :**



To keep it simple and even for better performances at data retrieval, I would personally recommend using attributes to represent such relationship. For instance, let us consider the case where the Person has or optionally has one wife. You can place the primary key of the wife within the table of the Persons which we call in this case Foreign key as shown below.

Person(personid , name, lastname, email , ***wifeid***)

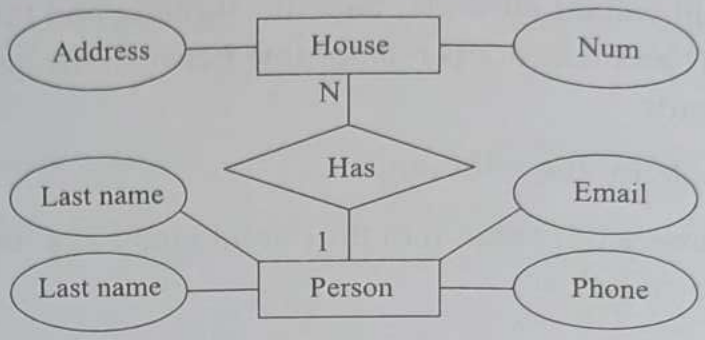
Wife (***wifeid*** , name)

Or vice versa to put the personid as a foreign key within the Wife table as shown below :

Person(personid , name, lastname, email)

Wife (wifeid , name, ***personid***)

4. **1 : N Relationships :** For simplicity, use attributes in the same way as 1:1 relationship but we have only one choice as opposed to two choices. For instance, the Person can have a House from zero to many, but a House can have only one Person. To represent such relationship the personid as the Parent node must be placed within the Child table as a foreign key but not the other way around as shown next :

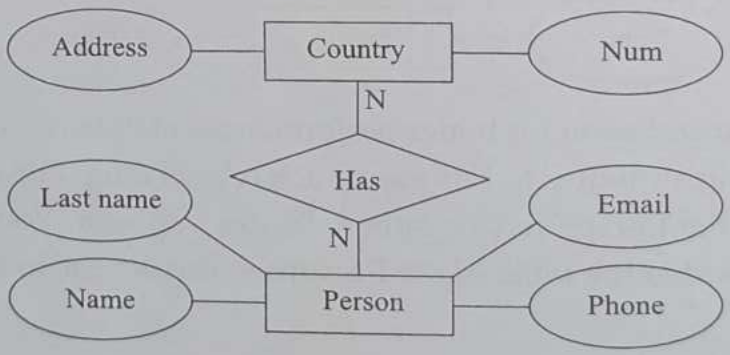


It should convert to :

Persons(personid, name, lastname, email)

House (houseid, num, address, ***personid***)

5. **N : N Relationships :** We normally use tables to express such type of relationship. This is the same for N – ary relationship of ER diagrams. For instance, The Person can live or work in many countries. Also, a country can have many people. To express this relationship within a relational schema we use a separate table as shown below :

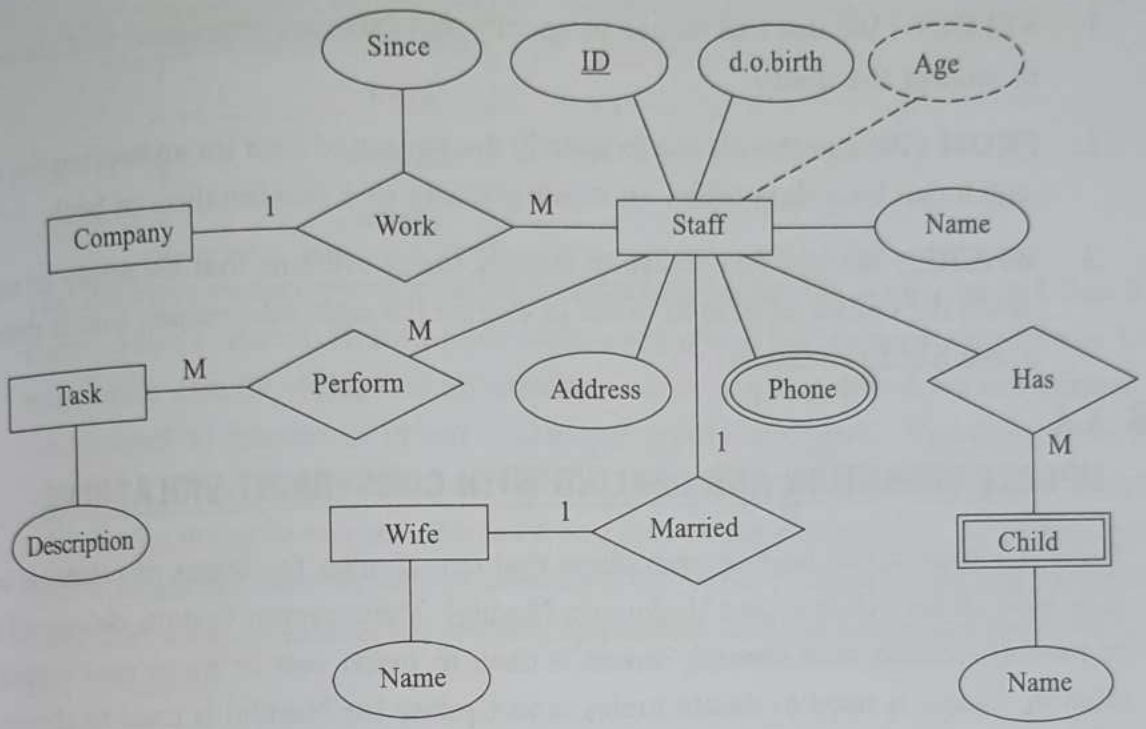


Person(personid, name, lastname, email)

Countrie(countryid, name, code)

HasRelation (hasrelatid, ***personid, countryid***)

Example to convert ER-diagram to relational schema



The relational schema for the ER Diagram is given below as:

Company(CompanyID, name, address)

Staff(StaffID, dob, address, ***WifeID***)

Child(ChildID, name, ***StaffID***)

Wife (WifeID, name)

Phone(PhoneID, phoneNumber, ***StaffID***)

Task (TaskID, description)

Work(WorkID, ***CompanyID, StaffID***, since)

Perform(PerformID, ***StaffID, TaskID***)

Interaction with Databases : The Structured Query Language or SQL makes it easy for the user to interact with the database. It is a comprehensive database language and contains statements for data definition, query, and update.

SQL also facilitates the migration of the users from one database application to another database application. Data Query Language (DQL) is a subset of SQL and is mostly in use to get the answer of most of the basic queries.

The basic set of queries consists of :

1. **SELECT** : We use this clause to specify the data or information that we require to answer the query.
2. **FROM** : We use this clause to specify the source of data for answering the query which can be a data table, an existing query or a combination of both.
3. **WHERE** : We use this clause to specify the conditions that we apply to narrow down the choice of data in order to extract the data information that is desirable in the SELECT clause.

3.4 UPDATE OPERATION AND DEALING WITH CONSTRAINT VIOLATIONS

There are three basic update operations that can change the states of relations in the data-base : Insert, Delete, and Update (or Modify). They insert new data, delete old data, or modify existing data records. Insert is used to insert one or more new tuples in a relation, Delete is used to delete tuples, and Update (or Modify) is used to change the values of some attributes in existing tuples. Whenever these operations are applied, the integrity constraints specified on the relational database schema should not be violated.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Update or Modify allows you to change the values of some attributes in existing tuples.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

Insert Operation : The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

Consider the following relations Customer and Department

Customer :

CustomerID	Customer Name	Status	DeptID
101	Amazon	Active	1
102	Google	Active	2
103	Flipkart	Active	3
104	Apple	Active	3

Department :

DeptID	Location
1	Delhi
2	Hyderabad
3	Mumbai

The Insert operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R. Insert can violate any of the four types of constraints. Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type. Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R). Entity integrity can be violated if any part of the primary key of the new tuple t is Null. Referential integrity can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation. Here are some examples to illustrate this discussion.

Operation :

Insert <NULL, 'TCS', 'Active',2> into Customer.

Result : This insertion violates the entity integrity constraint (NULL for the primary key CustomerID), so it is rejected.

Operation :

Insert <104, 'CTS', 'InActive',1> into Customer.

Result : This insertion violates the key constraint because another tuple with the same CustomerID value already exists in the Customer relation, and so it is rejected.

Operation :

Insert <106, 'Ebay','Active',4> into Customer.

Result : This insertion violates the referential integrity constraint specified on DeptID in Customer because no corresponding referenced tuple exists in Department with DeptID=4.

Operation :

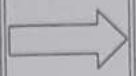
Insert <106,'Ebay', 'Active',3> into Customer.

Result : This insertion satisfies all constraints, so it is acceptable.

Update Operation :

You can see that in the below-given relation table CustomerName= 'Google' is updated from Active to Inactive.

Customer ID	Customer	Status Name	DeptID
101	Amazon	Active	1
102	Google	Active	2
103	Flipkart	Active	3
104	Apple	Active	3



Customer ID	Customer	Status Name	Dept ID
101	Amazon	Active	1
102	Google	Inactive	2
103	Flipkart	Active	3
104	Apple	Active	3

The Update (or Modify) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation R. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified. Here are some examples.

Operation :

Update the status of Customer tuple with CustomerName = 'Google' from 'Active' to 'Inactive'

Result : Aceptable.

Operation :

Update the DeptID of the Customer tuple with CustomerID=101 to 2.

Result : Acceptable.

Operation :

Update the DeptID of the Customer tuple with CustomerID = 102 to 5.

Result : Unacceptable, because it violates referential integrity.

Operation :

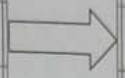
Update the CustomerID of the Customer tuple with CustomerID = 102 to 103.

Result : Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple.

Delete Operation :

To specify deletion, a condition on the attributes of the relation selects the tuple to be delete.

Customer ID	Customer	Status Name	DeptID
101	Amazon	Active	1
102	Google	Active	2
103	Flipkart	Active	3
104	Apple	Active	3



Customer ID	Customer	Status Name	Dept ID
101	Amazon	Active	1
102	Google	Active	2
103	Flipkart	Active	3

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

The Delete operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted. Here are some examples.

Operation :

Delete the Customer tuple with CustomerName='Apple'

Result : Acceptable.

Operation :

Delete the Department tuple with DeptID = 1.

Result : This deletion is not acceptable, because there are tuples in Customer relation that refer to this tuple. Hence, if the tuple in Departmet is deleted, referential integrity violations will result.

3.5 INFORMAL DESIGN GUIDELINES FOR RELATION SCHEMAS

There are four informal measures of quality for relation schema design. These measures are not always independent of one another. These Four informal measures are :

1. Meaning (semantics) of the relation attributes.
2. Reducing the redundant (repetitive) values in tuples.
3. Reducing the null values in tuples.
4. Not allowing the possibility of generating spurious tuples.

1. **Meaning of the relation attributes :** When the attributes are grouped to form a relation schema, it is assumed that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them. This meaning (Semantics) specifies how the attribute values in a tuple relate to one another. The conceptual design should have a clear meaning, if it is done carefully, followed by a systematic mapping into relations and most of the semantics will have been accounted for. Thus the easier it is to explain the meaning of the relation, the better the relation schema design will be.

   GUIDELINE 1 : Design a relation schema so that it is easy to explain its meaning. The attributes from multiple entity types and relationship types should not be combined into a single relation. Thus a relation schema that corresponds to one entity type or

Employee :

EID	Name	Salary	Dept.No	Dept.Name
1	Shivi Goyal	10,000	2	Accounts
2	Amit Chopra	9,000	2	Accounts
3	Deepak Gupta	11,000	1	Sales
4	Sandeep Sharma	8,500	5	Marketing
5	Vikas Malik	7,000	5	Marketing
6	Gaurav Jain	15,000	2	Accounts
7	Lalit Parmar	14,000	5	Marketing
		10	10	Finance
8	Vishal Bamel	10,500	2	Accounts

Cannot be inserted

   (i) ***Insertion Anomaly :*** Suppose you want to add new information in any relation but cannot enter that data because of some constraints. This is known as Insertion anomaly. In relation Employee, you cannot add new department Finance unless there is an employee in Finance department. Addition of this information violates Entity Integrity Rule1. (Primary Key cannot be NULL). In other words, when you depend on any other information to add new information then it leads to insertion anomaly.

   (ii) ***Deletion Anomaly :*** The deletion anomaly occurs when you try to delete any existing information from any relation and this causes deletion of any other undesirable information.

In relation Employee, if you try to delete tuple containing Deepak this leads to the deletion of department "Sales" completely (there is only one employee in sales department).

(iii) ***Updation Anomaly*** : The updation anomaly occurs when you try to update any existing information in any relation and this causes inconsistency of data.

In relation Employee, if you change the Dept.No. of department Accounts. We can update only one tuple at a time.

This will cause inconsistency if you update Dept.No. of single employee only otherwise you have to search all employees working in Accounts department and update them individually.

GUIDELINE 2 : Design the base relation schema in such a way that no updation anomalies (insertion, deletion and modification) are present in the relations. If present, note them and make sure that the programs that update the database will operate correctly.

2. **Redundant information in tuples and update anomalies :** One major goal of schema design is to minimize the storage space needed by the base relations. A significant effect on storage space occurred, when we group attributes into relation schemas. The second major problem, when we use relations as base relations is the problem of update anomalies. There are mainly three types of update anomalies in a relation i.e., Insertion Anomalies, deletion anomalies and modification Anomalies. Various dependencies in relational database cause these anomalies.

Anomalies : Anomalies refer to the undesirable results because of modification of data. Consider the relation Employee with attributes EID, Name, Salary, Dept.No, and Dept.Name as shown in below Figure. The various anomalies are as follows :

3. **Null values in tuples :** When many attributes are grouped together into a very big relation and many of the attributes do not apply to all tuples in the relation, then there exists many NULL's in those tuples. This wastes a lot of space. It is also not possible to understand the meaning of the attributes having NULL Values. Another problem occurs when specifying the join operation. One major and most important problem with Null's is how to consider them when aggregate functions (i.e., COUNT or SUM) are applied. The Null's can have multiple interpretations, like :

   (a) The attribute does not apply to this rule.

   (b) The attribute is unknown for this tuple.

   (c) The value is known but not present i.e., cannot be recorded.

GUIDELINE 3 : Try to avoid, placing the attributes in a base relation whose value may usually be NULL. If Null's are unavoidable, make sure that apply in exceptional cases only and majority of the tuples must have some value which is not NULL.

4. **Generation of spurious tuples :** The Decomposition of a relation schema R into two relations R1 and R2 is undesirable, because if we join them back using NATURAL Join, we do not get the correct original information. The join operation generates spurious tuples that represent the invalid information.

GUIDELINE 4 : The relation Schemas are designed in such a way that they can be joined with equality conditions on attributes that are either primary key or foreign key. This guarantees that no spurious tuples will be generated. Matching attributes in relations that are not (foreign key, primary key) combinations must be avoided because joining on such attributes may produce spurious tuples. one relationship type has a straight forward meaning.

3.6 FUNCTIONAL DEPENDENCIES

Functional dependencies are the result of interrelationship between attributes or in between tuples in any relation.

Definition : In relation R, X and Y are the two subsets of the set of attributes, Y is said to be functionally dependent on X if a given value of X (all attributes in X) uniquely determines the value of Y (all attributes in Y).

It is denoted by $X \rightarrow Y$(Y depends upon X).

Determinant : Here X is known as determinant of functional dependency.

Consider the example of Employee relation :

Employee :

EID	Name	Salary
1	Aditya	15,000
2	Manoj	16,000
3	Sandeep	9,000
4	Vikas	10,000
5	Manoj	9,000

In Employee relation, EID is primary key. Suppose you want to know the name and salary of any employee. If you have EID of that employee, then you can easily find information of that employee. So, Name and Salary attributes depend upon EID attribute. Here, X is (EID) and Y is (Name, Salary)

X (EID) : Y (Name, Salary)

The determinant is EID

Suppose X has value 5 then Y has value (Manoj, 9000)

A functional dependency exists when the value of one thing is fully determined by another.

For example : "Given the relation

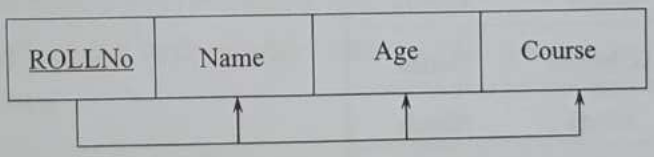
EMP (EMPNo, EmpName, Salary), attribute EmpName is functionally dependent on attribute EmpNo.

If we know EmpNo, we also know the EmpName. It is represented by EMPNo → EmpName.

A Functional dependency is a constraint between two sets of attributes in a relation from a database.

TYPES OF FUNCTIONAL DEPENDENCY

1. **Trivial FD :** A functional dependency of the form $X \rightarrow Y$ is trivial if Y is trivial if $Y \subseteq X$.
2. **Full Functional Dependency :** A FD $X \rightarrow Y$ is a full functional dependency if removal of any



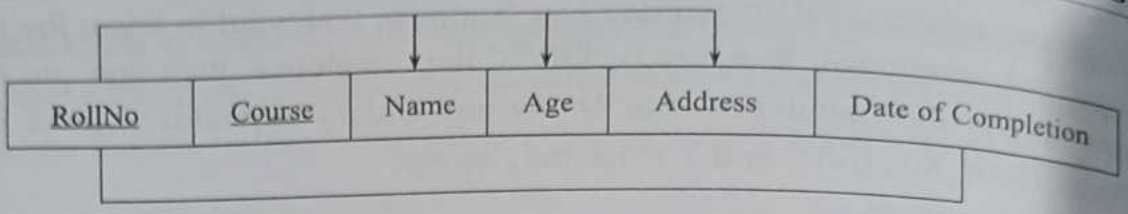
Attribute A from X means that the dependency does not hold any more. This is

Example Full/FD:

for any attribute $A \in X$, $(X - \{A\})$ does not functionally determine Y.

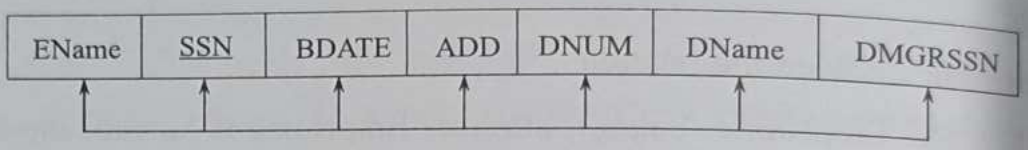
$(X - \{A\}) \rightarrow Y$ is called full functional dependency.

3. **Partial FD :** A FD $X \rightarrow Y$ is a partial dependency if some attribute $A \in X$ can be removed from X and then the dependency still hold.



This is if for some A ∈ X, $(X - (A)) \rightarrow Y$, then it is called partial dependency.

4. **Transitive Dependency :** A functional dependency $X \rightarrow Y$ in a relation scheme R is a transitive dependence if there is a set attributes Z that is neither a candidate key nor a subset of any key of



R and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

Ex :

The dependency SSN → DMGRSSN is transitive through

SSN → DNUM and DNUM → DMGRSSN

5. **Multivalued Dependency :** Let R be a relation schema and let a ⊆ R and b ⊆ R. The multivalued dependency a->>b hold on R if any legal relation r (R), for all pairs of tuples t1 and t2 in r s.t. t1 [a] = t2 [a], there exist tuples t3 and t4 in r s.t.

Example : A table with schema (name, address, car)

Name	Address	Car
Vijay	Noida	Toyota
Vijay	G. Noida	Honda
Vijay	Noida	Honda
Vijay	G. Noida	Toyota

(name, address, car) where

name ->> address

name ->> car

Closure of Functional Dependency :

The set of all functional dependency are logically implied by 'F' and the closure of 'F' is denoted by F^+

We can find all the closures by applying the following rules.

1. **Reflexivity** : If $B \rightarrow A$ then $A \rightarrow B$
2. **Augmentation** : If $A \rightarrow B$ then $CA \rightarrow CB$
3. **Transitivity** : If $A \rightarrow B$, $B \rightarrow C$, then $A \rightarrow C$
4. **Union** : If $A \rightarrow B$ holds and $A \rightarrow C$ holds then $A \rightarrow BC$ holds
5. **Decompoisition** : If $A \rightarrow BC$ holds then $A \rightarrow B$, $A \rightarrow C$ holds
6. **Psuedotransitivity** : If $x \rightarrow y$, $wy \rightarrow z$ then $wx \rightarrow z$ then $wx \rightarrow z$

3.7 NORMAL FORMS BASED ON PRIMARY KEYS

Introduction : Normalization is based on the analysis of functional dependencies. A functional dependency is a constraint between two attributes or two sets of attributes. The purpose of the database design is to arrange the various data items into an organized structure so that it generates set of relationships and stores the information without any repetition. A bad database design may result into redundant and spurious data and information.

Normalization is a process for deciding which attributes should be grouped together in a relation. It is a tool to validate and improve a logical design, so that it satisfies certain constraints that avoid redundancy of data. Furthermore, Normalization is defined as the process of decomposing relations with anomalies to produce smaller, well-organized relations. Thus, in normalization process, a relation with redundancy can be refined by decomposing it or replacing it with smaller relations that contain the same information, but without redundancy.

The goal of a relational database design is to generate a set of relation schemas that allows us to store information without any redundant (repeated) data. It also allows us to retrieve information easily and more efficiently.

For this we use a approach normal form as the set of rules. These rules and regulations are known as ***Normalization***.

Database normalization is data design and organization process applied to data structures based on their functional dependencies and primary keys that help build relational databases.

Normalization Helps :

- Minimizing data redundancy.
- Minimizing the insertion, deletion and update anomalies.
- Reduces input and output delays
- Reducing memory usage.
- Supports a single consistent version of the truth.
- It is an industry best method of tables or entity design.

Uses : Database normalization is a useful tool for requirements analysis and data modeling process of software development. Thus

The normalization is the process to reduce the all undesirable problems by using the functional dependencies and keys.

Here we use the following normal forms :

- First Normal Form (lNF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form

3.8 FIRST, SECOND AND THIRD NORMAL FORMS, BOYCE-CODD NORMAL FORM

First Normal Form (1NF)

A relation schema R is said to be in First Normal Form (INF) if the values in the domain of each attribute of the relation are atomic (single value).

For Ex: Course table

Fact-Dept	Professor	Course Prefrences	
		Course	Course-Dept
Comp-Sci	Vijay	353	Comp Sci
		370	Comp Sci
		310	Physics

	Santosh	353	Comp Sci
		320	Comp Sci
		370	Comp Sci
Chemistry	Gopal	456	Chemistry
		410	Mathematics
		370	Comp Sci

After converting to INF: Course table

Professor	Course	Fact-Dept	Course
Vijay	353	Comp Sci	Comp Sci
Vijay	370	Comp Sci	Comp Sci
Vijay	310	Comp Sci	Physics
Santosh	353	Comp Sci	Comp Sci
Santosh	320	Comp Sci	Comp Sci
Santosh	370	Comp Sci	Comp Sci
Gopal	456	Chemistry	Chemistry
Gopal	410	Chemistry	Mathematics
Gopal	370	Chemistry	Comp Sci

Second Normal Form (2NF)

- 2NF is a normal form in database normalization. It requires that all data elements in a table are full functionally dependent on the table's primary key.
- If data clement only dependent on part of primary key, then they are parsed out to separate tables.

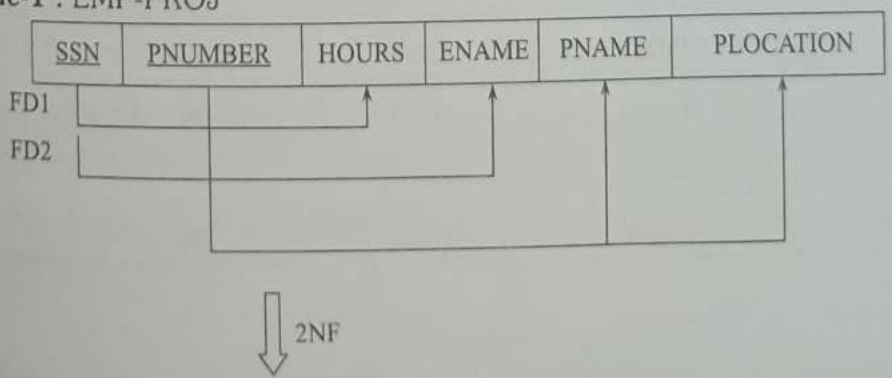
If the table has a single field as the primary key, it is automatically in 2NF.

A table is in 2NF if and only if

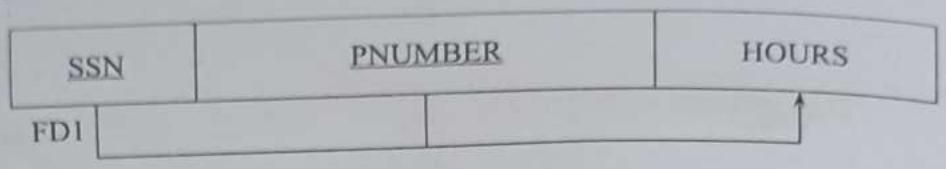
(i) It is in INF

(ii) Each non primary key attribute is full functionally dependent on the primary key.

Example-1 : EMP-PROJ



EP1 table :

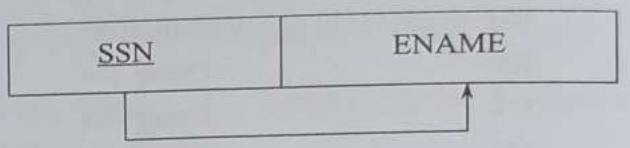


{SNN, PNUMBER} is a primary key and Hours is non key attribute.

(SSN, PNUMBER) 'HOURS

Thus HOURS is full FDS on primary key (SSN, PNUMBER)

EP2 table :

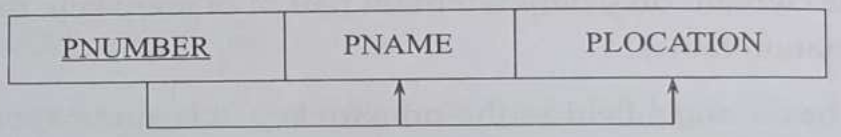


SSN is primary key and ENAME is a non key attribute.

Since non key attribute ENAME is full FDS on primary key attribute SSN. Thus, it is in 2NF.

EP3 table:

SSN → ENAME



PNUMBER → {PNAME, PLOCATION}

Example-2 : TEACHER

Course	Prof	Room	Room-Cap	Enrol-Unit
353	Vijay	A532	45	40
351	Vijay	C320	100	60
355	Santosh	H940	50	45
456	Santosh	B278	50	45
459	Gopal	D110	300	200

TEACHER Relation in Second Normal Form

T1 table :

Course	Prof	Enrol-Unt
353	Vijay	40
351	Vijay	60
355	Santosh	45
456	Santosh	45
459	Gopal	200

T2 table :

Course	Room
353	Vijay
351	Vijay
355	Santosh
456	Santosh
459	Gopal

T3 table :

Room	Room–Cap
353	Vijay
351	Vijay
355	Santosh
456	Santosh
459	Gopal

Third Normal Form (3NF)

The 3NF is a normal form used in database normalization to check if all the non key attributes of a relation depend only on the candidate keys of the relation.

This means that all non-key attributes are mutually independent or in other words that a non key attribute cannot be transitively dependent on another non-key attribute.

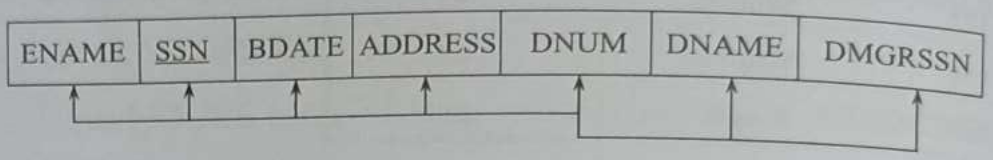
A relation schema R is in 3NF if every non prime attribute of R meets both of the following.

(i) It is in 2NF.

(ii) It is full functionally dependency on every key of *R*.

(iii) It is non transitively dependent on every key of *R*.

OR we can say : A relation schema R is in 3NF if, whenever a non trivial functional dependency



$X \rightarrow A$ holds in R.

Either

(a) X is a super key R. OR

(b) A is a prime attribute of R.

Example-1 : EMP-DEP

We can normalize EMP-DEP by decomposed into two 3NF said ED1 and ED2 respectively. Hence in 3NF:

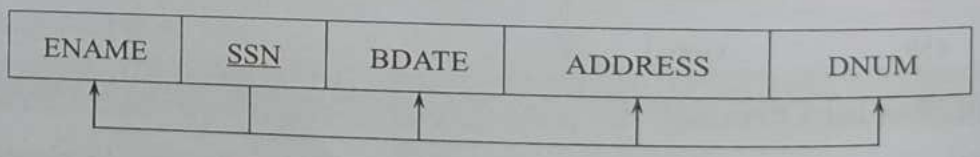
(i) ED1

(ii) ED2

The dependency SSN $\rightarrow$ DMGRSSN is transitive through the FDS

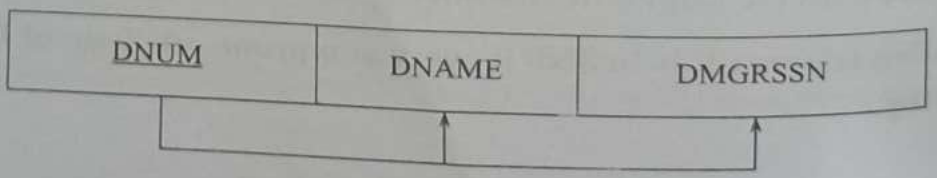
SSN $\rightarrow$ DNUM and DNUM $\rightarrow$ DMGRSSN

ED1 :



Thus EMP-DEP is not in 3NF because of the transitive dependency of DMGRSSN on SSN Via DNUM.

ED2 :



Example-2 : Student relation

Roll_No	Name	Dept.	Year	Hostel_Name
1784	Raman	Physics	1	Ganga
1648	Krishnan	Chemistry	1	Ganga
1768	Gopal	Maths	2	Kaveri
1848	Raja	Botany	2	Kaveri
1682	Maya	Geology	3	Krishna
1485	Singh	Zoology	4	Godavari

Here the dependency Roll-No → HOSTAL NAME is transitive through

ROLL-NO → YEAR AND YEAR → HOSTAL NAME

Thus the student Relation is not 3NF.

So we can normalize student Relation by decomposition into two 3NF, STUD1 AND STUD2 respectively.

(i) **STUDI relation**

Roll-No	Name	Dept.	Year
1784	Raman	Physics	1
1648	Krishnan	Chemistry	1
1768	Gopal	Maths	2
1848	Raja	Botany	2
1682	Maya	Geology	3
1485	Singh	Zoology	4

(ii) **STUD2 relation**

Year	Hostel_Name
1	Ganga
2	Kaveri
3	Krishna
4	Godavari

Boyce-Codd Normal Form (BCNF)

BCNF is a normal form used in database normalization. It is slightly stronger version of the 3NF.

A table is in BCNF if and only if :

(a) It is in 3NF and

(b) For every of its nontrivial functional dependency X 'Y, X is a super key.

OR A relation schema *R* is in BCNF if whenever a nontrivial functional dependency X'A holds in R then

(1) X is a super key of *R*.

Example-1: Student

Stud_Id	SName	Subject	Grade
10	Vijay	Computer	A
10	Vijay	Physics	B
10	Vijay	Maths	B
20	Gopal	Computer	A
20	Gopal	Physics	A
20	Gopal	Maths	C

There are two candidate keys (Stud_Id, Subject) and (SName, Subject)

In the above relation following FDS are exist :

SName, Subject → Grade

Stud_Id, Subject → Grade

Stud_Id → SName

Now BCNF decompose *R* into *R1* and *R2*.

R1

Stud_Id	Subject	Grade
10	Computer	A
10	Physics	B
10	Maths	B
20	Computer	A
20	Physics	A
20	Maths	C

R2

Stud_ID	SName
10	Vijay
20	Gopal

Example-2 : Normalize the relation professor so as it is in BCNF.

PROFESSOR

Prof. Code	Department	HOD	Present Time
P_1	Physics	Ghosh	50
P_1	Maths	Krishnan	50
P_2	Chemistry	Rao	25
P_2	Physics	Ghosh	75
P_3	Maths	Krishnan	100
P_4	Maths	Krishnan	30
P_4	Physics	Ghosh	70

PROF2

Department	HOD
Physics	Ghosh
Maths	Krishnan
Chemistry	Rao

Note : Every relation in BCNF is also in 3NF, but a relation is 3NF is not necessarily in BCNF

Ex : The above relations are in BCNF and 3NF also.

TEACH

Student	Course	Instructor
Narayan	Database	Pallaw
Vijay	Database	Navathe
Vijay	OS	Galvin
Vijay	Computer	Gopal
Santosh	OS	Ahmad
Santosh	Database	Pallaw

The dependencies show as :

{STUDENT, COURSE} ' INSTRUCTOR

INSTRUCTOR' COURSE

But the TEACH Relation is in 3NF but not in BCNF.

3.9 TRANSACTION IN DBMS

A database application program running against a relational database typically executes one or more transactions. A **transaction** is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema.

A single transaction may involve any number of retrieval operations and any number of update operations. These retrievals and updates will together form an atomic unit of work against the database. For example, a transaction to apply a bank withdrawal will typically read the user account record, check if there is a sufficient balance, and then update the record by the withdrawal amount.

A large number of commercial applications running against relational databases in **OnLine Transaction Processing (OLTP)** systems are executing transactions at rates that reach several hundred per second.

Transaction is a logical unit of work that represents the real-world events. A transaction is also defined as any one execution of a user program in a Database Management System (DBMS). The transaction management is the ability of a database management system to manage the different transactions that occur within it. A DBMS has to interleave the actions of many transactions due to performance reasons. The interleaving is done in such a way that the result of the concurrent execution is equivalent to some serial execution of the same set of transactions.

Concurrency control is the activity of coordinating the actions of transactions that operate, simultaneously or in parallel to access the shared data. How the DBMS handles concurrent executions is an important aspect of transaction management. Other related issues are how the DBMS handles partial transactions, or transactions that are interrupted before successful completion. The DBMS makes it sure that the modifications done by such partial transactions are not seen by other transactions. Thus, transaction processing

and concurrency control form important activities of any Database Management System (DBMS).

A transaction is a unit of program execution that accesses and possibly updates various data items.

(or)

A transaction is an execution of a user program and is seen by the DBMS as a series or list of actions i.e., the actions that can be executed by a transaction includes the reading and writing of database.

Transaction Operations : Access to the database is accomplished in a transaction by the following two operations,

1. **read(X) :** Performs the reading operation of data item X from the database.
2. **write(X) :** Performs the writing operation of data item X to the database.

Example : Let T1 be a transaction that transfers $50 from account A to account B.

This transaction can be illustrated as follows :

```
T1 : read (A);
A : = A – 50;
Write (A);
read (B);
B : = B + 50;
write (B);
```

Transaction Concept : The concept of transaction is the foundation for concurrent execution of transaction in a DBMS and recovery from system failure in a DBMS.

A user writes data access/updates programs in terms of the high-level query language supported by the DBMS.

To understand how the DBMS handles such requests, with respect to concurrency control and recovery, it is convenient to regard an execution of a user program or transaction, as a series of reads and writes of database objects.

To read a database object, it is first brought in to main memory from disk and then its value is copied into a program. This is done by read operation.

To write a database object, in-memory, copy of the object is first modified and then written to disk. This is done by the write operation.

3.10 ACID PROPERTIES OF TRANSACTIONS

Collection of operations that form a single logical unit of work are called transactions.

A transaction is a unit of program execution that accesses and possibly updates various data items.

Transactions should possess several properties, often called the ACID properties; they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties :

A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after transaction, certain properties are followed. These are called ***ACID properties.***

Atomicity :

By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves following two operations.

Abort : If a transaction aborts, changes made to database are not visible.

Commit : If a transaction commits, changes made are visible.

Atomicity is also known as the ***'All or nothing rule'.***

Consider the following transaction T consisting of T1 and T2 : Transfer of 100 from account X to account Y.

Before : X : 500	**Y : 200**
Transaction T	
T1	**T2**
Read (X)	Read (Y)
X : X – 100	Y : Y + 100
Write (X)	Write (Y)
After : X : 400	Y : 300

If the transaction fails after completion of T1 but before completion of T2.(say, after write(X) but before write(Y)), then amount has been deducted from X but not added to Y. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

Consistency : This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to correctness of a database. Referring to the example above, The total amount before and after the transaction must be maintained.

Total before T occurs = 500 + 200 = 700.

Total after T occurs = 400 + 300 = 700.

Therefore, database is consistent. Inconsistency occurs in case T1 completes but T2 fails. As a result T is incomplete.

Isolation : This property ensures that multiple transactions can occur concurrently without leading to inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Let X = 500, Y = 500.

Consider two transactions T and T".

T	T"
Read (X)	Read (X)
X: = X*100	Read (Y)
Write (X)	Z; = X + Y
Read (Y)	Write (Z)
Y : Y – 50 Write	

Suppose T has been executed till Read (Y) and then T" starts. As a result, interleaving of operations takes place due to which T" reads correct value of X but incorrect value of Y and sum computed by

T" : (X+Y = 50,000 + 500 = 50,500)

is thus not consistent with the sum at end of transaction :

T : (X+Y = 50,000 + 450 = 50,450).

This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability : This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if system failure occurs. These updates now become permanent and are stored in a non-volatile memory. The effects of the transaction, thus, are never lost.

The ACID properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

3.11 COMMIT, ROLLBACK, AND SAVE POINT

Transaction Control Language (TCL) commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

COMMIT command : COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

COMMIT ;

ROLLBACK command : This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

```
ROLLBACK TO savepoint_name;
```

SAVEPOINT command : SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

```
SAVEPOINT savepoint_name;
```

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

Using Savepoint and Rollback :

Following is the table class,

ID	Name
1	Abhi
2	Adam
4	Alex

Lets use some SQL queries on the above table and see the results.

```
INSERT INTO class VALUES(5, 'Rahul');
COMMIT;
UPDATE class SET name = 'Abhijit'  WHERE id = '5';
SAVEPOINT A;
INSERT INTO class VALUES(6, 'Chris');
SAVEPOINT B;
INSERT INTO class VALUES(7, 'Bravo');
SAVEPOINT C;
SELECT * FROM class;
```

ID	Name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris
7	Bravo

Now let's use the ROLLBACK command to roll back the state of data to the savepoint B.

ROLLBACK TO B;

SELECT * FROM class;

Now our class table will look like,

ID	Name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris

Now let's again use the ROLLBACK command to roll back the state of data to the savepoint A

ROLLBACK TO A;

SELECT * FROM class;

Now the table will look like,

ID	Name
1	Abhi
2	Adam
4	Alex
5	Abhijit

3.12 CONCEPT OF SERIALIZABILITY

A non serial schedule is said to be serializable, if it is conflict equivalent or view-equivalent to a serial schedule.

Consider a set of transactions (T1, T2, ..., Ti). S1 is the state of database after they are concurrently executed and successfully completed and S2 is the state of database after they are executed in any serial manner (one-by-one) and successfully completed. If S1 and S2 are same then the database maintains serializability.

Serializability is a widely accepted standard that ensures the consistency of a schedule. A schedule is consistent if and only if it is serializable. A schedule is said to be serializable if the interleaved transactions produces the result, which is equivalent to the result produced by executing individual transactions separately.

Example :

Transaction T_1	Transaction T_2		Transaction T_1	Transaction T_2
read (X)			read(X)	
write (X)			write(X)	
read (Y)				read(X)
write (Y)				write (X)
	read(X)		read(Y)	
	write (X)		write (Y)	read (Y)
	read (Y)			write (Y)
	write (Y)			

Serial Schedule Two interleaved transaction schedule

The above two schedules produce the same result, these schedules are said to be serializable. The transaction may be interleaved in any order and DBMS doesn't provide any guarantee about the order in which they are executed.

There two different types of Serializability. They are,

1. Conflict Serializability
2. View Serializability

1. **Conflict Serializability** : Consider a schedule S1, consisting of two successive instructions IA and IB belonging to transactions TA and TB refer to different data items then it is very easy to swap these instructions.

The result of swapping these instructions doesn't have any impact on the remaining instructions in the schedule. If IA and IB refers to same data item then the following four cases must be considered,

Case-1 : IA = read(x), IB = read(x),

Case-2 : IA = read(x), IB = write(x),

Case-3 : IA = write(x), IB = read(x),

Case-4 : IA = write(x), IB = write(x),

Case-1 : Here, both IA and IB are read instructions. In this case, the execution order of the instructions is not considered since the same data item x is read by both the transactions TA and TB.

Case-2 : Here, IA and IB are read and write instructions respectively. If the execution order of instructions is IA → IB, then transaction TA cannot read the value written by transaction TB in instruction IB. but order is IB → IA, then transaction TA can read the value written by transaction TB. Therefore in this case, the execution order of the instructions is important.

Case-3 : Here, IA and IB are write and read instructions respectively. If the execution order of instructions is IA → IB, then transaction TB can read the value written by transaction TA, but order is IB → IA, then transaction TB cannot read the value written by transaction TA. Therefore in this case, the execution order of the instructions is important.

Case-4 : Here, both IA and IB are write instructions. In this case, the execution order of the instructions doesn't matter. If a read operation is performed before the write operation, then the data item which was already stored in the database is read.

Conflicting operations :

Two operations are said to be in conflict, if they satisfy all the following three conditions:

(i) Both the operations should belong to different transactions.

(ii) Both the operations are working on same data item.

(iii) At least one of the operation is a write operation.

Let's see some examples to understand this :

Example-1 : Operation W(X) of transaction T1 and operation R(X) of transaction T2 are ***conflicting operations***, because they satisfy all the three conditions mentioned above. They belong to different transactions, they are working on same data item X, one of the operation in write operation.

Example-2 : Similarly Operations W(X) of T1 and W(X) of T2 are ***conflicting operations.***

Example-3 : Operations W(X) of T1 and W(Y) of T2 are non-conflicting operations because both the write operations are not working on same data item so these operations don't satisfy the second condition.

Example-4 : Similarly R(X) of T1 and R(X) of T2 are ***non-conflicting operations*** because none of them is write operation.

Example-5 : Similarly W(X) of T1 and R(X) of T1 are ***non-conflicting operations*** because both the operations belong to same transaction T1.

Conflict Equivalent Schedules :

Two schedules are said to be conflict Equivalent if one schedule can be converted into other schedule after swapping non-conflicting operations.

Conflict Serializable check :

Let's check whether a schedule is conflict serializable or not. If a schedule is conflict Equivalent to its serial schedule then it is called **Conflict Serializable schedule**. Lets take few examples of schedules.

Example of Conflict Serializability :

Lets consider this schedule :

T1	T2
R(A)	
R(B)	
	R(A)
	R(B)
	W(B)
W(A)	

To convert this schedule into a serial schedule we must have to swap the R(A) operation of transaction T2 with the W(A) operation of transaction T1. However we cannot swap these two operations because they are conflicting operations, thus we can say that this given schedule is **not Conflict Serializable.**

Lets take another example :

```
T1      T2
-----   ------
R(A)
        R(A)
        R(B)
        W(B)
R(B)
W(A)
```

Lets **swap non-conflicting operations :**

After swapping R(A) of T1 and R(A) of T2 we get :

```
T1      T2
-----   ------
R(A)
        R(A)
        R(B)
        W(B)
R(B)
W(A)
```

```
T1      T2
-----   ------
        R(A)
R(A)
        R(B)
        W(B)
R(B)
W(A)
```

After swapping R(A) of T1 and R(B) of T2 we get:

```
T1      T2
-----   ------
        R(A)
```

R(A)

R(B)

W(B)

R(B)

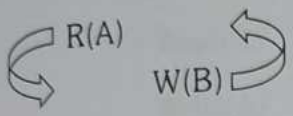
W(A)

T1 T2

----- ------

R(A)

R(B)



R(B)

W(A)

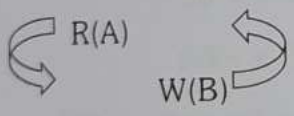
After swapping R(A) of T1 and W(B) of T2 we get:

T1 T2

----- ------

R(A)

R(B)



R(B)

W(A)

T1 T2

----- ------

R(A)

R(B)

W(B)

R(A)

R(B)

W(A)

We finally got a serial schedule after swapping all the non-conflicting operations so we can say that the given schedule is **Conflict Serializable.**

2. **View Serializability :**

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

View Equivalent :

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

(i) Initial Read :

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

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
Read(A)	
	Write(A)

Schedule S2

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

(ii) Updated Read :

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

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

Schedule S1

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

Schedule S2

Above two schedules are not view equal because, in S1, T3 is reading A updated by T2 and in S2, T3 is reading A updated by T1.

(iii) Final Write :

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

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

Schedule S1

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

Schedule S2

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

Example : Schedule S

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

With 3 transactions, the total number of possible schedule = 3! = 6

S1 = <T1 T2 T3>

S2 = <T1 T3 T2>

S3 = <T2 T3 T1>

S4 = <T2 T1 T3>

S5 = <T3 T1 T2>

S6 = <T3 T2 T1>

Taking first schedule S1 :

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

Step-1 : Final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2 : Initial Read

The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3 : Final Write

The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.

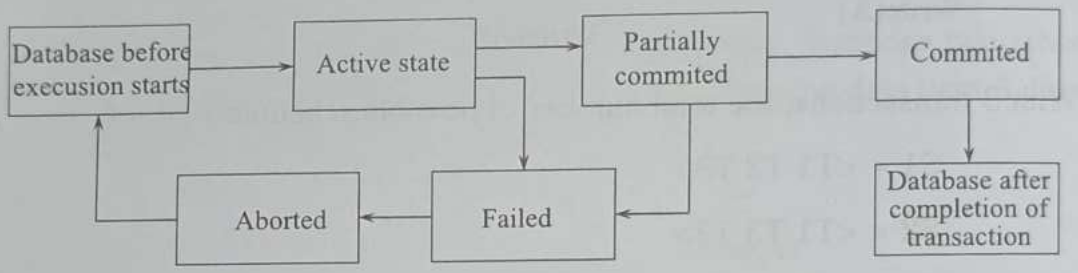
The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.

Hence, view equivalent serial schedule is :

$$T1 \rightarrow T2 \rightarrow T3$$

3.13 STATES OF TRANSACTIONS

A transaction must be in one of the following states as shown in Figure.

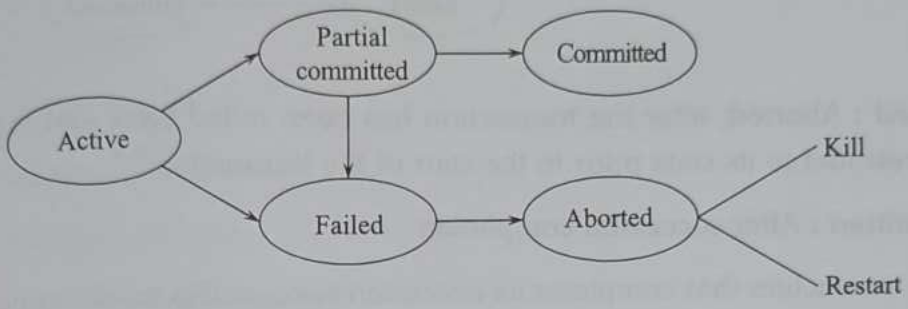


1. **Active state :** It is the initial state of transaction. During execution of statements, a transaction is in active state.
2. **Partially committed :** A transaction is in partially committed state, when all the statements within transaction are executed but transaction is not committed.
3. **Failed :** In any case, if transaction cannot be preceded further then transaction is in failed state.
4. **Committed :** After successful completion of transaction, it is in committed state.

The various states of a Database Transaction are listed below

State	Transaction Types
Active State	A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed.
Partially Committed	A transaction goes into the partially committed state after the end of a transaction.
Committed State	When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.

Failed State	A transaction considers failed when any one of the checks fails or if the transaction is aborted while it is in the active state.
Terminated State	State of transaction reaches terminated state when certain transactions which are leaving the system can't be restarted.



Let's study a state transition diagram that highlights how a transaction moves between these various states.

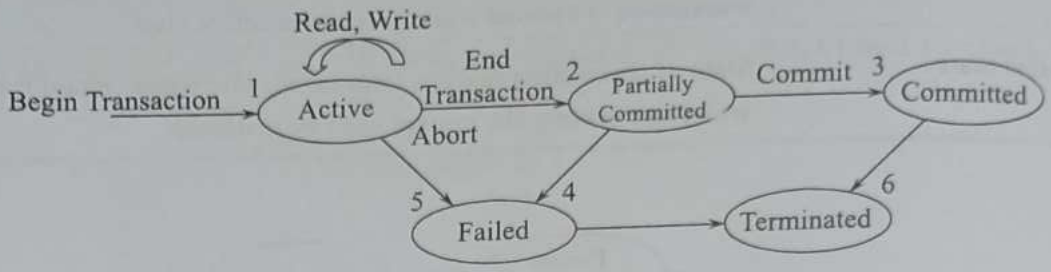
1. Once a transaction states execution, it becomes active. It can issue READ or WRITE operation.
2. Once the READ and WRITE operations complete, the transactions becomes partially committed state.
3. Next, some recovery protocols need to ensure that a system failure will not result in an inability to record changes in the transaction permanently. If this check is a success, the transaction commits and enters into the committed state.
4. If the check is a fail, the transaction goes to the failed state.
5. If the transaction is aborted while it's in the active state, it goes to the failed state. The transaction should be rolled back to undo the effect of its write operations on the database.
6. The terminated state refers to the transaction leaving the system.

A transaction must be in one of the following states :

Active : This state is the initial state, the transaction says in this state while it is executing.

Partial Committed : After the final statement has been executed.

Failed : Failed, after the discovery that normal execution can no longer proceed.



Aborted : Aborted, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

Committed : After successful completion.

i.e., A transaction that completes its execution successfully is said to be committed.

- Once a transaction has been committed we cannot undo its effect by aborting it.
- A transaction said to be terminated if it has either committed or aborted.
- A transaction starts in the active state. When it finished its final statements, it enters the partially committed state. At this point, the transaction has completed its execution, but it is still possible that it may have to be aborted.

REVIEW QUESTIONS

One Mark Questions :

1. Define relation schema.
2. Define tuple.
3. Define degree.
4. Define cardinality.
5. What is NULL value?
6. Define relation instance.
7. Define update operation.
8. What is functional dependency?
9. Define full functional dependency
10. Define transitive dependency.
11. What is normalization?
12. What is first Normal Form?
13. What is second Normal Form?
14. Define Boyce Codd Normal Form.
15. Define transaction.
16. Define Atomicity.
17. Define Consistency.
18. Define Isolation.
19. Define Durability.
20. What is serializability?

Three Mark Questions :

1. Write the properties of relations.
2. List the relational model constraints.
3. Write about Referential Integrity rule.
4. Write about Domain constraints.

5. Write an example to convert ER-diagram to relational schema.
6. Describe the types of functional dependencies.
7. List different types of normal forms.
8. Write about ACID properties of transactions.
9. Describe Commit, Rollback and Save point.
10. Write about the types of Serializabilities.
11. List the states of transaction.

Five Mark Questions :

1. Explain relational model constraints.
2. Explain relational database schema.
3. Give the steps to convert ER diagram to relational database schema/table.
4. Explain different update operation constraint violations.
5. Explain Informal design guidelines for relation schemas.
6. Explain 1NF with an example.
7. Illustrate 2NF with an example.
8. Explain 3NF with an example.
9. Illustrate BCNF with an example.
10. Explain the Commit, Rollback and Save Point statements with an example.
11. Explain about Conflict Serializability with an example.
12. Explain about View Serializability with an example.
13. Illustrate in detail about the states of transactions.

CHAPTER

Four

CONCEPT OF SQL

CHAPTER OUTLINE

4.1 INTRODUCTION TO SQL

SQL (Structured Query Language) :

DBMS requires a language to access the data.

Structured Query Language (SQL) is the language used by relational database system (RDBMS)

- It is developed by IBM company in early 1970's
- Initially it is named as System R and later it is named as SQL.

4.1.1 FEATURES OF SQL

1. SQL can be used by a range of users, including those with little or no programming experience.
2. It is a non procedural language.
3. It reduces the amount of time required for creating and maintaining systems.
4. It is an English-Like Language.
5. It is used to access, create and retrieve data from database.
6. It is used to execute queries.
7. It is used to insert, update and delete records from database.

4.1.2 BENEFITS OF SQL

1. SQL is an english like language
2. It is easy to learn.
3. SQL is non procedural language.

   i.e., SQL specifices only what is required and not how it should be done.
4. More productivity

   i.e., with a single statement we can perform multiple work/task like creation of tables etc.
5. With SQL statements we can retrive multiple rows at a time and even modify multiple rows at a time.
6. It is more powerful then COBOL.
7. All RDBMS supports SQL.
8. It can be used by specialist and non specialist person.

   Structured Query Language (SQL) is a language that provides an interface to relational database systems. SQL is the standard language for Relation Database System. All

relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Why SQL ?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

In common usage SQL also encompasses :

- DML (Data Manipulation Languages) for INSERT, UPDATE, DELETE operations
- DDL (Data Definition Language) used for creating and modifying tables and other database structures.

4.1.3 COMPONENTS OF SQL

There are following components of SOL.

1. **DDL (Data Definition Language) :** It is a set of SQL commands used to create, modify and delete data base structure but not data.

   For Example :

   (i) **Create :** To create objects in the database.

   (ii) **Alter :** Alters the structure of the database.

   (iii) **Drop :** Delete objects from the database.

   (iv) **Truncate :** Remove all records from a table, including all spaces allocated for the records are removed.

   (v) **Comment :** Add comments to the data dictionary.

2. **DML (Data Manipulation Language) :** It is the area of SOL that allows changing data within the database.

   For Example :

   (i) **Insert :** Insert data into a table.

   (ii) **Update :** Updates existing data within a table.

   (iii) **Delete :** Deletes all records from a table, the space for the records remain.

   (iv) **Lock TableL :** Control concurrency.

3. **DCL (Data Control Language) :** It is the components of SQL statements that control access to data and to the database.

   For Example :

   (i) **Commit :** Save work done

   (ii) **Save Point :** Identify a point in a transaction to which you can later roll back.

   (iii) **Roll Back :** Restore database to original since the last COMMIT.

   (iv) **Grant/Revoke :** Grant or take back permissions to or from the oracle users.

   (v) **Set Transaction :** Change transaction options like what rollback segment to use.

4. **DQL (Data Query Language) :** It is the component of SQL statement that allows getting data from the database and imposing ordering upon it.

   For Example :

   (i) **Select :** Retrieve data from the database.

4.2 NULL AND PSEUDO COLUMNS IN SQL

Null : A null value is a value which is unavailable, unassigned or inapplicable. A null value is not seen as zero, zero is a number.

Null values are handled by SQL. When a column name is defined as not null that column becomes a mandatory column and the user must enter data into it.

Pseudo Column : A pseudo-column behaves like a table column but is not actually stored in the table. You can select from pseudo-columns, but you cannot insert, update, or delete their values. A pseudo-column is also similar to a function without arguments.

Some of the Pseudo Columns in SQL are :

1. **NULL :** A null value.
2. **SYSDATE :** It returns system date. With the help of Sysdate pseudo column we can find the current date of our system. This helps us to run a quick check on our system.

   Syntax :

   SELECT SYSDATE() as an alias;

   Example: Let us now find the current date of the system we are using right now.

   Query :

   SELECT SYSDATE() as System_Date;

3. **UID :** User ID (or) User Identification number it returns a unique number assigned to your session used by DBA.

4. **USER :** We use the USER pseudo column to pull the name of the user of our database. With the help of the USER pseudo column, we can find who is currently accessing our database. This helps us to increase the security of our system and monitor it, to avoid any illegal access to our database.

Syntax :

SELECT USER() AS alias;

Example: Let us now find the user who is accessing the database on our system.

Query :

SELECT USER() AS UserName;

5. **ROWID** : It is a pointer to where row is stored and it contains the complete row description. For each row in the database, the ROWID pseudocolumn returns a row's address. The ROWID contains 3 information about row address :

FileNo: FileNo means Table Number.

DataBlockNo: DataBlockNo means the space assigned by the oracle sql engine to save the record.

RecordNo: Oracle engine maintains the record number for each record.

The syntax and usage of the ROWID pseudo column are as follows.

Syntax:

SELECT rowid(columnName) as alias from tableName;

Example: Let us now find the addresses of each row in our database location.

Query:

SELECT rowid(location_id) as address from location;

SELECT ROWID, ename FROM emp WHERE deptno = 20;

6. **ROWNUM:** We use the ROWNUM to get the desired number of rows in the order they are inserted in the database. The syntax of the Rownum pseudo column is as follows.

Syntax:

SELECT * FROM tableName where ROWNUM condition;

Example: Let us now find the data from the database such that the ROWNUM is equal to the value 2.

Query :

SELECT * from location where ROWNUM=2;

7. **CURRVAL and NEXTVAL :** A sequence is a schema object that can generate unique sequential values. These values are often used for primary and unique keys. You can refer to sequence values in SQL statements with these pseudocolumns:

   CURRVAL: Returns the current value of a sequence.

   NEXTVAL: Increments the sequence and returns the next value.

   Examples :

   SELECT STUDENTSEQ.currval FROM DUAL;

   INSERT INTO STUDENT VALUES (STUDENTSEQ.nextval, BISHAL, JAVA, 7902);

8. **LEVEL :** For each row returned by a hierarchical query, the LEVEL pseudocolumn returns 1 for a root node, 2 for a child of a root, and so on.

   Uses of Pseudo Columns in SQL

   Some of the important uses of the Pseudo Columns are as follows:

   - Used to get the system-related data.
   - Helps us to get the metadata of our database.
   - Allow us to get the details of access rights on our database.
   - Helps us to keep a check on our database.

4.3 SQL DATA TYPES

We have a huge volume of data available to us, and it would be complicated to handle this data if we store all the data in a single data type. Data type comes in handy when we need to handle a variety of data available to us.

For example: If we consider the data of a student, we will see that we have a diversity of data types being used.

The name of the candidate is a string or varchar, the roll number is an int value, date of birth is a date object. With this example, we can clearly understand the need for data types in the creation of a database.

Available Datatypes : Some of the data types available in SQL are :

1. Binary Datatype

Binary datatype has further divisions as follows :

S.No.	Datatype	Description
1.	binary	Stores fixed-length binary data with a maximum capacity of 8000 bytes.
2.	varbinary	Stores variable-length binary data with a maximum capacity of 8000 bytes.
3.	varbinary(max)	Stores variable-length binary data with a maximum capacity of 231 bytes.
4.	image	Stores variable-length binary data with a maximum capacity of 2,147,483,647 bytes.

2. Exact Numeric Datatype

Further divisions of an exact numeric datatype are as follows :

S.No.	Datatype	Description	Range
1.	bigint	Holds the big integer values. Both inclusive	[-9,223,372,036,854,775,808 to 9,223,372,036,854,775,808]
2.	int	Holds the integer values.	[-2,147,483,648 to 2,147,483,648] Both inclusive
3.	smallint	Holds small integer values.	[-32,768 to 32,768] Both inclusive
4.	tinyint	Holds a tiny range of int from 0 to 255.	[0 to 255] Both inclusive
5.	bit	Holds the value 0 or 1.	0 or 1
6.	decimal	Holds the decimal values.	[-10^38+1 to 10^38-1] Both inclusive
7.	numeric	Holds the numerical values	[-10^38+1 to 10^38-1] Both inclusive
8.	money	Holds the monetary values.	[-922,337,203,685,477.5808 to 922,337,203,685,477.5808]Both inclusive
9.	smallmoney	Holds small monetary values.	[-214,748.3648 to 214,748.3648] Both inclusive

3. Approximate Numeric Datatype

The divisions of an approximate numeric datatype are as follows:

S.No.	Datatype	Description	Range
1	float	Holds the float values.	-1.79E+308 to 1.79E+308
2	real	Holds the real numerical values.	-3.40E+38 to 3.40E+38

4. Character String Datatype

Further classifications of character and string data types are as follows:

S.No.	Datatype	Description
1.	char	Holds fixed length characters with a maximum capacity of 8000 characters.
2.	varchar	Holds variable-length non-Unicode characters with a max capacity of 8000 characters.
3.	varchar(max)	Holds Variable-length non-Unicode characters with a max capacity of 231 characters.
4.	text	Holds Variable-length non-Unicode data with a max capacity of 2,147,483,647 characters.

5. Unicode character String Datatype

Further divisions of Unicode character datatype are as follows:

S.No.	Datatype	Description
1	nchar	Stores Unicode fixed length characters with a max capacity of 4000 characters.
2.	Nvarchar	Stores Unicode variable-length characters with a max capacity of 4000 characters.
3.	nvarchar(max)	Stores Unicode variable-length characters with a max capacity of 231 characters.
4.	ntext	Stores Unicode variable-length characters with a max capacity of 1 GB characters.

6. Date and time Datatype

Further classifications of date and time datatype are as follows :

S.No.	Datatype	Description	Range
1	datetime	Holds the value of date and time both in a single variable.	Jan 1, 1753 to Dec 31, 9999
2	smalldatetime	Holds the smaller range of values of date and time both in a single variable.	Jan 1, 1753 to Jun 6, 2079
3	date	Holds the value of the date.	No Limits
4	time	Holds the value of time.	No Limits
5	year	Holds the two-digit or four-digit value for the variables of type year.	No Limits
6	timestamp	Holds the value of time in the variable.	No Limit

7. Miscellaneous Data Types

Further classifications of miscellaneous data types are as follows:

S.No.	Datatype	Description
1	clob	Holds large character objects with a max capacity of 2Gb.
2	blob	Holds large binary objects.
3	XML	Holds the incoming XML data.
4	JSON	Holds the incoming JSON data.
5	cursor	Holds the cursor object.
6	UUID	Holds the universally unique identifiers.
7	bfile	Holds the binary data stored in external files.

Datatype	Description	Size
Number(p, s)	It is used to store numeric data types. p stands for precision (total number of decimal digits) and s stands for scale (total number of digits after decimal point).	Range of p is from 1 to 38. And s is from - 84 to 127.
Date	It is used to store date and time values.	Range of date is from jan 1, 47 B.C. to Dec. 31. 9999 A.D.
Char (size)	It is used to store fixed size character data.	Range of char is 1 (By default) to 2000 bytes.
Varchar 2 (size)	It is used to store variable size character data.	Range of varchar 2 is 1 (By default) to 4000 bytes.
Long	It is used to store variable size character data.	Range of long is upto 2 GB.
Clob	It is used to store variable size character data.	Range of clob is upto 4 GB.
Raw (size)	It is used to store fixed binary data.	Maximum size is upto 2000 bytes.
Long raw	It is used to store variable binary data.	Maximum size is upto 2 GB.

4.4 DATA DEFINITION LANGUAGE STATEMENTS IN SQL

Shema : It is actually a logical representation of a database. When we are using relational database management system we are storing data logically in the form of tables. When we are using E-R model we are storing data in the form of entities.

Examples :

Student table schema

Rollnumber	name	address

Course table schema

Cid	Cname	Cduration

We can implement the above schema using SQL data definition language (DDL) commands.

- CREATE
- ALTER
- DROP

Create :

SQL Create Statement : It is basically used to create tables.

Syntax :

CREATE TABLE TABLENAME (column1name datatype (size), column2name datatype (size),

:

columnNname datatype (size));

EXAMPLE - 1

Create a student table with PIN, Name, Date of birth, Age, Sex, Marks, Phone no, Address.

```
>CREATE   TABLE   STUDENT
          (PIN    Varchar(20),
          Name    Varchar(20),
```

```
                    dob         Date,
                    age         Number(2),
                    sex         Char(1),
                    marks       Number(3),
                    phno.       Number(18),
                    address     Varchar(50));
```

EXAMPLE - 2

Create employee table having fields name, id, salary, address, phno, department.

```
> CREATE    TABLE       EMPLOYEE
                (NAME       Varchar(20),
                 Id         Number(8),
                 salary     Number(5),
                 address    Varchar(50),
                 phno       Number(10),
                 department Varchar(20),
                );
```

SQL ALTER Statement : It is used basically for adding, deleting (or) modifying columns in a existing table.

You can add a column (or) delete a column (or) modify a column in existing table.

Adding a column :

Syntax :

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example :

```
ALTER TABLE EMP
ADD deptname varchar2(10);
```

Delete a column :

Syntax :

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

Example :

```
ALTER TABLE emp
DROP COLUMN deptname;
```

Change datatype :

Syntax :

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

Example :

```
ALTER TABLE EMP
ALTER COLUMN deptname NUMBER (2);
```

SQL DROP Statment : It is used to delete indexes, tables, database from the system.

DROP Indexes :

Syntax : DROP INDEX index-name ON table_name;

DROP Table : used to delete table from database.

Syntax :

```
DROP TABLE table_name;
DROP TABLE EMP;
```

DROP Database : It is used to delete database from system.

Syntax :

```
DROP DATABASE database_name;
DROP DATABASE Employee;
```

TRUNCATE TABLE : It is used to delete the data inside the table

Not the table structure/schema.

```
TRUNCATE TABLE table_name;
TRUNCATE TABLE Emp;
```

DROP View : It is used to delete view.

```
DROP VIEW view_name;
DROP VIEW viewzo;
```

4.5 DATA MODIFICATION LANGUAGE STATEMENTS IN SQL

INSERT Command :

Once a table is created, the most natural thing to do is load this table with data to be manipulated later.

Syntax : INSERT INTO table_name(column_name1, column_name2...) VALUES (expression, expression);

Example :

INSERT INTO Employee(E-ID, ENAME, ADDRESS, CITY,STATE, PINCODE) VALUES (115, 'VIJAY', 'C-6 Gupta Road', 'New Delhi', 'DELHI');

Another method: INSERT INTO Employee VALUES(& E-ID, '& NAME', '& ADDRESS', '& CITY', '& STATE');

Note :

(i) The character or varchar expression must be enclosed in single quotes (').

(ii) In the insert into SQL statement the columns and values have a one to one relationship.

Inserting Data into a Table from another Table

Syntax : INSERT INTO table_name SELECT column_name, column_name, FROM table_name;

Example : Insert into table student1 from the table student;

INSERT INTO student 1 SELECT Roll-No, Name, Address, Sex, City, Ph-No, State FROM Student;

Insertion of a Data Set into a Table from another Table

Syntax : INSERT INTO table_name SELECT column_name, column_name FROM table_ name WHERE Column = Expression;

Example : Insert records into the table student 1 from the table student where the field Roll-No contains the value '115';

INSERT INTO Student1 SELECT Roll - No, Name, Address, Sex, City, Ph-No, State FROM Student WHERE Roll-No = '115';

DELETE Command :

The DELETE commands deletes rows from the table that satisfies the condition provided by its WHERE clause, and returns the number of records deleted.

The Verb DELETE in SQL is used to remove rows from table. To remove

- All the rows from a table.

OR

- A select set of rows from a table.

Removal of All Rows :

Syntax : DELETE FROM table_name;

Example : Delete all rows from the table student

DELETE FROM Student;

Removal of a Specified Rows :

Syntax : DELETE FROM table_name WHERE search condition;

Example : Delete rows from the table student where the Roll-No> 115.

DELETE FROM student where Roll-No> 115;

UPDATE Command : Updating the Contents of a Table: The UPDATE command is used to change or modify data values in a table.

To update :

- All the rows from a table.

OR

- A select set of rows from a table.

Updating of All Rows :

Syntax : UPDATE table_name SET column_name = expression, column_name = expression;

Example : Give every employee a bonus of 10%. Update the values held in the column net-salary.

UPDATE Employee SET Netsal = net-salary + Basic salary * 0.10;

Updating Records Conditionally :

Syntax : UPDATE table_name SET column_name = expression, column_name = expression

WHERE column_name = expression;

Example : Update the table student change, the contents of the field name to 'Vijay Krishna' and the contents of field Address to 'Hyderabad' for the record identified by the field Roll-No containing the value 115;

UPDATE student SET name = 'Krishna', Address = 'Hyderabad'

WHERE Roll-No = 115;

4.6 BASIC QUERIES WITH EXAMPLES IN SQL

CREATE TABLE COMMAND :

Syntax : CREATE TABLE table_name(column_name datatype (size), column_name data type (size), column_name datatype (size));

EXAMPLE - 1

Create a Employee Table.

CREATE TABLE Employee (E-ID number (6), ENAME char (15), ADDRESS varchar (15), CITY char (15), STATE char (15), PINCODE number (6));

Output : Table Created

EXAMPLE - 2

Create a Student table.

CREATE TABLE Student (Roll-No number (15), Name char (15), Address varchar (15), Sex char (2), City char (15), Phone number (15), State char (1));

Output : Table Created.

INSERTION OF DATA INTO TABLES :

Once a table is created, the most natural thing to do is load this table with data to be manipulated later.

Syntax : INSERT INTO table_name(column_name1, column_name2 ...) VALUES (expression, expression);

EXAMPLE - 1

INSERT INTO Employee (E-ID, ENAME, ADDRESS,CITY,STATE, PINCODE) VALUES (115, 'VIJAY', 'C-6 Gupta Road', 'New Delhi', 'DELHI');

Another method: INSERT INTO Employee VALUES(& E-ID, '&NAME', '&ADDRESS', '& CITY', '& STATE');

Note :

(i) The character or varchar expression must be enclosed in single quotes (').

(ii) In the insert into SQL statement the columns and values have a one to one relationship.

SELECT COMMAND :

Once data has been inserted into a table, the next most logical operation would be to view what has been entered. This is achieved by SELECT SQL Verb.

(i) View global table data the syntax is :

SELECT * FROM table_name;

e.g., SELECT * FROM Employee;

(ii) Retrieve ID, name, city of the employee

SELECT E-ID, ENAME, CITY FROM Employee;

Selected Columns and Selected Rows:

WHERE Clause

Syntax : SELECT * FROM table_name

WHERE search condition;

(i) SELECT * FROM Employee WHERE E-ID > 112;

(ii) SELECT Roll-No, Name FROM Student WHERE Roll No. < = 150;

Elimination of Duplicates from the Select Statement

A table could contain duplicate rows. We can eliminate using select statement.

Syntax : SELECT DISTINCT column_name1, column_name2 FROM tablename;

Syntax : SELECT DISTINCT * FROM table_name

EXAMPLE - 1

Select only unique rows from the table student :

SELECT DISTINCT * FROM Student;

SORTING DATA IN A TABLE :

Oracle allows data from a table to be viewed in a sorted order. The rows retrieved from the table will be sorted in either ascending or descending order depending on the

condition specified in the select statement.

Syntax : SELECT * FROM table_name ORDER BY column_name1, column_name2 [sortorder]; .

EXAMPLE - 1

Retrieve all rows from student and display this data sorted on the value contained in the field Roll-No. in ascending order;

SELECT * FROM Student ORDER BY Roll-No;

Note : Oracle engine sorts in ascending order by default.

EXAMPLE - 2

For viewing the data in descending sorted order the word desc.

SELECT * FROM Student ORDER BY Roll-No desc;

CREATING A TABLE FROM A TABLE :

Syntax : CREATE TABLE table_name [(column_name, column_name)] AS SELECT column name, columnname FROM Tablename;

EXAMPLE - 1

Create a table student1 from student.

CREATE TABLE Student1(SRoll-No, SName, Address, Sex, City, Ph.No, State)

AS SELECT Roll-No, Name, Address, Sex, City, Ph. No., State FROM Student;

INSERTING DATA INTO A TABLE_FROM ANOTHER TABLE :

Syntax : INSERT INTO table_name SELECT column_name, column_name, FROM table name;

EXAMPLE - 1

Insert into table student1 from the table student;

INSERT INTO student1 SELECT Roll-No, Name, Address, Sex, City, Ph-No, State FROM Student;

INSERTION OF A DATA SET INTO A TABLE FROM ANOTHER TABLE :

Syntax : INSERT INTO table_name

SELECT column name, column name FROM table name WHERE Column = Expression;

EXAMPLE - 1

Insert records into the table student1 from the table student where the field Roll-No contains the value '115';

INSERT INTO Student1 SELECT Roll-No, Name, Address, Sex, City, Ph-No, State FROM Student WHERE Roll-No = '115';

DELETE OPERATIONS :

The DELETE commands deletes rows from the table that satisfies the condition provided by its WHERE clause, and returns the number of records deleted.

The Verb DELETE in SQL is used to remove rows from table. To remove

- All the rows from a table.

OR

- A select set of rows from a table.

Removal of All Rows :

Syntax : DELETE FROM table_name;

EXAMPLE - 1

Delete all rows from the table student

DELETE FROM Student;

Removal of a Specified Rows :

Syntax : DELETE FROM table_name WHERE search condition;

EXAMPLE - 2

Delete rows from the table student where the Roll-No> 115.

DELETE FROM student where Roll-No> 115;

UPDATE COMMAND :

Updating the Contents of a Table : The UPDATE command is used to change or modify data values in a table.

To update :

- All the rows from a table.

OR

- A select set of rows from a table.

Updating of All Rows :

Syntax : UPDATE table_name SET column_name = expression, column_name = expression;

EXAMPLE - 1

Give every employee a bonus of 10%. Update the values held in the column net-salary.

UPDATE Employee SET Netsal = net-salary + Basic salary * 0.10;

Updating Records Conditionally :

Syntax : UPDATE table_name SET column_name = expression, column_name = expression WHERE column_name = expression;

EXAMPLE - 2

Update the table student change, the contents of the field name to 'Vijay Krishna' and the contents of field Address to 'Hyderabad' for the record identified by the field Roll-No containing the value 115;

UPDATE student SET name = 'Krishna', Address = 'Hyderabad' WHERE Roll-No = 115;

MODIFYING THE STRUCTURE OF TABLES :

Adding New Columns :

Syntax : ALTER TABLE table_name

ADD (new_column_name data type/size, new_column_name data type (size ...);

EXAMPLE - 1

Add the field Fax which is a field that can hold number upto 15 digits in length and Mobile-No, which is a field that can hold a number upto 10 digits in length.

ALTER TABLE student

ADD (Mobile-No number (10), Fax number (15));

Modifying Existing Columns :

Syntax : ALTER TABLE table_name

MODIFY (column_name new data type (New size));

EXAMPLE - 2

Modify the field fax of the table student to now hold maximum of 25 character values.

ALTER TABLE student MODIFY (Fax Varchar (25));

Limitation of the ALTER TABLE : Using the ALTER TABLE clause the following tasks cannot be performed:

- Change the name of the table.
- Change the name of the column.
- Drop a column.
- Decrease the size of a column if table data exists.

RENAMING COMMAND :

To rename a table, the syntax is :

Syntax : RENAME old_table_name to new_table_name

EXAMPLE - 1

Rename the table Employee to Employee 1;

RENAME Employee TO Employee1;

DESTROYING TABLES :

Syntax : DROP TABLE table_name;

EXAMPLE - 1

Destroy the table Employee and all the data held in it;

DROP TABLE Employee;

DESCRIBE COMMAND :

To find information about the column defined in the table uses the following syntax;

Syntax : DESCRIBE table name;

This command displays the column names, the data types and the special attributes connected to the table.

EXAMPLE - 1

Displays the columns and their attributes of the table student.

DESCRIBE Student;

QUERIES USING LOGICAL OPERATORS :

There are following logical operators used in SQL.

- **The AND Operator :** The oracle engine will process all rows in a table and display the result only when all of the conditions specified using the AND operator are satisfied.

EXAMPLE - 1

Retrieve the contents of the columns product-no, profit-percent, sell-price from the table product-master where the values contained in the field profit percent in between 10 and 20.

SELECT Product-no, profit-percent, sell-price

FROM Product-master

WHERE profit-percent> = 10 AND profit-percent < = 20;

- **The OR Operator :** The oracle engine will process all rows in a table and diplay the result only when any of the conditions specified using the OR operator are satisfied

EXAMPLE - 2

Retrieve the all fields of the table student where the field Roll-No has the value 115 OR 200;

SELECT Roll-No, Name, Address, Sex, City, Ph-No,

State FROM Student WHERE (Roll-No = 115 OR Roll-No = 200);

- The NOT Operator: The oracle engine will process all rows in a table and display the result only when none of the conditions specified using the NOT operator are satisfied.

EXAMPLE - 3

Retrieve specified student information for the clients, who are NOT in 'New-Delhi' OR 'Noida';

SELECT Roll-No, Name, Address, City, State

FROM Student WHERE NOT (City = 'New Delhi' OR City = 'Noida');

Range Searching

BETWEEN Operator :

EXAMPLE - 4

Retrieve Roll-No, Name, Address, Ph-No, State from the table student where the values contained within the field Roll-No is between 100 and 200 both inclusive.

SELECT Roll-No, Name, Address, Ph-No, State FROM Student

WHERE Roll-No BETWEEN 100 AND 200;

Pattern Matching : The use of the LIKE predicate: The LIKE predicate allows for a comparison of one string value with another string value, which is not identical.

For the character data types :

The percent sign (%) matches any string.

The underscore (_) matches any single character.

EXAMPLE - 5

1. Retrieve all information about students whose names begins with the letters 'vi' from student table.

   SELECT * FROM Student

   WHERE Name LIKE 'vi %';

2. Retrieve all information about students where the second character of names are either 'V' (or) 'S'.

   SELECT * FROM Student

   WHERE Name LIKE '_V%' OR

   Name LIKE '_S%';

THE IN PREDICATES :

In case of value needs to be compared to a list of values then the IN predicate is used.

EXAMPLE - 1

Retrieve the Roll-No, Name, Address, City, Ph-No from the table student where name is either Vijay or Santosh or Gopal or Sanjay.

SELECT Roll-No, Name, Address, City, Ph-No

FROM Student WHERE Name IN ('Vijay', 'Santosh', 'Gopal', 'Sanjay');

The NOT IN prediates: The NOT IN predicate is the opposite of the IN predicate. This will select all the rows where values do not match all of the values in the list.

EXAMPLE - 2

SELECT Roll-No, Name, Address, City, Ph-No FROM Student

WHERE Name NOT IN ('Vijay', 'Santosh', 'Gopal', 'Sanjay');

4.7 OPERATORS IN SQL

There are different types of operators are available in SQL,

1. **Arithmetic Operator :**

   Arithmetic operators are +, –, *, /, %, **

2. **Logical Operator :**

   Logical operator are AND, OR, NOT

3. **Comparison Operator :**

   Comparison operator are <, >, =, ≥, ≤

4. **Between Operator :**

   Between – – AND

5. **NOT Between Operator :**

   NOT between – – AND

6. **Partial Equality :**

   IN, NOT, IS, NULL, IS NOT NULL

7. **Set Operator :**

   Union, Intersect

1. **Examples on Arithmetic Operators :**

   > Select 2 + 3;

2+3
5

   > Select 4 + 3;

4+3
7

   > Select 15 – 5;

15–5
10

> Select 5 – 2;

5–2
3

> Select 5 * 2; (or) Select 2 * 5;

5*2
10

2*5
10

> Select 6/3 (or) Select 2/2;

6/3
2

2/2
1

2. Logical Operators Examples :

> Select 1 && 1;

1&&1
1

> Select 1 && 0;

1&&0
0

> Select 1 && NULL;

1&&NULL
NULL

> Select 0 && NULL;

0&&NULL
0

> Select NULL && 0;

NULL&&0
0

> Select 1 || 0;

1||0
1

> Select 0 || 0;

0||0
0

> Select 1 || NULL;

1||NULL
1

> Select 0 || NULL;

0||NULL
NULL

3. **SQL SET Operators :** The SQL SET operators are used to combine the two or more SQL SELECT statements.

Types of Set Operation :

1. Union
2. UnionAll
3. Intersect
4. Minus.

(i) **Union :**

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

Syntax :

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

Example :

The First table

ID	Name
1	Jack
2	Harry
3	Jackson

The Second table

ID	Name
3	Jackson
4	Stephen
5	David

Union SQL query will be :

```
SELECT * FROM First
UNION
```

SELECT * FROM Second;

The resultset table will look like :

ID	Name
1	Jack
2	Harry
3	Jackson
4	Stephen
5	David

(ii) **Union All :** Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax :

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

Example :

Using the above First and Second table.

Union All query will be like:

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

The resultset table will look like :

ID	Name
1	Jack
2	Harry
3	Jackson
3	Jackson
4	Stephen
5	David

(iii) **Intersect :**

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

Syntax :

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example :

Using the above First and Second table

Intersect query will be:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

The resultset table will look like :

ID	Name
3	Jackson

(iv) **Minus :**

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

Syntax :

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

Example :

Using the above First and Second table.

Minus query will be :

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

The resultset table will look like :

ID	Name
1	Jack
2	Harry

4. **SQL BETWEEN Operator :**

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax :

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

5. **SQL IN Operator :**

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax :

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
(or)
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

6. SQL IS NULL and IS NOT NULL Operators :

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

4.8 PROCESS OF SPECIFYING CONSTRAINTS IN SQL

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into the following two types :

1. Column level constraints: Limits only column data.
2. Table level constraints: Limits whole table data.

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- PRIMARY KEY
- CHECK
- UNIQUE
- FOREIGN KEY
- DEFAULT

NOT NULL Constraint : NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

UNIQUE Constraint : UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. This constraint can be applied at column level or table level.

Primary Key Constraint : Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Foreign Key Constraint : FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables :

CHECK Constraint : CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. It's like condition checking before saving data into a column.

SQL Constraints : SQL constraints are used to specifiy rules for the data in a table. Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the date in the table.

- Constraints can be columnlevel or table level.

The following constraints are commonly used in SQL

(i) NOT NULL (ii) UNIQUE

(iii) PRIMARY KEY (iv) FOREIGN KEY

(v) CHECK (vi) DEFAULT.

SQL NOT NULL : By default, a column can hold null values. The NOTNULL constraint enforces a column to not accept null values. This enforces a field to always contain a value.

EXAMPLE - 1

```
> CREATE TABLE Hero(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  fname varchar(30) NOT NULL,
  age int);
```

SQL UNIQUE CONSTRAINT : It ensures that all values in a column are different. Both unique and primary key constraints provide a guarantee for uniqueness for a column or set of columns. However you can have many unique constraints for table.

EXAMPLE - 2

Oracle :

```
> CREATE TABLE Hero(
```

```
ID  int  NOT NULL  UNIQUE,
lname  varchar(30)  NOT NULL,
fname  varchar(30)  NOT NULL,
age int);
```

EXAMPLE - 3

MySQL :

```
> CREATE  TABLE  Hero(
  ID  int  NOT NULL,
  lname  varchar(30)  NOT NULL,
  fname  varchar(30)  NOT NULL,
  age  int,
  UNIQUE(ID);
```

Defining a unique constraint on multiple columns;

Syntax :

```
Constraint  Constraintname
```

```
> CREATE  TABLE  Hero(
  ID  int  NOT NULL,
  lname  varchar(30)  NOT NULL,
  fname  varchar(30)  NOT NULL,
  age  int,
    CONSTRAINT  UK  UNIQUE (ID, lname));
```

Creating unique key when table is already created

```
> ALTER  TABLE  student/hero
  ADD  UNIQUE (pin);
```

Syntax :

```
Alter  table  tablename
ADD  constraintname (columnname);
```

Defining a unique key on multiple column when table is already created

```
> ALTER  TABLE  student
  ADD
  CONSTRAINT  UK  UNIQUE (pin, phone no);
```

Deleting unique constraint :

```
> ALTER TABLE Student/Hero
  DROP
  UNIQUE UK;
```

SQL PRIMARY KEY CONSTRAINT :

It uniquely identify each record in a database table. Primary key must contain unique values, cannot contain null values.

EXAMPLE - 1

MySQL :

```
> CREATE TABLE Hero(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  age int,
  PRIMARY KEY (ID));
```

EXAMPLE - 2

Oracle :

```
> CREATE TABLE Hero(
  ID int NOT NULL PRIMARY KEY,
  lname varchar(30) NOT NULL,
  fname varchar(30) NOT NULL,
  age int);
```

Defining a primary key constraint on multiple column :

```
> CREATE TABLE Hero(
  ID int NOT NULL,
  lname varchar(30) NOT NULL,
  fname varchar(30) NOT NULL,
  age int,
  CONSTRAINT PK PRIMARY KEY(ID, lname);
```

Creating Primary key when table is already created :

```
> ALTER TABLE STUDENT/Hero
  ADD PRIMARY KEY(pin);
```

Defining creating primary key when table is already created on multiple columns:

```
> ALTER TABLE Student/Hero
  ADD
  CONSTRAINT PR PRIMARY KEY(pin, phoneno);
```

Deleting a primary key :

```
> AUTER TABLE Student/Hero
  Drop
  PRIMARY KEY PR;
```

FOREIGN KEY CONSTRAINT :

A foreign key is a key used to link two tables together. A foreign is a field or collection of fields in one table that refers to the primary key in another table.

The table containing the foreign key is called child table, and the table containing the candidate key is called referenced (or) parent table.

EXAMPLE - 1

Hero (id, lname, fname, age)

Heroine (hid, phone no, id)

MySQL :

```
> CREATE TABLE Heroine(
  hid int NOT NULL,
  phoneno bigint NOT NULL,
  id int,
  PRIMARY KEY(hid),
  FOREIGN KEY(ID) REFERENCES Hero(id));
```

Oracle :

```
> CREATE TABLE Heroine(
  hid int NOT NULL PRIMARY KEY,
  Phoneno bigint NOT NULL,
  id int FOREIGN KEY REFERENCES
  Hero(id));
```

Defining a Foreign key constraint on multiple columns :

```
> CREATE TABLE Heroinc(
```

```
hid int NOT NULL,
Phoneno bigint NOT NULL,
id int, PRIMARYKEY(hid),
CONSTRAINT FK FOREIGNKEY (id)
REFERENCES Hero(id));
> ALTER TABLE Heroine
ADD FOREIGN KEY(id) REFERENCES Hero(id);
```

Defining foreign key constraint on multiple columns when table is already created :

```
> ALTER TABLE Heroine
ADD CONSTRAINT FK FOREIGN KEY(id)
REFERENCES Hero(id);
```

Deleting a foreign key constraint :

```
> ALTER TABLE Heroine
DROP FOREIGN KEY FK;
```

SQL CHECK CONSTRAINT :

The check constraint is used to limit the value range that can be placed in a column. If you define a check constraint on a single column it allows only certain value for that column.

EXAMPLE - 1

MySQL :

```
> CREATE TABLE Vilan(
ID int NOT NULL,
lname varchar(30) NOT NULL,
fname varchar(30), age int,
CHECK (age > = 18));
```

EXAMPLE - 2

Oracle :

```
> CREATE TABLE Vilan(
```

```
ID  int  NOT NULL,
lname  varchar(30)  NOT NULL,
fname  varchar(30),
age  int  check (age > = 18));
```

Defining a check constraint on multiple columns :

```
> CREATE  TABLE  Vilan(
ID  int  NOT NULL,
lname  varchar(30)  NOT NULL,
fname  varchar(30),
age int,
city  varchar(30),
Constraint CHK CHECK (age > = 18 AND City = 'Hyderabad'));
```

Creating a check constraint when table is already created :

EXAMPLE - 3

```
> ALTER  TABLE  Vilan
  ADD  CHECK (age > = 18);
```

Defining a check constraint on multiple columns when table is already created

```
> ALTER  TABLE  Vilan  ADD
  CONSTRAINT  CHK  CHECK (age > = 18 AND city = 'Hyderabad');
```

Deleting a check constriant :

```
> ALTER  TABLE  Vilan
  DROP  CHECK  CHK; /DROP  CONSTRAINT  CHK;
```

DEFAULT CONSTARAINT :

The default constraint is used to provide a default value for a column. The default value will be added to all new records if no other value is specified.

EXAMPLE - 1

```
> CREATE  TABLE  Comedian(
  ID  int  NOT NULL,
  lname  varchar(30) NOT NULL,
```

```
fname  varchar(30),
age int,
city varchar(30) DEFAULT 'Hyderabad');
```

The default constraint can also be used to insert system values by using functions.

EXAMPLE - 2

```
> CREATE  TABLE  Comedian(
  ID  int  NOT NULL,
  lname  varchar(30)  NOT NULL,
  fname  varchar(30),
  age  int,
 city  varchar(30)  DEFAULT 'Hyderabad';
 dob  date  DEFAULT GETDATE( ));
```

Creating a default constraint when table is already created :

MySQL :

```
> ALTER  TABLE  Comedian
  ALTER  City  SET DEFAULT 'Hyderabad';
```

Oracle :

```
> ALTER  TABLE  Comedian
  MODIFY  city  DEFAULT 'Hyderabad';
```

Deleting a default constraint :

MySQL :

```
> ALTER  TABLE  Comedian
  ALTER  city  DROP  DEFAULT;
```

Oracle :

```
> ALTER  TABLE  Comedian
  ALTER  COLUMN  city  DROP  DEFAULT;
```

4.9 SQL FUNCTIONS

Functions are methods used to perform operations. SQL has many buil-in functions.

They are five types of SQL functions :

(i) Numeric functions.

(ii) Aggregate functions.

(iii) Scalar functions.

(iv) Date functions.

(v) String functions.

(i) **Numeric Function :** These are used primarly for numeric manipulation and mathematical calculation.

1. **ABS :** This function returns absolute value of (x).

   ABS (x)

   Ex : > Select ABS (– 2);

ABS (–2)
2

   Ex : ABS (x)

   > Select ABS(2);

ABS (2)
2

2. **CEIL (x) :** This function returns the smallest integer value and it is not smaller than x.

   Ex : > Select CEIL (3.4);

CEIL (3.4)
4

3. **FLOOR (x) :** This function returns the largest value.

   Ex : > Select FLOOR (3.6);

FLOOR (3.6)
3

4. **COS(x) :** It returns the cosine of x.

The value of x is given is radiants.

Ex : Select cos(90);

cos (90)
0

5. **GREATEST (n_1, n_2 .....) :** It returns the greatest value in set of input parameters.

Ex : > Select GREATEST (1, 10, 20, 5);

GREATEST (1,10,20,5)
20

6. **LEAST (n_1, n_2 .....) :** It returns the least value in set of input parameters.

Ex : > Select LEAST (1, 10, 20, 5);

LEAST (1,10,20,5)
1

7. **MOD (N, M) :** It returns the remainder of N divided by M.

Ex : > Select MOD (29, 3);

MOD (29,3)
2

8. **Power (x, y) :** It returns (x, y) value of x to the power of y.

Ex : > Select POWER (3, 3);

Power (3,3)
27

9. **Round (x, y) :** This function returns round-nearest integer and the second argument in this function returns x rounded to y decimal place.

Ex : > Select ROUND (5.7);

ROUND(5.7)
6

Ex : > Select Round (5.68145, 2);

Round (5.68145,2)
5.68

10. **PI :** It simply returns the value of Pi.

Ex : > Select PI ( );

PI
3.141593

11. **SQRT (x) :** It returns the non negative square root of x.

Ex : > Select SQRT (49);

SQRT (49)
7

12. **TRUNCATE (X, D) :** It is same as round as (x, y).

Ex : Select Truncate (7.53678, 3);

TRUNCATE (7.53678,3)
7.536

POWER :

Syntax : POWER (m, n)

Rcturns 'm' raised to 'nth' power 'n' must be an integer, else an error is returned.

Example-1 :

SELECT POWER (3,2) "RESULT =" FROM math;

Output : RESULT = 9

ABS :

Syntax : ABS (n)

Returns the absolute value of 'n'

Example-2

SELECT ABS (-10) "Absolute =" FROM math;

Output : Absolute = 10

(ii) **Aggregate SQL Functions :**The Aggregate Functions in SQL perform calculations on a group of values and then return a single value.

Following are a few of the most commonly used Aggregate Functions :

S.No.	Function	Description
1	SUM()	Used to return the sum of a group of values.
2.	COUNT()	Returns the number of rows either based on a condition, or without a condition.
3.	AVG()	Used to calculate the average value of a numeric column.
4.	MIN()	This function returns the minimum value of a column.
5.	MAX()	Returns a maximum value of a column.
6.	FIRST()	Used to return the first value of the column.
7.	LAST()	This function returns the last value of the column.

Let us look into each one of the above functions in depth. For your better understanding, we will be considering the following table to explain to you all the examples.

Student ID	Student Name	Marks
1	Sanjay	64
2.	Varun	75
3.	Akash	45
4.	Rohit	86
5.	Anjali	92

1. **SUM() :** Used to return a total sum of numeric column which you choose.

Syntax :

```
SELECT SUM(ColumnName)
FROM TableName;
```

EXAMPLE -1

Write a query to retrieve the sum of marks of all students from the Students table.

```
SELECT SUM(Marks)
FROM Students;
```

Output : 359

2. **COUNT() :** Returns the number of rows present in the table either based on some condition or without any condition.

Syntax :

```
SELECT COUNT(ColumnName)
FROM TableName
WHERE Condition;
```

EXAMPLE-1

Write a query to count the number of students from the Students table.

```
SELECT COUNT(StudentID)
FROM Students;
```

Output :

```
5
```

EXAMPLE-2

Write a query to count the number of students scoring marks > 75 from the Students table.

```
SELECT COUNT(StudentID)
FROM Students
WHERE Marks >75;
```

Output :

```
2
```

3. AVG() : This function is used to return the average value of a numeric column.

Syntax :

```
SELECT AVG(ColumnName)
FROM TableName;
```

EXAMPLE-1

Write a query to calculate the average marks of all students from the Students table.

```
SELECT AVG(Marks)
FROM Students;
```

Output :

```
71.8
```

4. **MIN()** : Used to return the minimum value of a numeric column.

Syntax :

```
SELECT MIN(ColumnName)
FROM TableName;
```

EXAMPLE-1

Write a query to retrieve the minimum marks out of all students from the Students table.

```
SELECT MIN(Marks)
FROM Students;
```

Output:

64

5. **MAX()** : Returns the maximum value of a numeric column.

Syntax :

```
SELECT MAX(ColumnName)
FROM TableName;
```

EXAMPLE-1

Write a query to retrieve the maximum marks out of all students from the Students table.

```
SELECT MAX(Marks)
FROM Students;
```

Output :

92

6. **FIRST()** :This function returns the first value of the column which you choose.

Syntax :

```
SELECT FIRST(ColumnName)
FROM TableName;
```

EXAMPLE-1

Write a query to retrieve the marks of the first student.

```
SELECT FIRST(Marks)
FROM Students;
```

Output :

64

7. **LAST() :** Used to return the last value of the column which you choose.

Syntax :

```
SELECT LAST(ColumnName)
FROM TableName;
```

EXAMPLE:

Write a query to retrieve the marks of the last student.

```
SELECT LAST(Marks)
FROM Students;
```

Output :

92

(iii) **Scalar Functions :**The Scalar Functions in SQL are used to return a single value from the given input value.

Following are a few of the most commonly used Scalar Functions:

S.No.	Function	Description
1	LCASE()	Used to convert string column values to lowercase
2.	UCASE()	This function is used to convert a string column values to Uppercase.
3.	LEN()	Returns the length of the text values in the column.
4.	MID()	Extracts substrings in SQL from column values having String data type.
5.	ROUND()	Rounds off a numeric value to the nearest integer.
6.	NOW()	This function is used to return the current system date and time.
7.	FORMAT()	Used to format how a field must be displayed.

1. **LCASE()** : Used to convert values of a string column to lowercase characters.

Syntax :

```
SELECT LCASE(ColumnName)
FROM TableName;
```

EXAMPLE:

Write a query to retrieve the names of all students in lowercase.

```
SELECT LCASE(StudentName)
FROM Students;
```

Output :

sanjay
varun
akash
rohit
anjali

2. **UCASE()** : Used to convert values of a string column to uppercase characters.

Syntax :

```
SELECT UCASE(ColumnName)
FROM TableName;
```

EXAMPLE :

Write a query to retrieve the names of all students in lowercase.

```
SELECT UCASE(StudentName)
FROM Students;
```

Output :

SANJAY
VARUN
AKASH
ROHIT
ANJALI

3. **LEN()** : Used to retrieve the length of the input string.

Syntax :

SELECT LENGTH(String) AS SampleColumn;

EXAMPLE :

Write a query to extract the length of the student name "Sanjay".

SELECT LENGTH("Sanjay") AS StudentNameLen;

Output :

6

4. **MID()** : This function is used to extract substrings from columns having string data type.

Syntax :

```
SELECT MID(ColumnName, Start, Length)
FROM TableName;
```

EXAMPLE:

Write a query to extract substrings from the StudentName column.

SELECT MID(StudentName, 2, 3)

FROM Students;

Output :

anj
aru
kas
ohi
nja

5. **ROUND()** : This function is used to round off a numeric value to the nearest integer.

Syntax :

```
SELECT ROUND(ColumnName, Decimals)
FROM TableName;
```

EXAMPLE :

For this example, let us consider the following Marks table in the Students table.

StudentID	StudentName	Marks
1	Sanjay	90.76
2	Varun	80.45
3	Akash	54.32
4	Rohit	72.89
5	Anjali	67.66

Write a query to round the marks to the integer value.

```
SELECT ROUND(Marks)
FROM Students;
```

Output :

91
80
54
73
68

6. **NOW() :** Used to return the current date and time. The date and time are returned in the "YYYY-MM-DD HH-MM-SS" format.

Syntax:

```
SELECT NOW();
```

EXAMPLE :

Write a query to retrieve the current date and time.

```
SELECT NOW();
```

Output :

NOW()
2022-05-05 09:16:36

7. **FORMAT() :** This function formats the way a field must be displayed.

Syntax:

FORMAT(InputValue, Format)

EXAMPLE:

Write a query to display the numbers "123456789" in the format "###-###-###"

SELECT FORMAT(123456789, "###-###-###");

Output :

123-456-789

(iv) **Date Functions :**

1. **CURDATE :** It returns current date as a value in YYY-MM-DD format.

   Ex : > Select CURDATE ( );

CURDATE( )
2019-05-04

2. **CURTIME :** It returns the current time as a value in HH:MM:SS

   Ex : > Select CURTIME ( );

CURTIME( )
11:05:30

3. **DAY OF YEAR :** It returns the day of the year

   Ex : > Select DAYOFYEAR ('2018 - 07 - 15');

DAYOFYEAR('2018-07-15')
196

(So range is upto 365)

4. **HOUR (TIME) :** It returns the hour for time.

   The range of the return value is 0 to 23 how ever the range of time values actually much large so hour can return vale > 23.

   Ex : > Select HOUR ('11:05:30');

HOUR('11:05:30')
11

5. **MONTH (DATE) :** It returns the month for date in the rage 1 to 12 for January to december.

Ex : > Select MONTH ('2014 - 07 - 30');

MONTH('2014-07-30')
07

6. **MINUTE (TIME) :** It returns the minute from the time in the range 0 to 59.

Ex : > Select MINUTE ('2014 - 07 - 30 11:15:42');

MINUTE
15

7. **NOW :** It returns the current date and time as the value in YYY-MM-DD, HH:MM:SS.

Ex : > Select NOW ( );

NOW( )
2014-07-31, 09 : 45 : 50

8. **SYSDATE :** Same as above function.

Ex : > Select SYSDATE ( );

SYSDATE( )
2014-07-31, 09 : 46 : 50

9. **TIMESTAMP :** It returns the date or date time expression.

TIMESTAMP (exp1, exp2)

Here expr is a date time value which as two arguments expr1, expr2, which is used to add the time and return the result as the date time value.

Ex : Select TIMESTAMP ('2014 - 07 - 31');

TIMESTAMP
2014-07-31, 00 : 00 : 00

Ex : > Select TIMESTAMP ('2014 - 12 - 31', '12:00:00')

TIMESTAMP
2015-01-01, 00 : 00 : 00

10. **YEAR :** It returns the current year.

Ex : > Select YEAR ('2014 - 07 - 31');

YEAR('2014-07-31')
2014

(v) **String Functions :**

1. **LENGTH (String) :** It returns number of characters in a string.

Ex : > Select LENGTH ('vani');

LENGTH('vani')
4

2. **LOWER (String) :** It converts string to lower case.

Ex : > Select LOWER ('SWETHA')

LOWER('Swetha')
Swetha

3. **UPPER (String) :** It converts into upper case.

Ex : Select UPPER ('Swetha');

UPPER('Swetha')
SWETHA

4. **INITCAP (String) :** It converts first character to capital letter.

Ex : > Select INITCAP ('Swetha')

INITCAP('swetha')
Swetha

5. **REVERSE (String) :** It returns the string in reverse order.

Ex : > Select REVERSE ('Shivani');

REVERSE('Shivani')
inavihS

6. **STRCMP (String1, String2) :** It compare two strings and returns 0 if both strings are equal, it returns-1 if first argument is smaller than second According to the current sort order otherwise it returns 1.

Ex : > Select STRCMP ('Rani', 'Raju');

STRCMP('Rani','Raju')
1

Ex : > Select STRCMP ('Kaif', 'Katrina');

STRCMP('Kaif','Katrina')
–1

Ex : > Select STRCMP ('Anushka', 'Arina');

STRCMP('Anushka','Arina')
–1

7. SUBSTRING :

(a) SUBSTRING (String, POS).

(b) SUBSTRING (String, from POS).

(c) SUBTRING (String, POS, LEN).

(a) SUBSTRING (String, POS) : The first function will return substring from the given string starting at position (POS).

Ex : > Select SUBSTRING ('bhavani', 4);

SUBSTRING('bhavani', 4)
Vani

(b) SUBSTRING (String, from POS) :

Ex : > Select SUBSTRING ('bhavani', From 4);

SUBSTRING('bhavani', from 4)
Vani

(c) SUBSTRING (String, POS, Len) :

Ex : > Select SUBSTRING ('bharat', 2, 6);

SUBSTRING('bharat', 2, 6)
harat

8. **TRIM :** It returns the string without prefix or suffix. It remove spaces.

Ex : SELECT TRIM (' Raju');

TRIM('Raju')
Raju

(a) **LTRIM :** It removes the left prefix (or) Space from the given string.

Ex : > Select LTRIM (' Raju')

LTRIM(Raju)
Raju

(b) **RTRIM :** It removes the right space from the given string.

Ex : > Select RTRIM ('Raju ')

RTRIM('Raju')
Raju

9. **CONCAT (strin1, string2) :** It returns the string that results from CONCATINATING (adding) the arguments.

Ex : > Select CONCAT ('kajol', 'agarwal');

CONCAT('kajol','agarwal')
Kajolagarwal

10. **INSERT (string, POS, len, new string) :** It returns a new string by adding a string from position and length to the given string.

Ex : > Select INSERT ('bharat', 4, 6, van)

INSERT(Bharat,4,6,'van')
Bhavan

> Select INSERT ('Bharat', 2, 4, 'indu');

INSERT('Bharat',2,4,'indu')
bindut

11. **REPLACE (string from string to string) :**

Ex : > Select REPLACE ('Vani', 'V', 'R');

REPLACE('Vani','V','R')
Rani

12. **LPAD (string, len, pad string) :** Padding (adding or pumping) the string onleft.

Ex : > Select LPAD ('hi', 4, '$$');

LPAD('hi',4,$$)
$$hi

Ex : > Select LPAD ('hi', 3, '$$');

LPAD('hi',3,'$$')
$hi

Ex : > Select LPAD ('hi', 2, '&&');

LPAD('hi',2,'& &')
hi

Ex : > Select LPAD ('hi', 6, '@@');

LPAD('hi',6,'@@')
@@@@hi

13. **RPAD (string, length, padstring) :** Padding (adding or pumping) the string on right.

Ex : > Select RPAD ('hi', 4, '$$')

RPAD('hi',4,'$$')
hi$$

REVIEW QUESTIONS

One Mark Questions :

1. What do you mean by SQL?
2. List any two DDL commands.
3. List any two DML commands.
4. List any two DCL commands.
5. Write the syntax of CREATE command.
6. Write the syntax of INSERT command.
7. Write the syntax of DELETE command.
8. Write the syntax of UPDATE command.
9. What is NULL column?
10. What is pseudo column?
11. List any two data types used in SQL.
12. List any two operators in SQL.
13. What is IN operator?
14. Define constraints in SQL.
15. What is primary key?
16. What is foreign key?
17. Define function.
18. Define query.
19. List any two categories of SQL functions.

Three Mark Questions :

1. List the features of SQL.
2. List the benefits of SQL.
3. List the components of SQL.
4. Give some pseudo columns in SQL.
5. Describe various data types in SQL.

6. Write about arithmetic operators in SQL.
7. Write about logical operators in SQL.
8. List SQL SET operators.
9. Write about BETWEEN operator.
10. Write about IS NULL, IS NOT NULL operators.
11. Write different types of constraints in SQL.
12. List any three numeric functions.
13. List any three aggregate functions.
14. List any three scalar functions.
15. List any three date functions.
16. List any three string functions.

Five Mark Questions :

1. Explain DDL statements in SQL with one example.
2. Explain DML statements in SQL with one example.
3. Illustrate the process of creating any three constraints with examples.
4. Illustrate any five operators in SQL.
5. Explain any five numeric functions each with an example.
6. Explain any five aggregate functions each with an example.
7. Explain any five date functions each with an example.
8. Explain any five string functions each with an example.