UNIT 2

RELATIONAL MODEL : Introduction to relational model, concepts of domain, attribute, tuple, relation, importance of null values, constraints (Domain, Key constraints, integrity constraints) and their importance.

BASIC SQL : Simple Database schema, data types, table definitions (create, alter), different DML operations (insert, delete, update), basic SQL querying (select and project) using where clause, arithmetic & logical operations, SQL functions(Date and Time, Numeric, String conversion).

INTRODUCTION TO RELATIONAL MODEL:

The relational model represents the database as a collection of *relations* also called tables. In the relational model, each row in the table represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help in interpreting the meaning of the values in each row. A row in a table represents the relationship among a set of values. A table of a relational database is an entity set and row or tuple is an entity.

In the formal relational model terminology, a row is called a *tuple,* a column header is called an *attribute,* and the table is called a *relation.* The data type describing the types of values that can appear in each column is represented by a *domain* of possible values.

CONCEPTS OF DOMAIN, ATTRIBUTE, TUPLE, RELATION:

Domain: A domain D is a set of atomic values (each value in the domain is indivisible). A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain, to help in interpreting its values.

Example:

Names: The set of character strings that represent names of persons.

Employee_ages: Possible ages of employees of a company; each must be a value between 15 and 80 years old.

Aadhar_number: The set of valid twelve-digit social security numbers.

Attribute: A relation schema R, denoted by R(A_1, A_2, ... , An) is made up of a relation name R and a list of attributes A_1, A_2, ..., An. Each attribute Ai is the name of a role played by some domain D in the relation schema R. D is called the domain of A_i and is denoted by dom(A). A relation schema is used to *describe* a relation; R is called the name of this relation.

The degree (or arity) of a relation is the number of attributes n of its relation schema.

Tuple: A relation (or relation state) r of the relation schema R(A_1, A_2, ... , An) also denoted by *r(R),* is a set of n-tuples r = $\{t_1,t_2,t_3...t_n\}$. Each n-tuple *t* is an ordered list of *n* values $t = <v_1,v_2,v_3...v_n>$, where each value V_i $1 \leq i \leq n$, is an element of dom(A) or is a special **null** value. The i^{th} value in tuple *t,* which corresponds to the attribute A_i is referred to as t[A_i] (or t[i] if we use the positional notation). The terms **relation intension** for the schema R and **relation extension** for a relation state *r(R)* are also commonly used.

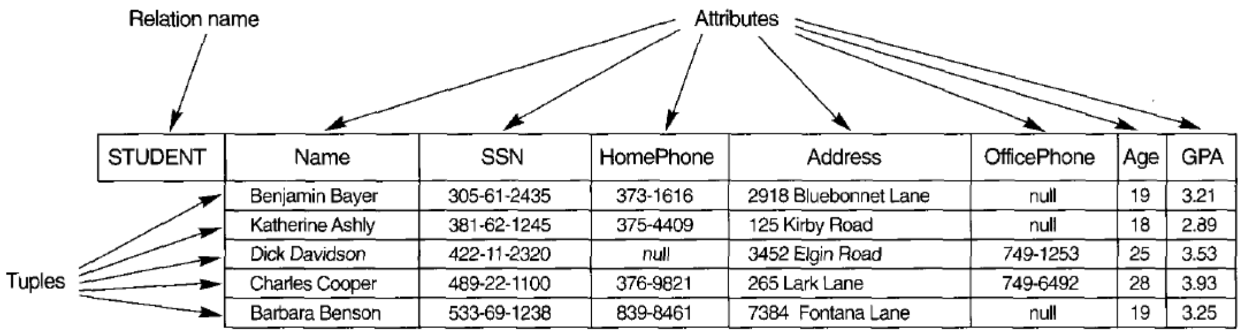
Relation: A (named) set of tuples all of the same form (i.e., having the same set of attributes). The term **table** is a loose synonym.

<u>Relational Schema</u>: used for describing (the structure of) a relation. E.g., R(A1, A2, ..., An) says that R is a relation with *attributes* A1, ... An. The **degree** of a relation is the number of attributes it has, here *n*.

Example: STUDENT(Name, SSN, Address)

<u>Relational Database</u>: A collection of **relations**, each one consistent with its specified relational schema.

Diagram shows an example of a STUDENT relation. Each tuple in the relation represents a particular student entity.



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Attributes and tuples of STUDENT Relation

We display the relation as a table, where each tuple is shown as a *row* and each attribute corresponds to a *column header* indicating a role or interpretation of the values in that column. *Null values* represent attributes whose values are unknown or do not exist for some individual STUDENT tuple.

Mathematically a relation can be defined as- A relation (or relation state) *r(R)* is a mathematical relation of degree n on the domains dom(A1) , dom(A_2), ... , dom(A_n), which is a subset of the Cartesian product of the domains that define R:

$$r(R) \subseteq (\text{dom}(A1) \text{ X } \text{dom}(Az) \text{ X } \ldots \text{ X } \text{dom}(An))$$

The Cartesian product specifies all possible combinations of values from the underlying domains. Hence, if we denote the total number of values, or cardinality, in a domain D by ID I (assuming that all domains are finite), the total number of tuples in the Cartesian product is $|\text{dom}(A1)| \text{ X } |\text{dom}(Az)| \text{ X } \ldots \text{ X } |\text{dom}(An)|$

Of all these possible combinations, a relation state at a given time-the current relation state-reflects only the valid tuples that represent a particular state of the real world.

<u>Characteristics of Relations:</u>

Ordering of Tuples: A relation is a *set* of tuples; hence, there is no order associated with them. That is, it makes no sense to refer to, for example, the 5th tuple in a relation. When a relation is depicted as a table, the tuples are necessarily listed in *some* order, of course, but you should attach no significance to that order. Similarly, when tuples are represented on a storage device, they must be organized in *some* fashion, and it may be advantageous, from a performance standpoint, to organize them in a way that depends upon their content.

Ordering of Attributes: A tuple is best viewed as a mapping from its attributes (i.e., the names we give to the roles played by the values comprising the tuple) to the corresponding values. Hence, the order in which the attributes are listed in a table is irrelevant. (Note that, unfortunately, the set theoretic operations in relational algebra (at least how E&N define them) make implicit use of the order of the attributes. Hence, E&N view attributes as being arranged as a sequence rather than a set.)

Values of Attributes: For a relation to be in *First Normal Form*, each of its attribute domains must consist of atomic (neither composite nor multi-valued) values. Much of the theory underlying the relational model was based upon this assumption. The **Null** value: used for *don't know*, *not applicable*.

Interpretation of a Relation: Each relation can be viewed as a **predicate** and each tuple in that relation can be viewed as an assertion for which that predicate is satisfied (i.e., has value **true**) for the combination of values in it. In other words, each tuple represents a fact. Some relations represent facts about entities (e.g., students) whereas others represent facts about relationships (between entities). (e.g., students and course sections).

Relational Model Notation:

R(A1, A2, ..., An) is a relational schema of degree *n* denoting that there is a relation *R* having as its attributes *A1, A2, ..., An*.

By convention,

- *Q*, *R*, and *S* denote relation names.
- *q*, *r*, and *s* denote relation states. For example, *r(R)* denotes one possible state of relation *R*. If *R* is understood from context, this could be written, more simply, as *r*.
- *t*, *u*, and *v* denote tuples.
- The "dot notation" *R.A* (e.g., STUDENT.Name) is used to qualify an attribute name, usually for the purpose of distinguishing it from a same-named attribute in a different relation(e.g., DEPARTMENT.Name).

CONSTRAINTS AND DATABASE SCHEMAS:

Constraints on databases can be categorized as follows:

Inherent Model-Based: Constraints that are inherent in the data model. characteristics of the relations are nothing but the inherent constraints. Example: no two tuples in a relation can be duplicates (because a relation is a set of tuples)

Application-Based: Constraints that *cannot* be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs. These are more general and are difficult to express and enforce within the data model, so they are usually checked within application programs.

Schema-Based: Constraints that can be directly expressed in the schemas of the data model, typically by specifying them in the DDL. These are again classified as:

- Domain Constraints
- Key Constraints
- Constraints on nulls
- Entity Integrity Constraints
- Referential Integrity Constraints

1. Domain Constraints:

Domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A). Each attribute value must be either **null** or drawn from the domain of that attribute.

2. Key Constraints:

A **key constraint** is a statement that a certain *minimal* subset of the fields of a relation is a unique identifier for a tuple. A set of fields that uniquely identifies a tuple according to a key constraint is called a **candidate key** for the relation; we often abbreviate this to just *key*. There are two parts to the definition of candidate key:

1. Two distinct tuples in a legal instance (an instance that satisfies all ICs, including the key constraint) cannot have identical values in all the fields of a key.
2. No subset of the set of fields in a key is a unique identifier for a tuple.

Every relation is guaranteed to have a key. Since a relation is a set of tuples, the set of *all* fields is always a superkey. If other constraints hold, some subset of the fields may form a key, but if not, the set of all fields is a key.

A relation may have several candidate keys. For example, the *login* and *age* fields of the Students relation may, taken together, also identify students uniquely. That is, *{login, age}* is also a key. It may seem that *login* is a key, since no two rows in the example instance have the same *login* value. However, the key must identify tuples uniquely in all possible legal instances of the relation. By stating that *{login, age}* is a key, the user is declaring that two students may have the same login or age, but not both.

Out of all the available candidate keys, a database designer can identify a **primary** key. Intuitively, a tuple can be referred to from elsewhere in the database by storing the values of its primary key fields. For example, we can refer to a Students tuple by storing its *sid* value. As a consequence of referring to student tuples in this manner, tuples are frequently accessed by specifying their *sid* value.

Primary key indirectly says that the set of attributes do not have duplicate and null values in them.

Specifying Key Constraints in SQL:

In SQL, we can declare that a subset of the columns of a table constitute a key by using the UNIQUE constraint. At most one of these candidate keys can be declared to be a *primary key,* using the PRIMARY KEY constraint.

```
CREATE TABLE Students
(
sid CHAR(20) ,
name CHAR (30) ,
login CHAR(20) ,
age INTEGER,
gpa REAL,
UNIQUE (name, age),
CONSTRAINT StudentsKey PRIMARY KEY (sid)
);
```

This definition says that *sid* is the primary key and the combination of *name* and *age* is also a key. The definition of the primary key also illustrates how we can name a constraint by preceding it with CONSTRAINT *constraint-name.* If the constraint is violated, the constraint name is returned and can be used to identify the error.

Foreign Key Constraints:

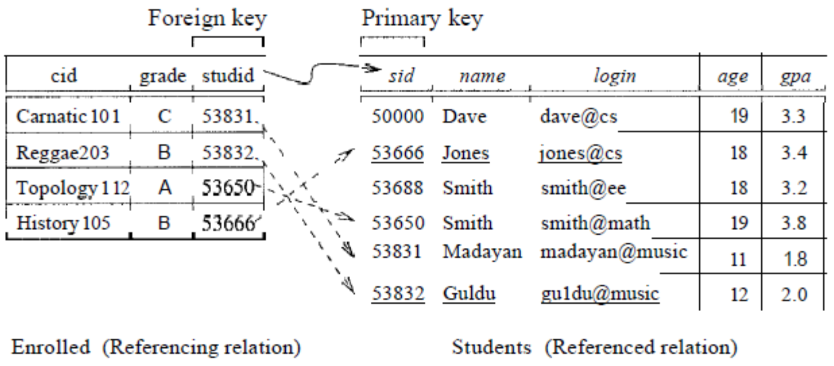
Sometimes the information stored in a relation is linked to the information stored in another relation. If one of the relations is modified, the other must be checked, and perhaps modified, to keep the data consistent.

An IC involving both relations must be specified if a DBMS is to make such checks. The most common IC involving two relations is a *foreign key* constraint.

Suppose that, in addition to Students, we have a second relation:

Enrolled(studid: string, *cid:* string, *gTade:* string)

To ensure that only bona fide students can enroll in courses, any value that appears in the *studid* field of an instance of the Enrolled relation should also appear in the *sid* field of some tuple in the Students relation. The *studid* field of Enrolled is called a foreign key and refers to Students. The foreign key in referencing relation (Enrolled) must have the same no. of columns and compatible datatypes, although the columns names can be different.



Foreign key
Primary key
Enrolled (Referencing relation)
Students (Referenced relation)

cid	grade	studid
Carnatic101	C	53831
Reggae203	B	53832
Topology112	A	53650
History105	B	53666

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	gu1du@music	12	2.0

Fig: referential Integrity

If we try to insert the tuple (55555, *Artl04, A)* into *E1,* the Ie is violated because there is no tuple in 51 with *sid* 55555; the database system should reject such an insertion.

Similarly, if we delete the tuple (53666, *Jones, jones@cs, 18,*3.4) from 51, we violate the foreign key constraint because the tuple (53666, *Historyl05, B)* in *El* contains *studid* value 53666, the *sid* of the deleted Students tuple. The DBMS should disallow the deletion or, perhaps, also delete the Enrolled tuple that refers to the deleted Students tuple.

Specifying Foreign Key Constraints in SQL:

Let us define Enrolled(studid: string, *cid:* string, *grade:* string):

```
CREATE TABLE Enrolled
(
studid CHAR(20) ,
cid CHAR(20),
grade CHAR(10),
PRIMARY KEY (studid, cid),
FOREIGN KEY (studid) REFERENCES Students
);
```

The foreign key constraint states that every *studid* value in Enrolled must also appear in Students, that is, *studid* in Enrolled is a foreign key referencing Students. Specifically, every *studid* value in Enrolled must appear as the value in the primary key field, *sid,* of Students. Incidentally, the primary key constraint for Enrolled states that a student has exactly one grade for each course he or she is enrolled in.

3. Constraints on nulls:

The null value is a member of all domains, and as a result is a legal value for every attribute in SQL by default. For certain attributes, however, null values may be inappropriate. Consider a tuple in the *student* relation where *name* is *null*. Such a tuple gives student information for an unknown student; thus, it does not contain useful information. Similarly, we would not want the department budget to be *null*. In cases such as this, we wish to forbid null values, and we can do so by restricting the domain of the attributes *name* and *budget* to exclude null values, by declaring it as follows:

name **varchar**(20) **not null**

budget **numeric**(12,2) **not null**

The **not null** specification prohibits the insertion of a null value for the attribute. Any database modification that would cause a null to be inserted in an attribute declared to be **not null** generates an error diagnostic.

4. Entity Integrity Constraints:

Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database.

Examples of integrity constraints are:

• An instructor name cannot be *null*.

• No two instructors can have the same instructor ID.

• Every department name in the *course* relation must have a matching department name in the *department* relation.

• The budget of a department must be greater than $0.00.

Unique Constraint:

SQL also supports an integrity constraint:

unique (*Aj*1 , *Aj*2, . . . , *Ajm*)

The unique specification says that attributes *Aj*1 , *Aj*2, . . . , *Ajm* form a candidate key; that is, no two tuples in the relation can be equal on all the listed attributes.

Example:

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

The relation Persons doesn't allow duplicate values for the attribute ID.

Check Constraint:

When applied to a relation declaration, the clause **check**(*P*) specifies a predicate *P* that must be satisfied by every tuple in a relation. A common use of the **check** clause is to ensure that attribute values satisfy specified conditions, in effect creating a powerful type system. For instance, a clause **check** (*budget* > 0) in the **create table** command for relation *department* would ensure that the value of **budget** is nonnegative.

Example:

```
CREATE TABLE Persons
(
ID int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int CHECK (Age>=18)
);
```

The above kind of Person relation doesn't allow the tuples with age <18.

5. Referential Integrity Constraints:

Ensuring that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. This condition is called **referential integrity**.

let $r1$ and $r2$ be relations whose set of attributes are $R1$ and $R2$, respectively, with primary keys $K1$ and $K2$. We say that a subset α of $R2$ is a **foreign key** referencing $K1$ in relation $r1$ if it is required that, for every tuple $t2$ in $r2$, there must be a tuple $t1$ in $r1$ such that $t1.K1 = t2.\alpha$. Requirements of this form are called **referential-integrity constraints**, or **subset dependencies**.

For a referential-integrity constraint to make sense, α and $K1$ must be compatible sets of attributes; that is, either α must be equal to $K1$, or they must contain the same number of attributes, and the types of corresponding attributes must be compatible. A referential integrity constraint does not require $K1$ to be a primary key of $r1$; as a result, more than one tuple in $r1$ can have the same value for attributes $K1$.

When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation (that is, the transaction performing the update action is rolled back). However, a **foreign key** clause can specify that if a delete or update action on the referenced relation violates the constraint, then, instead of rejecting the action, the system must take steps to change the tuple in the referencing relation to restore the constraint like cascade on delete or update.

BASIC SQL

SQL is widely popular because it offers the following advantages −

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed 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.

History of SQL

- **1970** − Dr. Edgar F. "Ted" Codd of IBM is known as the father of relational databases. He described a relational model for databases.
- **1974** − Structured Query Language appeared.
- **1978** − IBM worked to develop Codd's ideas and released a product named System/R.
- **1986** − IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software which later came to be known as Oracle.

SQL Process

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

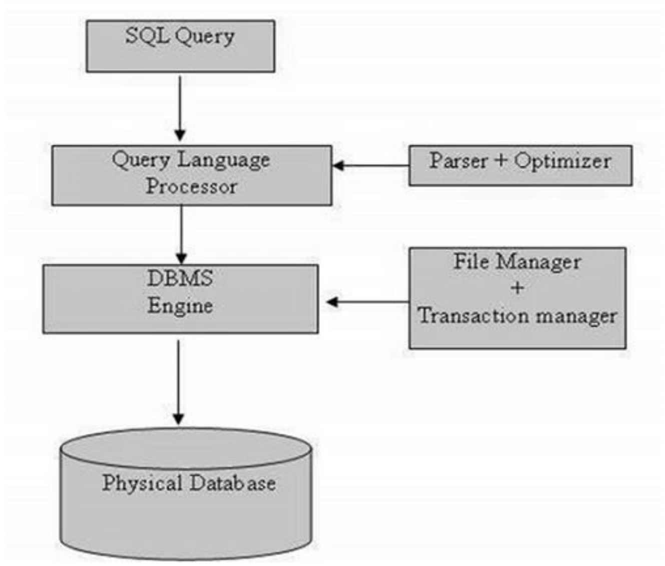
There are various components included in this process.

These components are −

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

A classic query engine handles all the non-SQL queries, but a SQL query engine won't handle logical files.

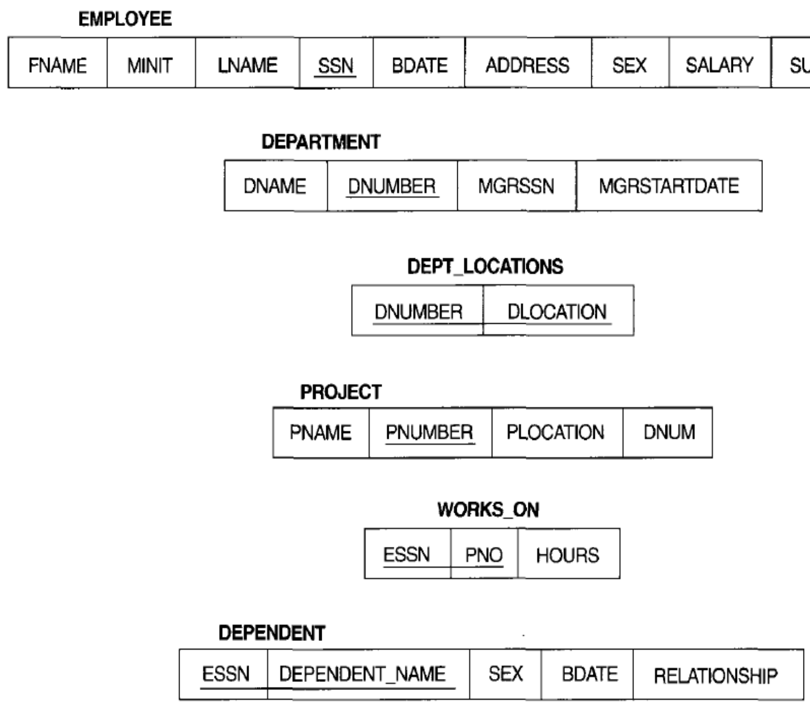
Following is a simple diagram showing the SQL Architecture −



RELATIONAL DATABASES AND RELATIONAL DATABASE SCHEMAS:

A relational database schema S is a set of relation schemas S = (R_1 , R_2, ... , *Rm}* and a set of integrity constraints IC. A relational database state'" DB of S is a set of relation states DB = *{r_1, r_2, ... , r_m }* such that each rj is a state of R_i, and such that the r_i relation states satisfy the integrity constraints specified in IC.

Bellow figure shows a relational database schema that we call COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}. The underlined attributes represent primary keys.


EMPLOYEE
FNAME MINIT LNAME SSN BDATE ADDRESS SEX SALARY SUPERSSN DNO
DEPARTMENT
DNAME DNUMBER MGRSSN MGRSTARTDATE
DEPT_LOCATIONS
DNUMBER DLOCATION
PROJECT
PNAME PNUMBER PLOCATION DNUM
WORKS_ON
ESSN PNO HOURS
DEPENDENT
ESSN DEPENDENT_NAME SEX BDATE RELATIONSHIP

Bellow figure shows a relational database state corresponding to the COMPANY schema.

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
		Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

A database state that does not obey all the integrity constraints is called an invalid state, and a state that satisfies all the constraints in IC is called a valid state. Attributes that represent the same real-world concept mayor may not have identical names in different relations. Alternatively, attributes that represent different concepts may have the same name in different relations.

Each relational DBMS must have a data definition language (DOL) for defining a relational database schema. Current relational DBMSs are mostly using SQL for this purpose. Integrity constraints are specified on a database schema and are expected to hold on every valid database state of that schema. In addition to domain, key, and NOT NULL constraints, two other types of constraints are considered part of the relational model: entity integrity and referential integrity.

SQL DATA TYPES:

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. You can use these data types while creating your tables. You can choose a data type for a table column based on your requirement. SQL Server offers six categories of data types for use which are listed below −

Exact Numeric Data Types:

Data Type	From	To
Bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38+1	10^38-1
numeric	-10^38+1	10^38-1
Money	-922,337,203,685,477.5808	922,337,203,685,477.5807
smallmoney	-214,748.3648	214,748.3647

Approximate Numeric Data Types:

Data Type	From	To
Float	-1.79E+308	1.79E+308
Real	-3.40E+38	3.40E+38

Date and Time Data Types:

Data Type	From	To
Datetime	Jan 1,1753	Dec 31,9999
smalldatetime	Jan 1, 1900	June 6, 2079
Date	Stores a date like June 30,1991	
Time	Stores a time of day like 12:30 P.M.	

Note − Here, datetime has 3.33 milliseconds accuracy where as smalldatetime has 1 minute accuracy.

Character Strings Data Types:

Char: Maximum length of 8,000 characters.(Fixed length non-Unicode characters)

varchar: Maximum of 8,000 characters.(Variable-length non-Unicode data).

varchar(max): Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).

text: Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types:

Nchar: Maximum length of 4,000 characters.(Fixed length Unicode)

Nvarchar: Maximum length of 4,000 characters.(Variable length Unicode)

nvarchar(max): Maximum length of 231characters (SQL Server 2005 only).(Variable length Unicode)

Ntext: Maximum length of 1,073,741,823 characters. (Variable length Unicode)

Binary Data Types:

Binary: Maximum length of 8,000 bytes(Fixed-length binary data)

Varbinary: Maximum length of 8,000 bytes.(Variable length binary data)

varbinary(max): Maximum length of 231 bytes (SQL Server 2005 only). (Variable length Binary data)

Image: Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

Misc Data Types:

sql_variant: Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.

Timestamp: Stores a database-wide unique number that gets updated every time a row gets updated

Uniqueidentifier: Stores a globally unique identifier (GUID)

Xml: Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only).

Cursor: Reference to a cursor object

Table: Stores a result set for later processing

SQL Operations:

There are five types of SQL statements. They are:

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Retrieval Language (DRL) Or Data Query Language (DQL)
4. Transactional Control Language (TCL)
5. Data Control Language (DCL)

1. DATA DEFINITION LANGUAGE (DDL): The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project. Let's take a look at the structure and usage of four basic DDL commands:

1. CREATE
2. ALTER
3. DROP
4. RENAME

1. CREATE:

(a) CREATE TABLE: This is used to create a new relation and the corresponding

Syntax:

```
CREATE TABLE relation_name
(
field_1 data_type(Size),
field_2 data_type(Size),
 .. . );
```

Example:

```
SQL>  CREATE TABLE Student (
          sno NUMBER(3) PRIMARY KEY ,
          sname VARCHAR2(10),
          dob DATE,
          class CHAR(5)
          );
```

Result: Table created.

(b)CREATE TABLE..AS SELECT....: This is used to create the structure of a new relation from the structure of an existing relation.

Syntax:

```
CREATE TABLE relation_name_1 ( field_1,
field_2, .....field_n) AS SELECT field_1,
field_2,...........field_n FROM relation_name_2;
```

Example:

```
SQL> CREATE TABLE std(rno, sname)
AS SELECT sno, sname FROM student;
```

Result: Table Created.

DESC: It is used to describe a schema as well as to retrieve rows from table in descending order.

Syntax: DESC schema_name; /*Describes the schema */

EX: SQL> DESC Student;

```
NAME                NULL?               TYPE
------------------- ------------------- --------------------
SNO                 NOT NULL            NUMBER(3)
SNAME                                   VARCHAR2(10)
DOB                                     DATE
CLASS                                   CHAR(5)
```

2. ALTER:

(a)ALTER TABLE ...ADD...: This is used to add some extra fields into existing relation.

Syntax:

```
ALTER TABLE relation_name ADD (
new_field_1 data_type(size),
new_field_2 data_type(size),
..);
```

Example :

```
SQL>ALTER TABLE Student ADD
        (Address CHAR(10)  );
```

Result: TABLE ALTERED.

/**To check whether column add or not check using desc command**/

```
SQL> DESC Student;
NAME                    NULL?                   TYPE
----------------------- ----------------------- ------------------------
SNO                     NOT NULL                NUMBER(3)
SNAME                                           VARCHAR2(10)
DOB                                             DATE
CLASS                                           CHAR(5)
ADDRESS                                         CHAR(10)
```

(b)ALTER TABLE...MODIFY...: This is used to change the width as well as data type of fields of existing relations.

Syntax:

```
ALTER TABLE relation_name MODIFY (
field_1 newdata_type(Size),
field_2 newdata_type(Size), ....
field_n_newdata_type(Size) );
```

Example:

```
SQL>ALTER TABLE Student MODIFY(
sname VARCHAR2(20),
sno NUMBER(5));
```

Result: TABLE ALTERED.

```
SQL> DESC Student;
NAME                    NULL?                   TYPE
----------------------- ----------------------- ------------------------
SNO                     NOT NULL                NUMBER(5)
SNAME                                           VARCHAR2(20)
DOB                                             DATE
CLASS                                           CHAR(5)
ADDRESS                                         CHAR(10)
```

3. DROP TABLE: This is used to delete the structure of a relation. It permanently deletes the records in the table.

Syntax:	**Example:**
DROP TABLE relation_name;	SQL>DROP TABLE Student; **Result:** Table dropped.

```
SQL> SELECT * FROM TAB;
TNAME                    TABTYPE    CLUSTERID
------------------------- ---------------- -----------------
BONUS                    TABLE
DEPT                     TABLE
EMP                      TABLE
EMPLOYEE1                TABLE
SALGRADE                 TABLE
5 ROWS SELECTED.
```

4. RENAME: It is used to modify the name of the existing database object.

Syntax:	**Example:**
RENAME TABLE old_relation_name TO new_relation_name;	SQL>RENAME TABLE EMP1 TO EMP2; **Result:** Table renamed.

5. TRUNCATE: This command will remove the data permanently. But structure will not be removed.

Syntax: TRUNCATE TABLE <Table name>

Example TRUNCATE TABLE EMP1;

Difference between Truncate & Delete:-

- By using truncate command data will be removed permanently & will not get back whereas by using delete command data will be removed temporally & get back by using roll back command.
- By using delete command data will be removed based on the condition where as by using truncate command there is no condition.
- Truncate is a DDL command & delete is a DML command.

2. DATA MANIPULATION LANGUAGE (DML): The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

1. INSERT
2. UPDATE

3. DELETE

1. INSERT INTO: This is used to add records into a relation. These are three type of INSERT INTO queries. They are:

a) Inserting a single record

Syntax: INSERT INTO relationname (field_1,field_2,.field_n)
VALUES (data_1,data_2,........data_n);

Example: SQL> INSERT INTO student (sno,sname,class,address)
VALUES(1,'satya','5','GNT');

b) Inserting all records from another relation

Syntax: INSERT INTO relation_name_1
SELECT Field_1,field_2,field_n
FROM relation_name_2
WHERE field_x=data;

Example: SQL> INSERT INTO stu1 SELECT sno,sname
FROM student WHERE name = 'satya';

c) Inserting multiple records

Syntax: INSERT INTO relation_name field_1,field_2,.....field_n)
VALUES (&data_1,&data_2,........&data_n);

Example: SQL>INSERT INTO Stu1 (sno,sname,dob,class,address)
VALUES (&sno,&sname,&dob,&class, &address);

Enter value for sno: 101

Enter value for sname: Ramesh

Enter value for dob: 10-10-2010

Enter value for class: 1

Enter value for address: vij

1 row insesrted.

2. UPDATE-SET-WHERE: This is used to update the content of a record in a relation.

Syntax: UPDATE relation_name SET Field_name1=data,field_name2=data,
WHERE field_name=data;

Example: SQL>UPDATE emp1
SET ename = ‘kumar’
WHERE empno=1;

3. DELETE-FROM: This is used to delete all the records of a relation but it will retain the structure of that relation.

a) DELETE-FROM: This is used to delete all the records of relation.

Syntax: DELETE FROM relation_name;

Example: SQL>DELETE FROM Stu1;

b) DELETE -FROM-WHERE: This is used to delete a selected record from a relation.

Syntax: DELETE FROM relation_name WHERE condition;

Example: SQL>DELETE FROM Stu1 WHERE empno = 2;

3. DRL(DATA RETRIEVAL LANGUAGE): Retrieves data from one or more tables.

1. SELECT FROM: To display all fields for all records.

Syntax : SELECT * FROM relation_name;

Example : SQL> select * from dept;

```
DEPTNO          DNAME                   LOC
---------------- -------------------------- ----------
10              ACCOUNTING              NEW YORK
20              RESEARCH                DALLAS
```

Example: SQL> select * from emp1;

EMPNO	ENAME	JOB	DEPTNAME	DEPTNO	HIREDATE	SALARY	EXP	ADDRESS
101	ramesh	asst.prof	It	10	10-DEC-96	20000	2	GNT
102	Ramu	asst.prof	It	10	05-JUL-97	10000	0	Vij
103	Rakesh	asst.prof	It	10	12-AUG-97	5000	0	GNT

2. SELECT FROM: To display a set of fields for all records of relation.

Syntax: SELECT a set of fields FROM relation_name;

Example: SQL> select empno, ename from emp1;

```
EMPNO   ENAME
------------ --------------------
101       ramesh
102       ramu
103       rakesh
```

3. SELECT - FROM -WHERE: This query is used to display a selected set of fields for a selected set of records of a relation.

Syntax: SELECT a set of fields FROM relation_name WHERE condition;

Example: SQL> select * from emp1 where deptno<20;

EMPNO	ENAME	JOB	DEPTNAME	DEPTNO	HIREDATE	SALARY	EXP	ADDRESS
101	ramesh	asst.prof	It	10	10-DEC-96	20000	2	GNT
102	Ramu	asst.prof	It	10	05-JUL-97	10000	0	Vij
103	Rakesh	asst.prof	It	10	12-AUG-97	5000	0	GNT

There are many constructs used for data retrieval like grouping, ordering, aggregation and set operations which can be described in next chapters.

4. TRANSATIONAL CONTROL LANGUAGE (T.C.L):

A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be made permanent to the database only if they are committed a transaction begins with an executable SQL statement & ends explicitly with either rollback or commit statement. The TCL commands are:

1. Commit
2. Savepoint
3. Rollback

1. COMMIT: This command is used to end a transaction only with the help of the commit command transaction changes can be made permanent to the database.

Syntax: SQL>COMMIT;

Example: SQL>COMMIT;

2. SAVE POINT: Save points are like marks to divide a very lengthy transaction to smaller once. They are used to identify a point in a transaction to which we can latter rollback. Thus, save point is used in conjunction with rollback.

Syntax: SQL>SAVE POINT ID;

Example: SQL>SAVE POINT xyz;

3. ROLLBACK: A rollback command is used to undo the current transactions. We can rollback the entire transaction so that all changes made by SQL statements are undo (or) rollback a transaction to a save point so that the SQL statements after the save point are rollback.

Syntax: ROLLBACK(current transaction can be rollback)

ROLLBACK to save point ID;

Example: SQL>ROLLBACK;

SQL>ROLLBACK TO SAVE POINT xyz;

5. DATA CONTROL LANGUAGE (D.C.L):

DCL provides user with privilege commands the owner of database objects (tables), has the soul authority ollas them. The owner (data base administrators) can allow other data base uses to access the objects as per their requirement. DCL commands are:

1.Grant

2.Revoke

1. GRANT: The GRANT command allows granting various privileges to other users and allowing them to perform operations within their privileges.

For Example, if a uses is granted as 'SELECT' privilege then he/she can only view data but cannot perform any other DML operations on the data base object GRANTED privileges can also be withdrawn by the DBA at any time

Syntax: SQL>GRANT PRIVILEGES on object_name To user_name;

Example: SQL>GRANT SELECT, UPDATE on emp1 To hemanth;

2. REVOKE: To with draw the privileges that has been GRANTED to a uses, we use the REVOKE command

Syntax: SQL>REVOKE PRIVILEGES ON object-name FROM user_name;

Example: SQL>REVOKE SELECT, UPDATE ON emp FROM ravi;

CONSTRAINTS:

1. NOT NULL: When a column is defined as NOTNULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

Syntax: CREATE TABLE Table_Name(column_name data_type(*size*) **NOT NULL,**);

Example: SQL> CREATE Table emp2(eno number(5) not null,ename varchar2(10));

```
Table created.

SQL> desc emp2;

Name              Null?          Type
---------------- -----------------   -------------
ENO                NOT NULL   NUMBER(5)
ENAME                         VARCHAR2(10)
```

2. UNIQUE: The purpose of a unique key is to ensure that information in the column(s) is unique i.e. a value entered in column(s) defined in the unique constraint must not be repeated across the column(s). A table may have many unique keys.

Syntax: CREATE TABLE Table_Name(column_name data_type(*size*) **UNIQUE, ….**);

Example: SQL> CREATE Table emp3(eno number(5) unique,ename varchar2(10));

Table created.

```
SQL> desc emp3;
Name                      Null?          Type
------------------------- ------------------ --------------------
ENO                                      NUMBER(5)
ENAME                              VARCHAR2(10)

SQL> insert into emp3 values(&eno,'&ename');
Enter value for eno: 1
Enter value for ename: sss
old 1: insert into emp3 values(&eno,'&ename')
new 1: insert into emp3 values(1,'sss')
1 row created.

SQL> /
Enter value for eno: 1
Enter value for ename: sas
old 1: insert into emp3 values(&eno,'&ename')
new 1: insert into emp3 values(1,'sas')
insert into emp3 values(1,'sas')
ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C003006) violated
```

3. CHECK: Specifies a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null).

Syntax: CREATE TABLE Table_Name(column_name data_type(*size*) **CHECK(*logical expression*), ….**);

Example: CREATE TABLE student (sno **NUMBER (3),** name **CHAR**(**10**),class **CHAR(5),CHECK**(class **IN**('CSE','CAD','VLSI'));

4. PRIMARY KEY: A field which is used to identify a record uniquely. A column or combination of columns can be created as primary key, which can be used as a reference from other tables.

A table contains primary key is known as Master Table.

- It must uniquely identify each record in a table.
- It must contain unique values.
- It cannot be a null field.
- It cannot be multi port field.
- It should contain a minimum no. of fields necessary to be called unique.

Syntax: CREATE TABLE Table_Name(column_name data_type(*size*) **PRIMARY KEY, ….**);

Example: CREATE TABLE faculty (fcode NUMBER(3) PRIMARY KEY, fname CHAR(10));

5. FOREIGN KEY: It is a table level constraint. We cannot add this at column level. To reference any primary key column from other table this constraint can be used. The table in which the foreign key is defined is called a **detail table**. The table that defines the primary key and is referenced by the foreign key is called the **master table**.

Syntax: CREATE TABLE Table_Name (

col_name type(*size*)

FOREIGN KEY(col_name) **REFERENCES** table_name

);

Example:

CREATE TABLE subject (

scode **NUMBER (3) PRIMARY KEY,**

subname **CHAR**(**10**),fcode **NUMBER(3),**

FOREIGN KEY(fcode) **REFERENCE** faculty

);

Defining integrity constraints in the alter table command:

Syntax: ALTER TABLE Table_Name ADD PRIMARY KEY (column_name);

Example: ALTER TABLE student ADD PRIMARY KEY (sno);

(Or)

Syntax: ALTER TABLE table_name ADD CONSTRAINT constraint_name PRIMARY KEY(colname)

Example: ALTER TABLE student ADD CONSTRAINT SN PRIMARY KEY(SNO)

Dropping integrity constraints in the alter table command:

Syntax: ALTER TABLE Table_Name DROP constraint_name;

Example: ALTER TABLE student DROP PRIMARY KEY;

(or)

Syntax: ALTER TABLE student DROP CONSTRAINT constraint_name;

Example: ALTER TABLE student DROP CONSTRAINT SN;

OPERATORS IN SQL:

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operations, such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

SQL Arithmetic Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b will give 30
- Subtraction	Subtracts right hand operand from left hand operand.	a - b will give -10
* Multiplication	Multiplies values on either side of the operator.	a * b will give 200
/ Division	Divides left hand operand by right hand operand.	b / a will give 2
% Modulus	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

SQL Comparison Operators

Assume **'variable a'** holds 10 and **'variable b'** holds 20, then

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	a=b is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	a!=b is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	a<>b is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	a>b is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	a<b is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	a>=b is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	a<=b is true.

!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	a!<b is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	a!>b is true.

SQL Logical Operators

Here is a list of all the logical operators available in SQL.

- **ALL:** The ALL operator is used to compare a value to all values in another value set.
- **AND**: The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
- **ANY:** The ANY operator is used to compare a value to any applicable value in the list as per the condition.
- **BETWEEN**: The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
- **EXISTS**: The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
- **IN**: The IN operator is used to compare a value to a list of literal values that have been specified.
- **LIKE**: The LIKE operator is used to compare a value to similar values using wildcard operators.
- **NOT**: The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
- **OR**: The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
- **IS NULL**: The NULL operator is used to compare a value with a NULL value.
- **UNIQUE**: The UNIQUE operator searches every row of a specified table for uniqueness no duplicates.

DATE FUNCTIONS:

1) Sysdate:

SQL>SELECT SYSDATE FROM DUAL;

18-MAY-17

2) next_day:

SQL>SELECT NEXT_DAY(SYSDATE,'WED')FROM DUAL;

24-MAY-09

3) add_months:

SQL>SELECT ADD_MONTHS(SYSDATE,2)FROM DUAL;

28-JUL-09

4) last_day:

```
SQL>SELECT LAST_DAY(SYSDATE)FROM DUAL;
31-MAY-17
```

5) months_between:

```
SQL>SELECT MONTHS_BETWEEN(SYSDATE,18-AUG-17) FROM EMP;
3
```

6) Least:

```
SQL>SELECT LEAST('10-JAN-07','12-OCT-07')FROM DUAL;
10-JAN-07
```

7) Greatest:

```
SQL>SELECT GREATEST('10-JAN-07','12-OCT-07')FROM DUAL;
10-JAN-07
```

8) Trunc: returns starting day of the week if format specified is 'DAY'

```
SQL>SELECT TRUNC(SYSDATE,'DAY')FROM DUAL;
14-MAY-17
```

9) Round: returns starting day of the next week if format specified is 'DAY'

```
SQL>SELECT ROUND(SYSDATE,'DAY')FROM DUAL;
21-MAY-17
```

10) to_char:

```
SQL> select to_char(sysdate, "dd\mm\yy") from dual;
18-may-17
```

11) to_date:

```
SQL> select to_date(sysdate, "dd\mm\yy") from dual;
18-may-17
```

NUMERIC FUNCTIONS:

- **ABS( )** - Returns the absolute value of numeric expression.
- **ACOS( )** - Returns the arccosine of numeric expression. Returns NULL if the value is not in the range -1 to 1.
- **ASIN( )** - Returns the arcsine of numeric expression. Returns NULL if value is not in the range -1 to 1.
- **ATAN( )** - Returns the arctangent of numeric expression.
- **ATAN2( )** - Returns the arctangent of the two variables passed to it.
- **BIT_AND( )** - Returns the bitwise AND all the bits in expression.
- **BIT_COUNT( )** - Returns the string representation of the binary value passed to it.
- **BIT_OR( )** - Returns the bitwise OR of all the bits in passed expression..
- **CEIL( )** - Returns the smallest integer value that is not less than passed numeric expression.
- **CEILING( )** - Returns the smallest integer value that is not less than passed numeric expression.
- **CONV( )** - Convert numeric expression from one base to another.
- **COS( )** - Returns the cosine of passed numeric expression. The numeric expression should be expressed in radians.
- **COT( )** - Returns the cotangent of passed numeric expression.
- **DEGREES( )** - Returns numeric expression converted from radians to degrees.
- **EXP( )** - Returns the base of the natural logarithm (e) raised to the power of passed numeric expression.
- **FLOOR( )** - Returns the largest integer value that is not greater than passed numeric expression.
- **FORMAT( )** - Returns a numeric expression rounded to a number of decimal places.
- **GREATEST( )** - Returns the largest value of the input expressions.
- **INTERVAL( )** - Takes multiple expressions exp1, exp2 and exp3 so on.. and returns 0 if exp1 is less than exp2, returns 1 if exp1 is less than exp3 and so on.
- **LEAST( )** - Returns the minimum-valued input when given two or more.
- **LOG( )** - Returns the natural logarithm of passed numeric expression.
- **LOG10( )** - Returns the base-10 logarithm of passed numeric expression.
- **MOD( )** - Returns the remainder of one expression by diving by another expression.
- **OCT( )** - Returns the string representation of the octal value of passed numeric expression. Returns NULL if passed value is NULL.
- **PI( )** - Returns the value of pi.
- **POW( )** - Returns the value of one expression raised to the power of another expression.
- **POWER( )** - Returns the value of one expression raised to the power of another expression.
- **RADIANS( )** - Returns the value of passed expression converted from degrees to radians.
- **ROUND( )** - Returns numeric expression rounded to an integer. Can be used to round an expression to a number of decimal points.
- **SIN( )** - Returns the sine of numeric expression given in radians.

- **SQRT( )** - Returns the non-negative square root of numeric expression.
- **STD( )** - Returns the standard deviation of the numeric expression.
- **STDDEV( )** - Returns the standard deviation of the numeric expression.
- **TAN( )** - Returns the tangent of numeric expression expressed in radians.
- **TRUNCATE( )** - Returns numeric exp1 truncated to exp2 decimal places. If exp2 is 0, then the result will have no decimal point.

STRING FUNCTIONS:

1) Concat: CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes

SQL> SELECT CONCAT('ORACLE','CORPORATION')FROM DUAL;

ORACLECORPORATION

2) Lpad: LPAD returns expr1, left-padded to length n characters with the sequence of characters in expr2.

SQL>SELECT LPAD('ORACLE',15,'*')FROM DUAL;

*********ORACLE

3) Rpad: RPAD returns expr1, right-padded to length n characters with expr2, replicated as many times as necessary.

SQL>SELECT RPAD ('ORACLE',15,'*')FROM DUAL;

ORACLE*********

4) Ltrim: Returns a character expression after removing leading blanks.

SQL>SELECT LTRIM('SSMITHSS','S')FROM DUAL;

MITHSS

5) Rtrim: Returns a character string after truncating all trailing blanks.

SQL>SELECT RTRIM('SSMITHSS','S')FROM DUAL;

SSMITH

6) Lower: Returns a character expression after converting uppercase character data to lowercase.

SQL>SELECT LOWER('DBMS')FROM DUAL;

dbms

7) Upper: Returns a character expression with lowercase character data converted to uppercase.

SQL>SELECT UPPER('dbms')FROM DUAL;

DBMS

8) Length: Returns the number of characters, rather than the number of bytes, of the given string expression, excluding trailing blanks.

```
SQL>SELECT LENGTH('DATABASE')FROM DUAL;
8
```

9) Substr: Returns part of a character, binary, text, or image expression.

```
SQL>SELECT SUBSTR('ABCDEFGHIJ'3,4) FROM DUAL;
CDEF
```

10) Instr: The INSTR functions search string for substring. The function returns an integer indicating the position of the character in string that is the first character of this occurrence.

```
SQL>SELECT INSTR('CORPORATE FLOOR','OR',3,2)FROM DUAL;
14
```