UNIT- II

Relational Model

- Introduction to relational model
- Concepts of domain
- Attribute
- Tuples
- Relation
- Importance of null values

The Relational Database: Definitions

- A DBMS that manages data as collection of tables in which all data relationships are represented by common values in related tables."
- "A DBMS that follows all the twelve rules of CODD is called RDBMS".

All information must be represented explicitly in one and only one way: as values in tables and each & every datum in the database must be accessible by specifying a table name, a column name, and a primary key

Relational Model

- **Relational Model (RM)** represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values.
- The Relational Model developed by Dr. E. F. Codd at IBM in the late 1960s.
- The model built on mathematical concepts, which expounded in the famous work called "A Relational Model of Data for Large Shared Databanks ".
- At the core of the relational model is the concept of a table (also called a relation) in which all data is stored.
- Records (horizontal rows also known as tuples) & Fields (vertical columns also known as attributes).
- Table can be identified by a unique name.

A table is a collection of records and each record in a table contains the same fields

Properties of the Relational model

- Data is presented as a collection of relations.
- Each relation is depicted as a table.
- Columns are attributes that belong to the entity modelled by the table (ex. In a student

table, you could have name, address, student ID, major, etc.).

- Each row ("tuple") represents a single entity (ex. In a student table, John , NRT, 12345, Accounting, would represent one student entity).
- Every table has a set of attributes that taken together as a **"key"** (technically, a "superkey")
- Uniquely identifies each entity (Ex. In the student table, “student ID” would uniquely identify each student – no two students would have the same student ID).

Relational Model Concepts

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

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.

Tuple: It is nothing but a single row of a table, which contains a single record.

Relation Schema: A relation schema represents the name of the relation with its attributes.

Degree: The total number of attributes which in the relation is called the degree of the relation.

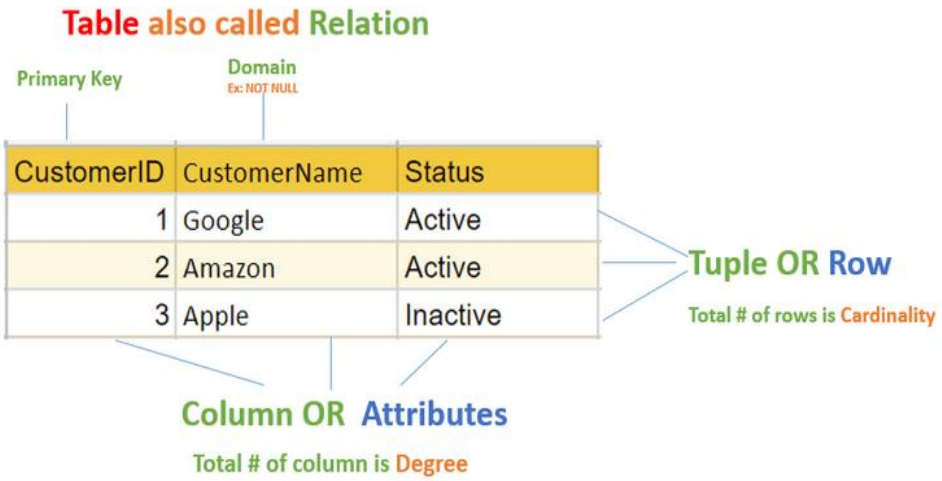
Cardinality: Total number of rows present in the Table.

Column: The column represents the set of values for a specific attribute.

Relation instance – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.

Relation key - Every row has one, two or multiple attributes, which is called relation key.

Attribute domain – Every attribute has some pre-defined value and scope which is known as attribute domain.

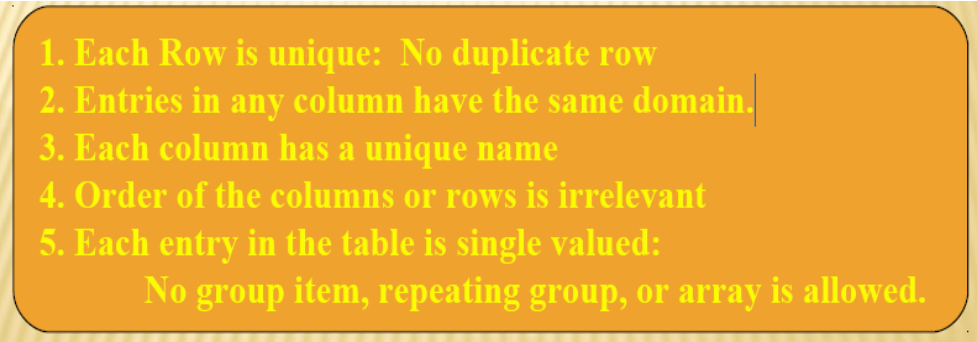


PRINCIPLES OF RELATIONAL MODEL

- Data about various entities and their relationships

are stored in a series of logical tables (also known as relations).

- A relation is a two-dimensional table with certain imposed restrictions:



Relation: made up of 2 parts:

- *Instance* (set of records): a *table*, with rows and columns.
  #Rows = *cardinality*, #fields = *degree / arity.*
    - *Schema* : specifies name of relation, plus name and domain (type) of each field (column).
        - E.G. Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).
- Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Cardinality = 3, degree = 5, all rows distinct

- ❖ Do all columns in a relation instance have to be distinct?

Domain constraints in DBMS

- ❖ A table in DBMS is a set of rows and columns that contain data.
- ❖ Columns in table have a unique name, often referred as attributes in DBMS.
- ❖ A domain is a unique set of values permitted for an attribute in a table.

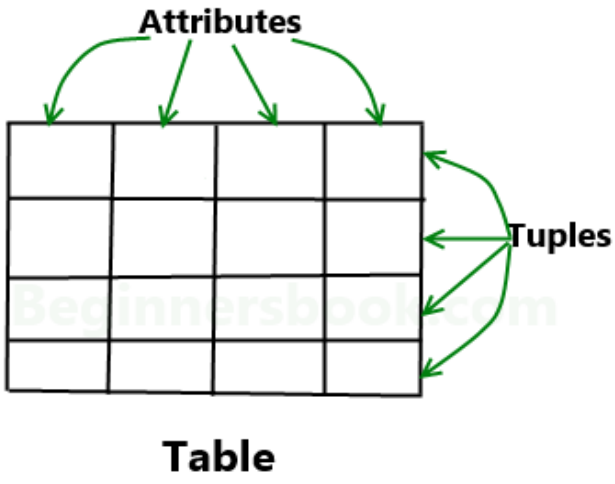
- For example, a domain of month-of-year can accept January, February….December as possible values,
- A domain of integers can accept whole numbers that are negative, positive and zero.
- **Definition:** Domain constraints are **user defined data type** and we can define them like this:
- Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)
- **Example:**
  For example i want to create a table "student_info" with **"stu_id"** field having value greater than **100**, I can create a domain and table like this:
- **create domain id_value int constraint id_test check(value > 100);**
- create table student_info (stu_id id_value PRIMARY KEY, stu_name varchar(30), stu_age int);
- **Another example:**
  I want to create a table "bank_account" with "account_type" field having value either "checking" or "saving":

  create domain **account_type** char(12) constraint acc_type_test check(value in ("Checking", "Saving"));

  create table bank_account (account_nbr int PRIMARY KEY, account_holder_name varchar(30), account_type **account_type**);

Attributes in DBMS

- In RDBMS, a table organizes data in rows and columns. The columns are known as attributes where as the rows are known as records.



Example: A school maintains the data of students in a table named **"student"**. Suppose the data they store in table is student id, student name & student age. To do this they have had three columns in the table: **student_id**, **student_age**, **student_name**. The table looks like this:

student_id	student_age	student_name
101	12	Jon
102	13	Arya
103	12	Sansa

Here student_id, student_age and student_name are the **attributes**

Importance of null values

- A **NULL** value in a table is a value in a field that appears to be blank.
- A field with a **NULL** value is a field with no value.
- It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Syntax

The basic syntax of **NULL** while creating a table.

```
CREATE TABLE CUSTOMERS
(
    ID INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2),
    PRIMAREY (ID)
```

Here, **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns where we did not use NOT NULL, which means these columns could be **NULL**.

A field with a **NULL** value is the one that has been left blank during the record creation.

Example

- The NULL value can cause problems when selecting data.
- However, because when comparing an unknown value to any other value, the result is always unknown and not included in the results.
- You must use the **IS NULL** or **IS NOT NULL** operators to check for a NULL value.
- Consider the following CUSTOMERS table having the records as shown below.

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

Now, following is the usage of the **IS NOT NULL**operator.

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
   FROM CUSTOMERS
   WHERE SALARY IS NOT NULL;
```

This would produce the following result –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
+----+----------+-----+-----------+----------+
```

Now, following is the usage of the **IS NULL** operator.

```
SQL> SELECT  ID, NAME, AGE, ADDRESS, SALARY
   FROM CUSTOMERS
   WHERE SALARY IS NULL;
```

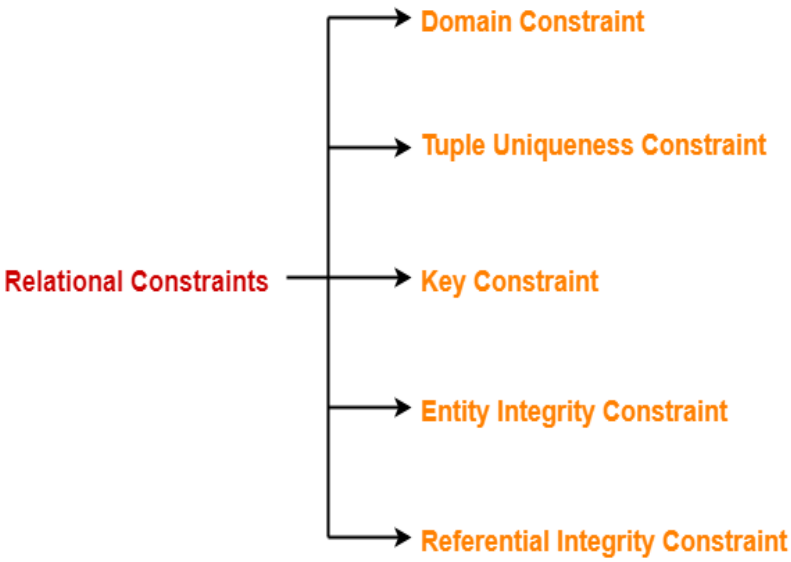
This would produce the following result –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  6 | Komal    |  22 | MP        |          |
|  7 | Muffy    |  24 | Indore    |          |
+----+----------+-----+-----------+----------+
```

Constraints in DBMS

- Constraints or nothing but the rules that are to be followed while entering data into columns of the database table.
- Constraints ensure that data entered by the user into columns must be within the criteria specified by the condition.
- For example, if you want to maintain only unique IDs in the employee table or if you want to enter only age under 18 in the student table etc.

Types of Constraints in DBMS



Domain Constraint

- Domain constraint defines the domain or set of values for an attribute.
- It specifies that the value taken by the attribute must be the atomic value from its domain.
- **Example**- Consider the following Student table

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	A

Here, value **'A'** is not allowed since only integer values can be taken by the age attribute.

Tuple Uniqueness Constraint

- Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation.

Example-01: Consider the following Student table

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
S004	Rahul	20

This relation satisfies the tuple uniqueness constraint since here all the tuples are unique.

Example-02: Consider the following Student table

STU_ID	Name	Age
S001	Akshay	20
S001	Akshay	20
S003	Shashank	20
S004	Rahul	20

This relation does not satisfy the tuple uniqueness constraint since here all the tuples are not unique.

Key Constraints or Uniqueness Constraints

- These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
- A relation can have multiple keys or candidate keys(minimal superkey), out of which we choose one of the keys as primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes.

- Null values are not allowed in the primary key, hence Not Null constraint is also a part of key constraint.

Example- Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S001	Abhishek	21
S003	Shashank	20
S004	Rahul	20

This relation does not satisfy the key constraint as here all the values of primary key are not unique.

Entity Integrity Constraint

- Entity integrity constraint specifies that no attribute of primary key must contain a null value in any relation.
- This is because the presence of null value in the primary key violates the uniqueness property.

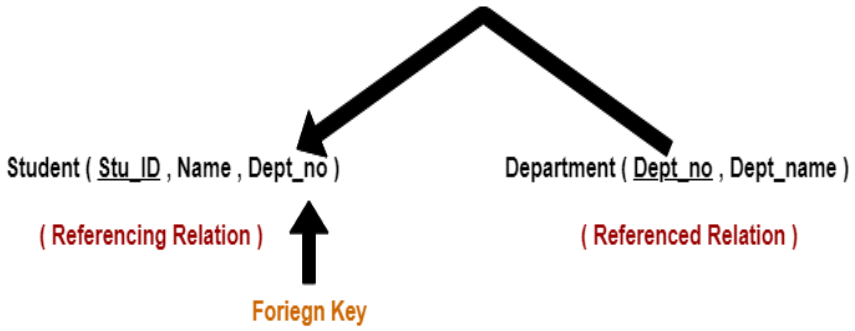
Example- Consider the following Student table-

STU_ID	Name	Age
S001	Akshay	20
S002	Abhishek	21
S003	Shashank	20
	Rahul	20

This relation does not satisfy the entity integrity constraint as here the primary key contains a NULL value.

Referential Integrity Constraints

- The Referential integrity constraints is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.
- This constraint is enforced through foreign key, when an attribute in the foreign key of relation R1 have the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.
- **Example-**
- Consider the following two relations- ‘Student’ and ‘Department’.
- Here, relation ‘Student’ references the relation ‘Department’.



Student

STU_ID	Name	Dept_no
S001	Akshay	D10
S002	Abhishek	D10
S003	Shashank	D11
S004	Rahul	D14

Department

Dept_no	Dept_name
D10	ASET
D11	ALS
D12	ASFL
D13	ASHS

Here,

The relation 'Student' does not satisfy the referential integrity constraint.

This is because in relation 'Department', no value of primary key specifies department no. 14.

Thus, referential integrity constraint is violated.

SQL CREATE TABLE Statement

- The CREATE TABLE statement is used to create a new table in a database.

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

- The column parameters specify the names of the columns of the table.
- The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

CREATE TABLE Example

- The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

Create Table Using Another Table

- A copy of an existing table can also be created using CREATE TABLE.
- The new table gets the same column definitions. All columns or specific columns can be selected.
- If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

Example

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

ALTER TABLE Statement

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

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

- **Example**
- The following SQL adds an "Email" column to the "Customers" table:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

- To delete a column in a table, use the following syntax:

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

Example

The following SQL deletes the "Email" column from the "Customers" table:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

ALTER TABLE - ALTER/MODIFY COLUMN

- To change the data type of a column in a table, use the following syntax:

SQL Server / MS Access:

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

My SQL / Oracle (prior version 10G):

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

Oracle 10G and later:

```
ALTER TABLE table_name
MODIFY column_name datatype;
```

DROP TABLE Statement

- The DROP TABLE statement is used to drop an existing table in a database.
- **Syntax**

```
DROP TABLE table_name;
```

TRUNCATE TABLE

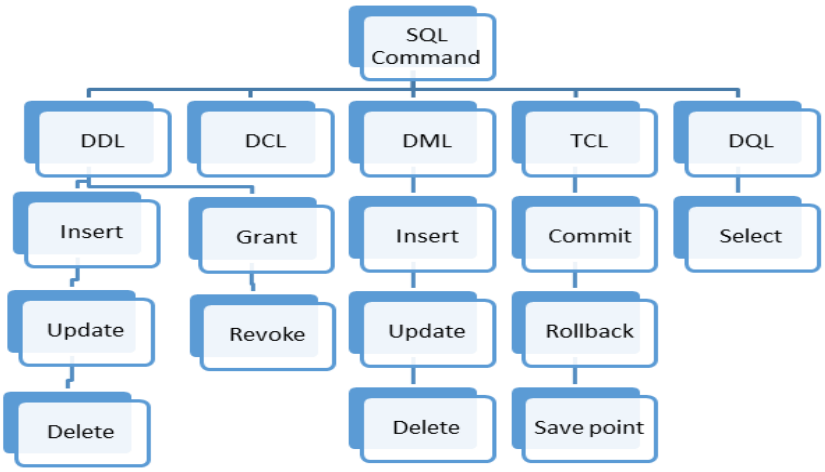
- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.
- **Syntax**

```
TRUNCATE TABLE table_name;
```

Types of SQL

Here are five types of widely used SQL queries.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language(DCL)
- Transaction Control Language(TCL)
- Data Query Language (DQL)



Types of SQL

Different DML operations

- Data Manipulation Language (**DML**) allows you to modify the database instance by inserting, modifying, and deleting its data.
- It is responsible for performing all types of data modification in a database.
- Here are some important DML commands in SQL:
- INSERT

- UPDATE
- DELETE

INSERT

- This is a statement is a SQL query. This command is used to insert data into the row of a table.

Syntax:

```
INSERT INTO TABLE_NAME  (col1, col2, col3,.... col N)
VALUES (value1, value2, value3, .... valueN);
Or
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3, .... valueN);
```

For example:

```
INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', E
richsen');
```

UPDATE

- This command is used to update or modify the value of a column in the table.

Syntax:

```
UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WH
ERE CONDITION]
```

For example:

```
UPDATE students
SET FirstName = 'Jhon', LastName= 'Wick'
WHERE StudID = 3;
```

DELETE

- This command is used to remove one or more rows from a table.

Syntax:

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

For example:

```
delete from Employee where e_id=5;
```

To delete all records from the table –

```
Delete * from <TABLE NAME>;
```

Select Command

- The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax

The basic syntax of the SELECT statement is as follows –

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

```
SELECT * FROM table_name;
```

Example

- Consider the **CUSTOMERS** table having the following records

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

- The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

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

This would produce the following result

```
+-----+-----------+-----------+
|  ID  |  NAME      |  SALARY    |
+-----+-----------+-----------+
|   1  |  Ramesh    |   2000.00  |
|   2  |  Khilan    |   1500.00  |
|   3  |  kaushik   |   2000.00  |
|   4  |  Chaitali  |   6500.00  |
|   5  |  Hardik    |   8500.00  |
|   6  |  Komal     |   4500.00  |
|   7  |  Muffy     |  10000.00  |
+-----+-----------+-----------+
```

- If you want to fetch all the fields of the **CUSTOMERS** table, then you should use the following query.

```
SQL> SELECT * FROM CUSTOMERS;
```

- This would produce the result as shown below.

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

Using **WHERE** clause

- The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables.
- If the given condition is satisfied, then only it returns a specific value from the table.
- You should use the **WHERE** clause to filter the records and fetching only the necessary records.
- The **WHERE** clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc.,
- **Syntax**
- The basic syntax of the **SELECT** statement with the **WHERE** clause is as shown below.

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc

Example

- Consider the **CUSTOMERS** table having the following records

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

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

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

- This would produce the following result.

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

- The following query is an example, which would fetch the ID, Name and Salary fields from the **CUSTOMERS** table for a customer with the name **Hardik**.
- Here, it is important to note that all the strings should be given inside single quotes **(' ')**. Whereas, numeric values should be given without any quote as in the above example.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME = 'Hardik';
```

This would produce the following result.

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  5 | Hardik   |  8500.00 |
+----+----------+----------+
```

What are SQL operators?

- SQL operators are reserved keywords used in the **WHERE** clause of a SQL statement to perform **arithmetic**, **logica**l and **comparison** operations. Operators act as conjunctions in SQL statements to fulfill multiple conditions in a statement.

- There are different types of operators in SQL
- Types of Operators:
    1. Arithmetic Operators
    2. Comparison Operators
    3. Logical Operators

Arithmetic Operators

These operators are used to perform operations such as addition, multiplication, subtraction etc.

Operator	Operation	Description
+	Addition	Add values on either side of the operator
–	Subtraction	Used to subtract the right hand side value from the left hand side value
*	Multiplication	Multiples the values present on each side of the operator
/	Division	Divides the left hand side value by the right hand side value
%	Modulus	Divides the left hand side value by the right hand side value; and returns the remainder

Addition (+) :

- It is used to perform **addition operation** on the data items, items include either single column or multiple columns.

- **Implementation**:

```
SELECT employee_id, employee_name, salary, salary + 100
   AS "salary + 100" FROM addition;
```

Output:

employee_id	employee_name	salary	salary+100
1	alex	25000	25100
2	rr	55000	55100
3	jpm	52000	52100
4	ggshmr	12312	12412

Here we have done addition of 100 to each Employee's salary i.e, addition operation on single column.

Let's perform **addition of 2 columns**:

```
SELECT employee_id, employee_name, salary, salary + employee_id
   AS "salary + employee_id" FROM addition;
```

Output:

employee_id	employee_name	salary	salary+employee_id
1	alex	25000	25001
2	rr	55000	55002
3	jpm	52000	52003
4	ggshmr	12312	12316

Here we have done addition of 2 columns with each other i.e, each employee's employee_id is added with its salary.

Subtraction (-) :

- It is use to perform **subtraction operation** on the data items, items include either single column or multiple columns.

Implementation:

```
SELECT employee_id, employee_name, salary, salary - 100
    AS "salary - 100" FROM subtraction;
```

Output:

employee_id	employee_name	salary	salary-100
12	Finch	15000	14900
22	Peter	25000	24900
32	Warner	5600	5500
42	Watson	90000	89900

Here we have done subtraction of 100 to each Employee's salary i.e, subtraction operation on single column.

- Let's perform **subtraction of 2 columns**:

```
SELECT employee_id, employee_name, salary, salary - employee_id
    AS "salary - employee_id" FROM subtraction;
```

Output:

employee_id	employee_name	salary	salary – employee_id
12	Finch	15000	14988
22	Peter	25000	24978
32	Warner	5600	5568
42	Watson	90000	89958

Here we have done subtraction of 2 columns with each other i.e, each employee's employee_id is subtracted from its salary.

Multiplication (*) :

- It is use to perform **multiplication** of data items.

Implementation:

```
SELECT employee_id, employee_name, salary, salary * 100
    AS "salary * 100" FROM addition;
```

Output:

employee_id	employee_name	salary	salary * 100
1	Finch	25000	2500000
2	Peter	55000	5500000
3	Warner	52000	5200000
4	Watson	12312	1231200

Let's perform **multiplication of 2 columns**:

```
SELECT employee_id, employee_name, salary, salary * employee_id
     AS "salary * employee_id" FROM addition;
```

Output:

employee_id	employee_name	salary	salary * employee_id
1	Finch	25000	25000
2	Peter	55000	110000
3	Warner	52000	156000
4	Watson	12312	49248

- Here we have done multiplication of 2 columns with each other i.e, each employee's employee_id is multiplied with its salary.

Modulus (%) :

- It is use to get **remainder** when one data is divided by another.

Implementation:

```
SELECT employee_id, employee_name, salary, salary % 25000
    AS "salary % 25000" FROM addition;
```

Output:

employee_id	employee_name	salary	salary % 25000
1	Finch	25000	0
2	Peter	55000	5000
3	Warner	52000	2000
4	Watson	12312	12312

- Here we have done modulus of 100 to each Employee's salary i.e, modulus operation on single column.

- Let's perform **modulus operation between 2 columns**:

```
SELECT employee_id, employee_name, salary, salary % employee_id
    AS "salary % employee_id" FROM addition;
```

Output:

employee_id	employee_name	salary	salary % employee_id
1	Finch	25000	0
2	Peter	55000	0
3	Warner	52000	1
4	Watson	12312	0

- Here we have done modulus of 2 columns with each other i.e, each employee's salary is divided with its id and corresponding remainder is shown.

Concept of NULL :

- If we perform any arithmetic operation on **NULL**, then answer is *always* null.
- **Implementation**:

```
SELECT employee_id, employee_name, salary, type, type + 100
    AS "type+100" FROM addition;
```

Output:

employee_id	employee_name	salary	type	type + 100
1	Finch	25000	NULL	NULL
2	Peter	55000	NULL	NULL
3	Warner	52000	NULL	NULL
4	Watson	12312	NULL	NULL

Here output always came null, since performing any operation on null will always result in a *null value*.

Comparison Operators

- **These operators are used to perform operations such as equal to, greater than, less than etc.**

Operator	Operation	Description
=	Equal to	Used to check if the values of both operands are equal or not. If they are equal, then it returns TRUE.
>	Greater than	Returns TRUE if the value of left operand is greater than the right operand.
<	Less than	Checks whether the value of left operand is less than the right operand, if yes returns TRUE.
>=	Greater than or equal to	Used to check if the left operand is greater than or equal to the right operand, and returns TRUE, if the condition is true.
<=	Less than or equal to	Returns TRUE if the left operand is less than or equal to the right operand.
<> or !=	Not equal to	Used to check if values of operands are equal or not. If they are not equal then, it returns TRUE.
!>	Not greater than	Checks whether the left operand is not greater than the right operand, if yes then returns TRUE.
!<	Not less than	Returns TRUE, if the left operand is not less than the right operand.

Example:

StudentID	FirstName	Age
1	Gopi	23
2	Priya	21
3	Rohan	21
4	Ramesh	20
5	Vaibhav	25

Equality Operator

you can use the = operator to test for equality in a query.

```
SELECT * FROM Students
WHERE Age = 20;
```

StudentID	FirstName	Age
4	Ramesh	20

Example[Use greater than]:

```
SELECT * FROM students
WHERE Age > 23;
```

Output:

StudentID	FirstName	Age
5	Vaibhav	25

Example[Use less than or equal to]:

```
SELECT * FROM students
WHERE Age <= 21;
```

Output:

StudentID	FirstName	Age
2	Priya	21
3	Rohan	21
4	Ramesh	20

Logical Operators

The logical operators are used to perform operations such as ALL, ANY, NOT, BETWEEN etc.

Operator	Description
ALL	Used to compare a specific value to all other values in a set
ANY	Compares a specific value to any of the values present in a set.
IN	Used to compare a specific value to the literal values mentioned.
BETWEEN	Searches for values within the range mentioned.
AND	Allows the user to mention multiple conditions in a WHERE clause.
OR	Combines multiple conditions in a WHERE clause.
NOT	A negate operators, used to reverse the output of the logical operator.
EXISTS	Used to search for the row's presence in the table.
LIKE	Compares a pattern using wildcard operators.
SOME	Similar to the ANY operator, and is used compares a specific value to some of the values present in a set.

Example[ANY]

```
SELECT * FROM Students
WHERE Age > ANY (SELECT Age FROM Students WHERE Age > 21);
```

Output:

StudentID	FirstName	Age
1	Gopi	23
5	Vaibhav	25

Example[BETWEEN & AND]

```
SELECT * FROM Students
WHERE Age BETWEEN 22 AND 25;
```

Output:

StudentID	FirstName	Age
1	Gopi	23

Example[IN]

```
SELECT * FROM Students
WHERE Age IN('23', '20');
```

Output:

StudentID	FirstName	Age
1	Gopi	23
4	Ramesh	20