Structured Query Language (SQL) & Indexing

-Unit 4

Overview of SQL

- **S**tructured **Q**uery **L**anguage or **SQL** is a standard Database language which is used to create, maintain and retrieve the data from **relational databases.**
- Examples of databases are MySQL, Oracle, SQL Server, PostGre, etc. The recent ISO standard version of SQL is SQL:2019.
- As the name suggests, it is used when we have structured data (in the form of tables).
- All databases that are not relational (or do not use fixed structure tables to store data) and therefore do not use SQL, are called NoSQL databases. Examples of NoSQL are MongoDB, DynamoDB, Cassandra, etc.

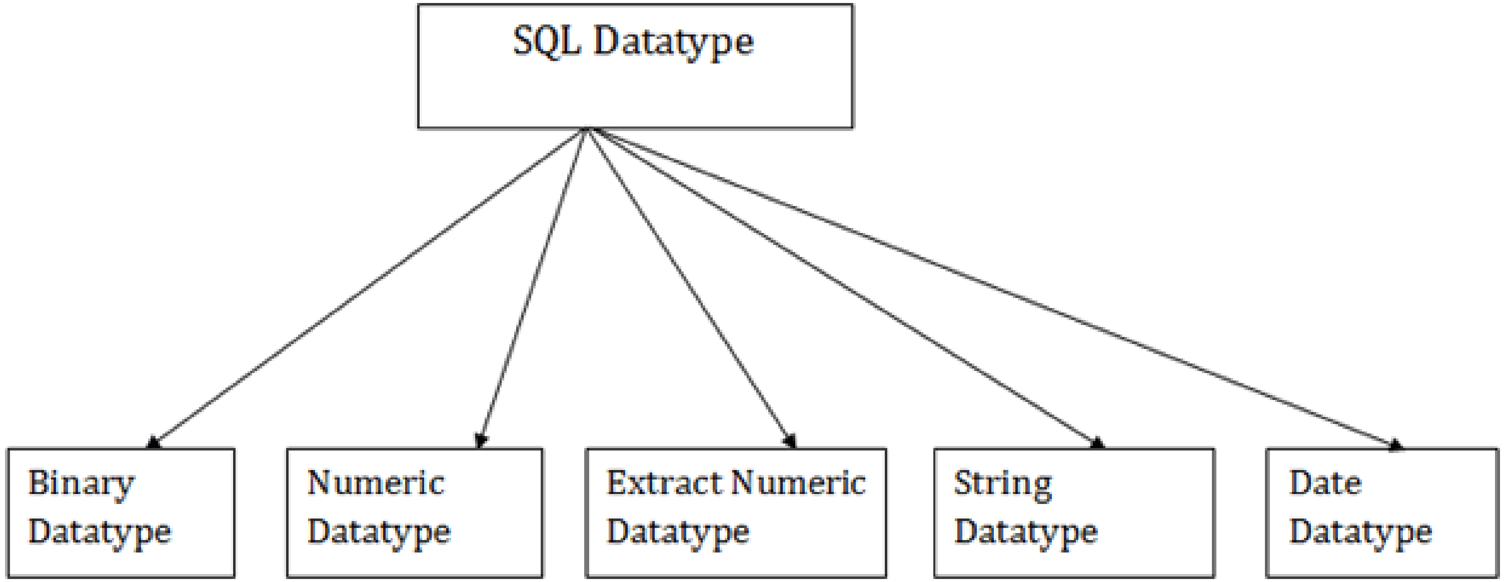
- SQL is used to perform operations on the records stored in the database, such as
  - updating records,
  - inserting records,
  - deleting records,
  - creating and
  - modifying database tables,
  - views, etc.
- SQL is not case sensitive. But it is a recommended practice to use keywords (like SELECT, UPDATE, CREATE, etc) in capital letters and use user defined things (liked table name, column name, etc) in small letters.

Advantages of SQL

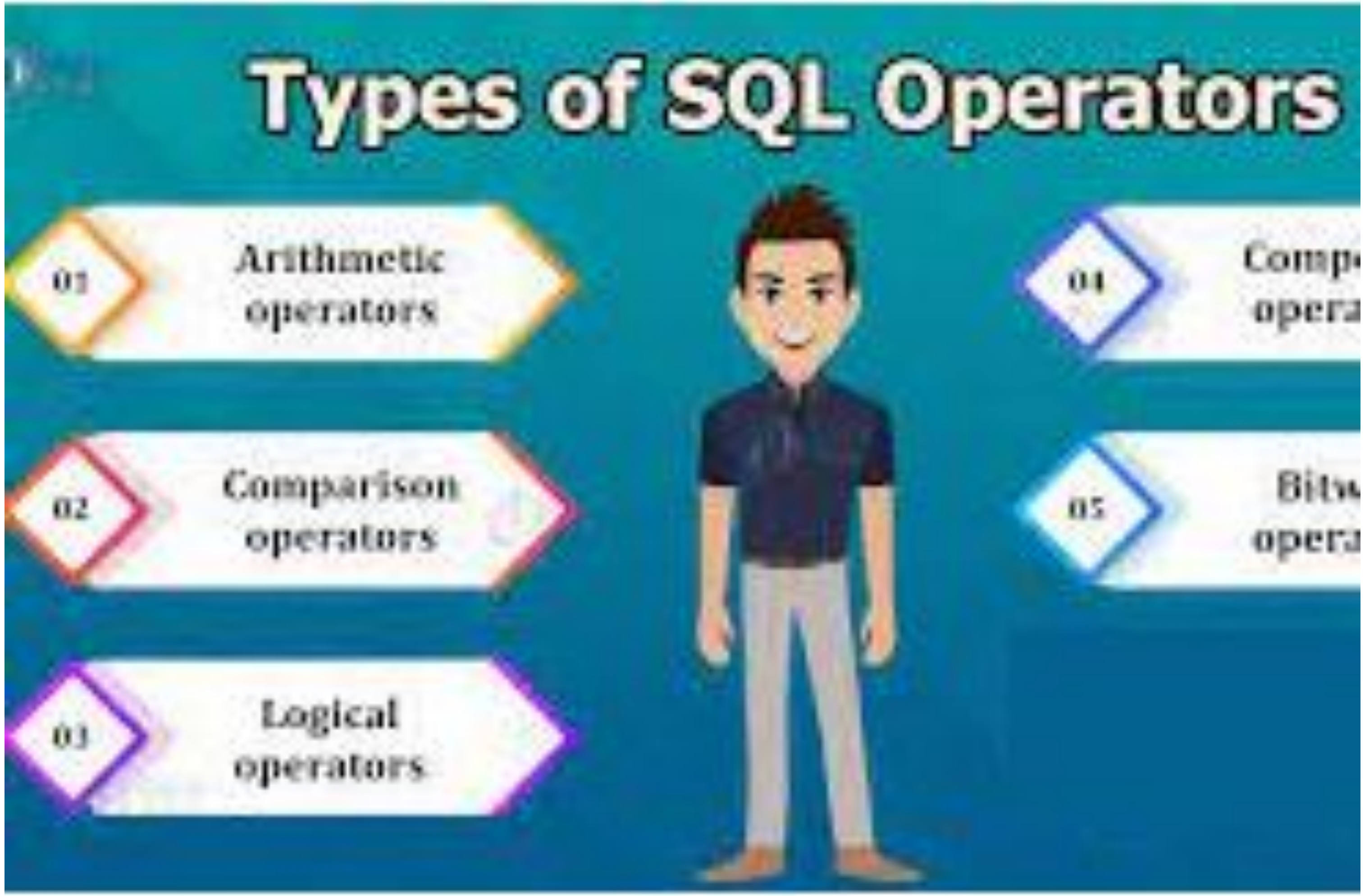
- **Faster Query Processing** (High speed)**–**
  - Large amount of data is retrieved quickly and efficiently.
  - Operations like Insertion, deletion, manipulation of data is also done in almost no time.
- **No Coding Skills –**
  - For data retrieval, large number of lines of code is not required.
  - All basic keywords such as SELECT, INSERT INTO, UPDATE, etc are used
- **Standardized Language –**
  - Due to documentation and long establishment over years, it provides a uniform platform worldwide to all its users.

- **Portable –**
  - It can be used in programs in PCs, server, laptops independent of any platform (Operating System, etc).
  - Also, it can be embedded with other applications as per need/requirement/use.
- **Interactive Language –**
  - Easy to learn and understand, answers to complex queries can be received in seconds.
- **Multiple data view**
  - Using the SQL language, the users can make different views of the database structure.

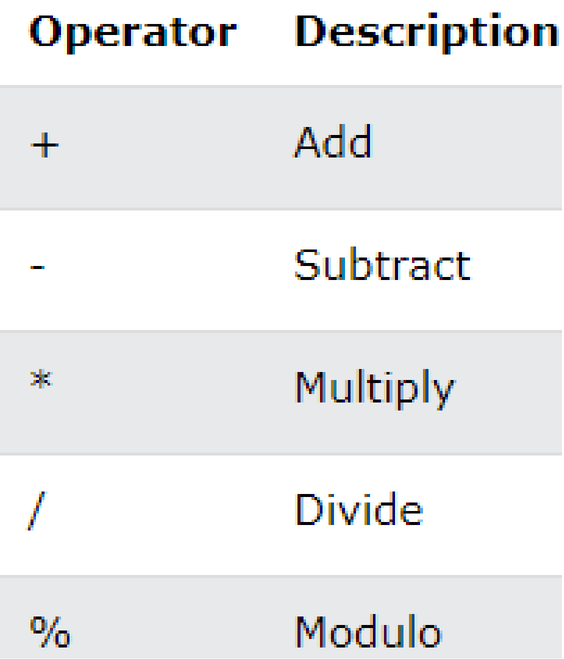
SQL Data types



SQL Operators

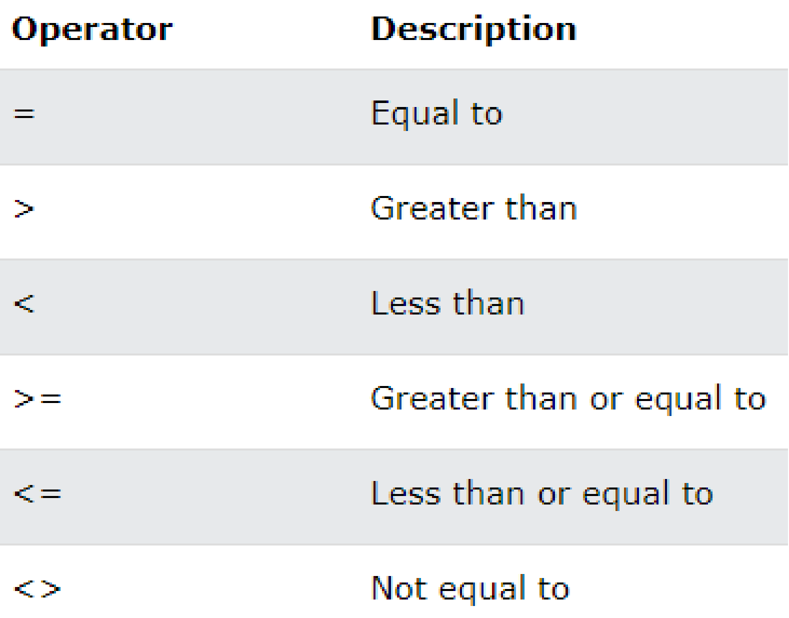


- Arithmetic Operators



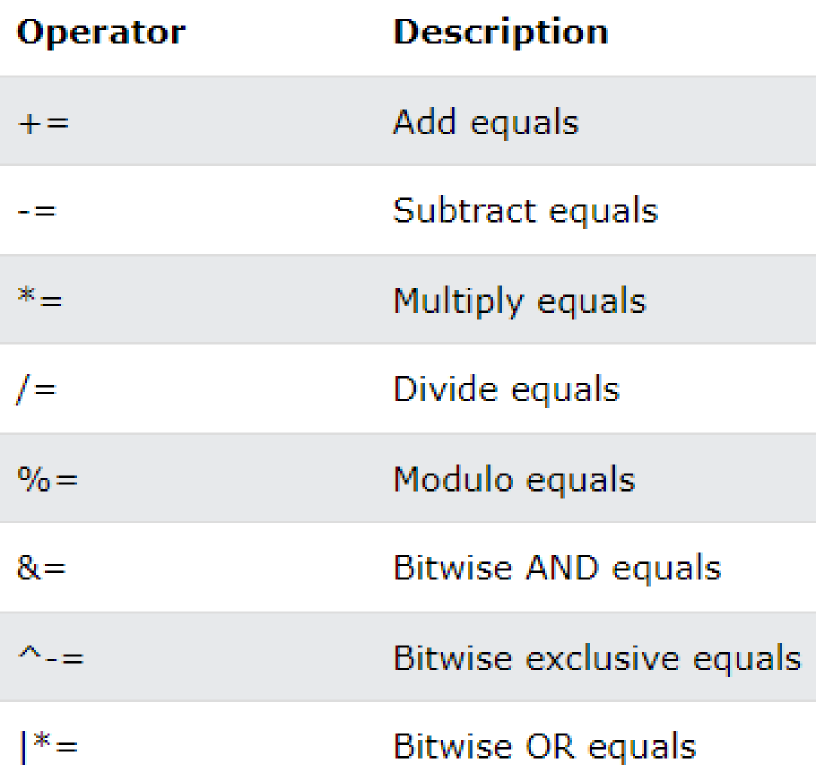
Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

- Comparison Operator



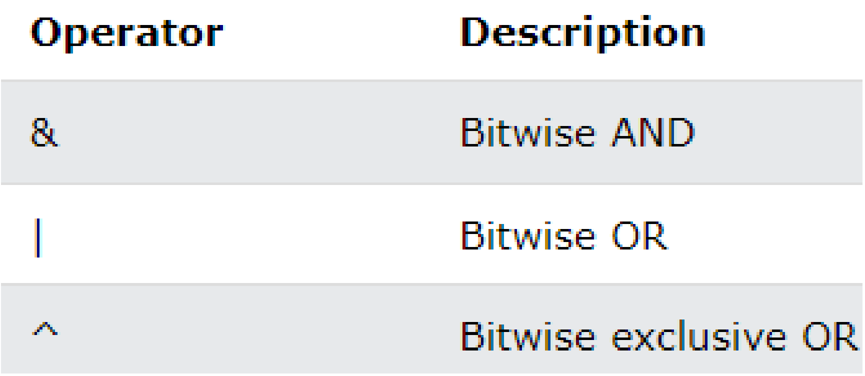
Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

- Compound Operators



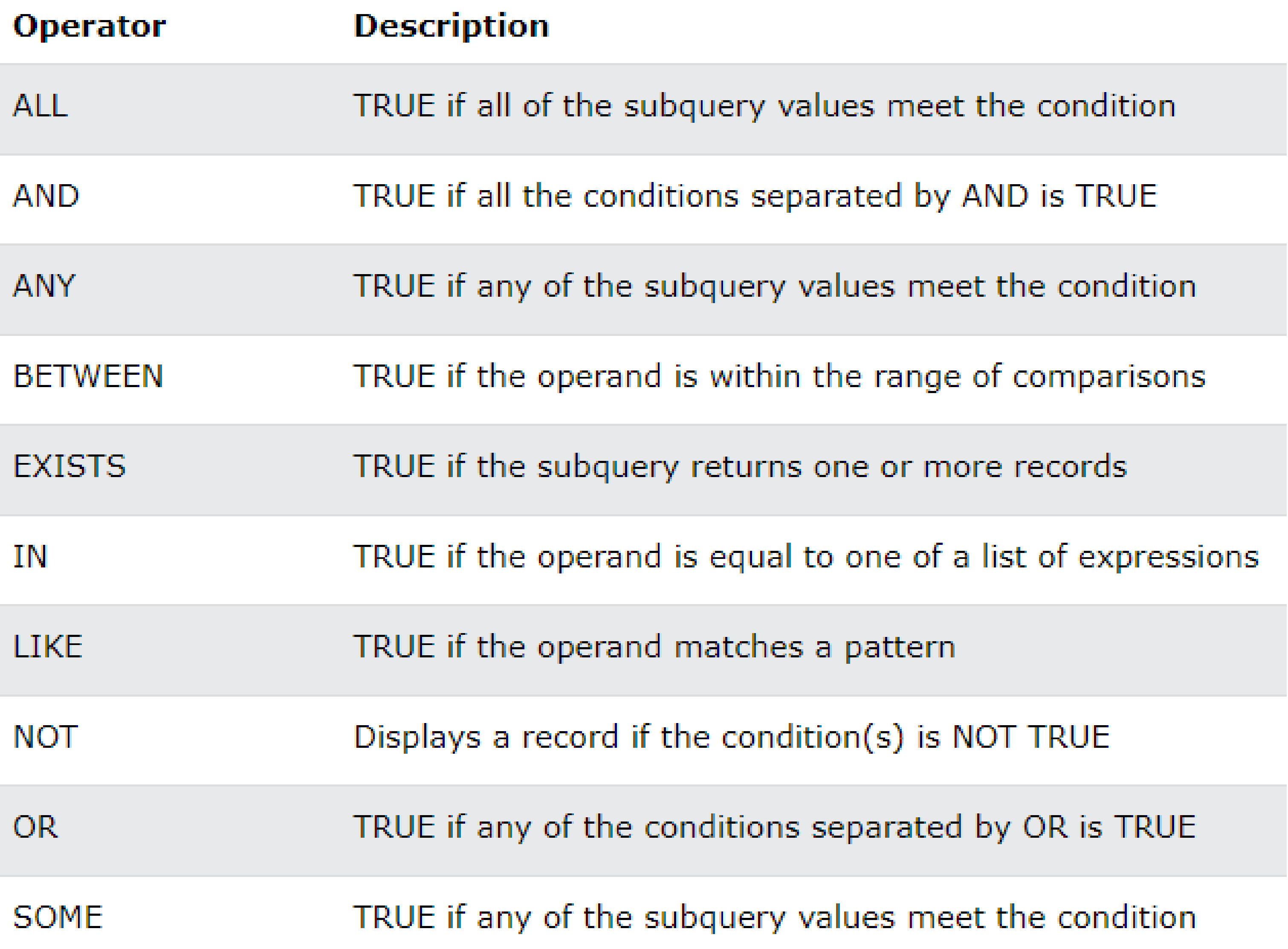
Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

- Bitwise Operator



Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

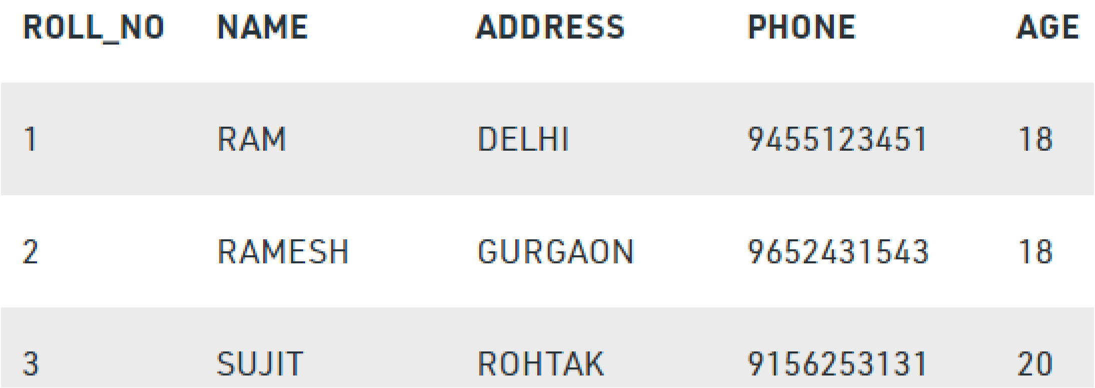
Logical Operators



Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

What is Relational Database?

- Relational database means the data is stored as well as retrieved in the form of relations (tables).
- Table 1 shows the relational database with only one relation called **STUDENT,**
- which stores ROLL_NO, NAME, ADDRESS, PHONE and AGE of students.
- **STUDENT**



ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20

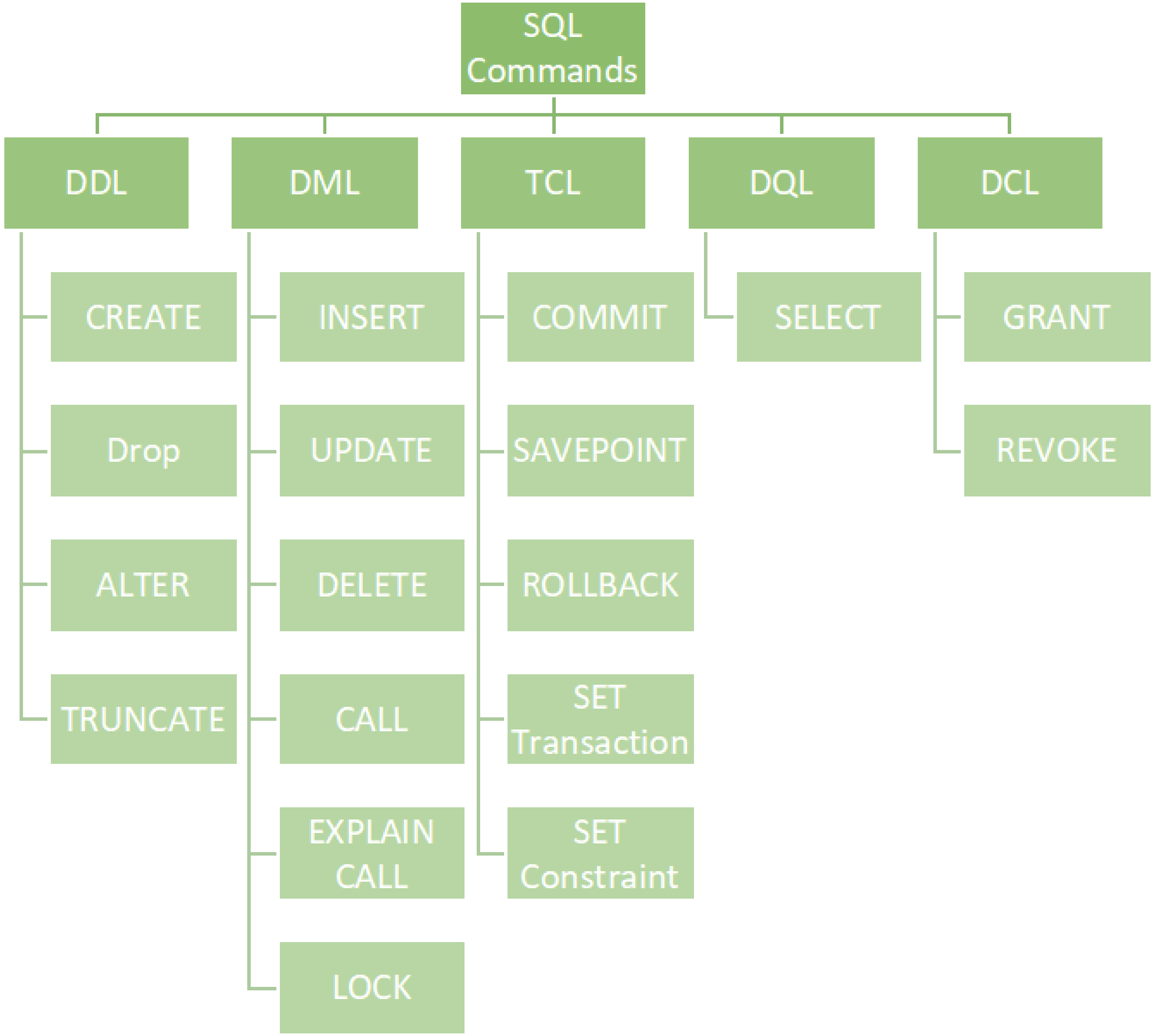
Important terminologies that are used in terms of relation.

- **Attribute (field):** Attributes are the properties that define a relation. e.g.; **ROLL_NO, NAME** etc.
- **Tuple:** Each row in the relation is known as tuple.
- **Degree:** The number of attributes in the relation is known as degree of the relation.

  Ex.The **STUDENT** relation defined above has degree 5.
- **Cardinality:** The number of tuples in a relation is known as cardinality. Ex.The **STUDENT** relation defined above has cardinality 3.
- **Column:** Column represents the set of values for a particular attribute.
- **A *query:*** *It* is an inquiry to the database for information.

Classification of SQL commands

- DDL (Data Definition Language)
- DML (Data Manipulation Language)
- DCL (Data Control Language)
- DQL (Data Query Language)
- Transactional control commands



- SQL Commands
  - DDL
    - CREATE
    - Drop
    - ALTER
    - TRUNCATE
  - DML
    - INSERT
    - UPDATE
    - DELETE
    - CALL
    - EXPLAIN CALL
    - LOCK
  - TCL
    - COMMIT
    - SAVEPOINT
    - ROLLBACK
    - SET Transaction
    - SET Constraint
  - DQL
    - SELECT
  - DCL
    - GRANT
    - REVOKE

DQL (Data Query Language):

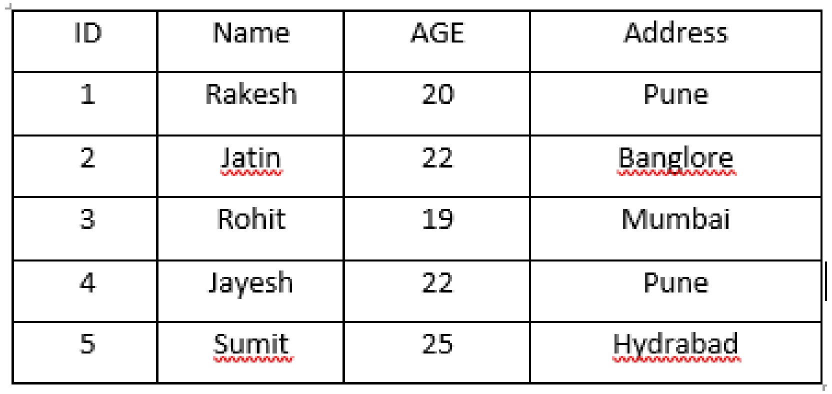
- It allows getting data from the database and imposing order upon it.
- It includes the SELECT statement.
- This command allows getting the data out of the database to perform operations with it.

List of DQL:

- **SELECT:** It is used to retrieve data from the database.
- It is also can be used with where clause to retrieve specific record.
- Syntax: Select column from <table_name> where <condition>;
- Ex. Select * from students where rno=5;

SELECT Command

- The SELECT command shows the records of the specified table.
- It also shows the particular record of a particular column by using the WHERE clause.
- Select statement retrieves the data from database according to the constraints specifies alongside.
- **Syntax:**
- SELECT <Col1 , col2 , col3 … , col N> FROM <TABLE NAME>
- Here, **column_Name_1, column_Name_2, ….., column_Name_N** are the names of those columns whose data we want to retrieve from the table.
- If we want to retrieve the data from all the columns of the table, we have to use the following SELECT command:
- **SELECT * FROM** table_name;
- Example:
- Select * from students;



ID	Name	AGE	Address
1	Rakesh	20	Pune
2	Jatin	22	Banglore
3	Rohit	19	Mumbai
4	Jayesh	22	Pune
5	Sumit	25	Hydrabad

USE OF DISTINCT

- SQL does not treat a relation as a set; duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

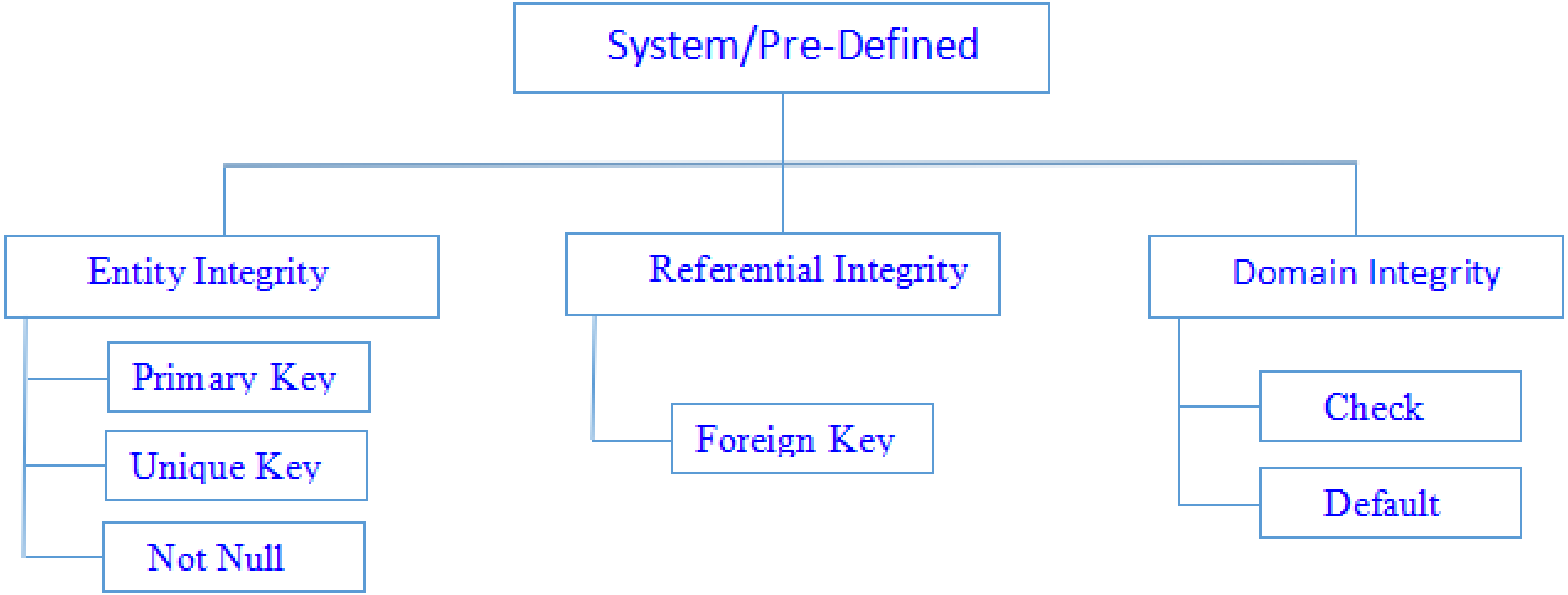
```
Q11:    SELECT    SALARY
        FROM      EMPLOYEE
Q11A:   SELECT    DISTINCT SALARY
        FROM      EMPLOYEE
```

USE OF DISTINCT

- SQL does not treat a relation as a set; duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

```
Q11:    SELECT    SALARY
        FROM      EMPLOYEE
Q11A:   SELECT    DISTINCT SALARY
        FROM      EMPLOYEE
```

Classification of constraints

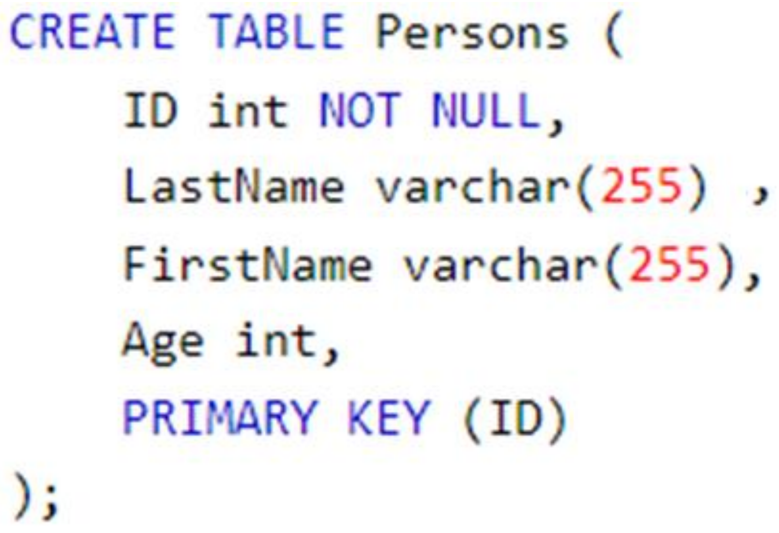


SQL constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- 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.
- Constraints can be specified when a table is created with the CREATE TABLE statement.
- you can use the ALTER TABLE statement to create constraints even after the table is created.
- Constraints can be defined in two ways
  - **column-level definition** The constraints can be specified immediately after the column definition.
  - **table-level definition** The constraints can be specified after all the columns are defined.

NOT NULL Constraints

- The NOT NULL constraint in a column means that the column cannot store NULL values.
- That is, you will be not allowed to insert a new row in the table without specifying any value to this field.



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

Primary key = { Unique + Not NULL}

Candidate Keys

- Phone No
- Adhar No
- PAN No.
- Reg_No
- RollNo

RollNO	Contact	Parents Contact	Email	AdharCard

Primary key constraints

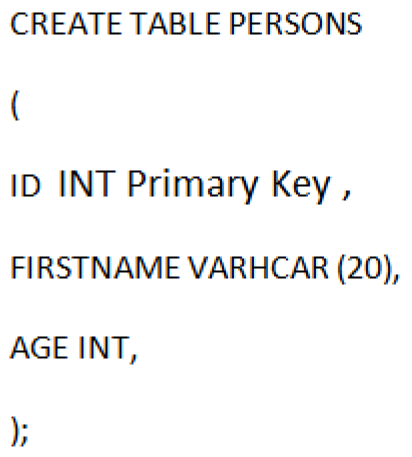
- This constraint defines a column or combination of columns which uniquely identifies each row in the table.

- Primary keys must contain UNIQUE values, and cannot contain NULL values.

  Primary key = { Unique + Not NULL}

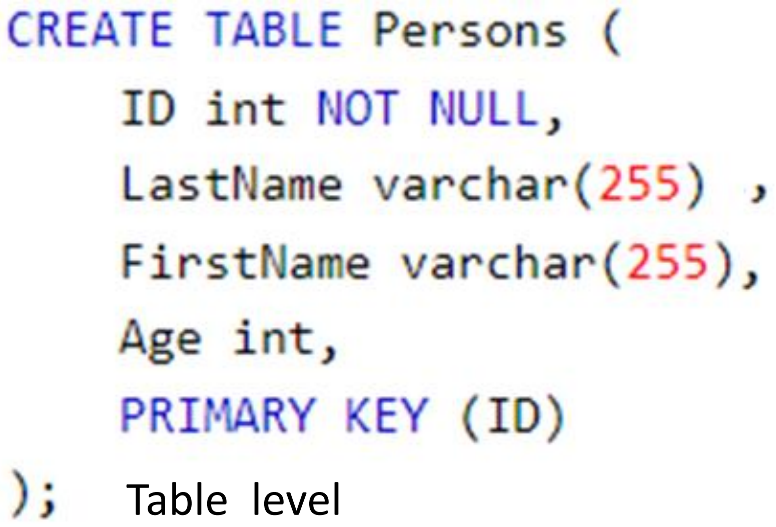
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

- Syntax:
- It creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:



```
CREATE TABLE PERSONS

(

ID INT Primary Key ,

FIRSTNAME VARHCAR (20),

AGE INT,

);
```

Column Level



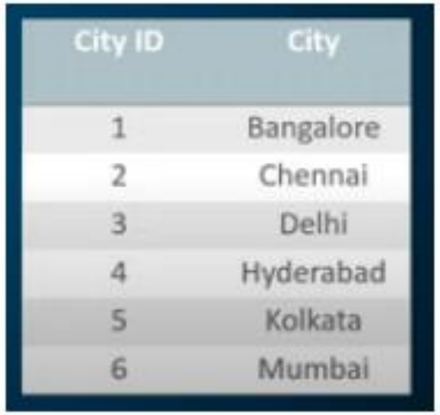
```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) ,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

Table level

Foreign Key constraints

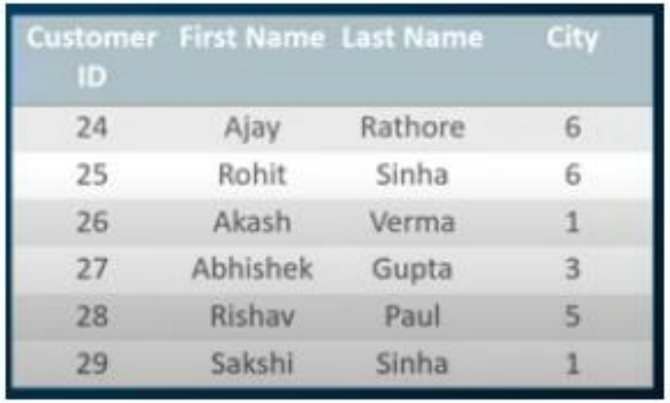
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

City



City ID	City
1	Bangalore
2	Chennai
3	Delhi
4	Hyderabad
5	Kolkata
6	Mumbai

Customer



Customer ID	First Name	Last Name	City
24	Ajay	Rathore	6
25	Rohit	Sinha	6
26	Akash	Verma	1
27	Abhishek	Gupta	3
28	Rishav	Paul	5
29	Sakshi	Sinha	1

Syntax:

- SQL FOREIGN KEY on CREATE TABLE

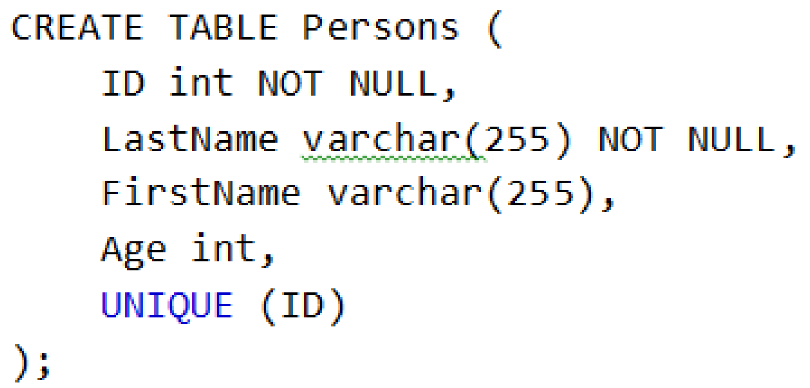
```
CREATE TABLE customer (
    customerID int NOT NULL,
    FirstName varchar(20),
    LastName varchar(20)
    City int,
    PRIMARY KEY (customerID),
    FOREIGN KEY (city) REFERENCES City(CityID)
);
```

Unique constraints

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

SQL UNIQUE Constraint on CREATE TABLE

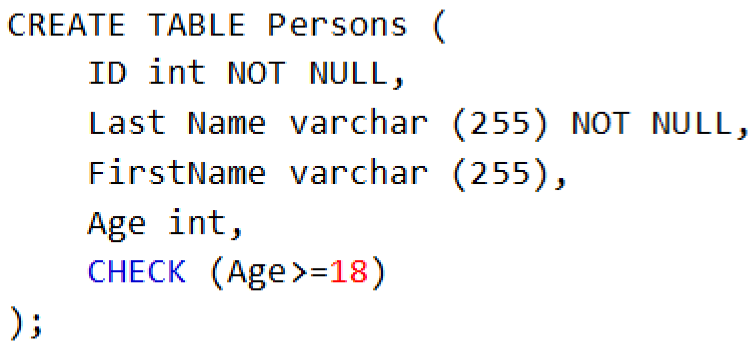
- The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:



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

Check constraints

- 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 column it will allow only certain values for this column.
- **SQL CHECK on CREATE TABLE**



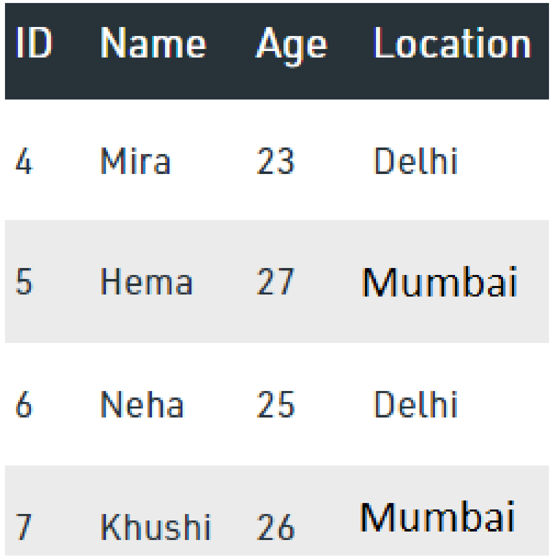
```
CREATE TABLE Persons (
    ID int NOT NULL,
    Last Name varchar (255) NOT NULL,
    FirstName varchar (255),
    Age int,
    CHECK (Age>=18)
);
```

Default constraints

- The DEFAULT constraint is used to set a default value for a column.
- The default value will be added to all new records, if no other value is specified.
- SQL DEFAULT on CREATE TABLE
- To set a DEFAULT value for the “Location” column when the “Venue” table is created –
- CREATE TABLE Venue (ID int NOT NULL, Name varchar(255), Age int, Location varchar(255) DEFAULT ‘Mumbai');

```
INSERT INTO Venue VALUES (4, 'Mira', 23, 'Delhi');
INSERT INTO Venue VALUES (5, 'Hema', 27);
INSERT INTO Venue VALUES (6, 'Neha', 25, 'Delhi');
INSERT INTO Venue VALUES (7, 'Khushi', 26);
```

- select * from Venue;



ID	Name	Age	Location
4	Mira	23	Delhi
5	Hema	27	Mumbai
6	Neha	25	Delhi
7	Khushi	26	Mumbai

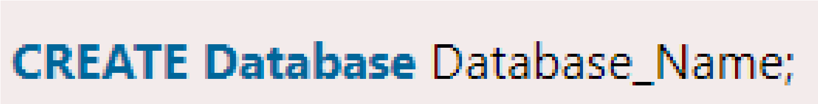
DDL COMMANDS

DDL (Data Definition Language):

- It is used to define the structure of the database .
- DDL actually consists of the SQL commands that can be used to define the database schema. (A schema is a collection of database objects like tables, triggers, stored procedures, etc.)
- *Data Definition Language, DDL*, is the part of SQL that allows a database user to create and restructure database objects, such as the creation or the deletion of a table.
- DDL commands :
  - **CREATE**
  - **ALTER TABLE**
  - **DROP TABLE**
  - **TRUNCATE TABLE**
  - **RENAME TABLE**
  - CREATE INDEX
  - ALTER INDEX
  - DROP INDEX
  - CREATE VIEW
  - DROP VIEW

CREATE, SHOW & DROP DATABASE

- The SQL CREATE DATABASE statement is used to create a new SQL database.
- Syntax:



```
CREATE Database Database_Name;
```

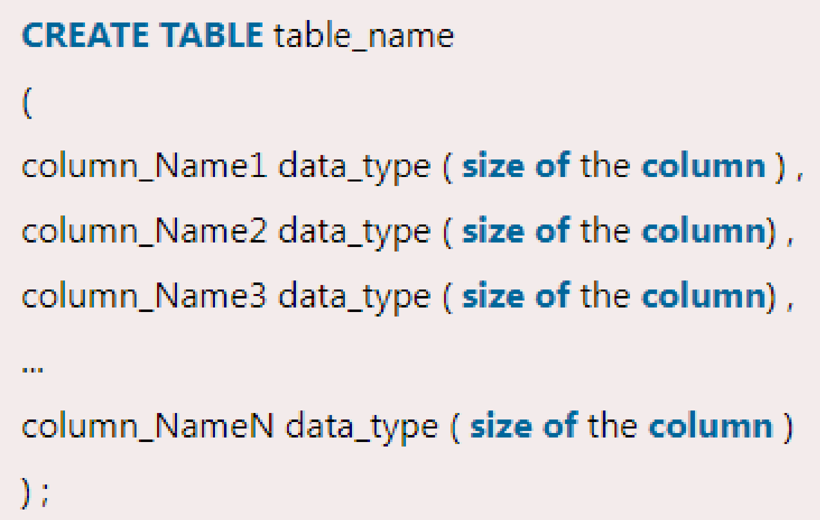
- Example
- If you want to create a new database <college>,
    - Posgres> CREATE DATABASE college;
- Once a database is created, you can check it in the list:
    - Postgres> \list;

DROP DATABASE

- The SQL DROP DATABASE statement is used to drop an existing database in SQL schema.
- Syntax: DROP DATABASE **DatabaseName**;
- Example:
    - SQL> DROP DATABASE college;

CREATE Table

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

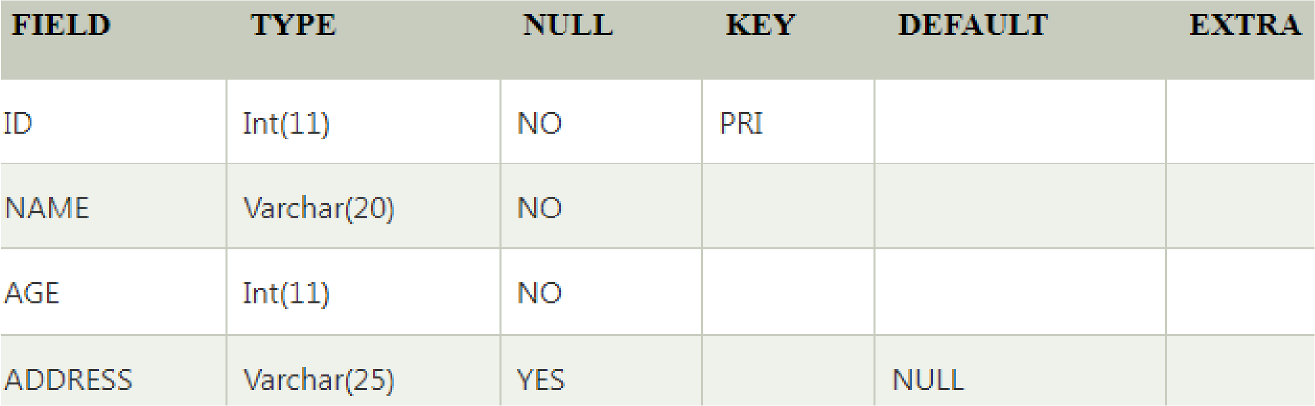


```
CREATE TABLE table_name
(
column_Name1 data_type ( size of the column ) ,
column_Name2 data_type ( size of the column) ,
column_Name3 data_type ( size of the column) ,
...
column_NameN data_type ( size of the column )
) ;
```

Note:

- The data type of the columns may vary from one database to another.
- For example, NUMBER is supported in Oracle database for integer value whereas INT is supported in MySQL.

- Example:
-
```
SQL> CREATE TABLE STUDENTS
(
   ID INT NOT NULL,
 NAME VARCHAR (20) NOT NULL,
   AGE INT  NOT NULL,
   ADDRESS VARCHAR (25),
   PRIMARY KEY (ID)
   );
```
- You can see the structure of your table by using disc command.
- Example: Postgres> \d STUDENTS;



FIELD	TYPE	NULL	KEY	DEFAULT	EXTRA
ID	Int(11)	NO	PRI		
NAME	Varchar(20)	NO			
AGE	Int(11)	NO			
ADDRESS	Varchar(25)	YES		NULL	

DROP , TRUNCATE table command

- **DROP:**
- It is used to delete a table definition and all data from a table.
- This is very important to know that once a table is deleted all the information available in the table is lost forever, so we have to be very careful when using this command.

  Syntax: **DROP TABLE** <table_name>;
- **Truncate:**
- It is used to delete all the rows from the table and free the containing space.

  Syntax: **TRUNCATE TABLE** <table_name>;

Alter table command

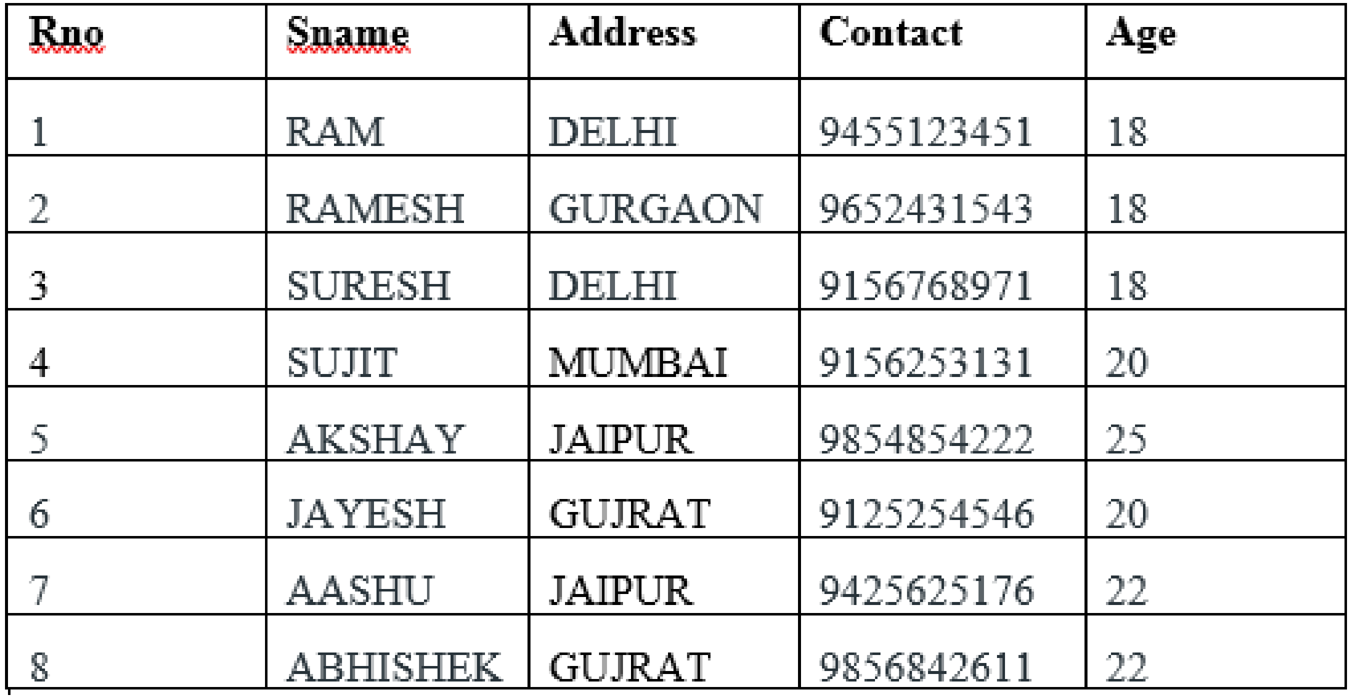
- syntax:
- To add new column:
  - ALTER TABLE table_name ADD column_name datatype;
- To delete existing column:
  - ALTER TABLE table_name DROP COLUMN column_name;
- To change column data type:
  - ALTER TABLE table_name ALTER COLUMN column_name TYPE datatype;
- ALTER TABLE to **ADD PRIMARY KEY** constraint to a table
  - ALTER TABLE table_name ADD CONSTRAINT **MyPrimaryKey** PRIMARY KEY (column_name);

AGGRATION FUNCTIONS

AGGRATION FUNCTIONS:

- Aggregation functions are used to perform mathematical operations on data values of a relation.
- take a collection (a set or multiset) of values as input and return a single value.
- Some of the common aggregation functions used in SQL are:
  - Average: **avg**
  - Minimum: **min**
  - Maximum: **max**
  - Total: **sum**
  - Count: **count**

Example Table : “Student”



Rno	Sname	Address	Contact	Age
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SURESH	DELHI	9156768971	18
4	SUJIT	MUMBAI	9156253131	20
5	AKSHAY	JAIPUR	9854854222	25
6	JAYESH	GUJRAT	9125254546	20
7	AASHU	JAIPUR	9425625176	22
8	ABHISHEK	GUJRAT	9856842611	22

- Count:
  - Query: SELECT COUNT (PHONE) FROM STUDENT;
  - Output: 8
- SUM:
  - Query: SELECT SUM (AGE) FROM STUDENT;
  - 163
- AVG:
  - Query: SELECT AVG (AGE) FROM STUDENT;
  - 20.37
- MAX:
  - Query: SELECT MAX (AGE) FROM STUDENT;
  - 25
- MIN:
  - Query: SELECT MIN (AGE) FROM STUDENT;
  - 18

Examples

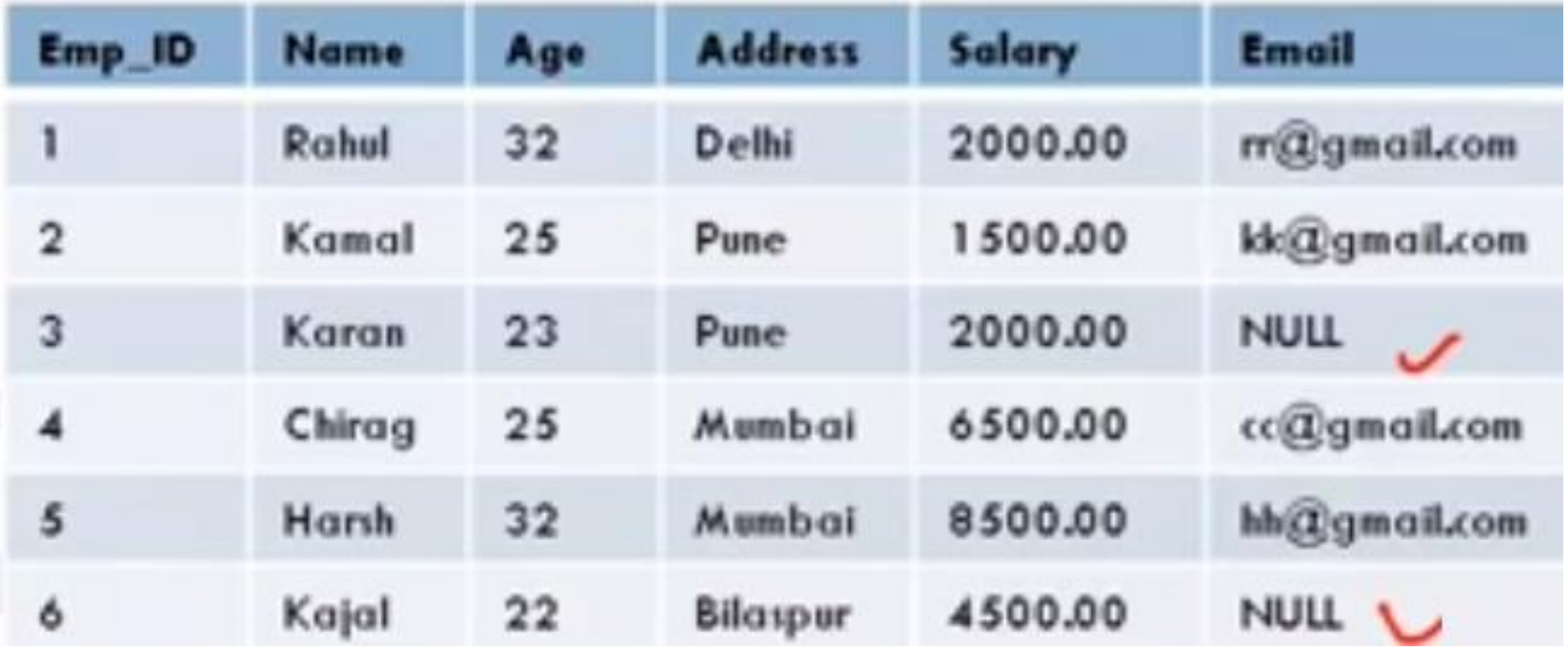
1. Consider following relational tables:

    1. Student (S_ID, Name, Dept_name)
    2. Course (course_ID, c_name, Dept_name)
    3. Trainer (ID, name, dept_name)
    4. Dept (Dept_ID, dept_name)

Solve above relation using DDL statements with primary key and Foreign key.

NULL Value

- A field with a NULL value is a field with no value.
- If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field.
- Then, the field will be saved with a NULL value.
- 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.



Emp_ID	Name	Age	Address	Salary	Email
1	Rahul	32	Delhi	2000.00	rr@gmail.com
2	Kamal	25	Pune	1500.00	kk@gmail.com
3	Karan	23	Pune	2000.00	NULL ✓
4	Chirag	25	Mumbai	6500.00	cc@gmail.com
5	Harsh	32	Mumbai	8500.00	hh@gmail.com
6	Kajal	22	Bilaspur	4500.00	NULL ✓

DML COMMANDS

DML(Data Manipulation Language)

- Data Manipulation Language, DML, is the part of SQL used to manipulate data within objects of a relational database.
- There are three basic DML commands:
    - INSERT
    - UPDATE
    - DELETE

Data Manipulation commands

- A **DML** is a language that enables users to access or manipulate data as organized by the appropriate data model.
- The types of access are**:**
  - Retrieval of information stored in the database
  - Insertion of new information into the database
  - Deletion of information from the database
  - Modification of information stored in the database

INSERT Command

- Insert statement is used to insert data into database tables.
- Syntax:

INSERT INTO <TABLE NAME> (<COLUMNS TO INSERT>) VALUES

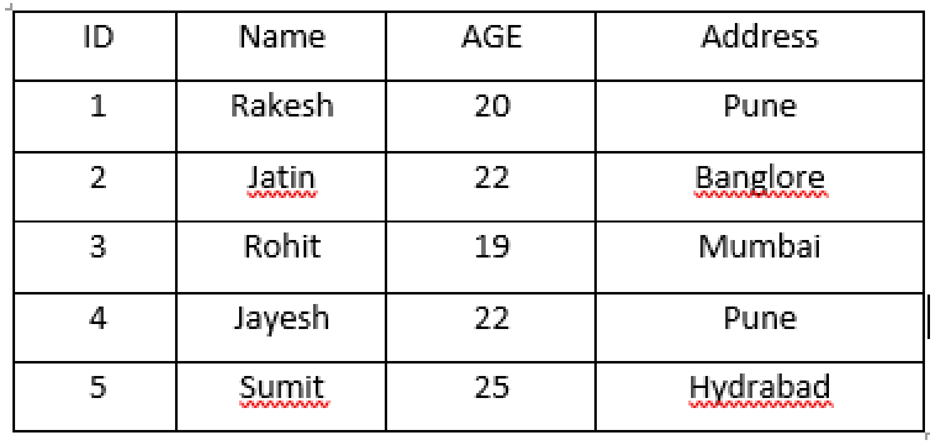
(<VALUES TO INSERT>)

OR

INSERT INTO <TABLE NAME> VALUES

(<VALUES TO INSERT>)

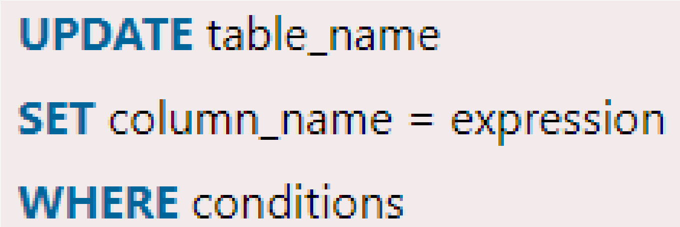
- Insert into students (ID, NAME, AGE, ADDRESS)values (1,’Rakesh’,20,’Pune’);
- Or
- Insert into students values (1,’Rakesh’,20,’Pune’);
- Output:



ID	Name	AGE	Address
1	Rakesh	20	Pune
2	Jatin	22	Banglore
3	Rohit	19	Mumbai
4	Jayesh	22	Pune
5	Sumit	25	Hydrabad

UPDATE command

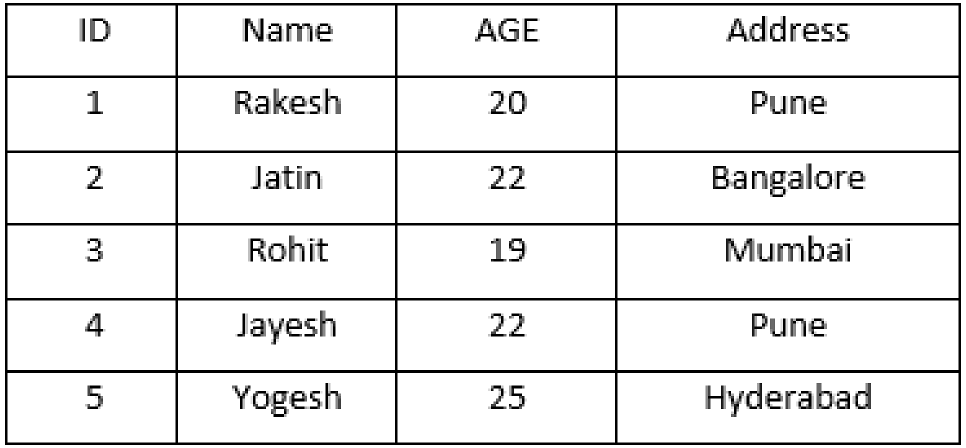
- The SQL commands (*UPDATE* and *DELETE*) are used to modify the data that is already in the database.
- `The SQL DELETE command uses a WHERE clause.
- **SQL UPDATE** statement is used to change the data of the records held by tables.
- Which rows is to be update, it is decided by a condition. To specify condition, we use WHERE clause.
- Syntax:
  UPDATE table_name **SET** [column_name1= value1,... column_nameN = valueN] [**WHERE** condition]



```
UPDATE table_name
SET column_name = expression
WHERE conditions
```

Example:

- **UPDATE** students
- **SET** Name = ’Yogesh'
- **WHERE** Student_Id = ’5’

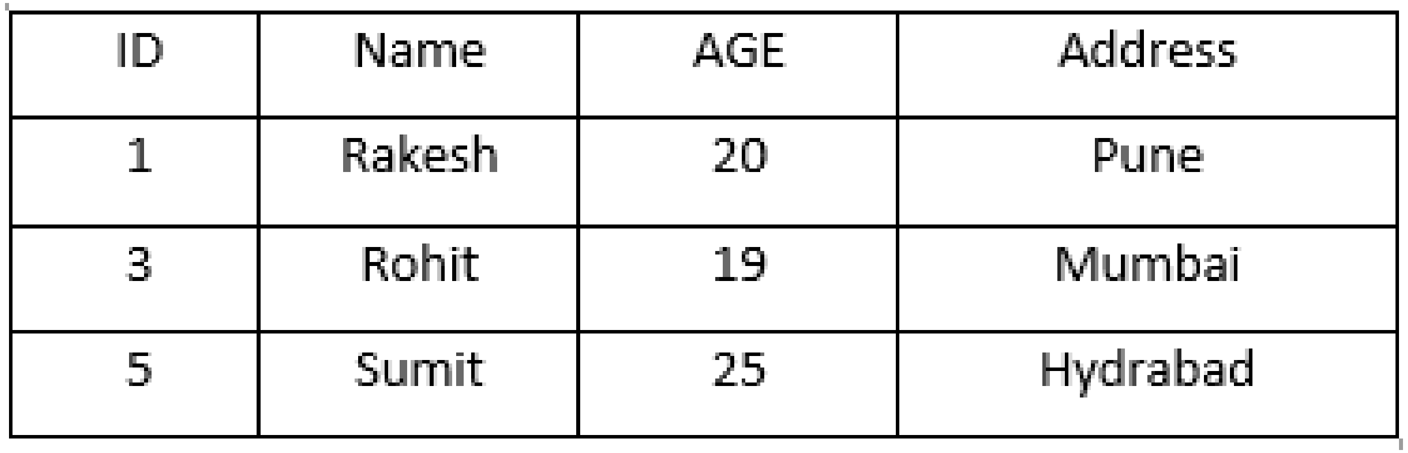


ID	Name	AGE	Address
1	Rakesh	20	Pune
2	Jatin	22	Bangalore
3	Rohit	19	Mumbai
4	Jayesh	22	Pune
5	Yogesh	25	Hyderabad

DELETE command

- **DELETE:**
- The DELETE statement is used to delete rows from a table.
- If you want to remove a specific row from a table you should use WHERE condition.
- Syntax: **DELETE FROM** table_name [**WHERE** condition];
- OR
- **DELETE FROM** table_name;

- Example:
- Delete * from students;
- **DELETE FROM** students where age="22";



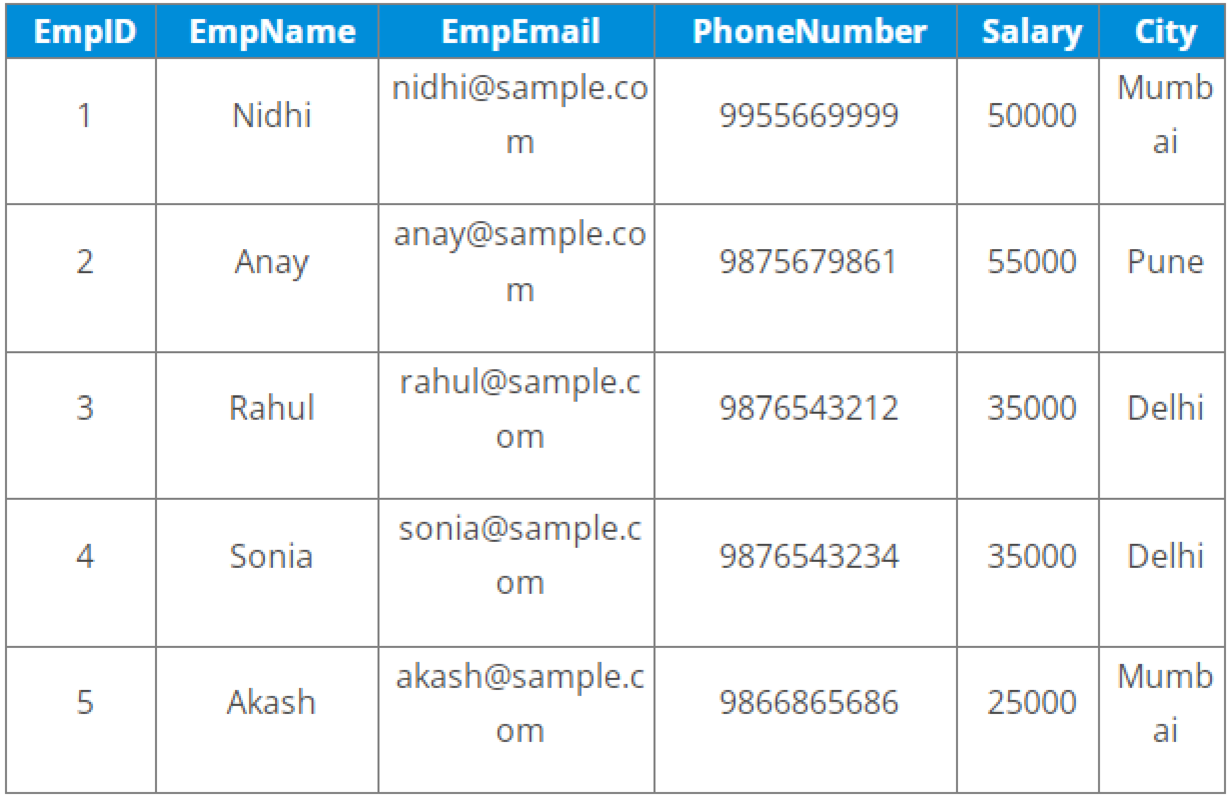
ID	Name	AGE	Address
1	Rakesh	20	Pune
3	Rohit	19	Mumbai
5	Sumit	25	Hydrabad

Complex Retrieval Queries using Group By

GROUP BY and HAVING Clause

- The GROUP BY clause is a SQL command that is used to **group rows that have the same values**.
- The GROUP BY clause is used in the **SELECT statement**.
- Optionally it is used in conjunction **with aggregate functions** to produce summary reports from the database.
- That’s what it does, **summarizing data** from the database.
- The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

- Examples:
  - Use GROUP BY on single column
  - GROUP BY on multiple columns
  - Use GROUP BY with ORDER BY
  - GROUP BY with HAVING clause
  - Use GROUP BY with JOINS



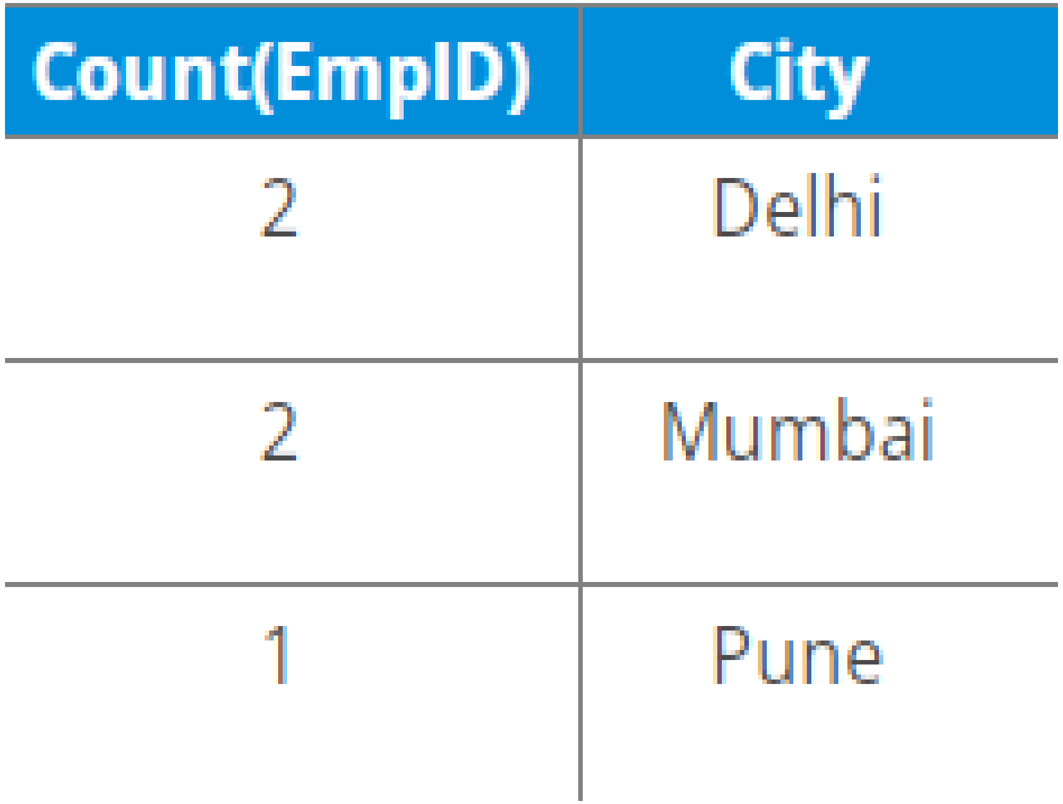
EmpID	EmpName	EmpEmail	PhoneNumber	Salary	City
1	Nidhi	nidhi@sample.com	9955669999	50000	Mumbai
2	Anay	anay@sample.com	9875679861	55000	Pune
3	Rahul	rahul@sample.com	9876543212	35000	Delhi
4	Sonia	sonia@sample.com	9876543234	35000	Delhi
5	Akash	akash@sample.com	9866865686	25000	Mumbai

GROUP BY on single column

- **Example:**
- **Find no. of employees per city.**

- Query:

```
SELECT COUNT(EmpID), City
FROM Employees
GROUP BY City;
```



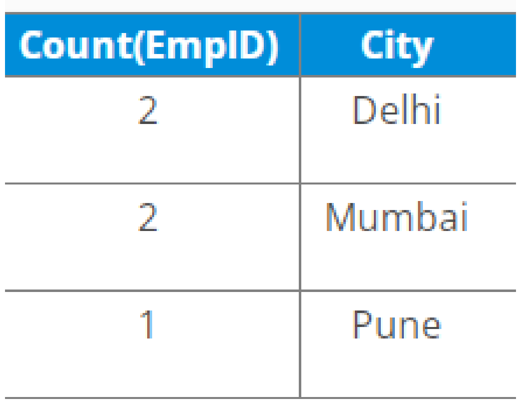
Count(EmpID)	City
2	Delhi
2	Mumbai
1	Pune

GROUP BY with ORDER BY

- When we use the SQL GROUP BY statement with the ORDER BY clause, the values get sorted either in ascending or descending order.

- Example:
- Write a query to retrieve the number of employees in each city, sorted in descending order.

```
SELECT COUNT(EmpID), City
FROM Employees
GROUP BY City
ORDER BY COUNT(EmpID) DESC;
```



Count(EmpID)	City
2	Delhi
2	Mumbai
1	Pune

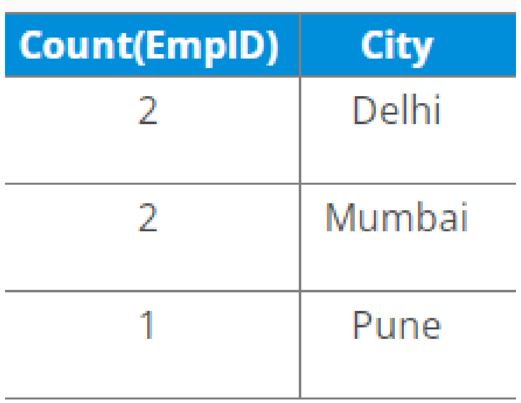
GROUP BY with HAVING clause

- The SQL GROUP BY statement is used with ‘HAVING’ clause to mention conditions on groups.
- Also, since we cannot use the aggregate functions with the WHERE clause, we have to use the ‘HAVING’ clause to use the aggregate functions with GROUP BY.

- Example:
- Write a query to retrieve the number of employees in each city, having salary > 15000

```
SELECT COUNT(EmpID), City
FROM Employees
GROUP BY City
HAVING SALARY > 15000;
```

(Since all are records in the Employee table have a salary > 15000, we will see the following table as output)

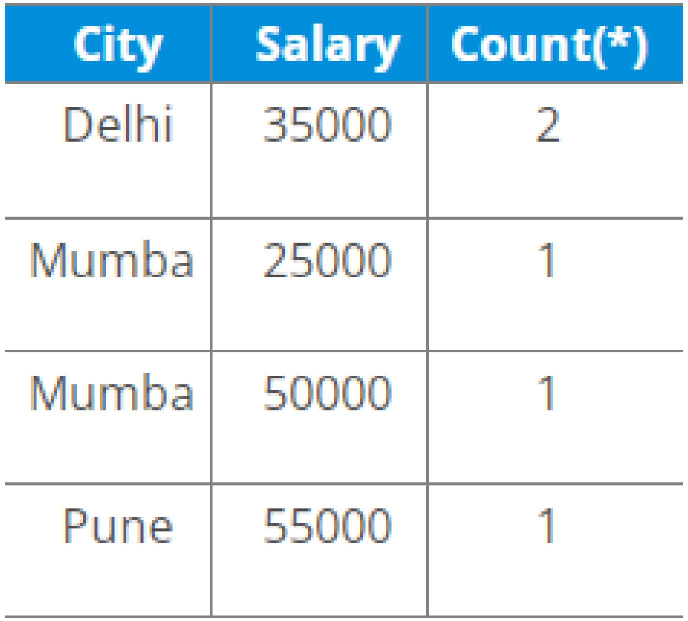


Count(EmpID)	City
2	Delhi
2	Mumbai
1	Pune

GROUP BY on multiple columns

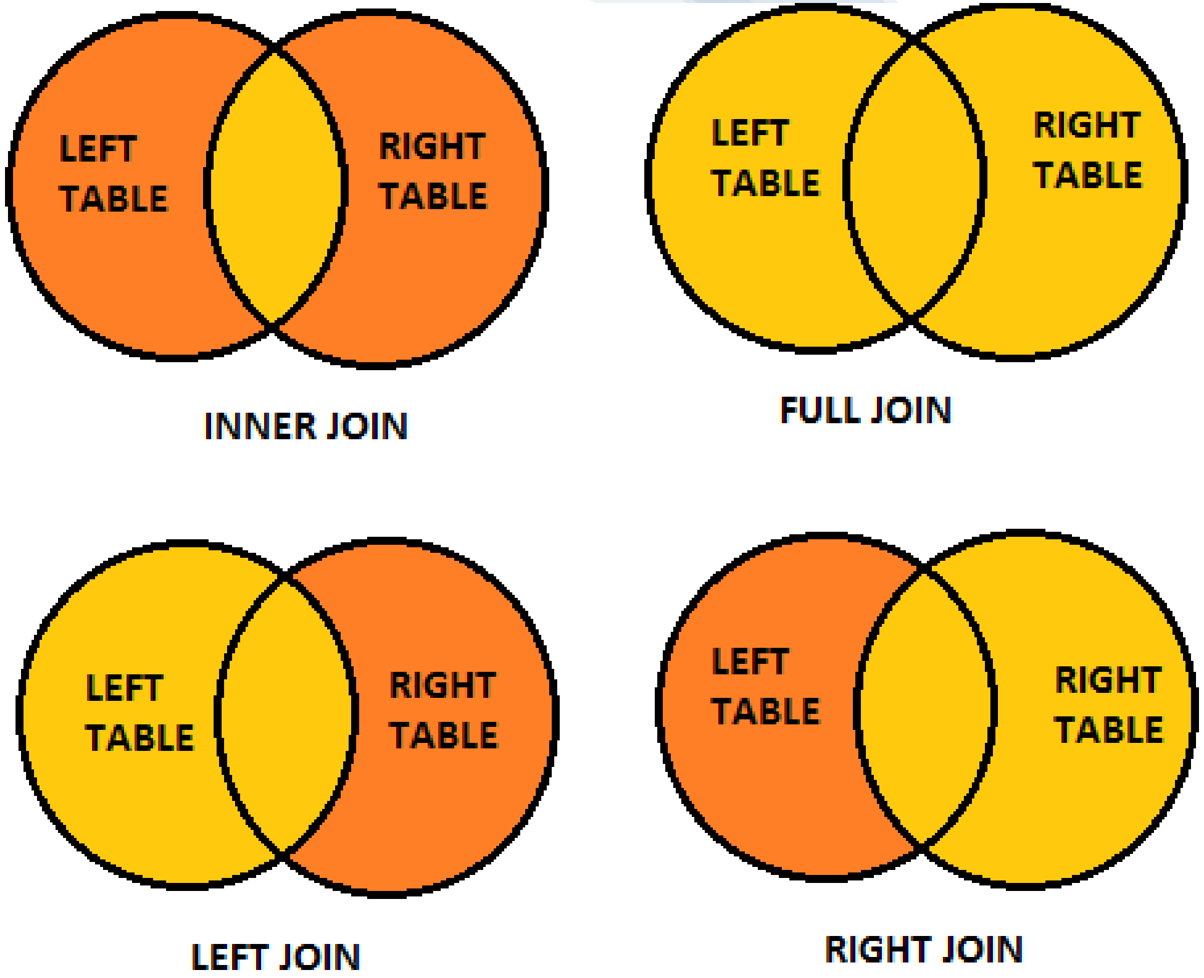
- Example:
- Write a query to retrieve the number of employees having different salaries in each city.

```
SELECT City, Salary, Count(*)
FROM Employees
GROUP BY City, Salary;
```



City	Salary	Count(*)
Delhi	35000	2
Mumba	25000	1
Mumba	50000	1
Pune	55000	1

JOIN



INNER JOIN

FULL JOIN

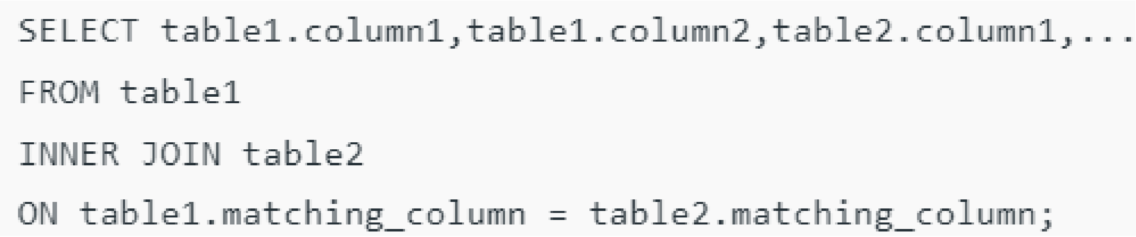
LEFT JOIN

RIGHT JOIN

- **SQL Join** statement is used to combine data or rows from two or more tables based on a common field between them.
- Different types of Joins are as follows:
  - INNER JOIN
  - LEFT JOIN
  - RIGHT JOIN
  - FULL JOIN

INNER JOIN

- Returns records that have matching values in both tables.
- *We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.*
- This keyword will create the result-set by combining all rows from both the tables where the condition satisfies
- i.e value of the common field will be the same.
- Syntax:



```
SELECT table1.column1,table1.column2,table2.column1,...
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

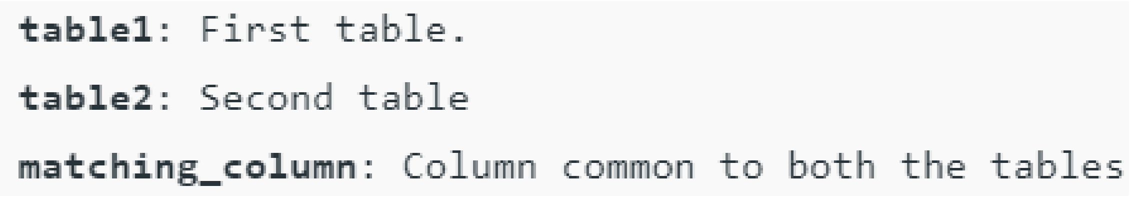
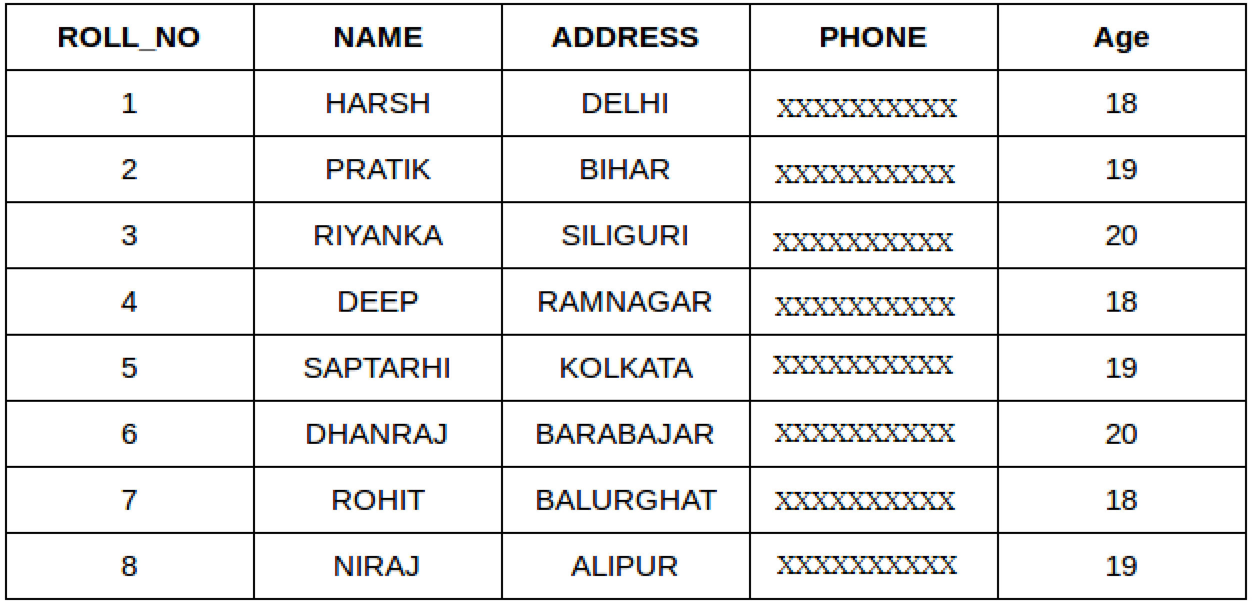


table1: First table.

table2: Second table

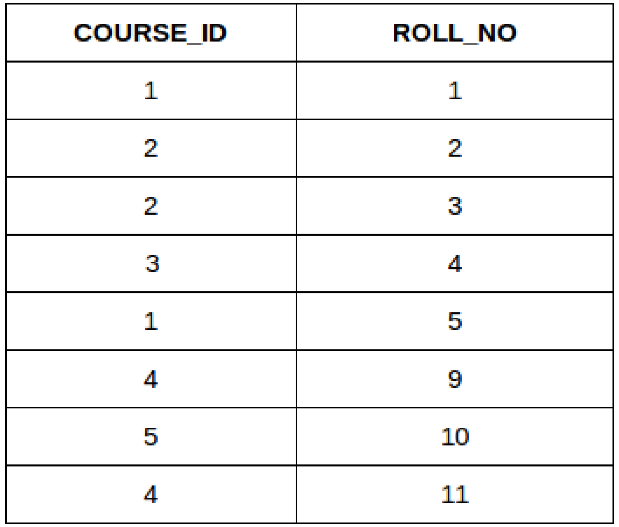
matching_column: Column common to both the tables

Student



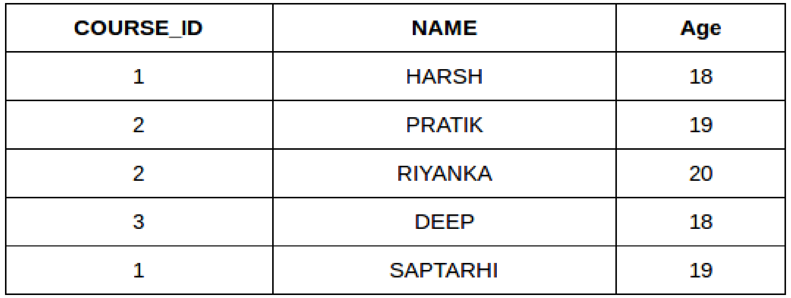
ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

Course



COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

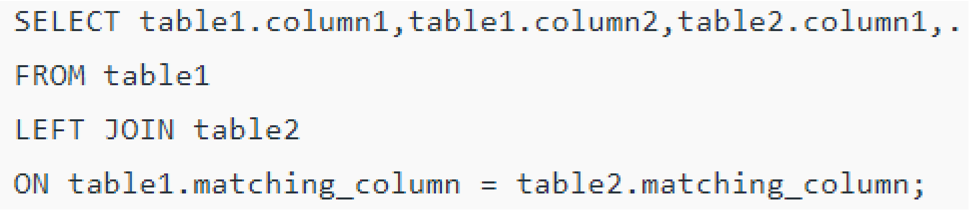
```
SELECT Course.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN Course
ON Student.ROLL_NO = Course.ROLL_NO;
```



COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

LEFT JOIN

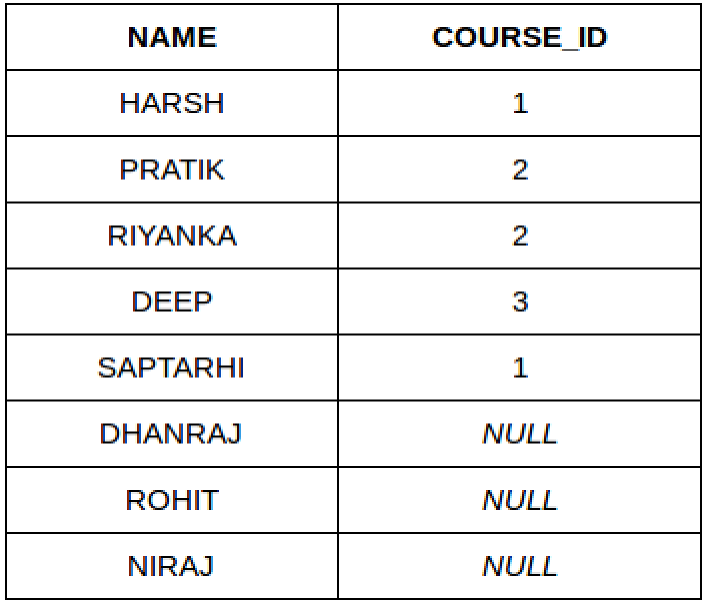
- This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join.
- For the rows for which there is no matching row on the right side, the result-set will contain *null*.
- LEFT JOIN is also known as LEFT OUTER JOIN.
- Syntax:



```
SELECT table1.column1,table1.column2,table2.column1,.
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Example

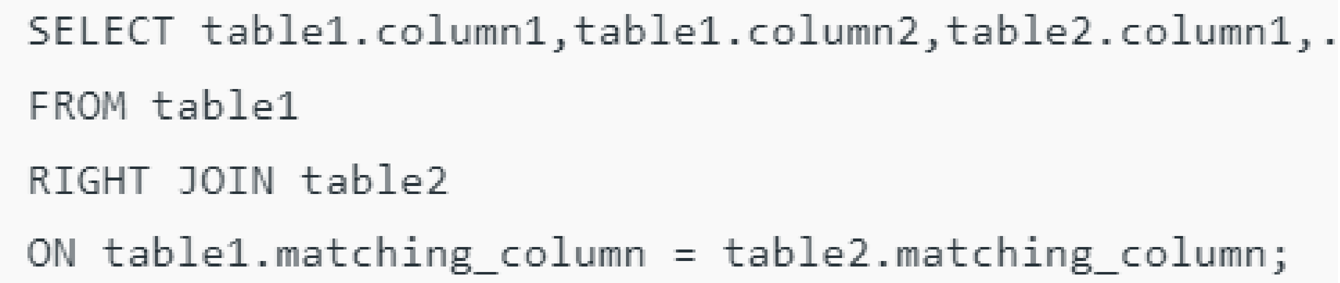
- SELECT Student.NAME,Course.COURSE_ID FROM Student
- LEFT JOIN Course
- ON Course.ROLL_NO = Student.ROLL_NO;



NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

RIGHT JOIN

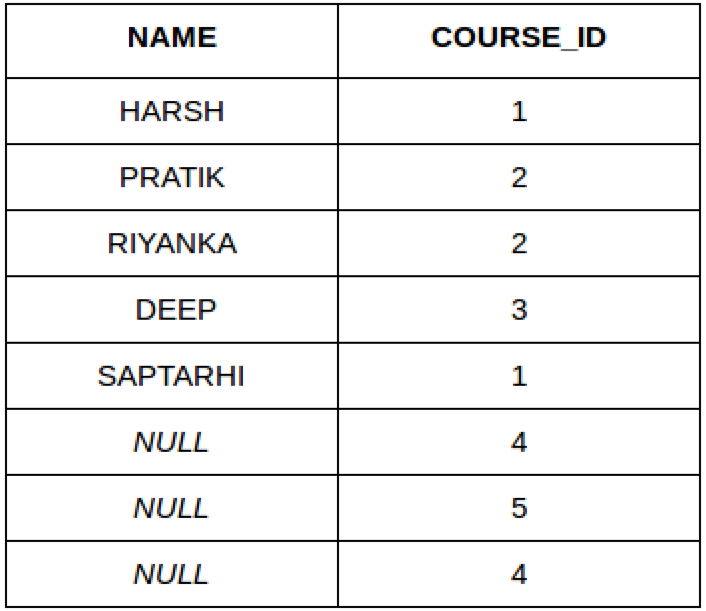
- This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join.
- For the rows for which there is no matching row on the left side, the result-set will contain *null*.
- RIGHT JOIN is also known as RIGHT OUTER JOIN.
- Syntax:



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

Example:

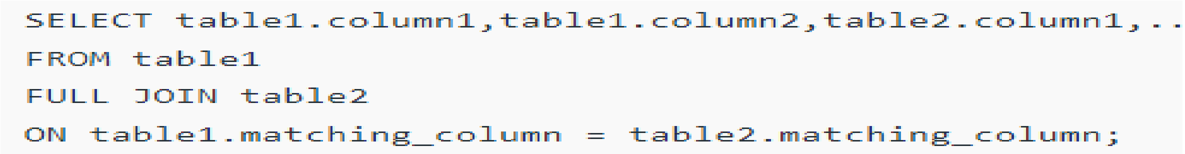
```
SELECT Student.NAME, Course.COURSE_ID
FROM Student
RIGHT JOIN Course
ON Course.ROLL_NO = Student.ROLL_NO;
```



NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

FULL JOIN

- FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN.
- The result-set will contain all the rows from both tables.
- For the rows for which there is no matching, the result-set will contain *NULL* values.

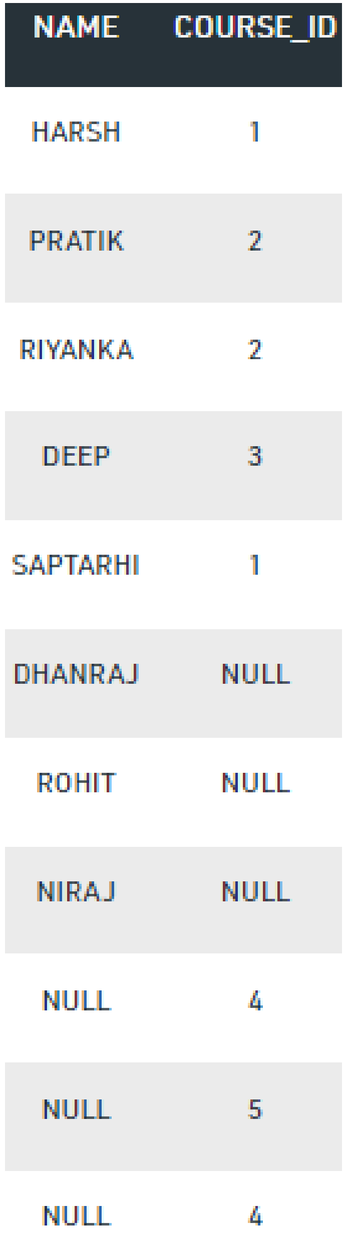


```
SELECT table1.column1,table1.column2,table2.column1,..
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```

- SELECT table1

Example:

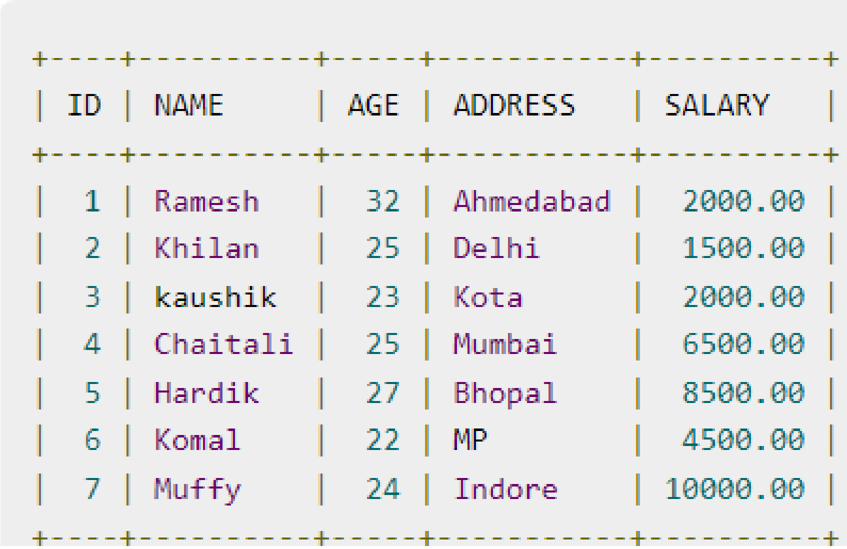
```
SELECT Student.NAME, Course.COURSE_ID
FROM Student
FULL JOIN Course
ON Course.ROLL_NO = Student.ROLL_NO;
```



NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	4
NULL	5
NULL	4

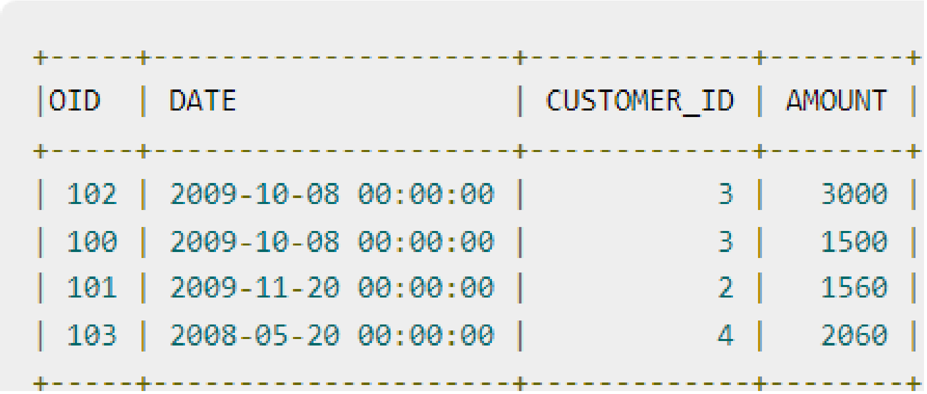
Example 2: full join

Table 1 – CUSTOMERS Table is as follows.

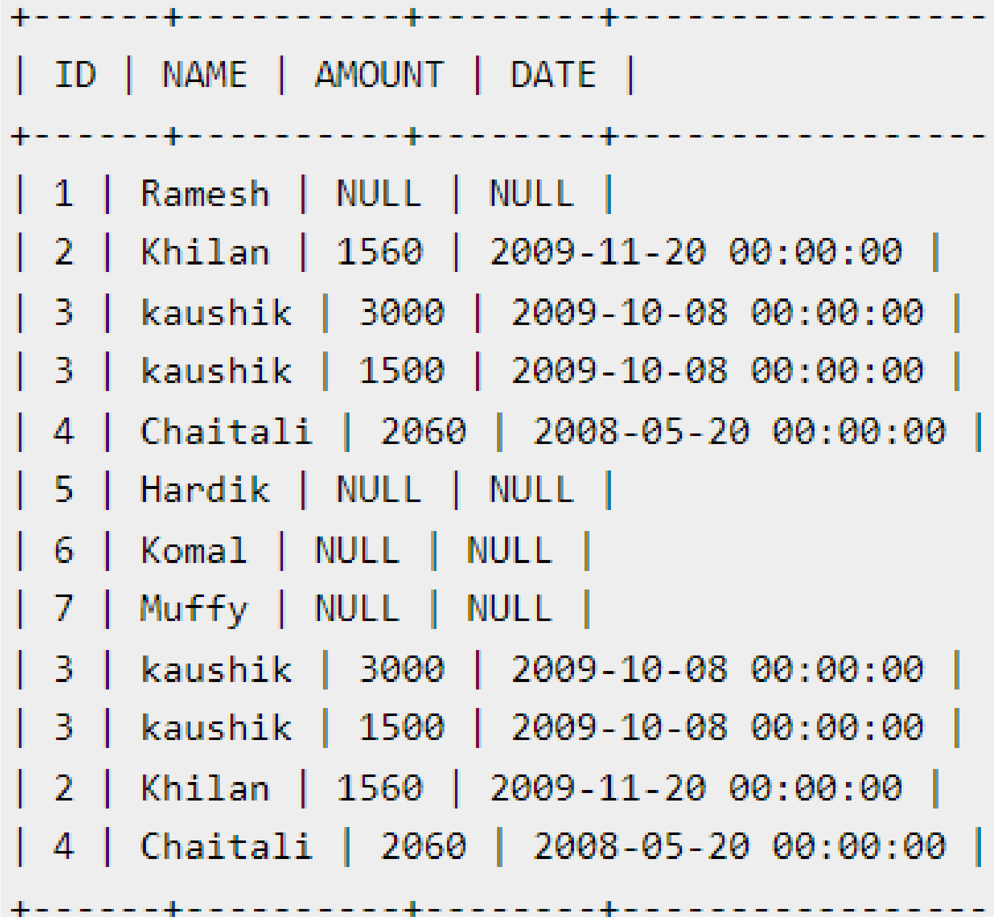


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali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

Table 2 – ORDERS Table is as follows.



```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```



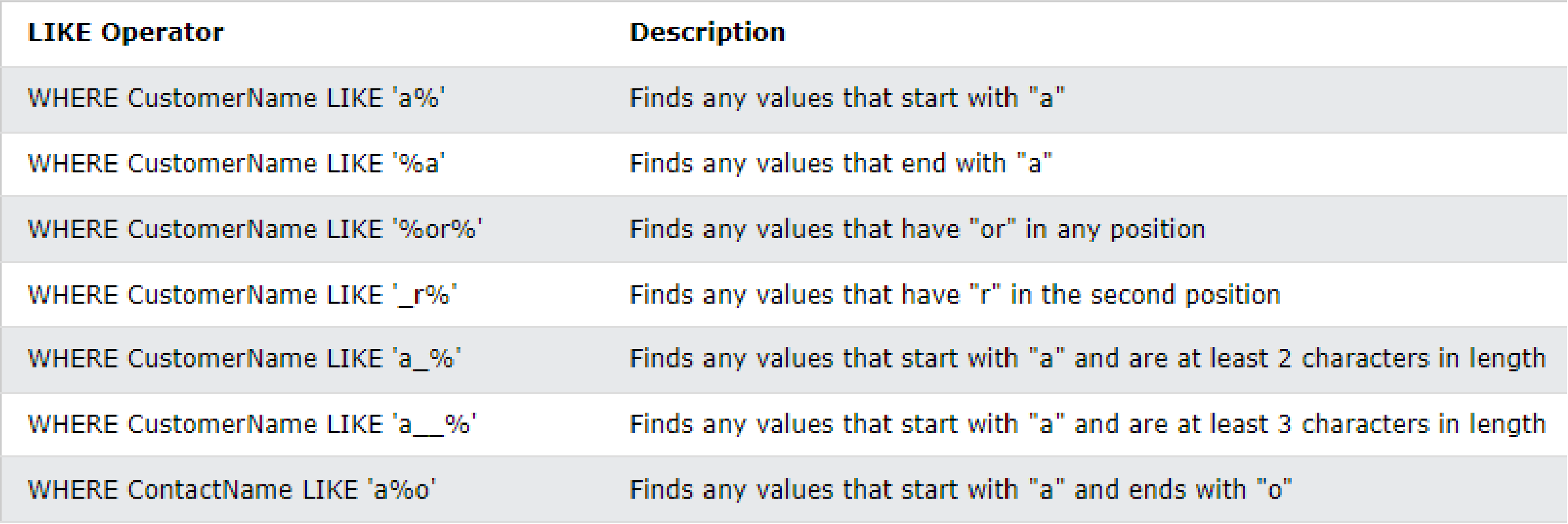
```
+------+-----------+--------+----------------
| ID | NAME | AMOUNT | DATE |
+------+-----------+--------+----------------
| 1 | Ramesh | NULL | NULL |
| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |
| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |
| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |
| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |
| 5 | Hardik | NULL | NULL |
| 6 | Komal | NULL | NULL |
| 7 | Muffy | NULL | NULL |
| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |
| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |
| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |
| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |
+------+-----------+--------+----------------
```

Logical Operators

LIKE condition

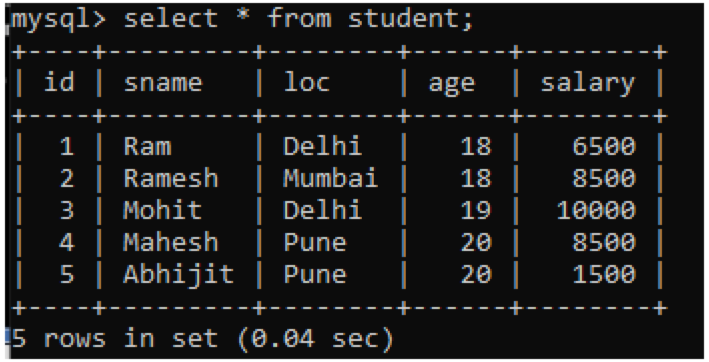
- LIKE condition is used to perform pattern matching to find the correct result.
- It is used in SELECT, INSERT, UPDATE and DELETE statement with the combination of WHERE clause.
- **Syntax:**
- expression LIKE pattern [**ESCAPE** 'escape_character']
- Parameters
- **expression:** It specifies a column or field.
- **pattern:** It is a character expression that contains pattern matching.
- **escape_character:** It is optional. It allows you to test for literal instances of a wildcard character such as % or _. If you do not provide the escape_character, MySQL assumes that "\" is the escape_character.

Different LIKE operators with '%' and '_' wildcards:



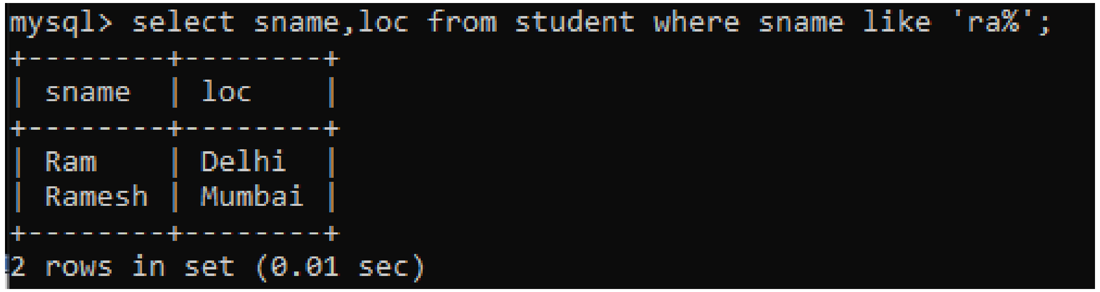
LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Examples



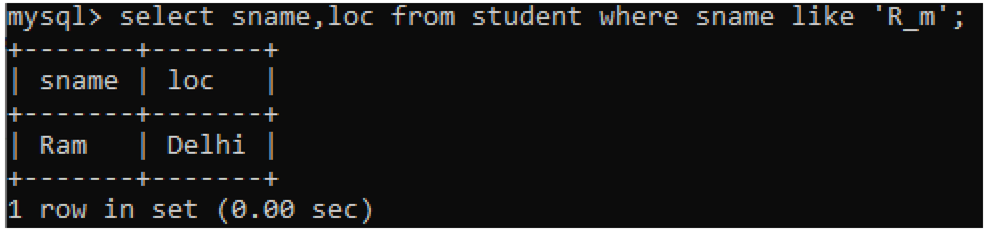
```
mysql> select * from student;
+----+---------+--------+------+--------+
| id | sname   | loc    | age  | salary |
+----+---------+--------+------+--------+
|  1 | Ram     | Delhi  |   18 |   6500 |
|  2 | Ramesh  | Mumbai |   18 |   8500 |
|  3 | Mohit   | Delhi  |   19 |  10000 |
|  4 | Mahesh  | Pune   |   20 |   8500 |
|  5 | Abhijit | Pune   |   20 |   1500 |
+----+---------+--------+------+--------+
5 rows in set (0.04 sec)
```

- 1) Using % (percent) Wildcard:



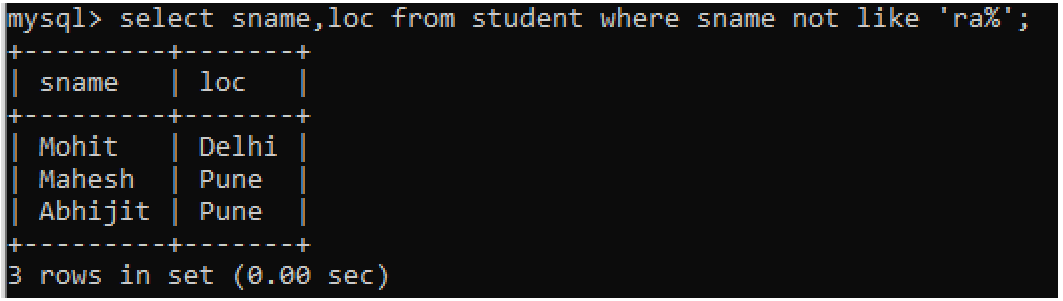
```
mysql> select sname,loc from student where sname like 'ra%';
+---------+--------+
| sname   | loc    |
+---------+--------+
| Ram     | Delhi  |
| Ramesh  | Mumbai |
+---------+--------+
2 rows in set (0.01 sec)
```

- 2) Using _ (Underscore) Wildcard:



```
mysql> select sname,loc from student where sname like 'R_m';
+-------+-------+
| sname | loc   |
+-------+-------+
| Ram   | Delhi |
+-------+-------+
1 row in set (0.00 sec)
```

- 3) using not like:



```
mysql> select sname,loc from student where sname not like 'ra%';
+---------+-------+
| sname   | loc   |
+---------+-------+
| Mohit   | Delhi |
| Mahesh  | Pune  |
| Abhijit | Pune  |
+---------+-------+
3 rows in set (0.00 sec)
```

IN Condition

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.

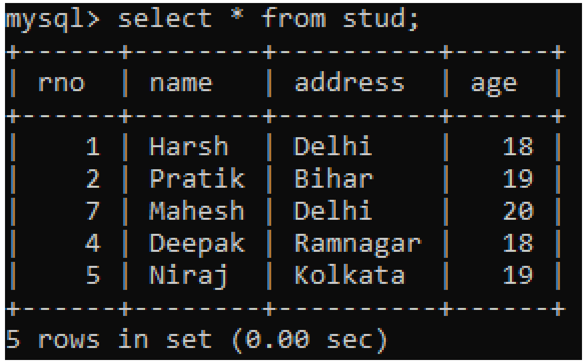
Syntax

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* IN (*value1*, *value2*, ...);

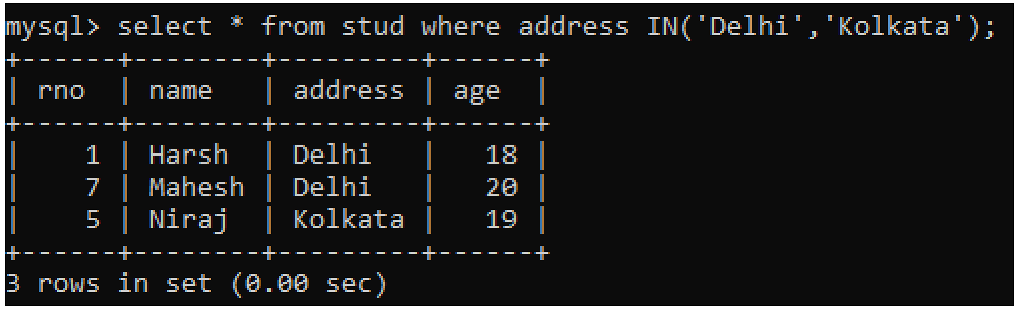
OR

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* IN (*SELECT STATEMENT*);

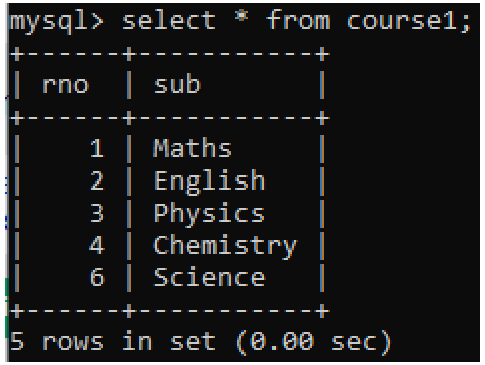
Examples



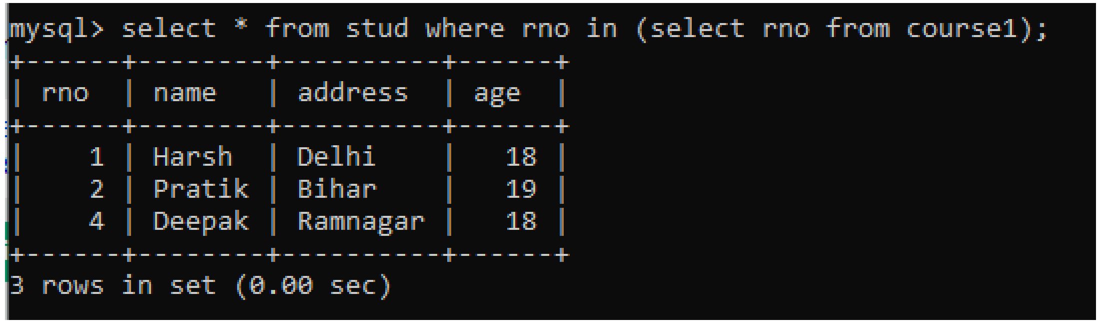
```
mysql> select * from stud;
+------+--------+----------+------+
| rno  | name   | address  | age  |
+------+--------+----------+------+
|    1 | Harsh  | Delhi    |   18 |
|    2 | Pratik | Bihar    |   19 |
|    7 | Mahesh | Delhi    |   20 |
|    4 | Deepak | Ramnagar |   18 |
|    5 | Niraj  | Kolkata  |   19 |
+------+--------+----------+------+
5 rows in set (0.00 sec)
```



```
mysql> select * from stud where address IN('Delhi','Kolkata');
+------+--------+---------+------+
| rno  | name   | address | age  |
+------+--------+---------+------+
|    1 | Harsh  | Delhi   |   18 |
|    7 | Mahesh | Delhi   |   20 |
|    5 | Niraj  | Kolkata |   19 |
+------+--------+---------+------+
3 rows in set (0.00 sec)
```



```
mysql> select * from course1;
+------+-----------+
| rno  | sub       |
+------+-----------+
|    1 | Maths     |
|    2 | English   |
|    3 | Physics   |
|    4 | Chemistry |
|    6 | Science   |
+------+-----------+
5 rows in set (0.00 sec)
```

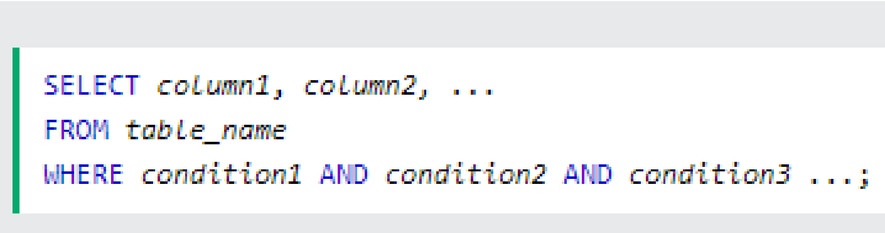


```
mysql> select * from stud where rno in (select rno from course1);
+------+--------+----------+------+
| rno  | name   | address  | age  |
+------+--------+----------+------+
|    1 | Harsh  | Delhi    |   18 |
|    2 | Pratik | Bihar    |   19 |
|    4 | Deepak | Ramnagar |   18 |
+------+--------+----------+------+
3 rows in set (0.00 sec)
```

AND, OR and NOT Operators

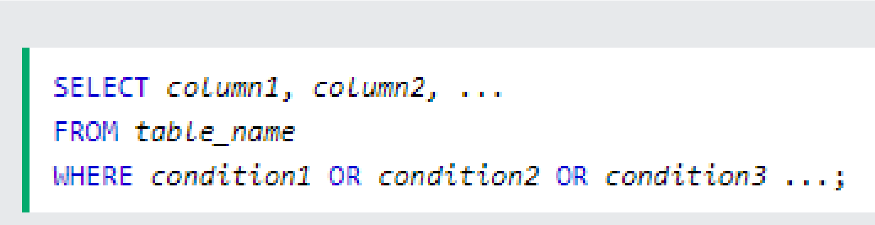
- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition:
    - The AND operator displays a record if all the conditions separated by AND are TRUE.
    - The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

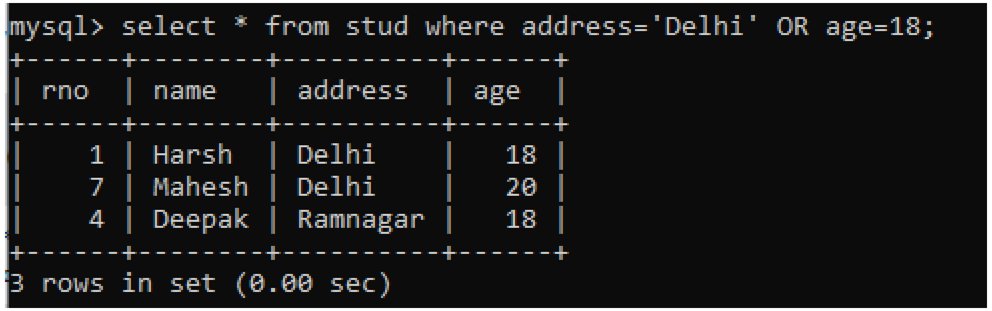


```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

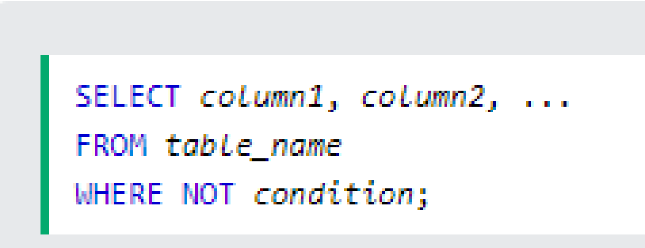


```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

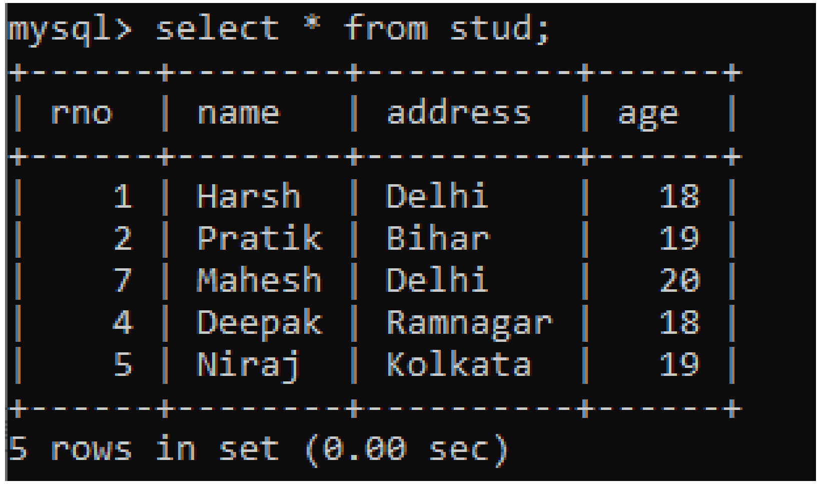


```
mysql> select * from stud where address='Delhi' OR age=18;
+------+--------+----------+------+
| rno  | name   | address  | age  |
+------+--------+----------+------+
|    1 | Harsh  | Delhi    |   18 |
|    7 | Mahesh | Delhi    |   20 |
|    4 | Deepak | Ramnagar |   18 |
+------+--------+----------+------+
3 rows in set (0.00 sec)
```

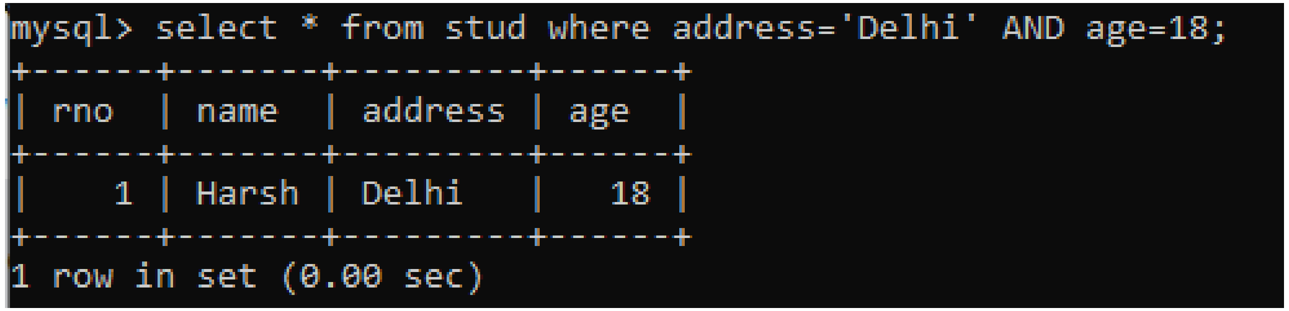
NOT Syntax



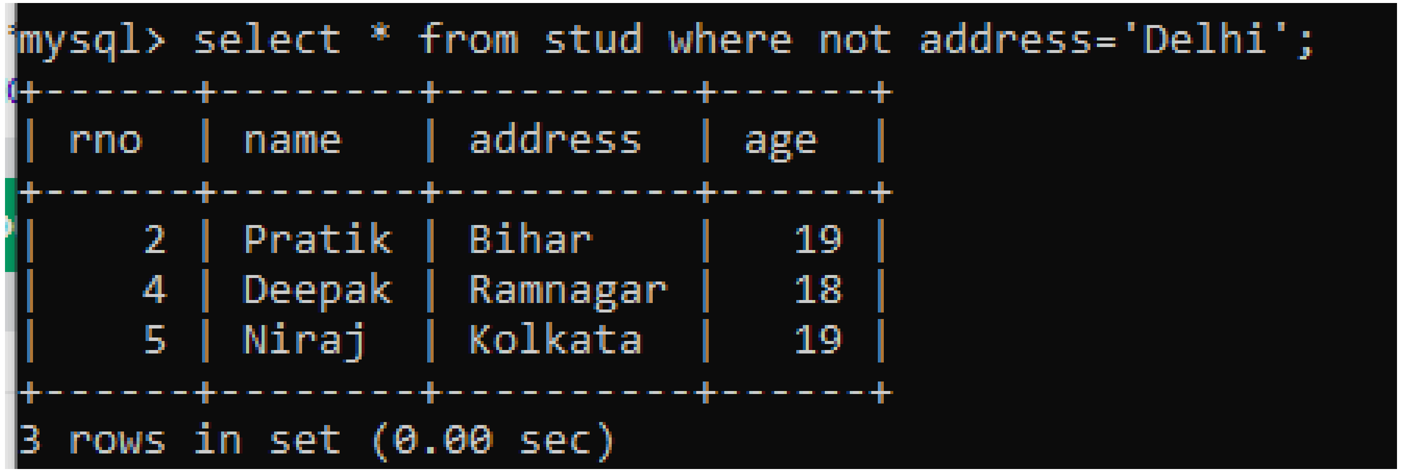
```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```



```
mysql> select * from stud;
+------+--------+----------+------+
| rno  | name   | address  | age  |
+------+--------+----------+------+
|    1 | Harsh  | Delhi    |   18 |
|    2 | Pratik | Bihar    |   19 |
|    7 | Mahesh | Delhi    |   20 |
|    4 | Deepak | Ramnagar |   18 |
|    5 | Niraj  | Kolkata  |   19 |
+------+--------+----------+------+
5 rows in set (0.00 sec)
```



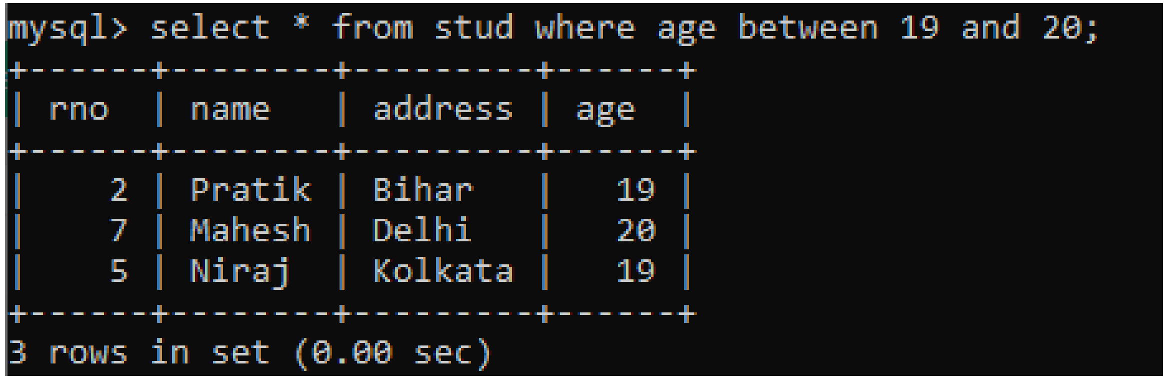
```
mysql> select * from stud where address='Delhi' AND age=18;
+------+-------+---------+------+
| rno  | name  | address | age  |
+------+-------+---------+------+
|    1 | Harsh | Delhi   |   18 |
+------+-------+---------+------+
1 row in set (0.00 sec)
```



```
mysql> select * from stud where not address='Delhi';
+------+--------+----------+------+
| rno  | name   | address  | age  |
+------+--------+----------+------+
|    2 | Pratik | Bihar    |   19 |
|    4 | Deepak | Ramnagar |   18 |
|    5 | Niraj  | Kolkata  |   19 |
+------+--------+----------+------+
3 rows in set (0.00 sec)
```

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.
- Syntax
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *column_name* BETWEEN *value1* AND *value2;*



```
mysql> select * from stud where age between 19 and 20;
+------+--------+---------+------+
| rno  | name   | address | age  |
+------+--------+---------+------+
|    2 | Pratik | Bihar   |   19 |
|    7 | Mahesh | Delhi   |   20 |
|    5 | Niraj  | Kolkata |   19 |
+------+--------+---------+------+
3 rows in set (0.00 sec)
```

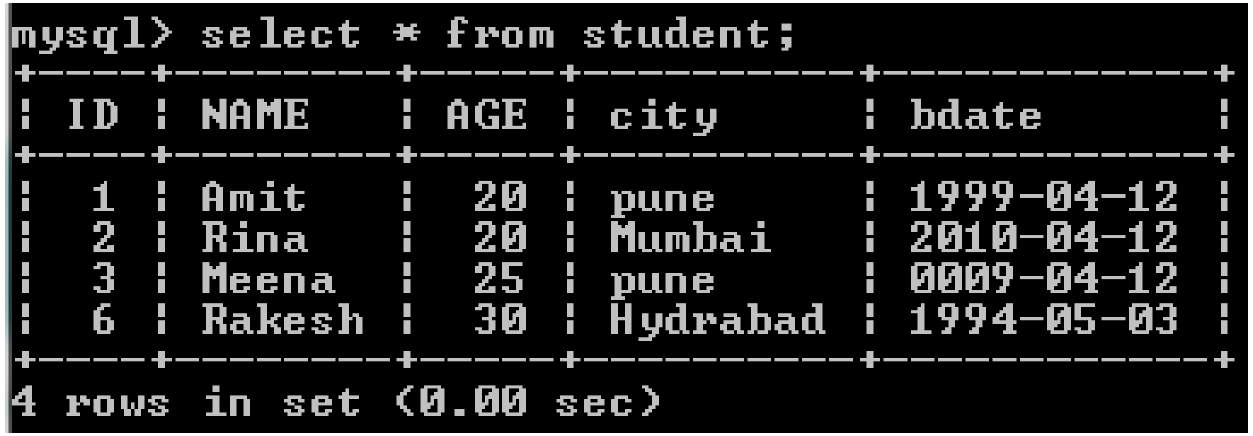
Nested Queries in SQL

Subqueries

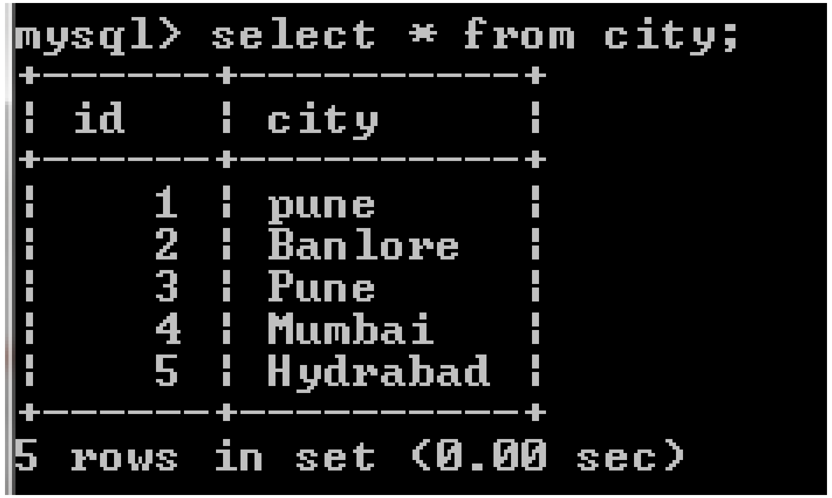
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- There are a few rules that subqueries must follow −
- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Nested Queries

- In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query.
- Here we use **STUDENT & city** tables for understanding nested queries.

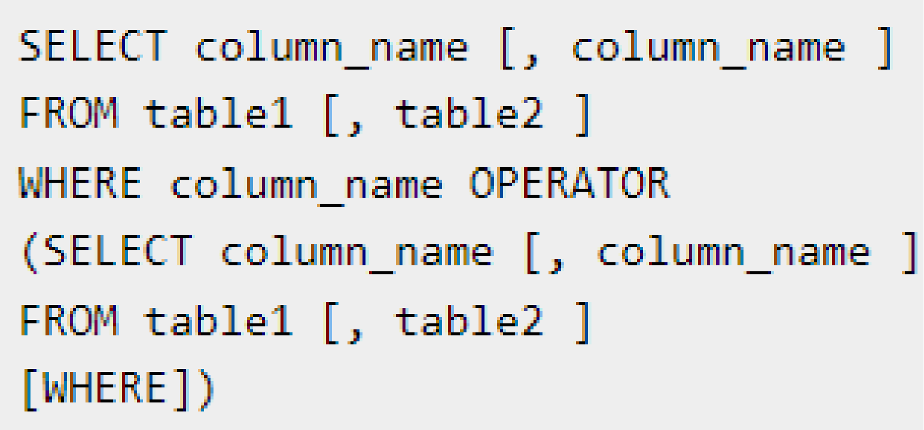


```
mysql> select * from student;
+----+--------+-----+----------+------------+
| ID | NAME   | AGE | city     | bdate      |
+----+--------+-----+----------+------------+
|  1 | Amit   |  20 | pune     | 1999-04-12 |
|  2 | Rina   |  20 | Mumbai   | 2010-04-12 |
|  3 | Meena  |  25 | pune     | 0009-04-12 |
|  6 | Rakesh |  30 | Hydrabad | 1994-05-03 |
+----+--------+-----+----------+------------+
4 rows in set (0.00 sec)
```



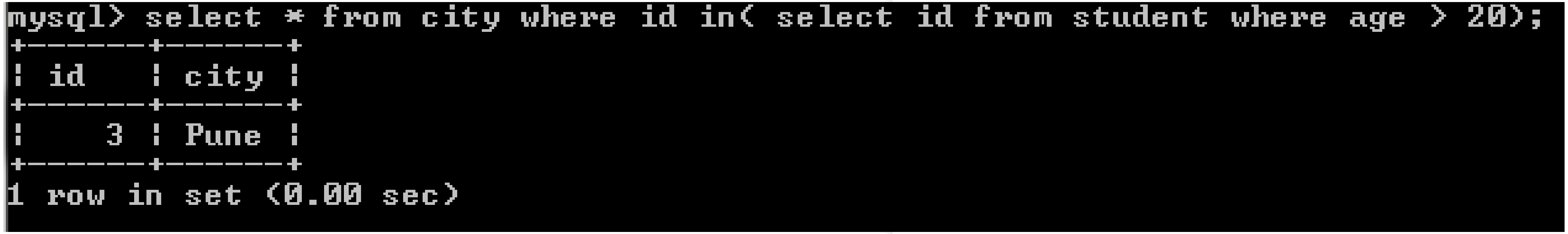
```
mysql> select * from city;
+------+----------+
| id   | city     |
+------+----------+
|    1 | pune     |
|    2 | Banlore  |
|    3 | Pune     |
|    4 | Mumbai   |
|    5 | Hydrabad |
+------+----------+
5 rows in set (0.00 sec)
```

Subqueries with the SELECT Statement



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

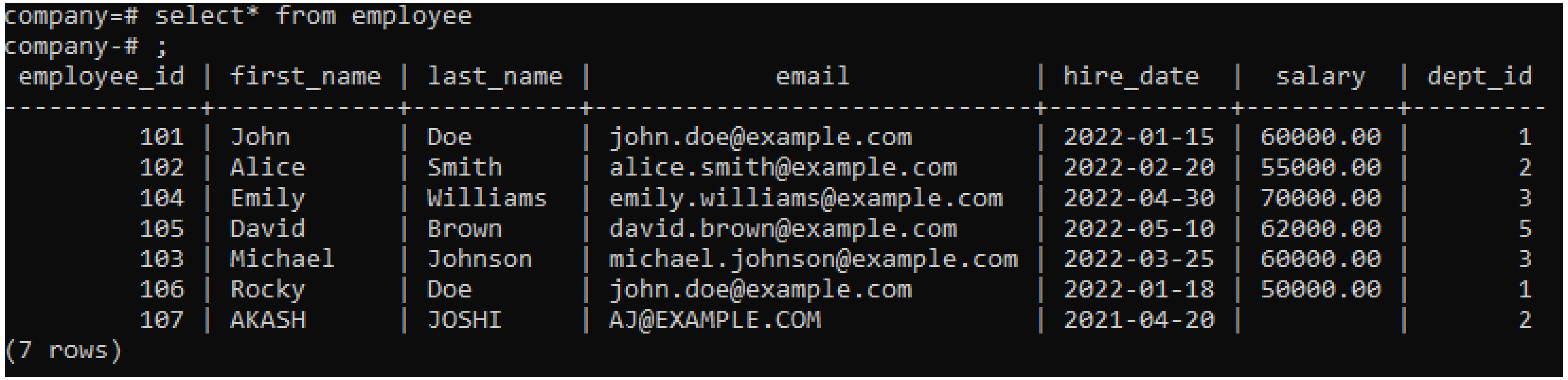
Retrieve id and city name of all students those age is greater than 20.



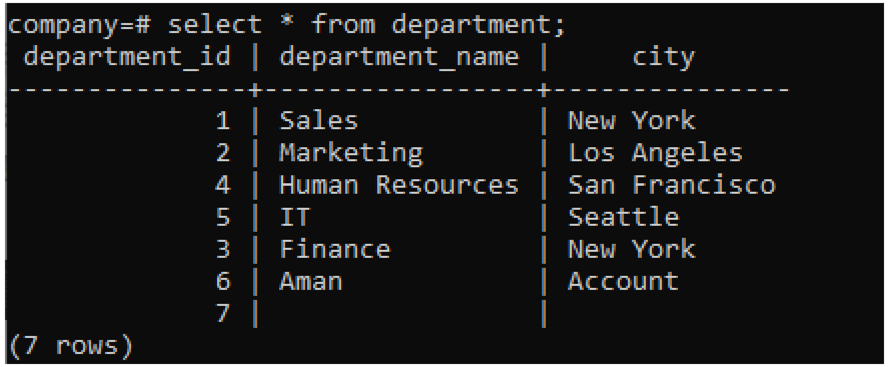
```
mysql> select * from city where id in( select id from student where age > 20);
+-------+-------+
| id    | city  |
+-------+-------+
|     3 | Pune  |
+-------+-------+
1 row in set (0.00 sec)
```

Subquery with delete statement

- To delete record from employees whose works in ‘IT’ Department.
- **Delete from employee where dept_id in(select department_id from department where department_name=‘IT’;**

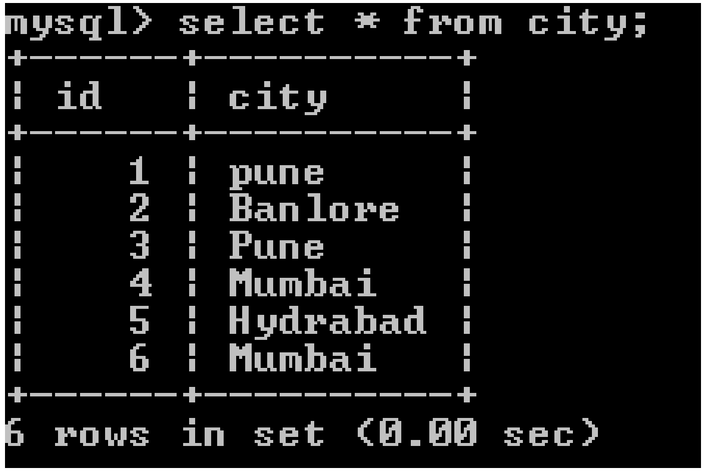


```
company=# select* from employee
company-# ;
 employee_id | first_name | last_name |            email             | hire_date  |  salary  | dept_id
-------------+------------+-----------+------------------------------+------------+----------+---------
         101 | John       | Doe       | john.doe@example.com         | 2022-01-15 | 60000.00 |       1
         102 | Alice      | Smith     | alice.smith@example.com      | 2022-02-20 | 55000.00 |       2
         104 | Emily      | Williams  | emily.williams@example.com   | 2022-04-30 | 70000.00 |       3
         105 | David      | Brown     | david.brown@example.com      | 2022-05-10 | 62000.00 |       5
         103 | Michael    | Johnson   | michael.johnson@example.com  | 2022-03-25 | 60000.00 |       3
         106 | Rocky      | Doe       | john.doe@example.com         | 2022-01-18 | 50000.00 |       1
         107 | AKASH      | JOSHI     | AJ@EXAMPLE.COM               | 2021-04-20 |          |       2
(7 rows)
```

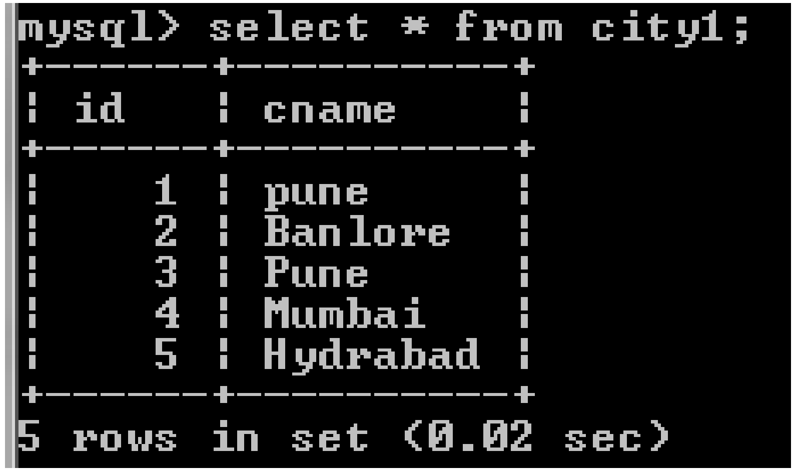


```
company=# select * from department;
 department_id | department_name |     city
---------------+-----------------+---------------
             1 | Sales           | New York
             2 | Marketing       | Los Angeles
             4 | Human Resources | San Francisco
             5 | IT              | Seattle
             3 | Finance         | New York
             6 | Aman            | Account
             7 |                 |
(7 rows)
```

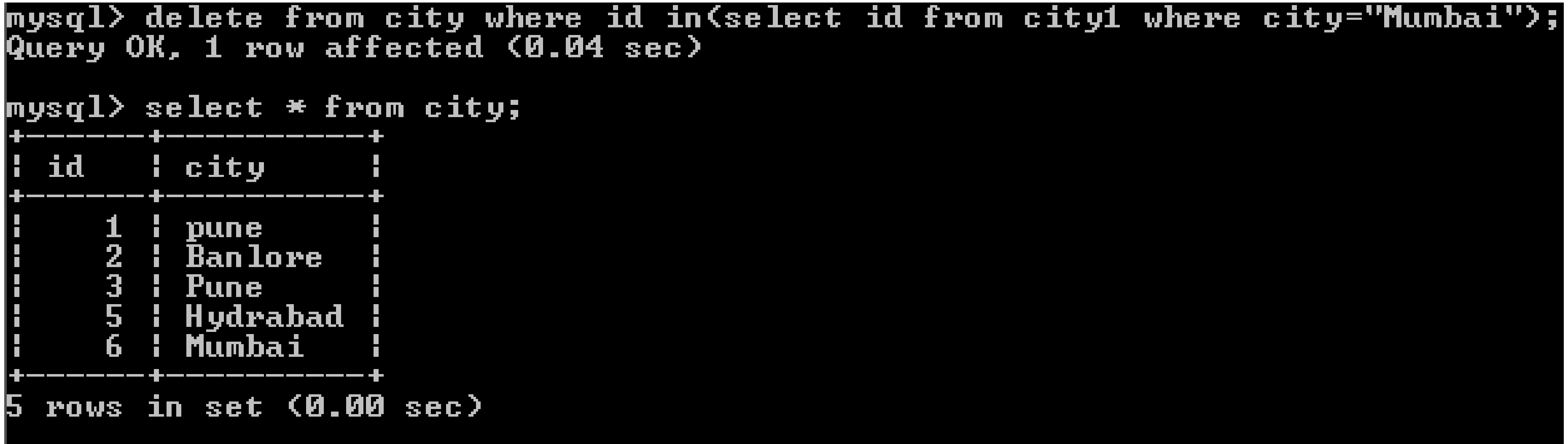
Example 2:



```
mysql> select * from city;
+-------+-----------+
| id    | city      |
+-------+-----------+
|     1 | pune      |
|     2 | Banlore   |
|     3 | Pune      |
|     4 | Mumbai    |
|     5 | Hydrabad  |
|     6 | Mumbai    |
+-------+-----------+
6 rows in set (0.00 sec)
```



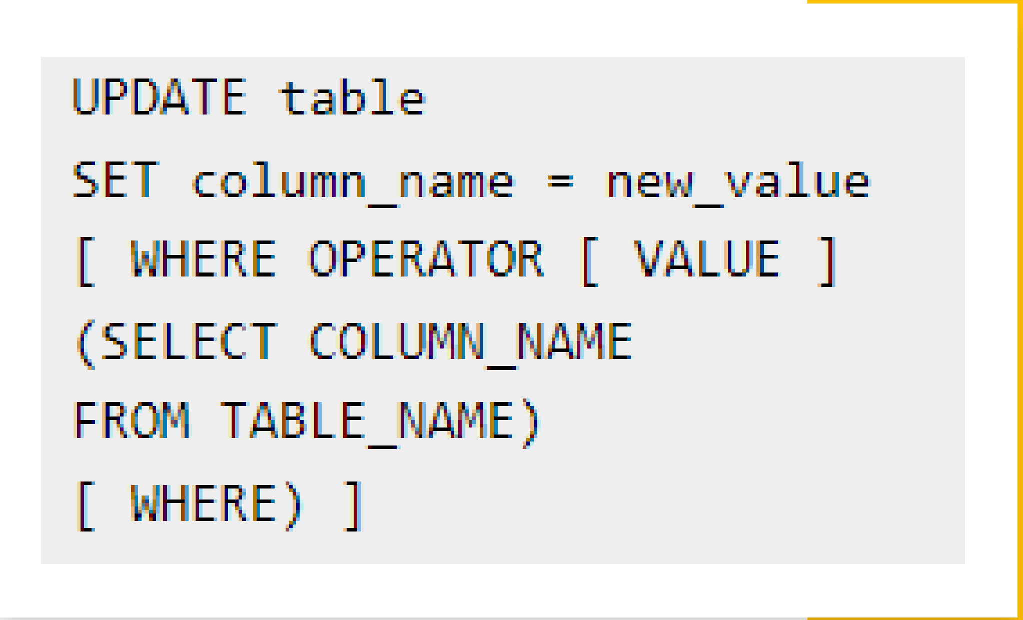
```
mysql> select * from city1;
+-------+-----------+
| id    | cname     |
+-------+-----------+
|     1 | pune      |
|     2 | Banlore   |
|     3 | Pune      |
|     4 | Mumbai    |
|     5 | Hydrabad  |
+-------+-----------+
5 rows in set (0.02 sec)
```



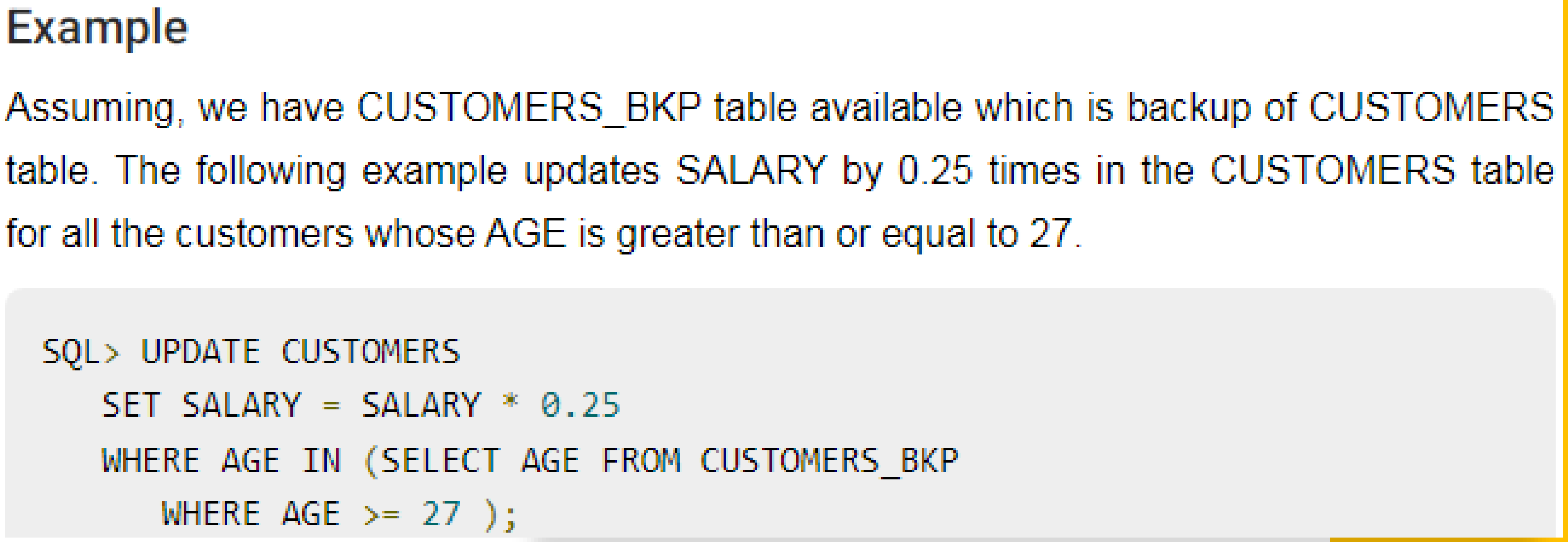
```
mysql> delete from city where id in(select id from city1 where city="Mumbai");
Query OK, 1 row affected (0.04 sec)

mysql> select * from city;
+-------+-----------+
| id    | city      |
+-------+-----------+
|     1 | pune      |
|     2 | Banlore   |
|     3 | Pune      |
|     5 | Hydrabad  |
|     6 | Mumbai    |
+-------+-----------+
5 rows in set (0.00 sec)
```

Subqueries with the UPDATE Statement

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



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



Example

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

```
SQL> UPDATE CUSTOMERS
   SET SALARY = SALARY * 0.25
   WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
      WHERE AGE >= 27 );
```

Example

- Consider the following employee database.
- Employee(emp_name, street,city,date_of_joining)
- Works(emp_name,company_name,salary)
- Company(company_name,city)
- Manages(emp_name,manager_name)
- Write SQL queries for following:

1. Modify the database so that ‘Deepa’lives in ‘Pune’;
2. Give all employees of ‘Aarya corporation’ a 10% rise in salary.
3. Display all employees who joined in the month of ‘March’;
4. Find all employees who earn more than average salary of all employees of their company.