Structured Query Languauge

Syllabus

Structured Query Language: introduction, Data Definition Language and Data Manipulation Language, data type (char(n), varchar(n), int, float, date), constraints (not null, unique, primarykey),

create database, use database, show databases, drop database, show tables, create table, describe table, alter table (add and remove an attribute, add and remove primary key), drop table, insert, delete, select, operators (mathematical, relational and logical), aliasing, distinct clause, where clause, in, between, order by, meaning of null, is null, is not null, like, update command, delete command

Syllabus

Aggregate functions (max, min, avg, sum, count), group by, having clause,

joins: Cartesian product on two tables, equi-join and natural join

Introduction to SQL

- Till now, we have discussed about the Database concepts and its need, and then we discussed about the Relational Database Management System (RDBMS) and its purpose.
- We also know that RDBMS allows us to store, retrieve and manipulate data on the database through queries.
- There are many RDBMS such as MySQL, Microsoft SQL Server, PostgreSQL, Oracle, etc. that allow us to create a database consisting of relations.

Introduction to SQL

- We already seen in topic “Data Handling in Python”, in order to access and manipulate the data from the files (text, binary or csv), we need to write the programs in Python.
- Similarly, in order to access and manipulate data from the database, we need to write commands in a query language.
- The Structured Query Language (SQL) is the most popular query language used by major Relational Database Managements systems such as MySQL, ORACLE, SQL Server, etc.

Why SQL?

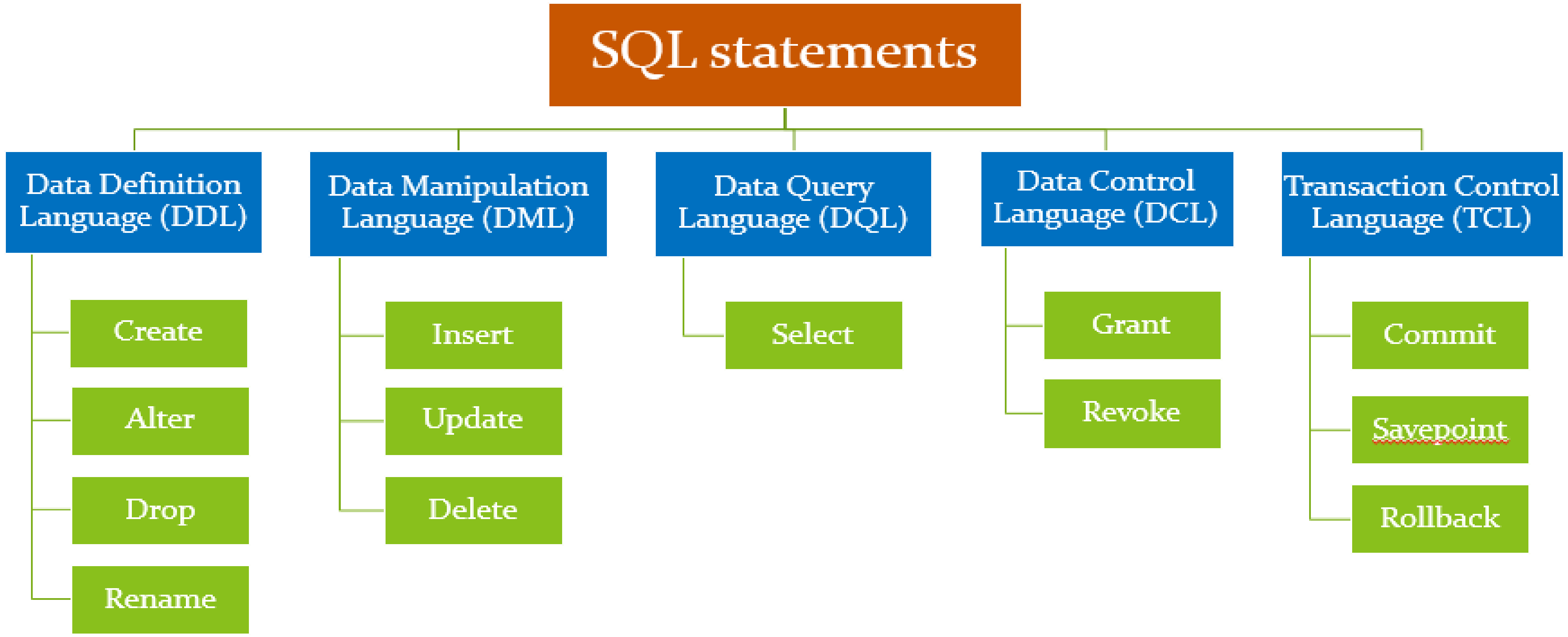
- SQL is easy to learn as the statements comprise of descriptive English words and are not case sensitive.

- We can create and interact with a database using SQL easily. Benefit of using SQL is that we do not have to specify how to get the data from the database. Rather, we simply specify what is to be retrieved, and SQL does the rest.

Why SQL?

- Although SQL is called a query language, SQL can do much more, besides querying.
- SQL provides statements for
  - defining the structure of the data,
  - manipulating data in the database,
  - declaring constraints and
  - retrieving data from the database in various ways, depending on our requirements.

Classification of SQL statements


SQL statements
Data Definition Language (DDL)
Create
Alter
Drop
Rename
Data Manipulation Language (DML)
Insert
Update
Delete
Data Query Language (DQL)
Select
Data Control Language (DCL)
Grant
Revoke
Transaction Control Language (TCL)
Commit
Savepoint
Rollback

Important points:

- Some important points to be kept in mind while using SQL:
  - SQL is case insensitive. For example, the column names ‘salary’ and ‘SALARY’ are the same for SQL.
  - SQL statements are always end with a semi-colon (;)
  - In order to enter multiline SQL statements, we don’t write “;” after the first line. We press the Enter key to continue on the next line.

Data types and Constraints in SQL

- We know that a database consists of one or more relations and each relation (table) is made up of attributes (column) and each attribute has a data type which indicates the type of data value that an attribute can have.

- Data type of an attribute also decides the operations that can be performed on the data of that attribute.
  For e.g.)
  arithmetic operations can be performed on numeric data but not on character data

Data type	Description
char (n)	char is of fixed length i.e., char(n) will have space reserve for n characters where n can be any value from 0 to 255. For e.g.) we have assigned char(50) to an attribute student name, and data 'Himanshu' has only 8 characters, it means MySQL fills the remaining 42 characters with spaces on the right of data.
varchar (n)	varchar(n) data type specifies character type data of length n where n could be any value from 0 to 65535. But unlike char, varchar(n) is a variable-length data type. That is, declaring varchar(30) means a maximum of 30 characters can be stored but the actual allocated bytes will depend on the length of entered string. For e.g.) 'city' in varchar(30) will occupy space needed to store 4 characters only.
int	int data type specifies an integer value and each int value occupies 4 bytes of storage. The range of unsigned values allowed in a 4 byte integer type are 0 to 4,294,967,295. For values larger than that, we have to use BIGINT, which occupies 8 bytes.
float	float holds numbers with decimal points and each float value occupies 4 bytes
date	In MySQL dates are stored in 'YYYY-MM-DD' format using date data type where YYYY is the 4-digit year, MM is the 2-digit month and DD is the 2-digit month. The supported range for date data type is '1000-01-01' to '9999-12-31'

Data types and Constraints in SQL

- Constraints are the certain types of restrictions on the data values that an attribute can have.
- Constraints are used to ensure correctness of data. However, it is not mandatory to define constraints for each attribute of a table.
- Some of the commonly used constraints in SQL are Unique, Not Null, Primary key etc.

Data types and Constraints in SQL

Constraint	Description
Not Null	Ensures that a column cannot have NULL values where NULL means missing/ unknown/not applicable value.
Unique	Ensures that all the values in a column are distinct/unique
Default	A default value specified for the column if no value is provided
Primary Key	The column which can uniquely identify each row/record in a table.
Foreign Key	The column which refers to value of an attribute defined as primary key in another table

SQL for Data Definition

- In order to be able to store data we need to first define the relation schema. Defining a schema includes
    - creating a relation and giving name to a relation,
    - identifying the attributes in a relation,
    - deciding upon the datatype for each attribute and
    - also specify the constraints as per the requirements.
- Sometimes, we may require to make changes to the relation schema also.

SQL for Data Definition

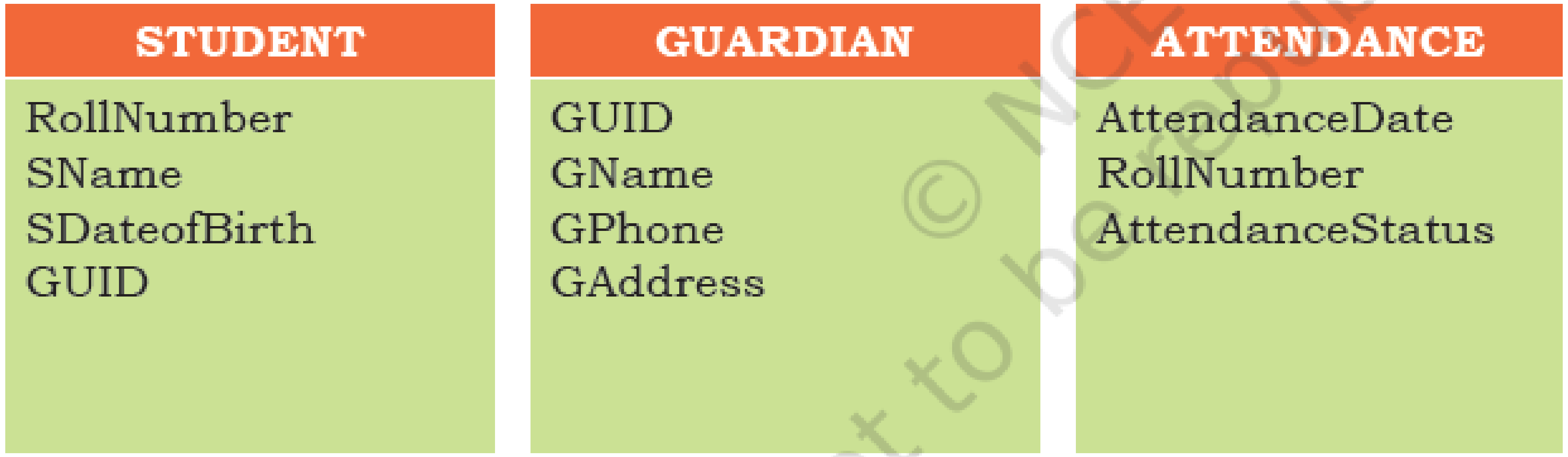
- SQL allows us to write statements for defining, modifying and deleting relation schemas. These are part of Data Definition Language (DDL).
- Commonly used DDL statements are:
  - create
  - alter
  - drop

Note:

DDL statements are auto committed i.e., changes will become permanent and database objects created are available to all users

SQL for Data Definition

- We will use the StudentAttendance database that we discussed in the previous chapter



STUDENT	GUARDIAN	ATTENDANCE
RollNumber	GUID	AttendanceDate
SName	GName	RollNumber
SDateofBirth	GPhone	AttendanceStatus
GUID	GAddress	

SQL for Data Definition: Create Statement

- We have already know that in a database, data are stored in relations or tables. Hence, we can say that a database is a collection of tables.

- The Create statement is used to create a database and its tables (relations).

Note:

Before creating a database, we should be clear about

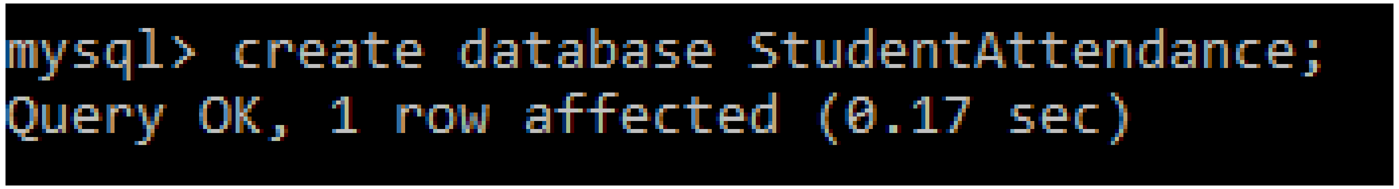
- the number of tables the database will have,
- the columns (attributes) in each table along with the data type of each column, and its constraint, if any.

SQL for Data Definition: Create Database

- To create a database, we use the CREATE DATABASE statement as shown in the following syntax:

  CREATE DATABASE databasename;

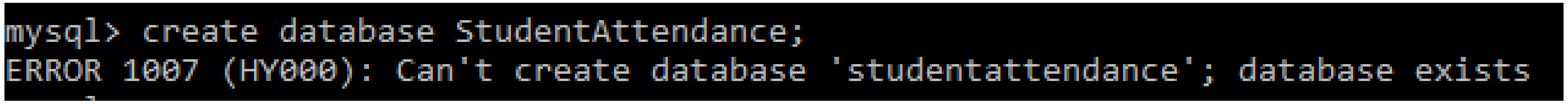
- To create a database called StudentAttendance, we will type following command at MySQL prompt.



```
mysql> create database StudentAttendance;
Query OK, 1 row affected (0.17 sec)
```

SQL for Data Definition: Create Database

- If we try to create a database, which is already been created, then we will get the following error:



```
mysql> create database StudentAttendance;
ERROR 1007 (HY000): Can't create database 'studentattendance'; database exists
```

Note:

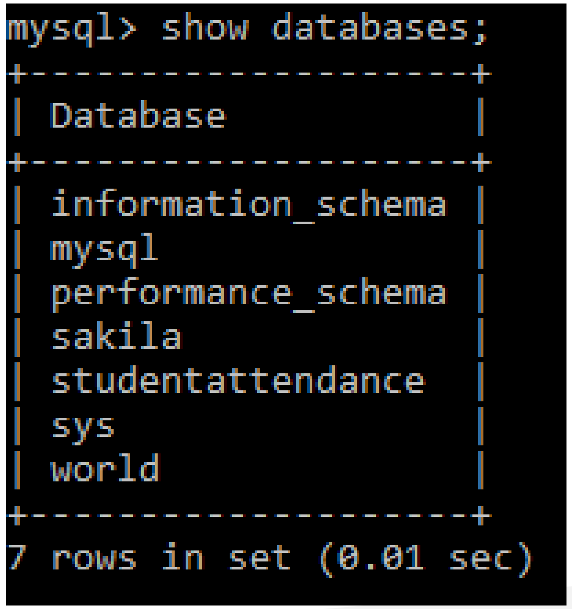
In LINUX environment, names for database and tables are case-sensitive whereas in WINDOWS, there is no such differentiation.

However, as a good practice, it is suggested to write database/table name in the same letter cases that were used at the time of their creation.

SQL for Data Definition: Show databases

- We know, a DBMS can manage multiple databases on one computer. Therefore, we need to select the database that we want to use.

- To know the names of existing databases, we use the statement SHOW DATABASES.

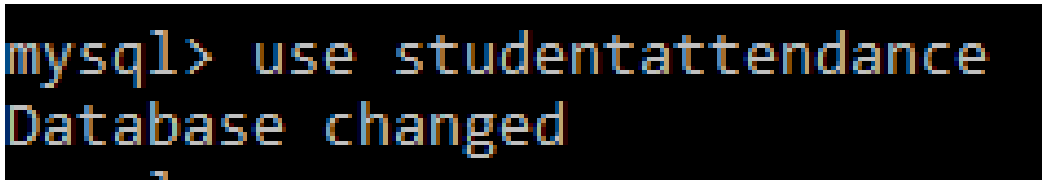


```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| studentattendance  |
| sys                |
| world              |
+--------------------+
7 rows in set (0.01 sec)
```

SQL for Data Definition: Use database

- From the listed databases, we can select the database to be used. Once the database is selected, we can proceed with creating tables or querying data.

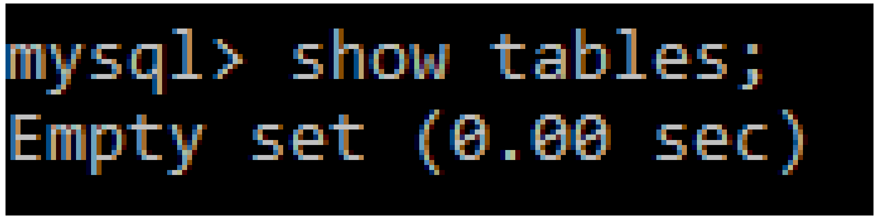
- In order to use the StudentAttendance database, the following SQL statement is required.



```
mysql> use studentattendance
Database changed
```

SQL for Data Definition: Show tables

- Initially, the created database is empty. It can be checked by using the show tables statement that lists names of all the tables within a database.



```
mysql> show tables;
Empty set (0.00 sec)
```

SQL for Data Definition: Create table

- After creating a database StudentAttendance, we need to define relations in this database and specify attributes for each relation along with data type and constraint (if any) for each attribute.

- This is done using the CREATE TABLE statement:

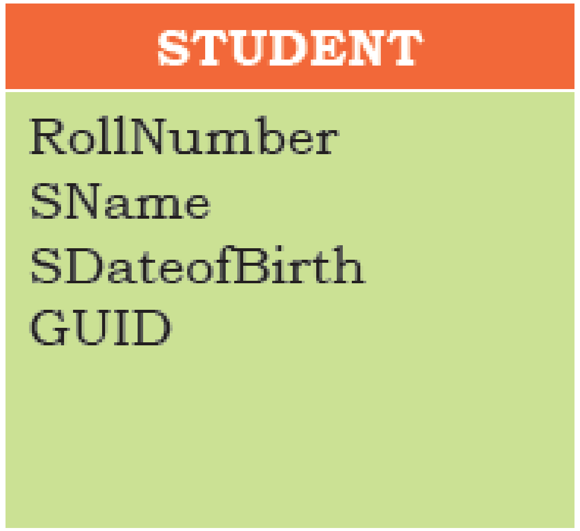
```
CREATE TABLE tablename(
attributename1 datatype constraint,
attributename2 datatype constraint,
:
attributenameN datatype constraint);
```

SQL for Data Definition: Create table

- It is important to observe the following points with respect to the CREATE TABLE statement:
  - The number of columns in a table defines the degree of that relation, which is denoted by N.
  - Attribute name specifies the name of the column in the table.
  - Datatype specifies the type of data that an attribute can hold.
  - Constraint indicates the restrictions imposed on the values of an attribute.
  - By default, each attribute can take NULL values except for the primary key.

SQL for Data Definition: Create table

- Let us identify data types of the attributes of table STUDENT along with their constraints (if any).



STUDENT
RollNumber
SName
SDateofBirth
GUID

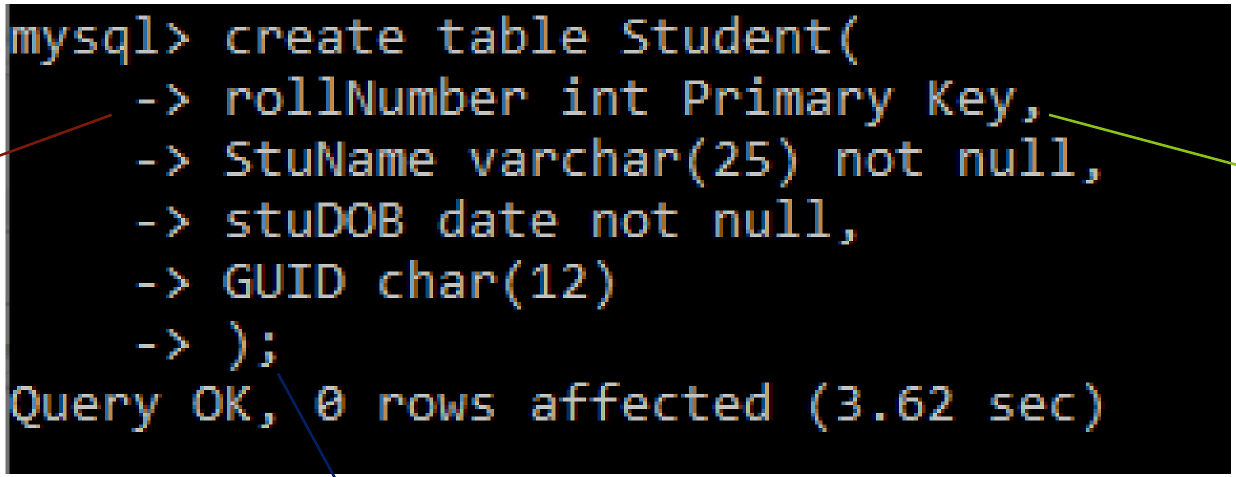
SQL for Data Definition: Create table

- Let us identify data types of the attributes of table STUDENT along with their constraints (if any).

Attribute Name	Expected Data	Data type	Constraint
RollNumber	Numeric value consisting of maximum 3 digits	INT	Primary Key
StuName	Variable length string of maximum 25 characters	VARCHAR(25)	Not Null
StuDOB	Date value	DATE	Not Null
GUID	Numeric value consisting of 12 digits	CHAR(12)	Foreign Key

SQL for Data Definition: Create table

- Once we have identified the data types and constraints along with the attribute names, let us create Student table using below SQL commands:



```
mysql> create table Student(
    -> rollNumber int Primary Key,
    -> StuName varchar(25) not null,
    -> stuDOB date not null,
    -> GUID char(12)
    -> );
Query OK, 0 rows affected (3.62 sec)
```

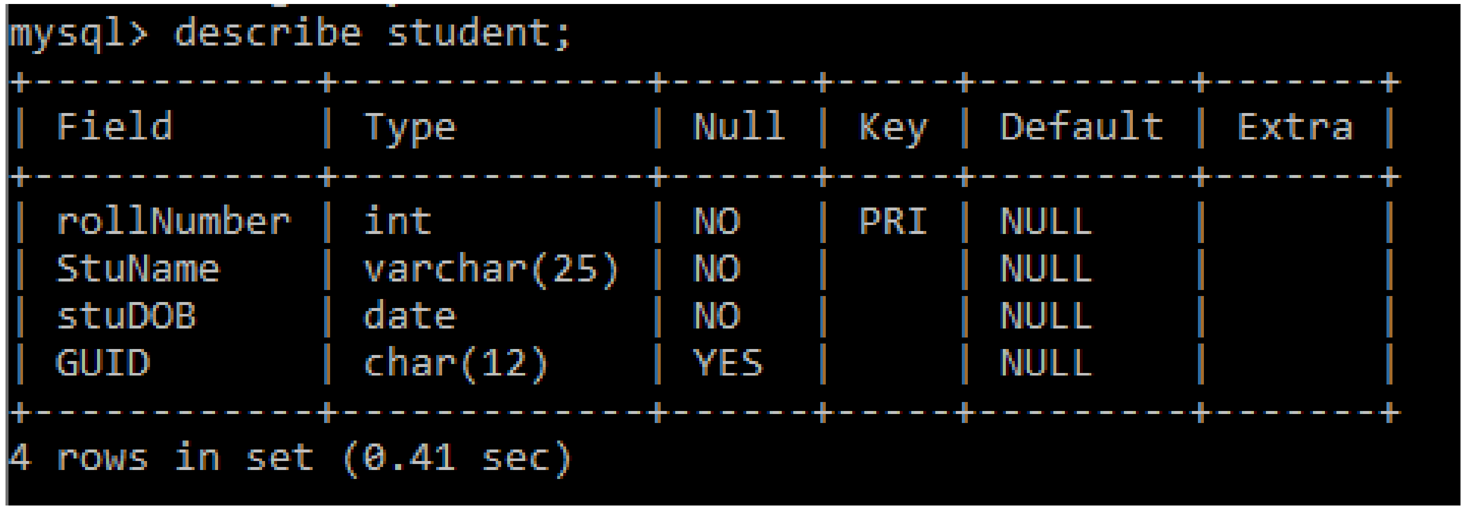
The arrow (->) is an interactive continuation prompt

“,” is used to separate two attributes

each statement in SQL terminates with a semi-colon (;)

SQL for Data Definition: Describe table

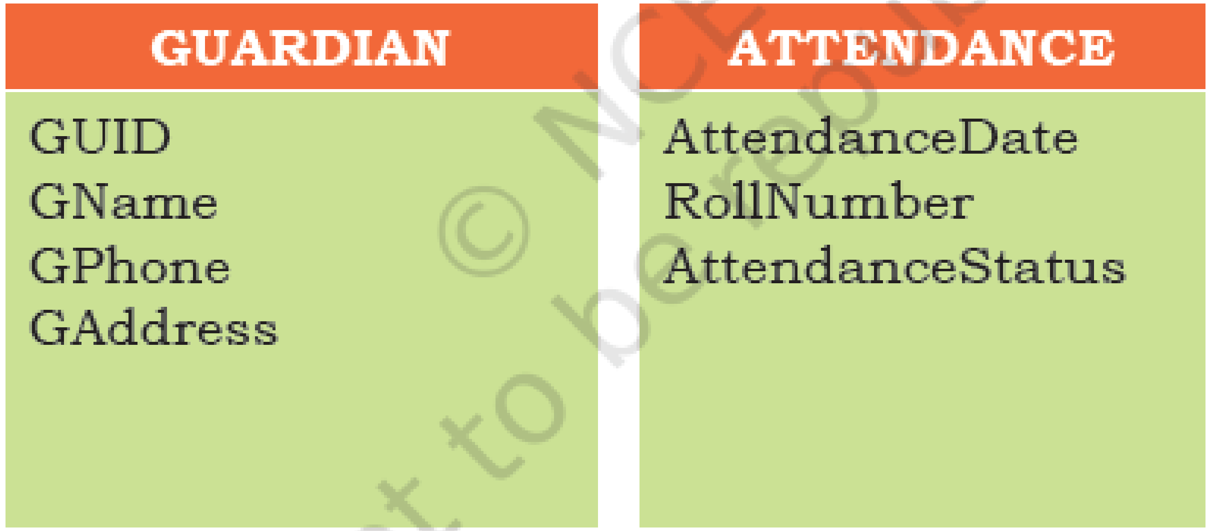
- Once we have created the table, we can structure of the newly created or already created table using the DESCRIBE statement



```
mysql> describe student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollNumber | int         | NO   | PRI | NULL    |       |
| StuName    | varchar(25) | NO   |     | NULL    |       |
| stuDOB     | date        | NO   |     | NULL    |       |
| GUID       | char(12)    | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.41 sec)
```

SQL for Data Definition: Create table

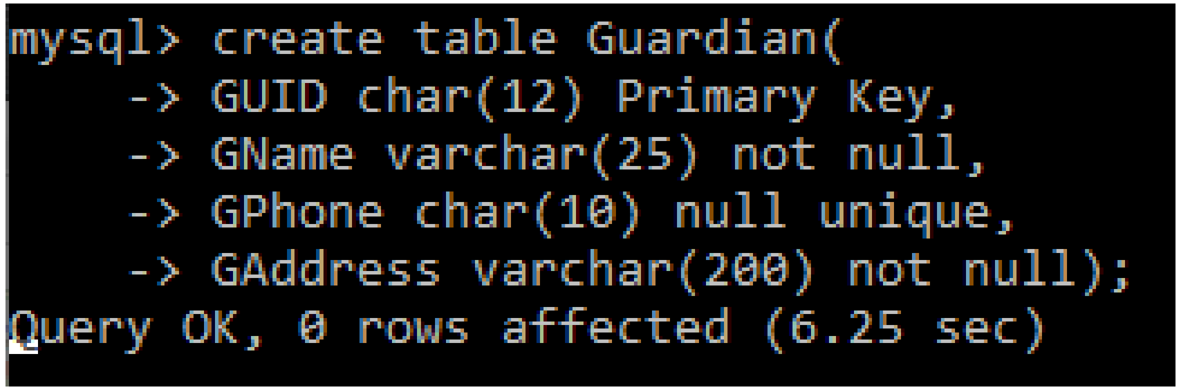
- Assignment: Create below two tables under studentAttendance database with appropriate data types and constraints



GUARDIAN	ATTENDANCE
GUID	AttendanceDate
GName	RollNumber
GPhone	AttendanceStatus
GAddress	

SQL for Data Definition: Create table

GUARDIAN
GUID
GName
GPhone
GAddress



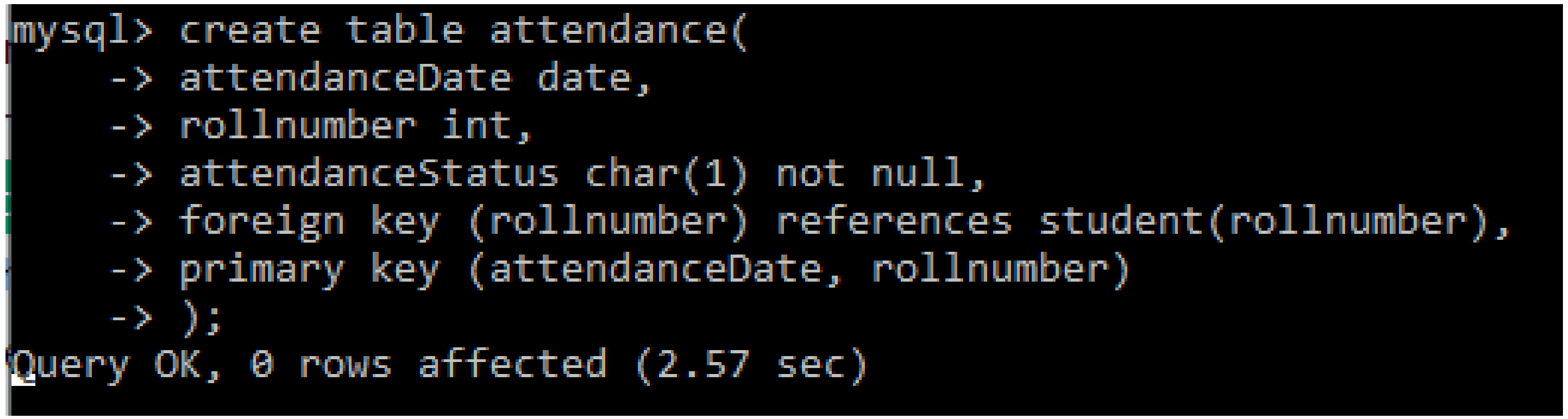
```
mysql> create table Guardian(
    -> GUID char(12) Primary Key,
    -> GName varchar(25) not null,
    -> GPhone char(10) null unique,
    -> GAddress varchar(200) not null);
Query OK, 0 rows affected (6.25 sec)
```

Attribute Name	Data expected to be stored	Data type	Constraint
GUID	Numeric value consisting of 12 digit Aadhaar number	CHAR (12)	PRIMARY KEY
GName	Variant length string of maximum 20 characters	VARCHAR(20)	NOT NULL
GPhone	Numeric value consisting of 10 digits	CHAR(10)	NULL UNIQUE
GAddress	Variant length String of size 30 characters	VARCHAR(30)	NOT NULL

SQL for Data Definition: Create table

ATTENDANCE

AttendanceDate
RollNumber
AttendanceStatus



```
mysql> create table attendance(
    -> attendanceDate date,
    -> rollnumber int,
    -> attendanceStatus char(1) not null,
    -> foreign key (rollnumber) references student(rollnumber),
    -> primary key (attendanceDate, rollnumber)
    -> );
Query OK, 0 rows affected (2.57 sec)
```

Attribute Name	Data expected to be stored	Data type	Constraint
AttendanceDate	Date value	DATE	PRIMARY KEY*
RollNumber	Numeric value consisting of maximum 3 digits	INT	PRIMARY KEY* FOREIGN KEY
AttendanceStatus	'P' for present and 'A' for absent	CHAR(1)	NOT NULL

means part of composite primary key.

SQL for Data Definition: Drop statement

- Sometimes a table in a database or the database itself needs to be removed. We can use a DROP statement to remove a database or a table permanently from the system.

  Syntax to drop a table:

  ```
  DROP TABLE table_name;
  ```

  Syntax to drop a database:

  ```
  DROP DATABASE database_name;
  ```

Note:

Using the DROP statement to remove a database will ultimately remove all the tables within it. So, one should be very cautious while using this statement as it cannot be undone.

SQL for Data Definition: Alter table

- After creating a table, we may realise that we need to add/remove an attribute or to modify the datatype of an existing attribute or to add constraint in attribute.
- In all such cases, we need to change or alter the structure (schema) of the table by using the alter statement.

Alter table: a.) adding primary key

- In order to add a primary key into the existing table, use the following syntax:

  alter table table_name add primary key (attribute_name);

Alter table: b.) adding foreign key

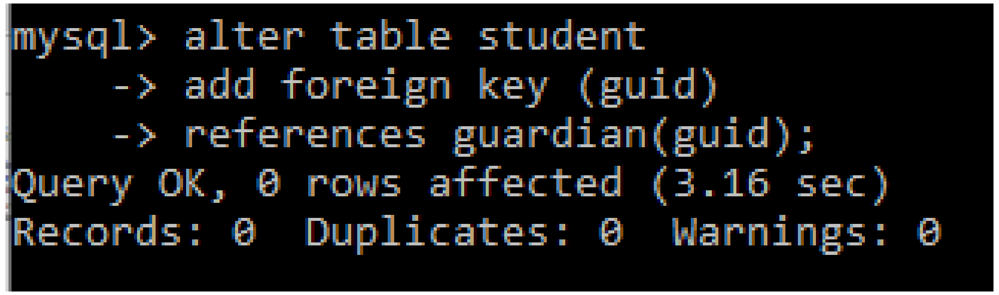
- Once primary keys are added, the next step is to add foreign keys to the relation (if any).

- Following points need to be observed while adding foreign key to a relation:
    - The referenced relation must be already created.
    - The referenced attribute(s) must be part of the primary key of the referenced relation.
    - Data types and size of referenced and referencing attributes must be the same.

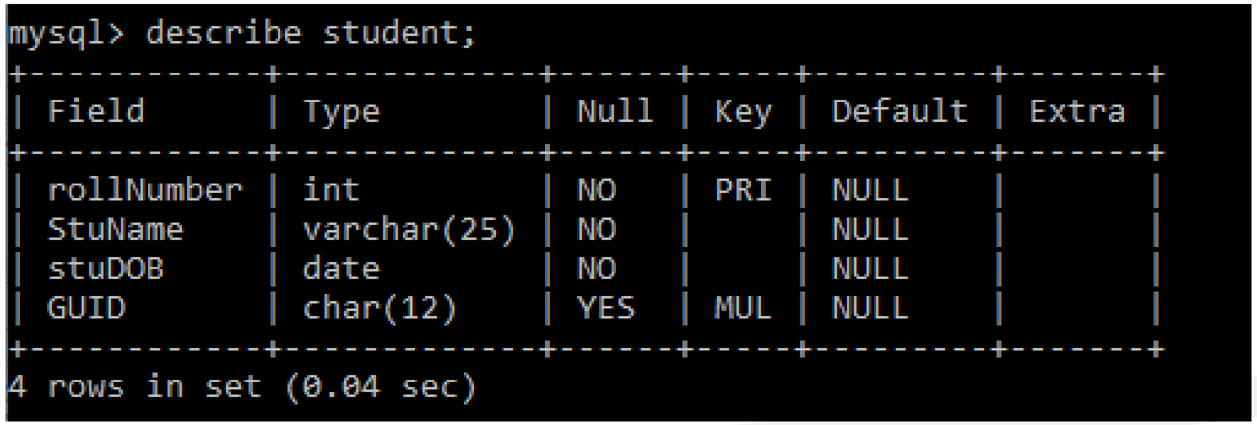
Alter table: b.) adding foreign key

- In order to add a foreign key into the existing table, use the following syntax:

  alter table table_name add foreign key (attribute_name) references referenced_table_name (attribute_name);



```
mysql> alter table student
    -> add foreign key (guid)
    -> references guardian(guid);
Query OK, 0 rows affected (3.16 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

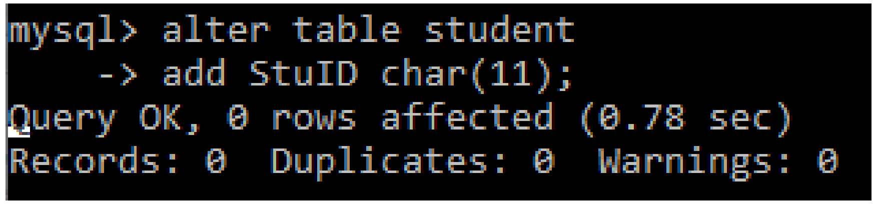


```
mysql> describe student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollNumber | int         | NO   | PRI | NULL    |       |
| StuName    | varchar(25) | NO   |     | NULL    |       |
| stuDOB     | date        | NO   |     | NULL    |       |
| GUID       | char(12)    | YES  | MUL | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.04 sec)
```

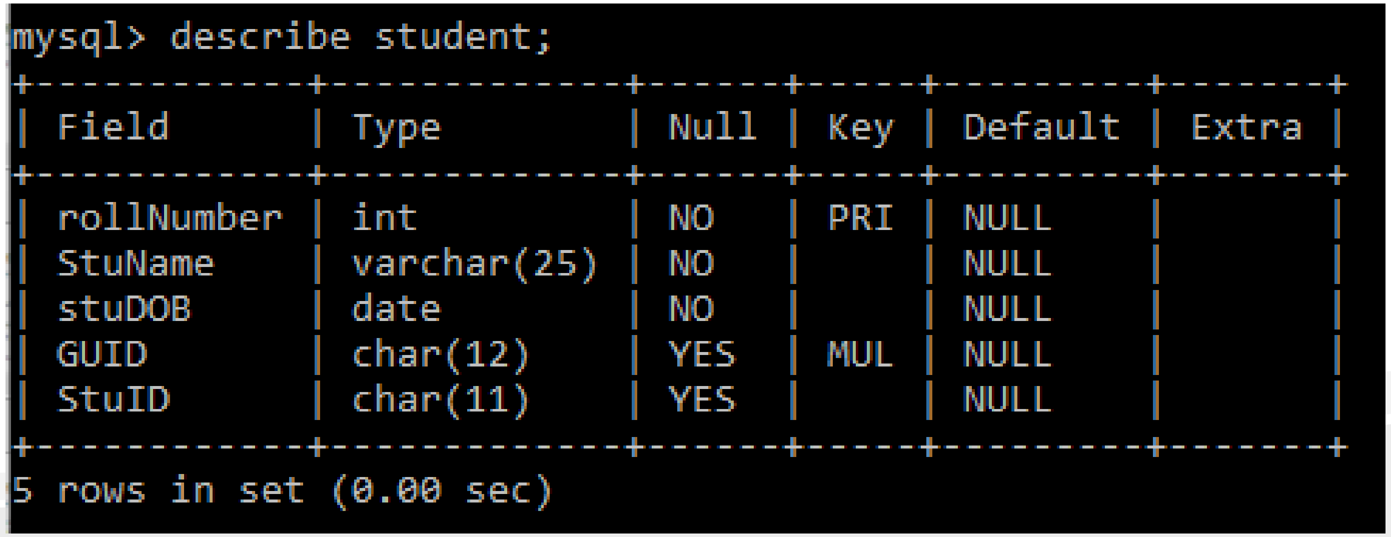
Alter table: c.) add an attribute to an existing table

- Sometimes, we may need to add an additional attribute in a table.
- It can be done using the ADD attribute statement as shown in the following syntax:

alter table table_name add attribute_name dataType;



```
mysql> alter table student
    -> add StuID char(11);
Query OK, 0 rows affected (0.78 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

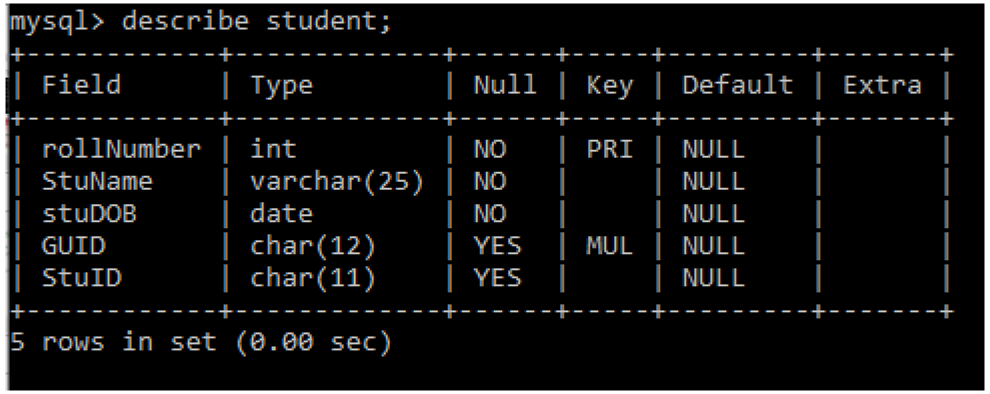


```
mysql> describe student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollNumber | int         | NO   | PRI | NULL    |       |
| StuName    | varchar(25) | NO   |     | NULL    |       |
| stuDOB     | date        | NO   |     | NULL    |       |
| GUID       | char(12)    | YES  | MUL | NULL    |       |
| StuID      | char(11)    | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

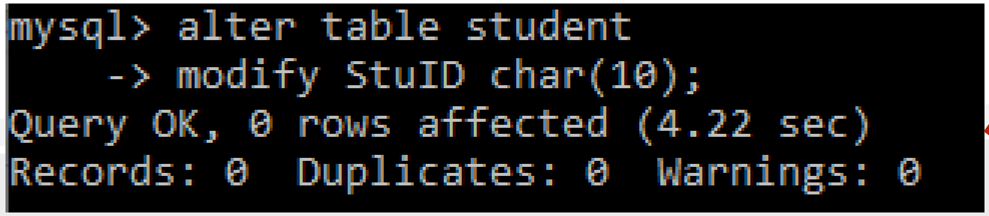
Alter table: d.) modify datatype of an attribute

- We can change data types of the existing attributes of a table using the following syntax:

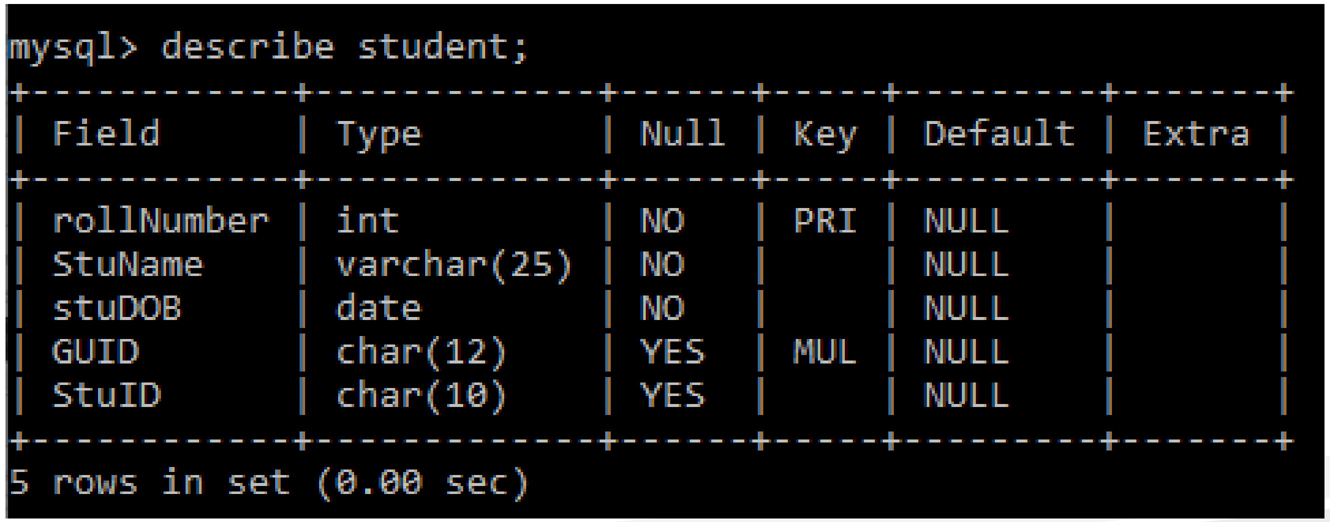
alter table table_name modify attribute_name dataType;



```
mysql> describe student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollNumber | int         | NO   | PRI | NULL    |       |
| StuName    | varchar(25) | NO   |     | NULL    |       |
| stuDOB     | date        | NO   |     | NULL    |       |
| GUID       | char(12)    | YES  | MUL | NULL    |       |
| StuID      | char(11)    | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```



```
mysql> alter table student
    -> modify StuID char(10);
Query OK, 0 rows affected (4.22 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

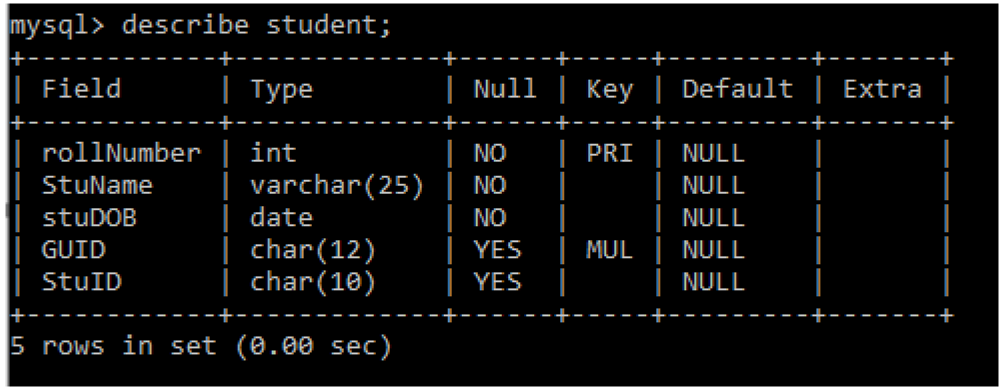


```
mysql> describe student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollNumber | int         | NO   | PRI | NULL    |       |
| StuName    | varchar(25) | NO   |     | NULL    |       |
| stuDOB     | date        | NO   |     | NULL    |       |
| GUID       | char(12)    | YES  | MUL | NULL    |       |
| StuID      | char(10)    | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

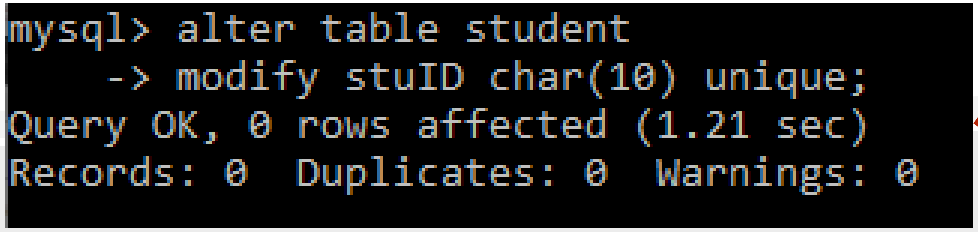
Alter table: e.) modify constraint of an attribute

- We can change constraints of the existing attributes of a table using the following syntax:

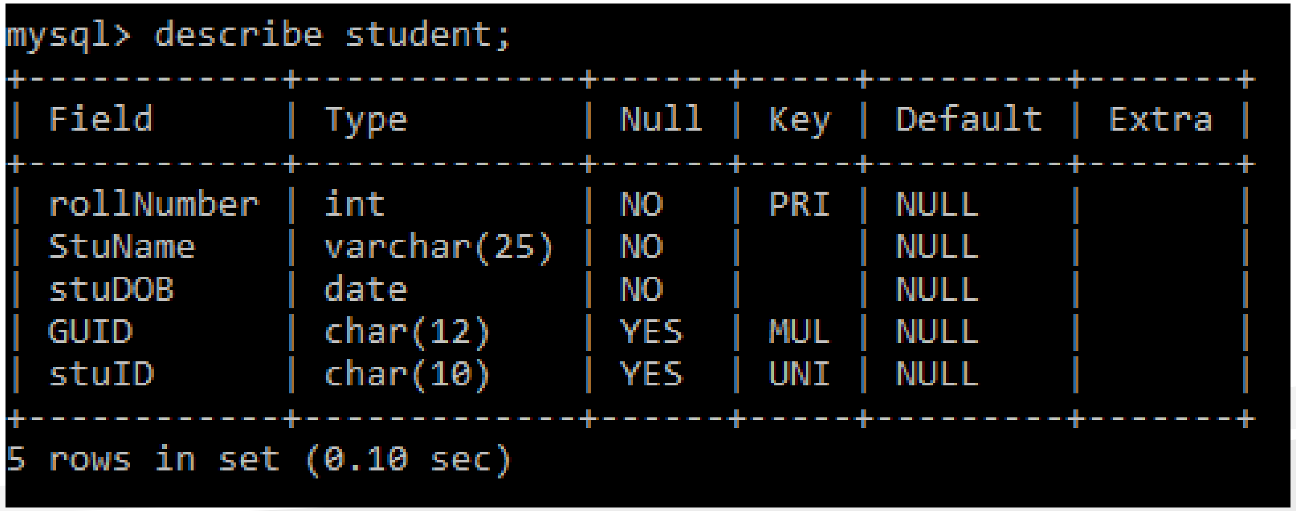
alter table table_name modify attribute_name datatype constraint;



```
mysql> describe student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollNumber | int         | NO   | PRI | NULL    |       |
| StuName    | varchar(25) | NO   |     | NULL    |       |
| stuDOB     | date        | NO   |     | NULL    |       |
| GUID       | char(12)    | YES  | MUL | NULL    |       |
| StuID      | char(10)    | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```



```
mysql> alter table student
    -> modify stuID char(10) unique;
Query OK, 0 rows affected (1.21 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

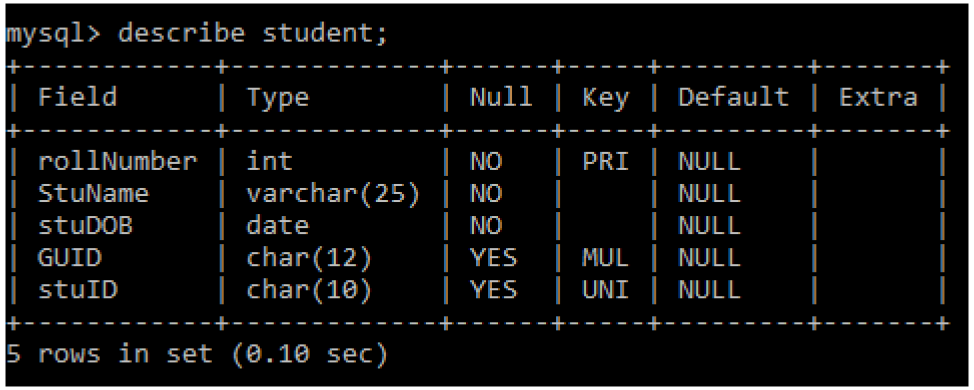


```
mysql> describe student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollNumber | int         | NO   | PRI | NULL    |       |
| StuName    | varchar(25) | NO   |     | NULL    |       |
| stuDOB     | date        | NO   |     | NULL    |       |
| GUID       | char(12)    | YES  | MUL | NULL    |       |
| stuID      | char(10)    | YES  | UNI | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.10 sec)
```

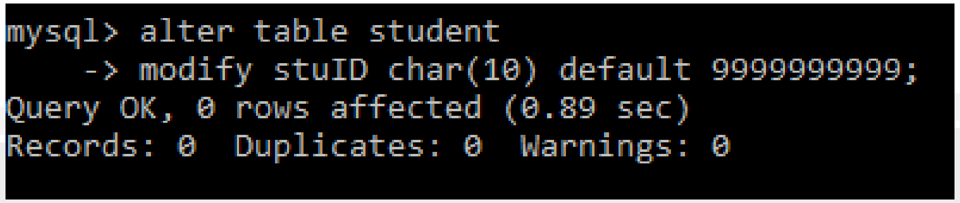
Alter table: f.) add default value to an attribute

- If we want to specify default value for an attribute, then use the following syntax:

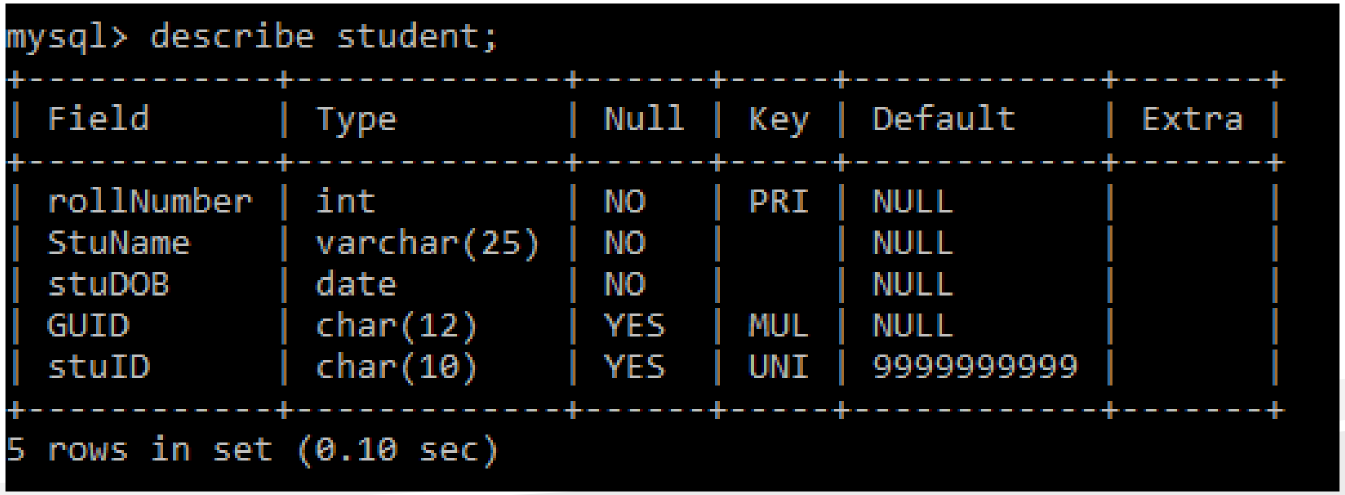
  alter table table_name modify attribute_name datatype default default_value;



```
mysql> describe student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollNumber | int         | NO   | PRI | NULL    |       |
| StuName    | varchar(25) | NO   |     | NULL    |       |
| stuDOB     | date        | NO   |     | NULL    |       |
| GUID       | char(12)    | YES  | MUL | NULL    |       |
| stuID      | char(10)    | YES  | UNI | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.10 sec)
```



```
mysql> alter table student
    -> modify stuID char(10) default 9999999999;
Query OK, 0 rows affected (0.89 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

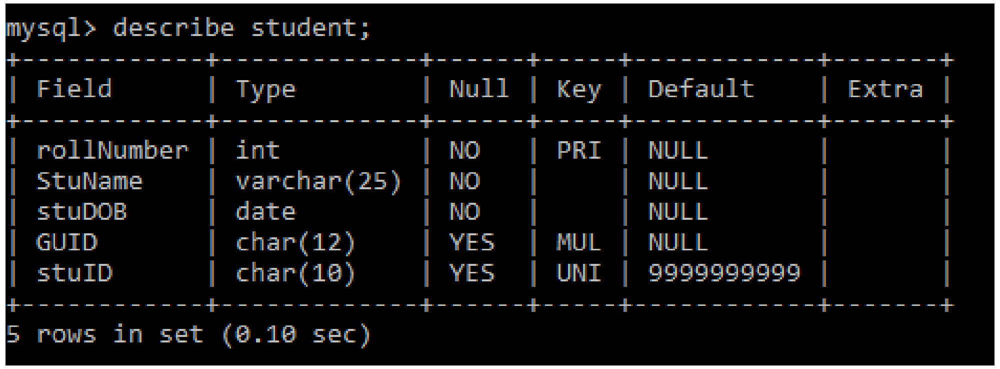


```
mysql> describe student;
+------------+-------------+------+-----+------------+-------+
| Field      | Type        | Null | Key | Default    | Extra |
+------------+-------------+------+-----+------------+-------+
| rollNumber | int         | NO   | PRI | NULL       |       |
| StuName    | varchar(25) | NO   |     | NULL       |       |
| stuDOB     | date        | NO   |     | NULL       |       |
| GUID       | char(12)    | YES  | MUL | NULL       |       |
| stuID      | char(10)    | YES  | UNI | 9999999999 |       |
+------------+-------------+------+-----+------------+-------+
5 rows in set (0.10 sec)
```

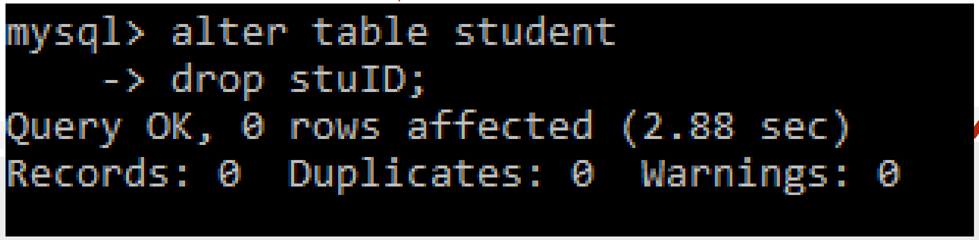
Alter table: g.) remove an attribute

- Using ALTER, we can remove attributes from a table, as shown in the following syntax:

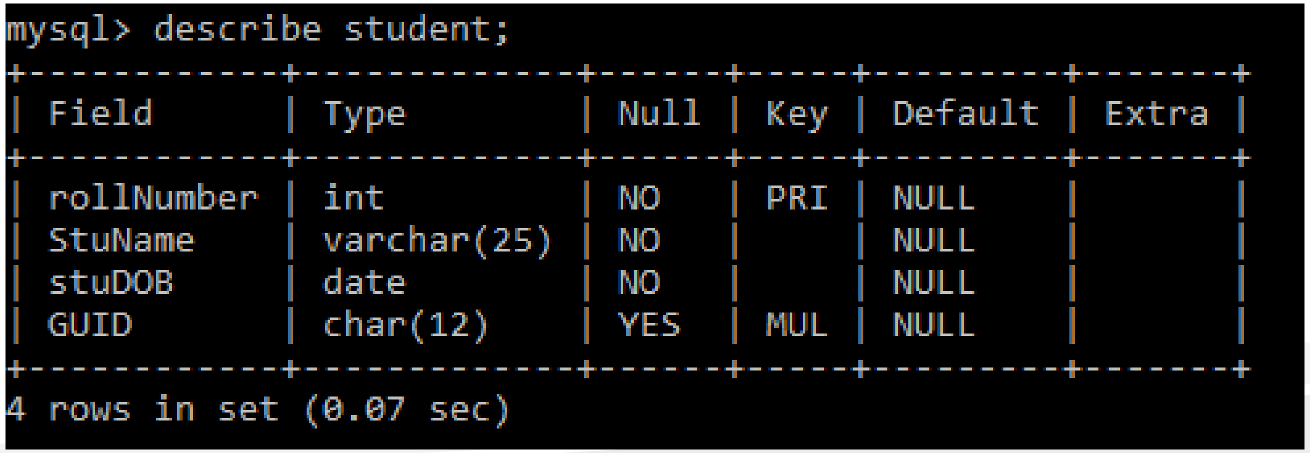
  alter table table_name DROP attribute_name;



```
mysql> describe student;
+------------+-------------+------+-----+------------+-------+
| Field      | Type        | Null | Key | Default    | Extra |
+------------+-------------+------+-----+------------+-------+
| rollNumber | int         | NO   | PRI | NULL       |       |
| StuName    | varchar(25) | NO   |     | NULL       |       |
| stuDOB     | date        | NO   |     | NULL       |       |
| GUID       | char(12)    | YES  | MUL | NULL       |       |
| stuID      | char(10)    | YES  | UNI | 9999999999 |       |
+------------+-------------+------+-----+------------+-------+
5 rows in set (0.10 sec)
```



```
mysql> alter table student
    -> drop stuID;
Query OK, 0 rows affected (2.88 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

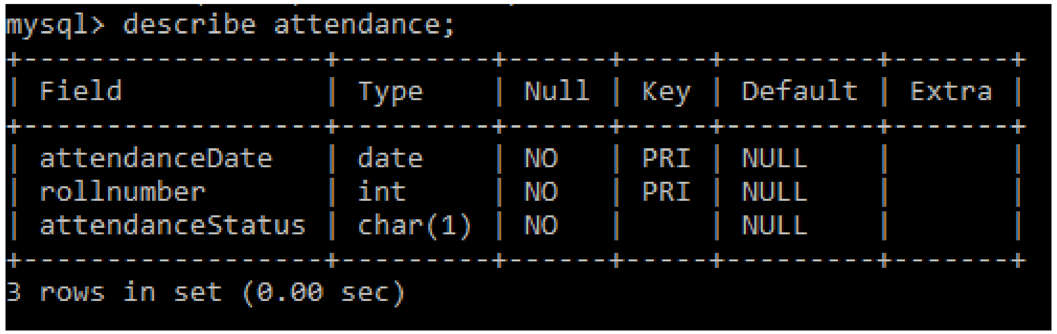


```
mysql> describe student;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| rollNumber | int         | NO   | PRI | NULL    |       |
| StuName    | varchar(25) | NO   |     | NULL    |       |
| stuDOB     | date        | NO   |     | NULL    |       |
| GUID       | char(12)    | YES  | MUL | NULL    |       |
+------------+-------------+------+-----+---------+-------+
4 rows in set (0.07 sec)
```

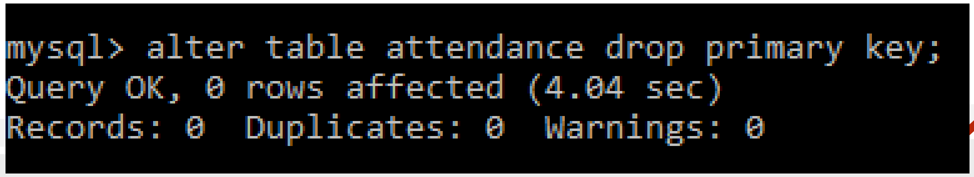
Alter table: h.) remove primary key from the table

- Sometime there may be a requirement to remove primary key constraint from the table.
  In that case, Alter table command can be used in the following way:

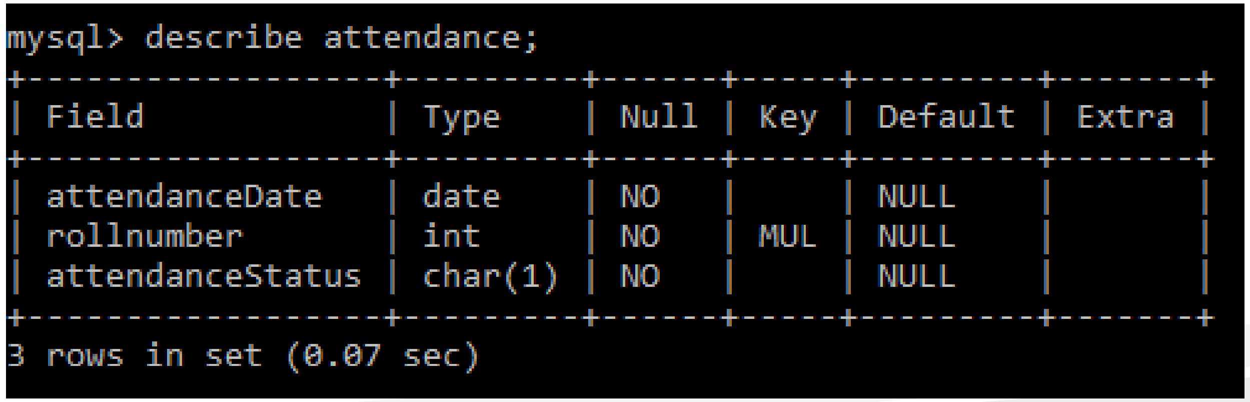
  alter table table_name DROP primary key;



```
mysql> describe attendance;
+-------------------+---------+------+-----+---------+-------+
| Field             | Type    | Null | Key | Default | Extra |
+-------------------+---------+------+-----+---------+-------+
| attendanceDate    | date    | NO   | PRI | NULL    |       |
| rollnumber        | int     | NO   | PRI | NULL    |       |
| attendanceStatus  | char(1) | NO   |     | NULL    |       |
+-------------------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```



```
mysql> alter table attendance drop primary key;
Query OK, 0 rows affected (4.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



```
mysql> describe attendance;
+-------------------+---------+------+-----+---------+-------+
| Field             | Type    | Null | Key | Default | Extra |
+-------------------+---------+------+-----+---------+-------+
| attendanceDate    | date    | NO   |     | NULL    |       |
| rollnumber        | int     | NO   | MUL | NULL    |       |
| attendanceStatus  | char(1) | NO   |     | NULL    |       |
+-------------------+---------+------+-----+---------+-------+
3 rows in set (0.07 sec)
```

Summary: Data Definition Language

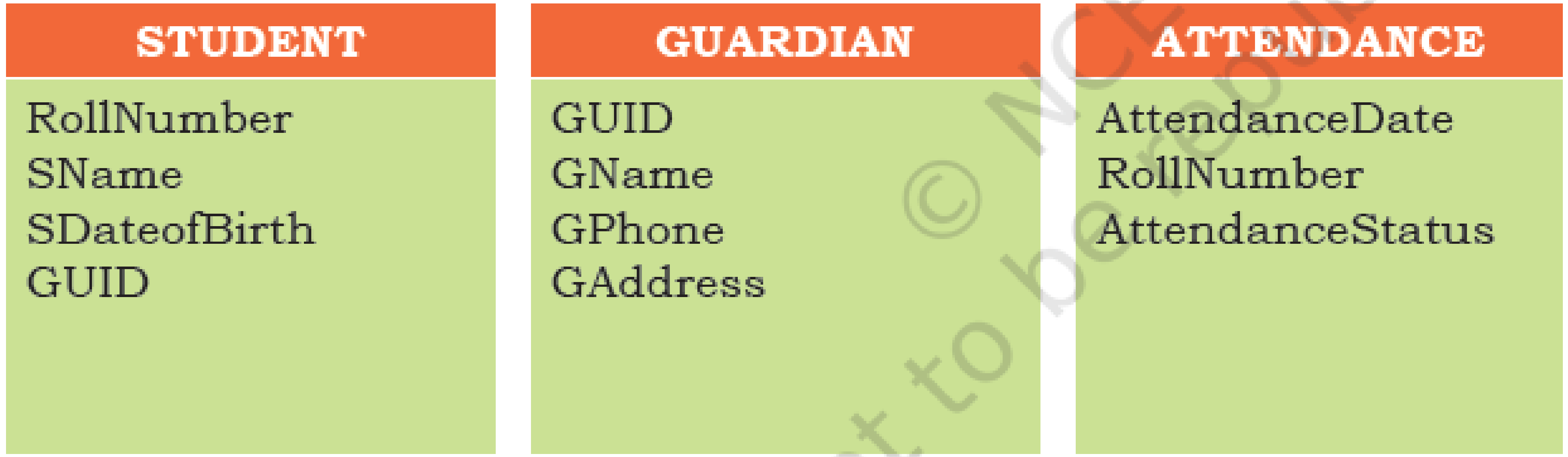
- Create statement: table/ database
- Drop statement: table/database
- Alter table:

  - adding primary/foreign key to a relation
  - adding attribute to an existing table
  - modify datatype/constraint of an attribute
  - adding default value to an attribute
  - removing an attribute
  - removing primary key from the table

Data Manipulation Language (DML)

SQL for Data Manipulation Language (DML)

- Till now, we have created the database StudentAttendance having three relations STUDENT, GUARDIAN and ATTENDANCE



STUDENT	GUARDIAN	ATTENDANCE
RollNumber	GUID	AttendanceDate
SName	GName	RollNumber
SDateofBirth	GPhone	AttendanceStatus
GUID	GAddress	

SQL for Data Manipulation Language (DML)

- When we create a table, only its structure is created but the table has no data.
- So, we may need to insert/modify/delete the data in the respective table which refers as the data manipulation in a database.
- So SQL statements under DML are:
  - INSERT statement (for insertion of new Data)
  - DELETE statement (for removal of existing data)
  - UPDATE statement (for modification of existing data)

DML: Insertion statement

- In order to insert a new record (tuple) in a table, we will use INSERT INTO statement and its syntax is:

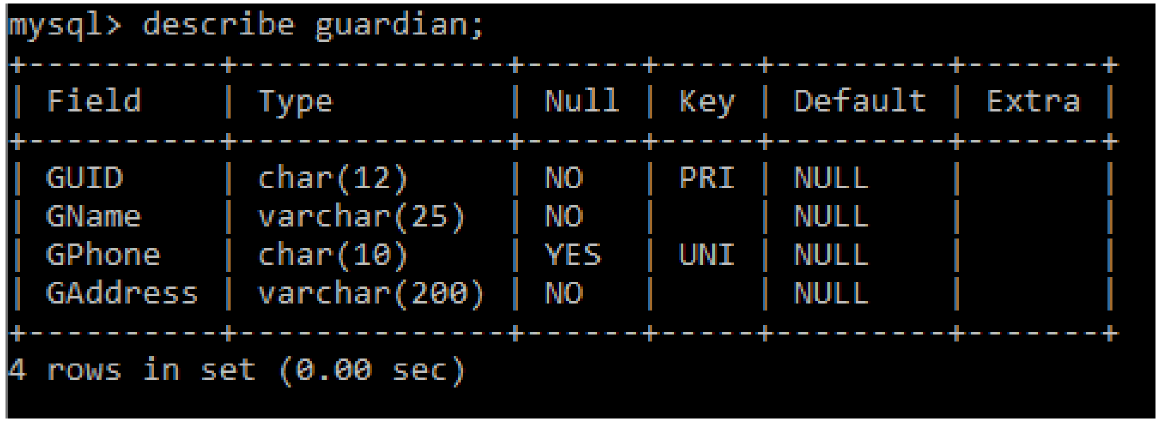
```
INSERT INTO table_name
VALUES (value 1, value 2, ....);
```

where value 1 corresponds to attribute 1, value 2 corresponds to attribute 2 and so on.

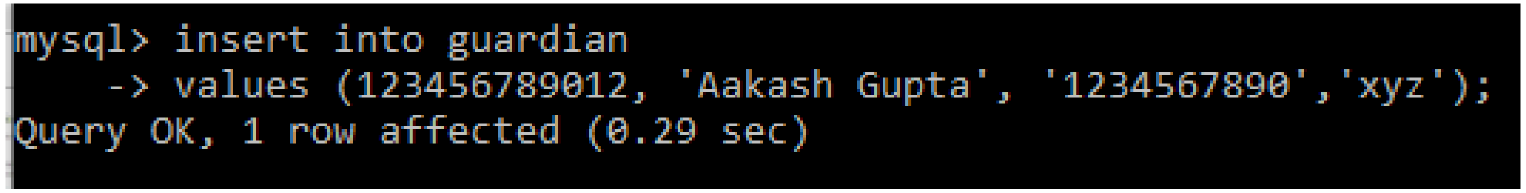
Note:

While populating records in a table with foreign key, ensure that records in referenced tables are already populated.

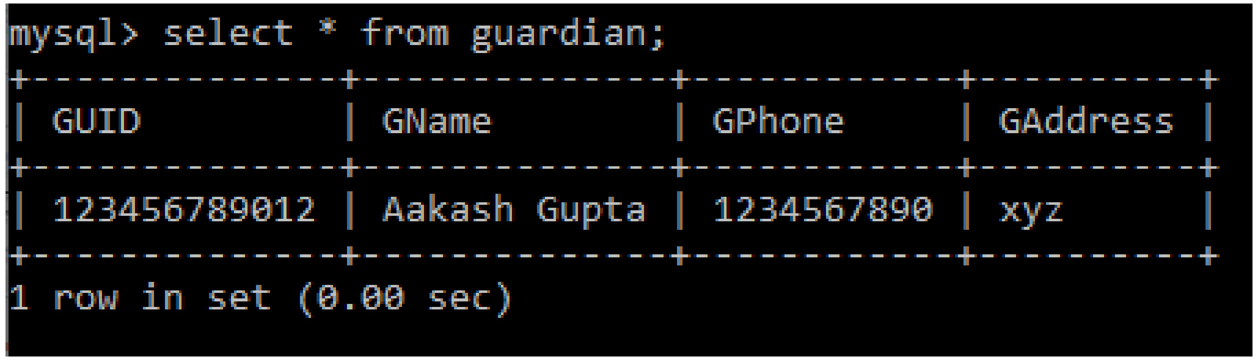
DML: Insertion statement



```
mysql> describe guardian;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| GUID     | char(12)     | NO   | PRI | NULL    |       |
| GName    | varchar(25)  | NO   |     | NULL    |       |
| GPhone   | char(10)     | YES  | UNI | NULL    |       |
| GAddress | varchar(200) | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```



```
mysql> insert into guardian
    -> values (123456789012, 'Aakash Gupta', '1234567890','xyz');
Query OK, 1 row affected (0.29 sec)
```



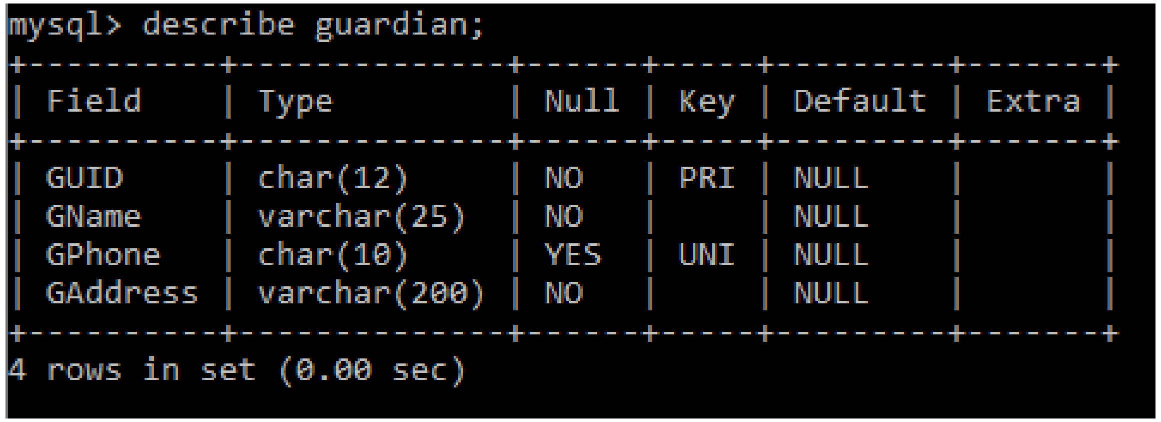
```
mysql> select * from guardian;
+--------------+--------------+------------+----------+
| GUID         | GName        | GPhone     | GAddress |
+--------------+--------------+------------+----------+
| 123456789012 | Aakash Gupta | 1234567890 | xyz      |
+--------------+--------------+------------+----------+
1 row in set (0.00 sec)
```

DML: Insertion statement

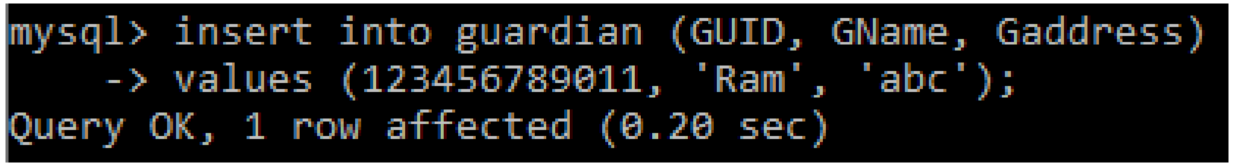
- In the INSERT statement, if there are same number of values as of the number of attributes in the table, then we don’t need to specify the names of the attributes.
- But if want to insert values for only some of the attributes in a table (assuming other attributes having NULL or any other default values), then we should specify the attributes names in which the values need to be inserted using below syntax:

```
INSERT INTO table_name (attribute 1, attribute 2, ....)
VALUES (value 1, value 2, ....);
```

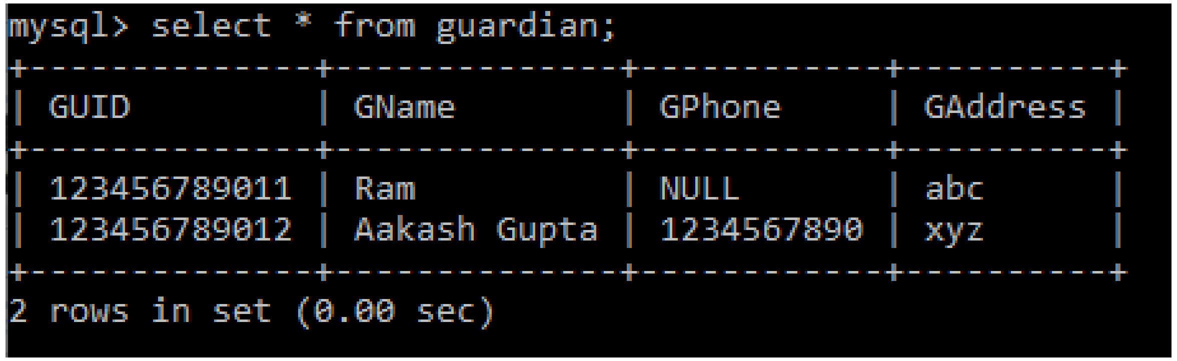
DML: Insertion statement



```
mysql> describe guardian;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| GUID     | char(12)     | NO   | PRI | NULL    |       |
| GName    | varchar(25)  | NO   |     | NULL    |       |
| GPhone   | char(10)     | YES  | UNI | NULL    |       |
| GAddress | varchar(200) | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```



```
mysql> insert into guardian (GUID, GName, Gaddress)
    -> values (123456789011, 'Ram', 'abc');
Query OK, 1 row affected (0.20 sec)
```



```
mysql> select * from guardian;
+--------------+--------------+------------+----------+
| GUID         | GName        | GPhone     | GAddress |
+--------------+--------------+------------+----------+
| 123456789011 | Ram          | NULL       | abc      |
| 123456789012 | Aakash Gupta | 1234567890 | xyz      |
+--------------+--------------+------------+----------+
2 rows in set (0.00 sec)
```

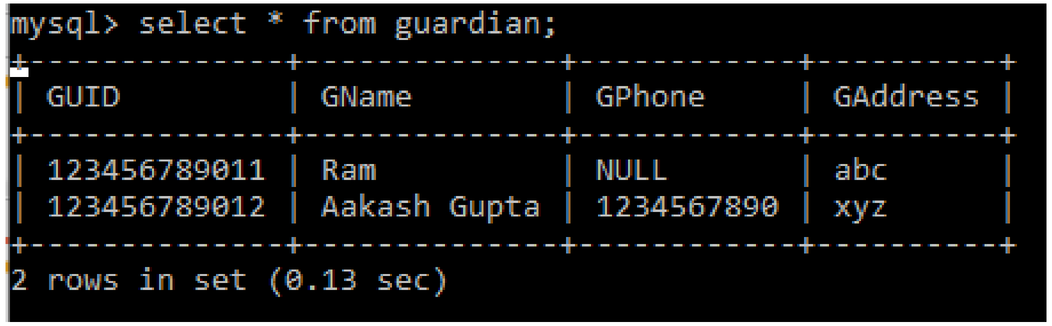
DML: Data Updation statement

- We may need to make changes in the value(s) of one or more columns of existing records in a table.
  For example, we may require some changes in address, phone number or spelling of name, etc.

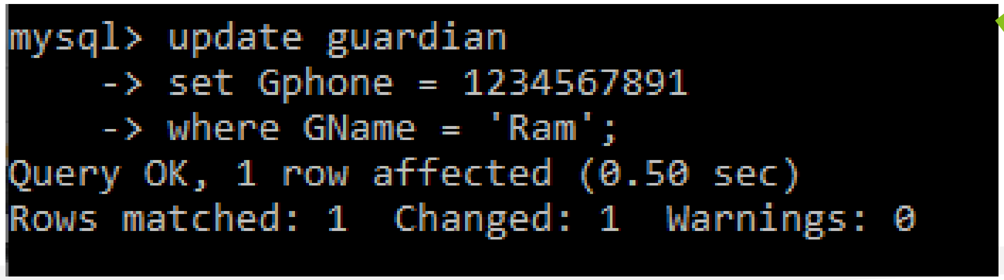
- The UPDATE statement is used to make such modifications in existing data and its syntax is:

```
UPDATE table_name
SET attribute1 = value1, attribute2 = value2, ....
WHERE condition;
```

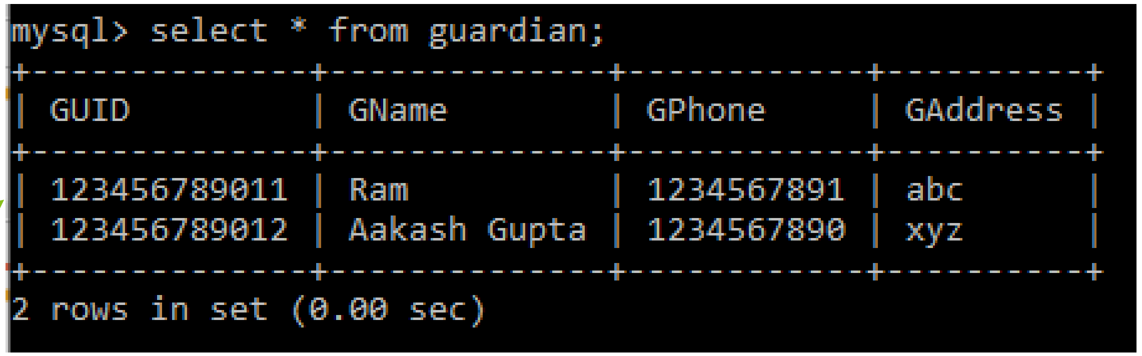
DML: Data Updation statement



```
mysql> select * from guardian;
+--------------+--------------+------------+----------+
| GUID         | GName        | GPhone     | GAddress |
+--------------+--------------+------------+----------+
| 123456789011 | Ram          | NULL       | abc      |
| 123456789012 | Aakash Gupta | 1234567890 | xyz      |
+--------------+--------------+------------+----------+
2 rows in set (0.13 sec)
```



```
mysql> update guardian
    -> set Gphone = 1234567891
    -> where GName = 'Ram';
Query OK, 1 row affected (0.50 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```



```
mysql> select * from guardian;
+--------------+--------------+------------+----------+
| GUID         | GName        | GPhone     | GAddress |
+--------------+--------------+------------+----------+
| 123456789011 | Ram          | 1234567891 | abc      |
| 123456789012 | Aakash Gupta | 1234567890 | xyz      |
+--------------+--------------+------------+----------+
2 rows in set (0.00 sec)
```

Note:
If we don't use the where clause in the update statement, change will be reflected for all tuples

DML: Data Deletion statement

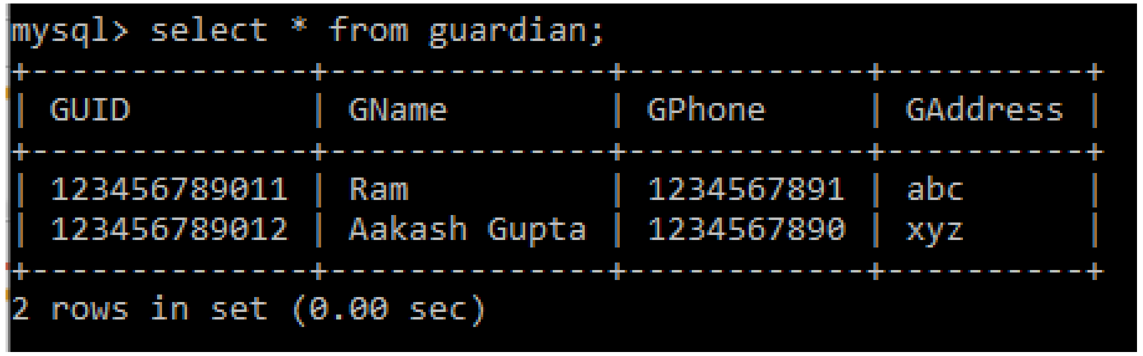
- The DELETE statement is used to delete/remove one or more records from a table and its syntax is:

```
DELETE from table_name
WHERE condition;
```

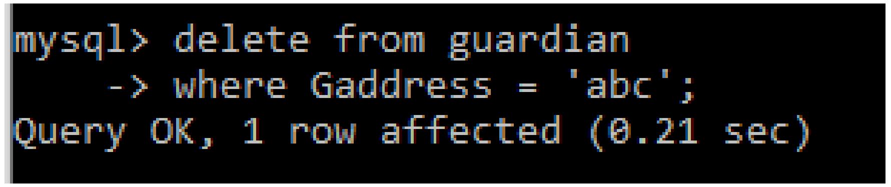
Note:

Like update statement, we need to be careful regarding the where clause, as if we don't include where clause then all the data from the table will be deleted.

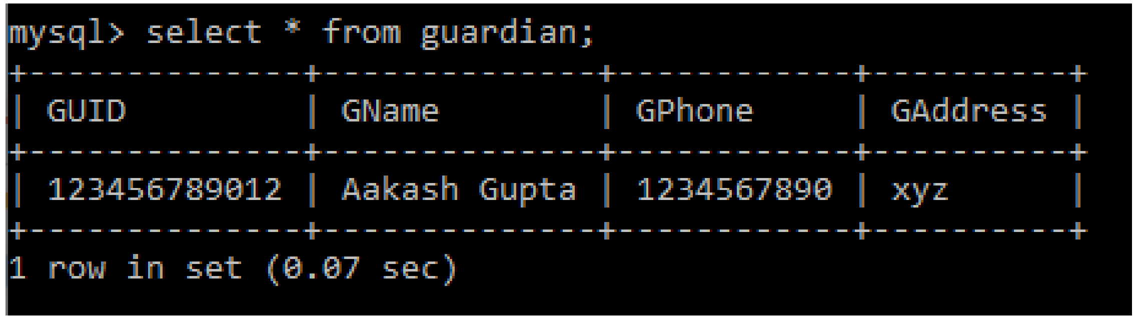
DML: Data Deletion statement



```
mysql> select * from guardian;
+--------------+--------------+------------+----------+
| GUID         | GName        | GPhone     | GAddress |
+--------------+--------------+------------+----------+
| 123456789011 | Ram          | 1234567891 | abc      |
| 123456789012 | Aakash Gupta | 1234567890 | xyz      |
+--------------+--------------+------------+----------+
2 rows in set (0.00 sec)
```



```
mysql> delete from guardian
    -> where Gaddress = 'abc';
Query OK, 1 row affected (0.21 sec)
```



```
mysql> select * from guardian;
+--------------+--------------+------------+----------+
| GUID         | GName        | GPhone     | GAddress |
+--------------+--------------+------------+----------+
| 123456789012 | Aakash Gupta | 1234567890 | xyz      |
+--------------+--------------+------------+----------+
1 row in set (0.07 sec)
```

Data Query Language (DQL)

SQL for Data Query Language (DQL)

- So far, we have learnt how to create a database and its associated relations, and also how to store and manipulate data in them. Now, we will learn about how to retrieve data from the database.
- SQL provides efficient mechanisms to retrieve data stored in multiple tables in MySQL database (or any other RDBMS).
- The SQL statement SELECT is used to retrieve data from the tables in a database and is also called a query statement.

DQL: SELECT statement

- The SQL statement SELECT is used to retrieve data from the tables in a database and the output is also displayed in tabular form and its syntax is:

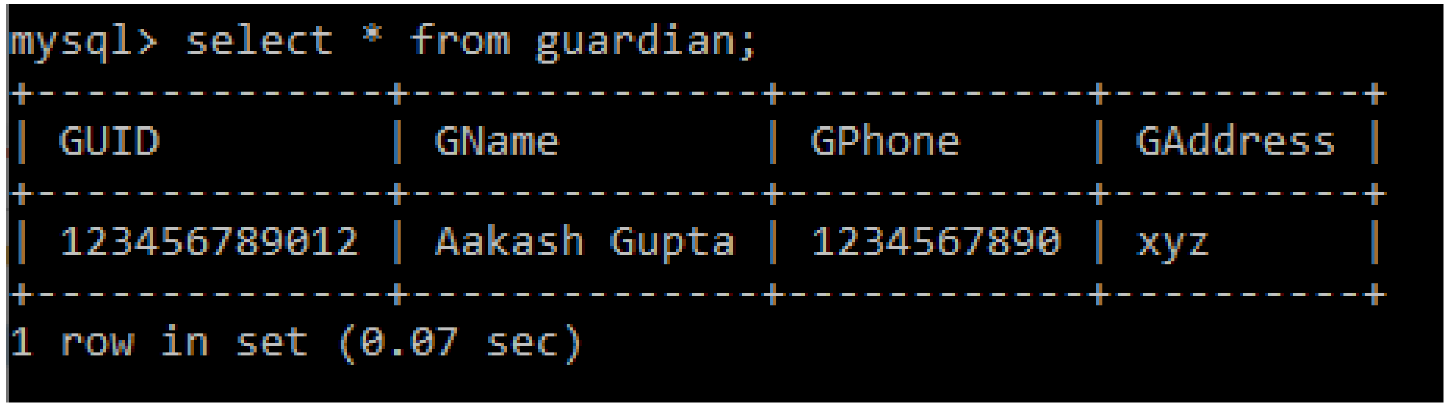
```
SELECT attribute1, attribute2, ...
FROM table_name
WHERE condition;
```

  - Here, the FROM clause is always written with SELECT clause as it specifies the name of the table from which data is to be retrieved
  - The WHERE clause is optional and is used to retrieve data that meet specified condition(s)

DQL: SELECT statement

- In order to select all the data available in the table, we can use following statement:

SELECT * FROM table_name ;



```
mysql> select * from guardian;
+--------------+--------------+------------+----------+
| GUID         | GName        | GPhone     | GAddress |
+--------------+--------------+------------+----------+
| 123456789012 | Aakash Gupta | 1234567890 | xyz      |
+--------------+--------------+------------+----------+
1 row in set (0.07 sec)
```

Creating Database for DQL

- Requirement: Create Database ‘OFFICE’ with two relations ‘DEPARTMENT and ‘EMPLOYEE’ with the following records:

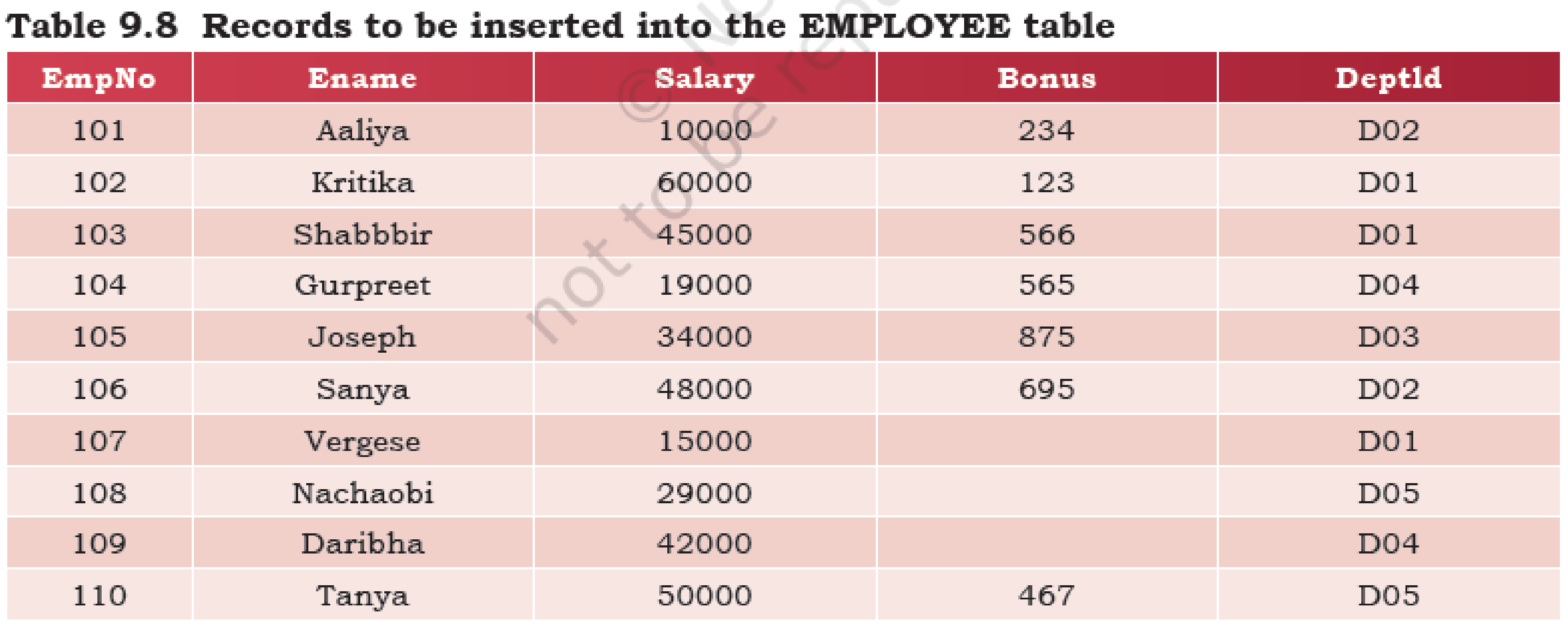
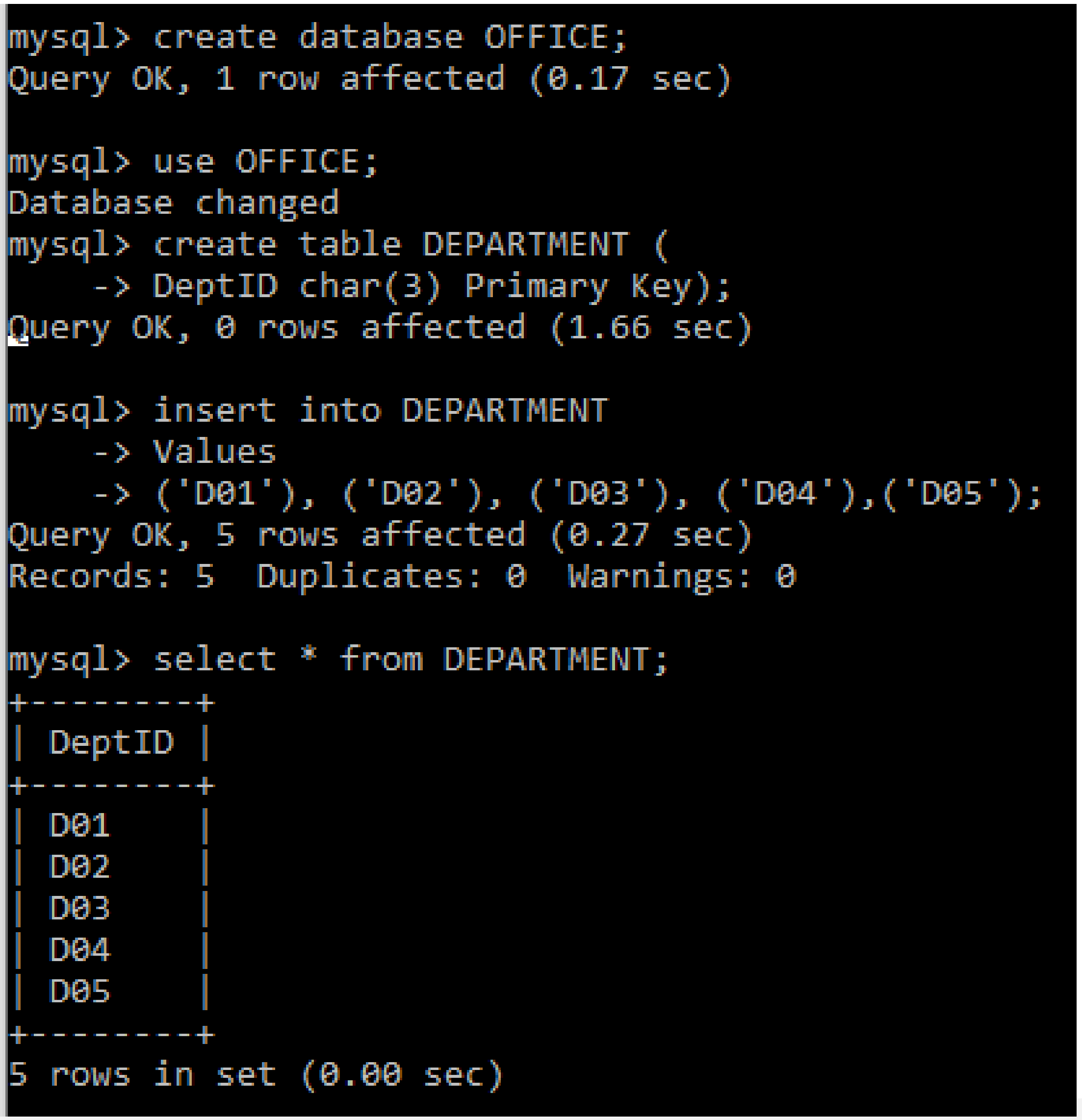
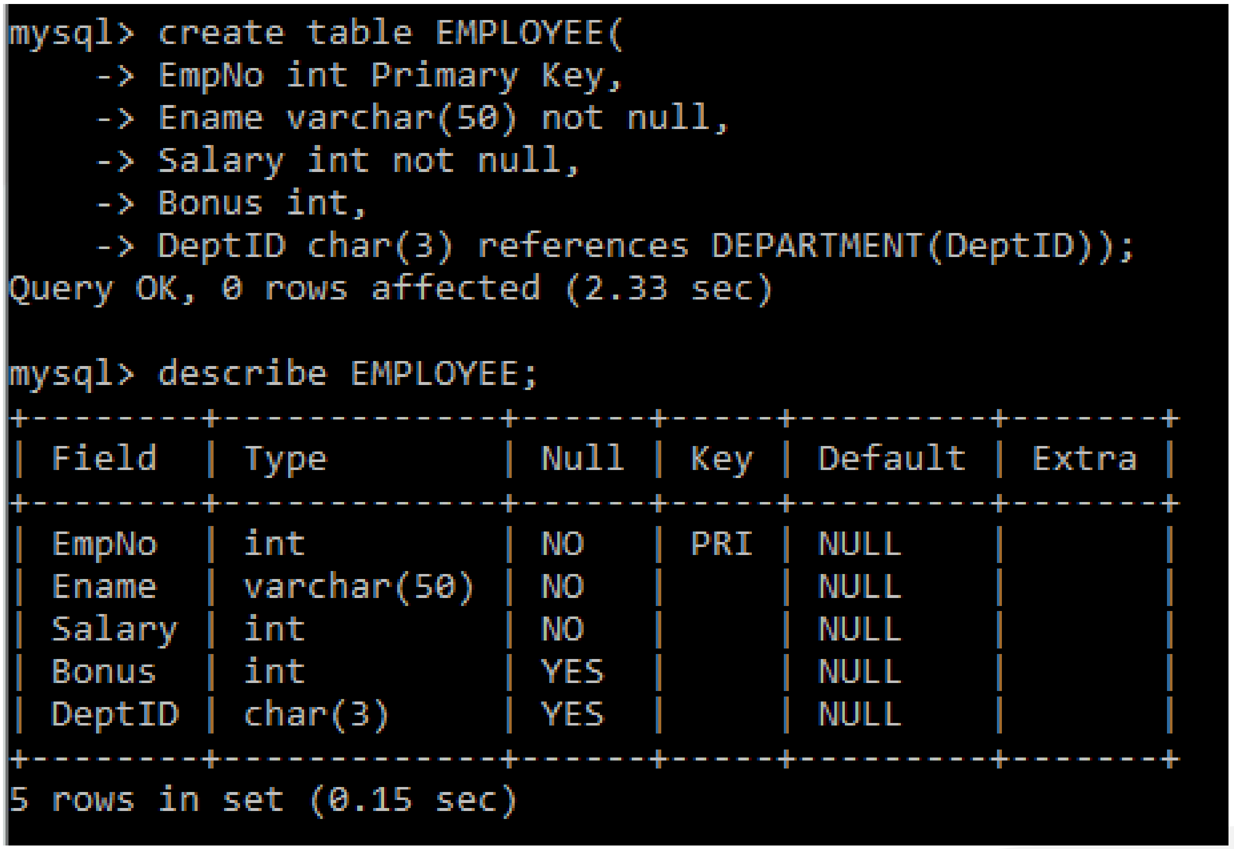


Table 9.8 Records to be inserted into the EMPLOYEE table

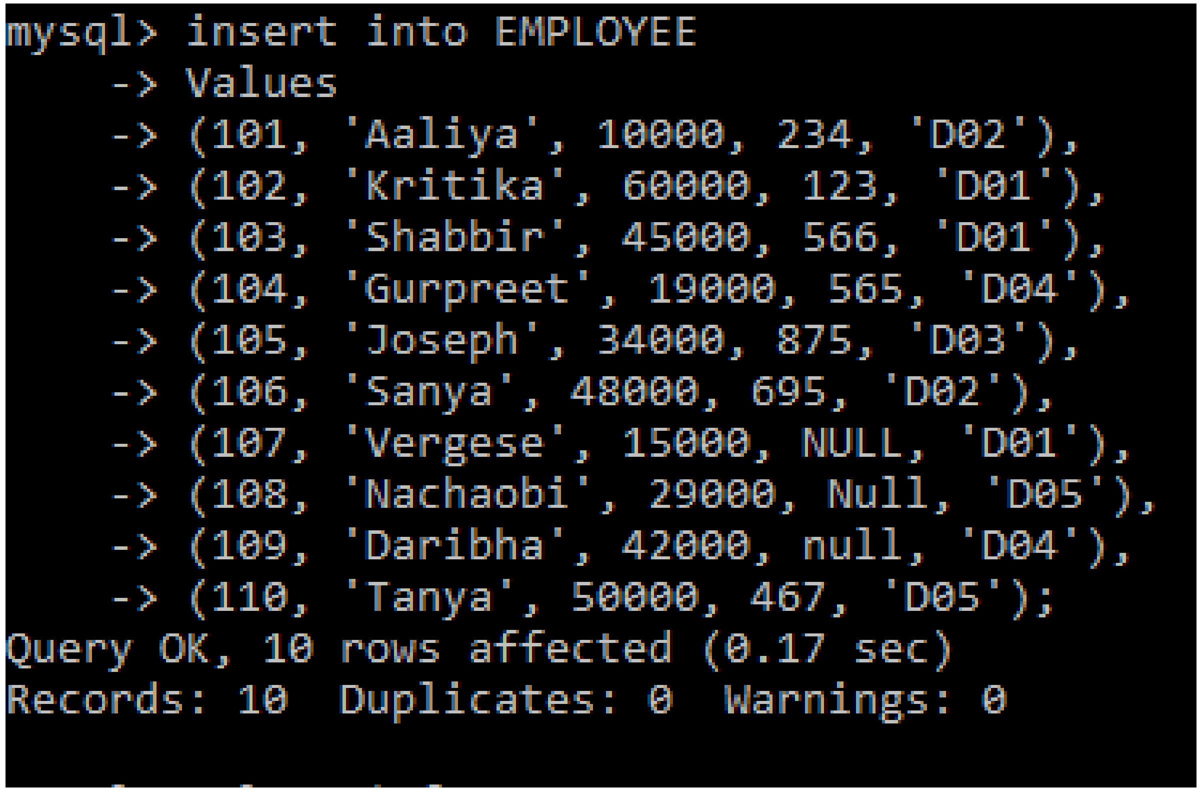
EmpNo	Ename	Salary	Bonus	DeptId
101	Aaliya	10000	234	D02
102	Kritika	60000	123	D01
103	Shabbbir	45000	566	D01
104	Gurpreet	19000	565	D04
105	Joseph	34000	875	D03
106	Sanya	48000	695	D02
107	Vergese	15000		D01
108	Nachaobi	29000		D05
109	Daribha	42000		D04
110	Tanya	50000	467	D05



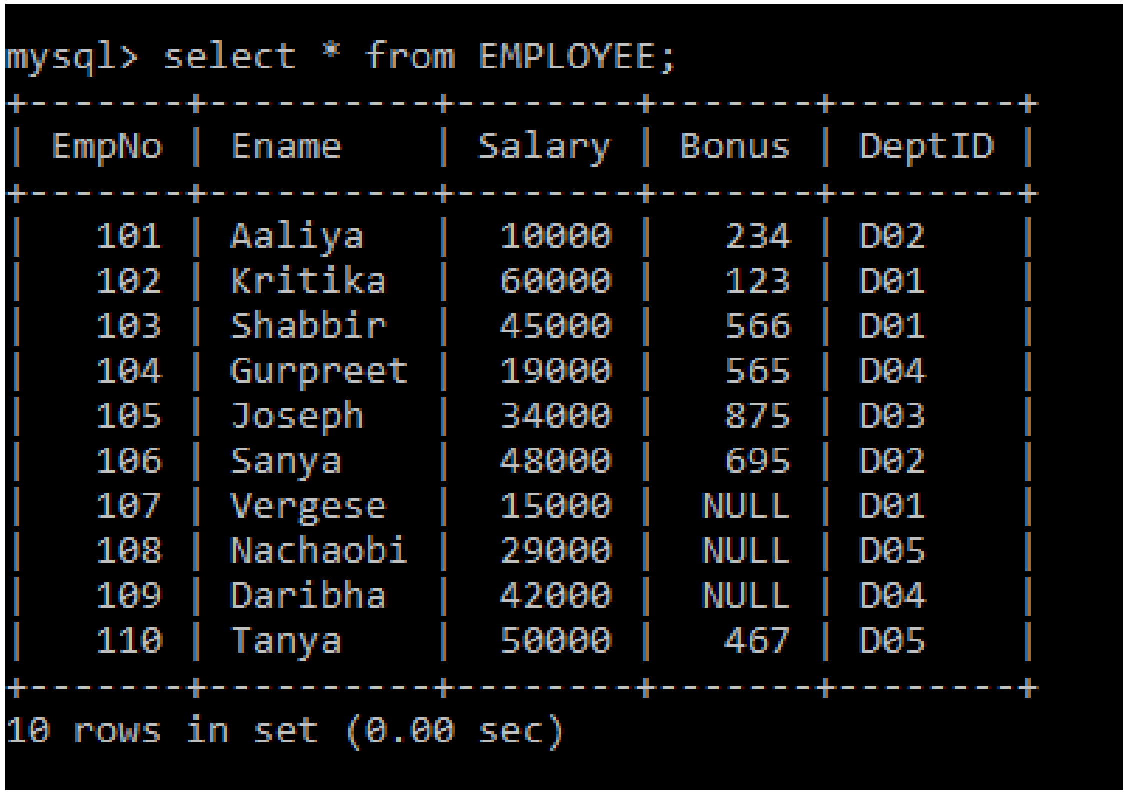
```
mysql> create database OFFICE;
Query OK, 1 row affected (0.17 sec)

mysql> use OFFICE;
Database changed
mysql> create table DEPARTMENT (
    -> DeptID char(3) Primary Key);
Query OK, 0 rows affected (1.66 sec)

mysql> insert into DEPARTMENT
    -> Values
    -> ('D01'), ('D02'), ('D03'), ('D04'),('D05');
Query OK, 5 rows affected (0.27 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from DEPARTMENT;
+--------+
| DeptID |
+--------+
| D01    |
| D02    |
| D03    |
| D04    |
| D05    |
+--------+
5 rows in set (0.00 sec)
```



```
mysql> create table EMPLOYEE(
    -> EmpNo int Primary Key,
    -> Ename varchar(50) not null,
    -> Salary int not null,
    -> Bonus int,
    -> DeptID char(3) references DEPARTMENT(DeptID));
Query OK, 0 rows affected (2.33 sec)

mysql> describe EMPLOYEE;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| EmpNo  | int         | NO   | PRI | NULL    |       |
| Ename  | varchar(50) | NO   |     | NULL    |       |
| Salary | int         | NO   |     | NULL    |       |
| Bonus  | int         | YES  |     | NULL    |       |
| DeptID | char(3)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
5 rows in set (0.15 sec)
```



```
mysql> insert into EMPLOYEE
    -> Values
    -> (101, 'Aaliya', 10000, 234, 'D02'),
    -> (102, 'Kritika', 60000, 123, 'D01'),
    -> (103, 'Shabbir', 45000, 566, 'D01'),
    -> (104, 'Gurpreet', 19000, 565, 'D04'),
    -> (105, 'Joseph', 34000, 875, 'D03'),
    -> (106, 'Sanya', 48000, 695, 'D02'),
    -> (107, 'Vergese', 15000, NULL, 'D01'),
    -> (108, 'Nachaobi', 29000, Null, 'D05'),
    -> (109, 'Daribha', 42000, null, 'D04'),
    -> (110, 'Tanya', 50000, 467, 'D05');
Query OK, 10 rows affected (0.17 sec)
Records: 10  Duplicates: 0  Warnings: 0
```



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

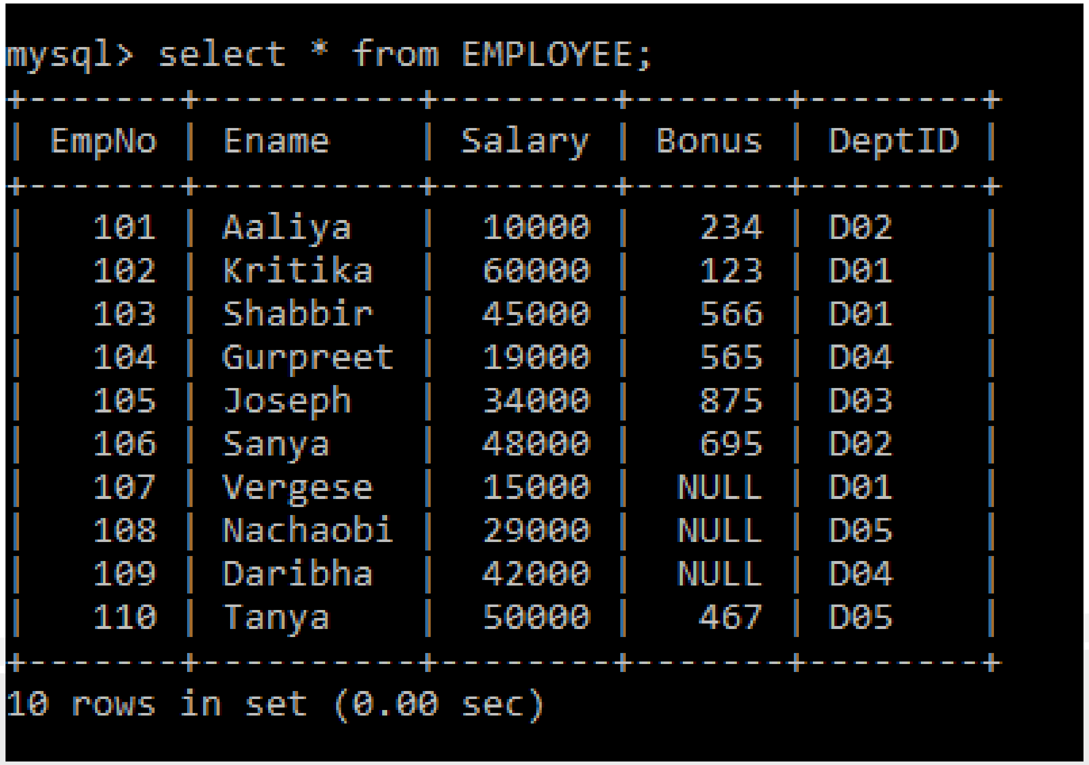
Operators in SQL

- Like Python, SQL too have operators like:
    - Arithmetic Operators (+,-, * , /, %)
    - Relational Operators (=, <, >, >=, <=, <>)
    - Logical Operators (AND, OR, NOT)
    - Membership Operators (IN, NOT IN), and
    - Some other operators (ANY, BETWEEN, LIKE, etc..)

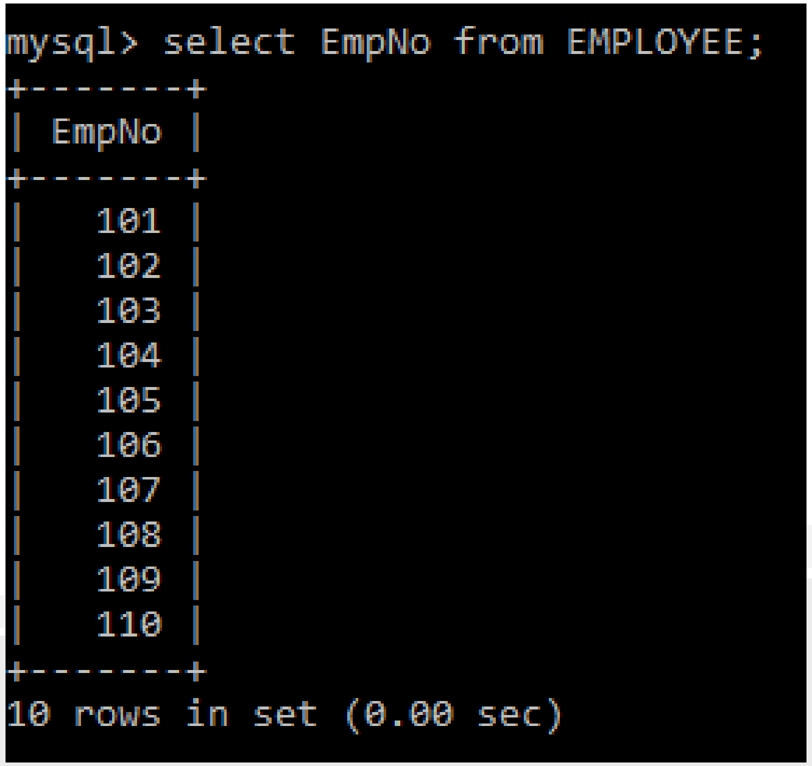
DQL: retrieving single column

- In order to select a single attribute/column, we can use the following statement:

SELECT attribute_name FROM table_name ;



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

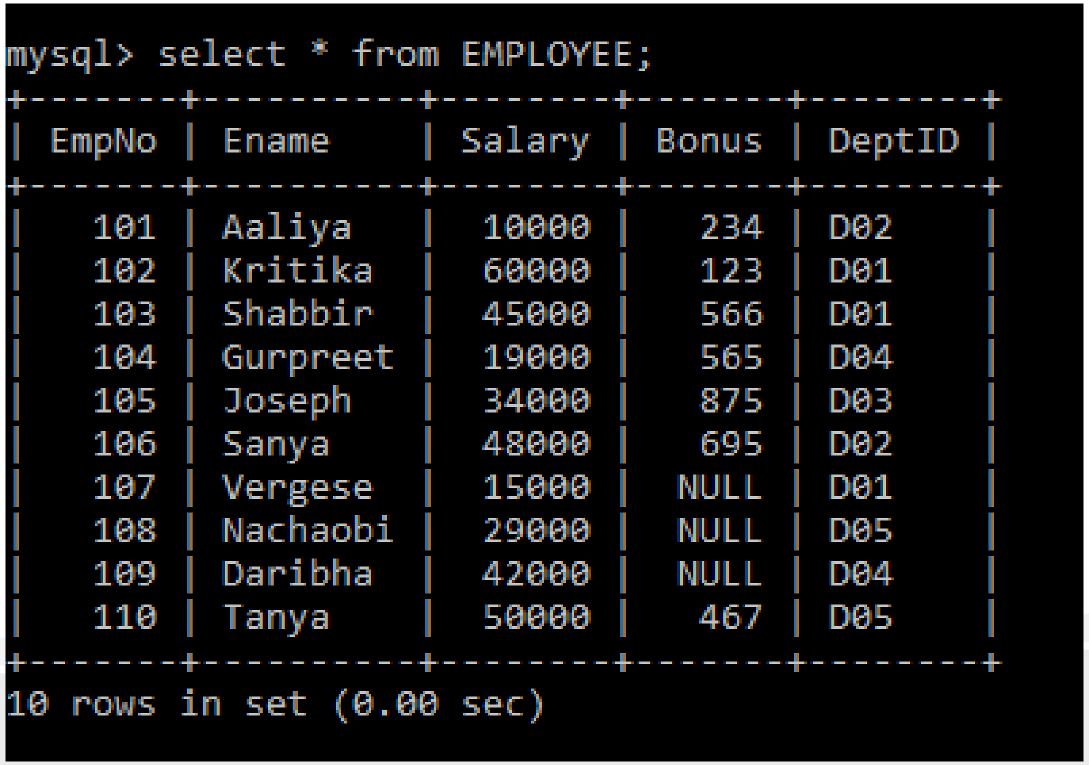


```
mysql> select EmpNo from EMPLOYEE;
+-------+
| EmpNo |
+-------+
|   101 |
|   102 |
|   103 |
|   104 |
|   105 |
|   106 |
|   107 |
|   108 |
|   109 |
|   110 |
+-------+
10 rows in set (0.00 sec)
```

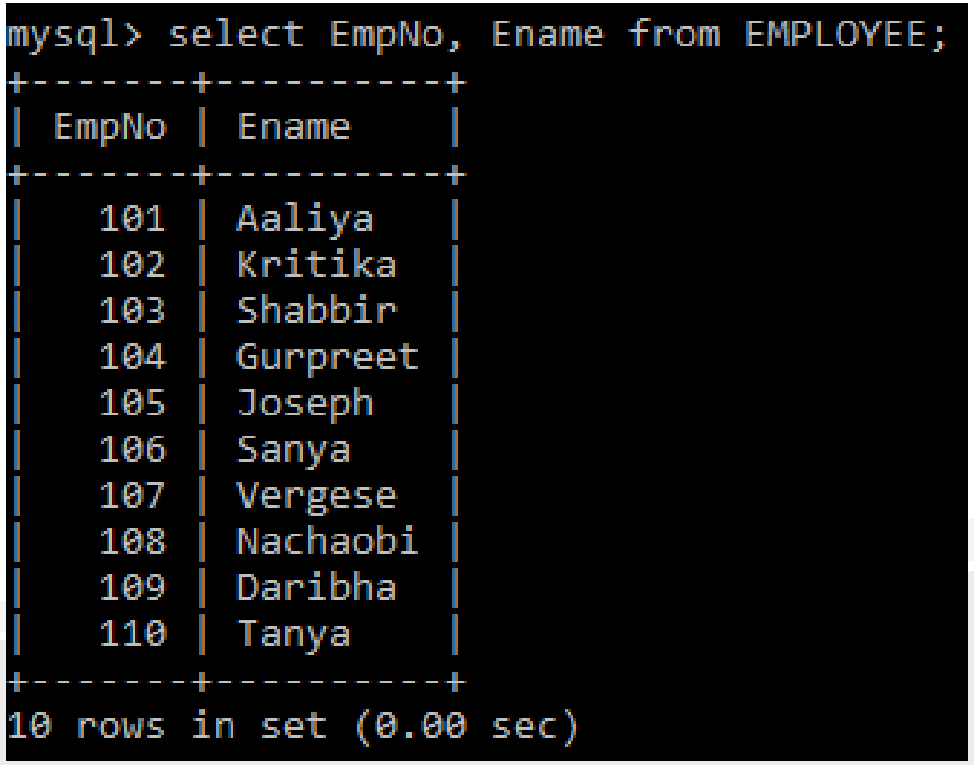
DQL: retrieving multiple columns

- In order to select multiple attributes/columns, we can use the following statement:

SELECT attribute_name1, attribute_name2, …. FROM table_name ;



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

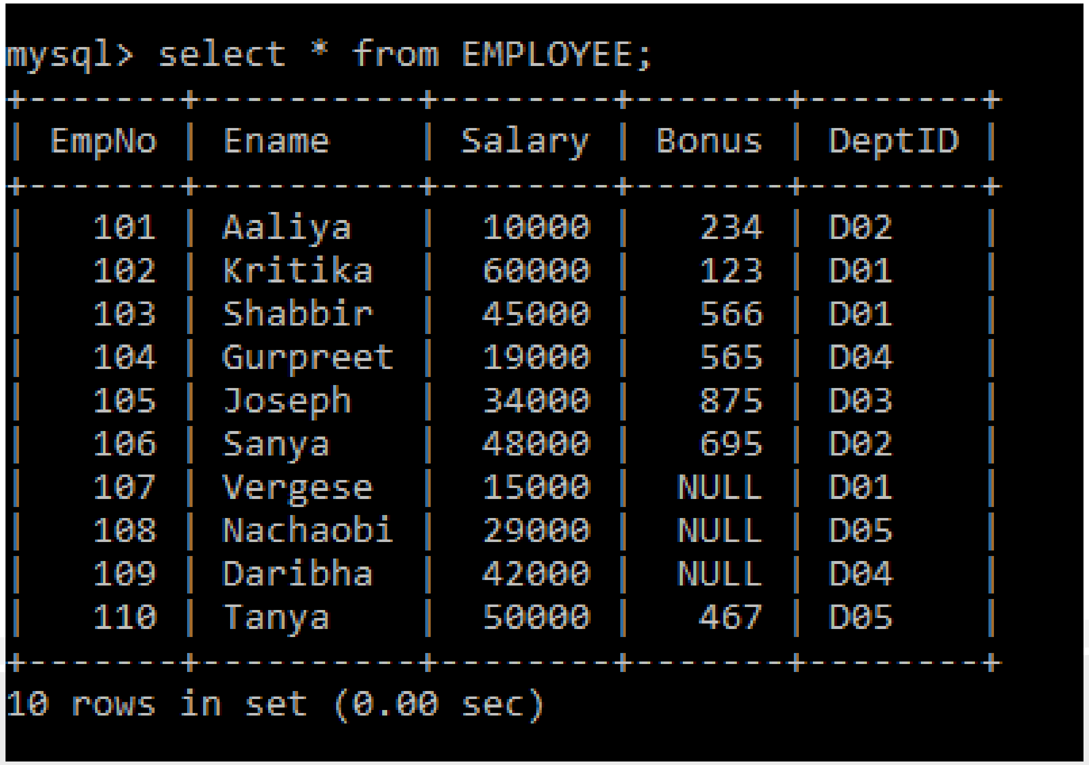


```
mysql> select EmpNo, Ename from EMPLOYEE;
+-------+----------+
| EmpNo | Ename    |
+-------+----------+
|   101 | Aaliya   |
|   102 | Kritika  |
|   103 | Shabbir  |
|   104 | Gurpreet |
|   105 | Joseph   |
|   106 | Sanya    |
|   107 | Vergese  |
|   108 | Nachaobi |
|   109 | Daribha  |
|   110 | Tanya    |
+-------+----------+
10 rows in set (0.00 sec)
```

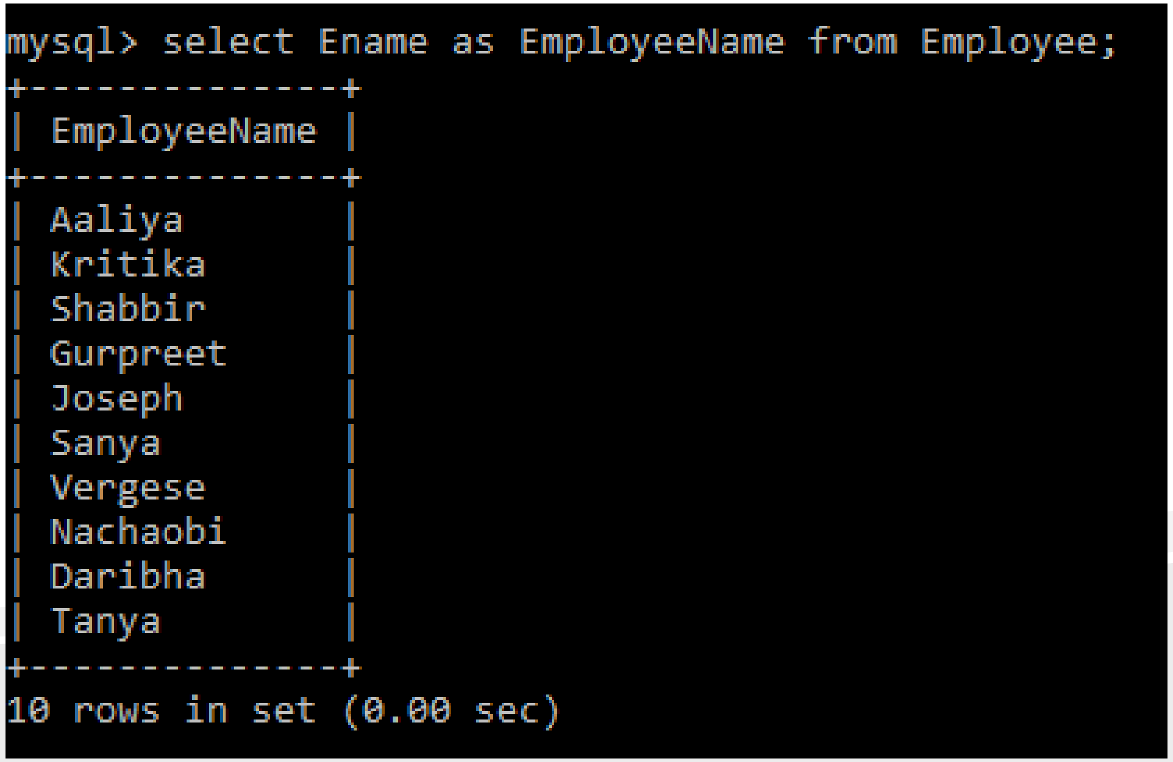
DQL: renaming of columns

- In case we want to rename any column while displaying the output, it can be done using the alias ‘AS’ and its syntax is

SELECT attribute_name AS desired_attribute_name FROM table_name ;



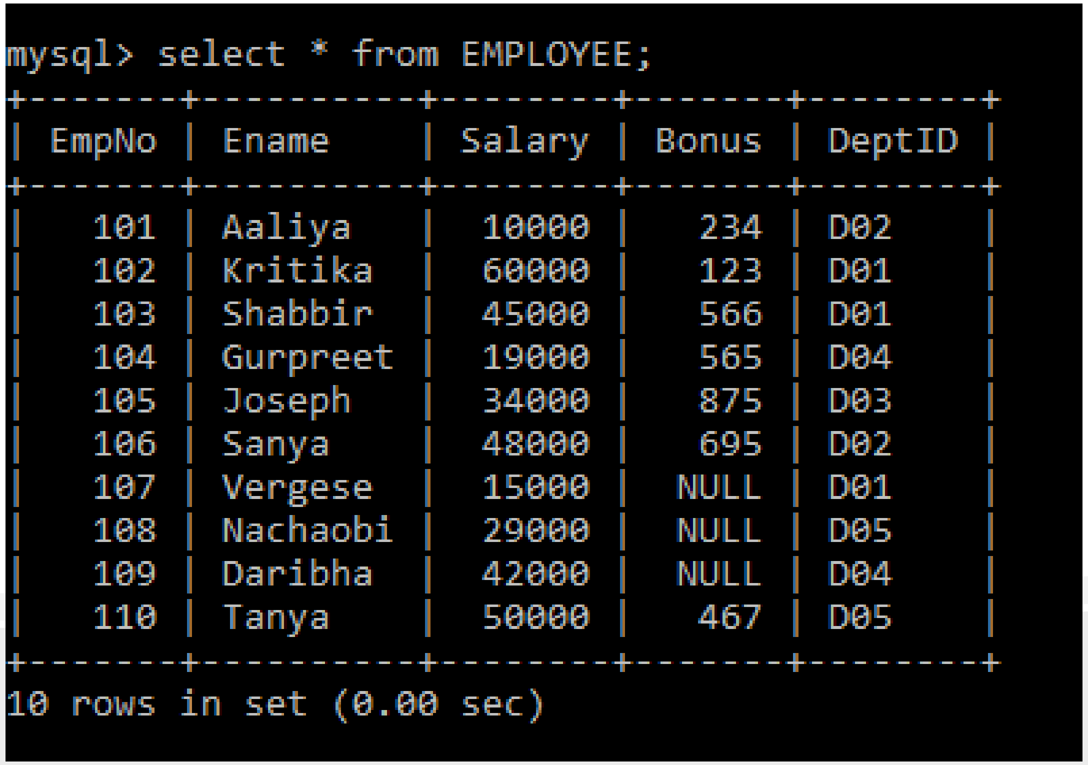
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



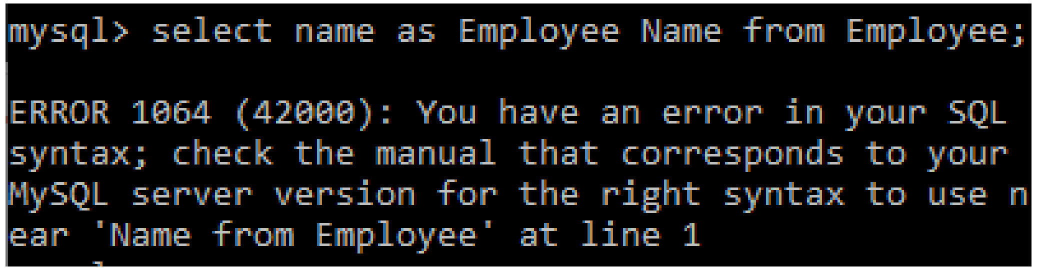
```
mysql> select Ename as EmployeeName from Employee;
+--------------+
| EmployeeName |
+--------------+
| Aaliya       |
| Kritika      |
| Shabbir      |
| Gurpreet     |
| Joseph       |
| Sanya        |
| Vergese      |
| Nachaobi     |
| Daribha      |
| Tanya        |
+--------------+
10 rows in set (0.00 sec)
```

DQL: renaming of columns

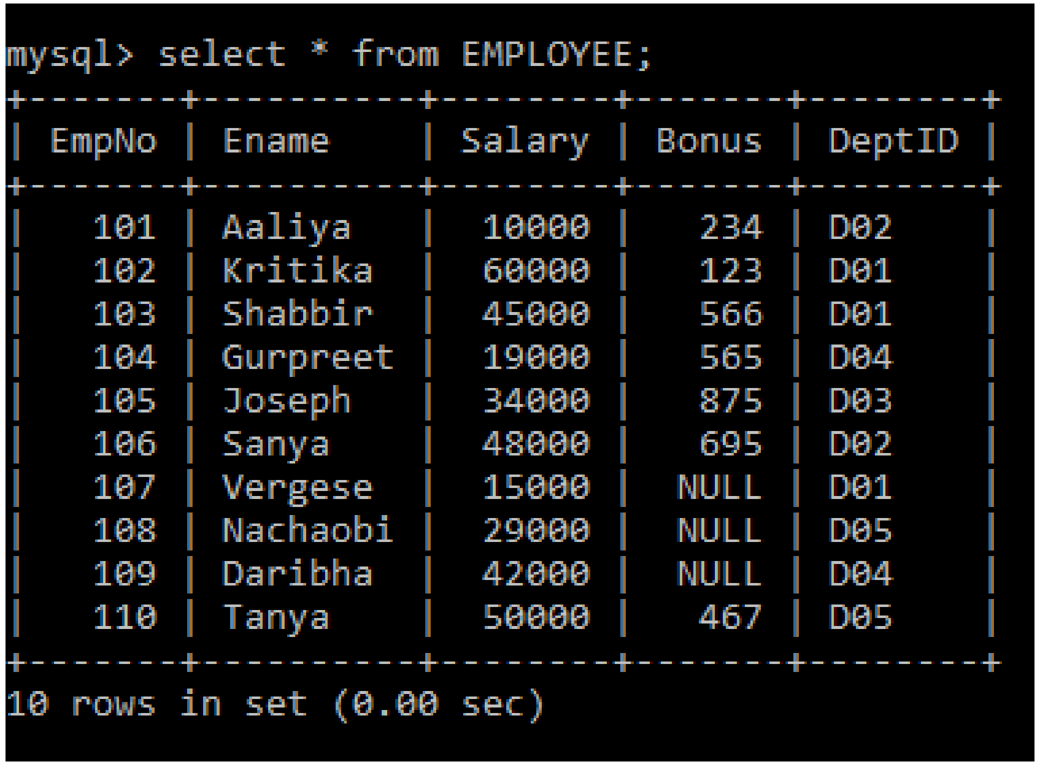
- While using alias ‘AS’ for renaming the column name, keep the following points in mind:
  - If an aliased column name has space, then it should be enclosed in the quotes (‘ ’) else it will lead to an error



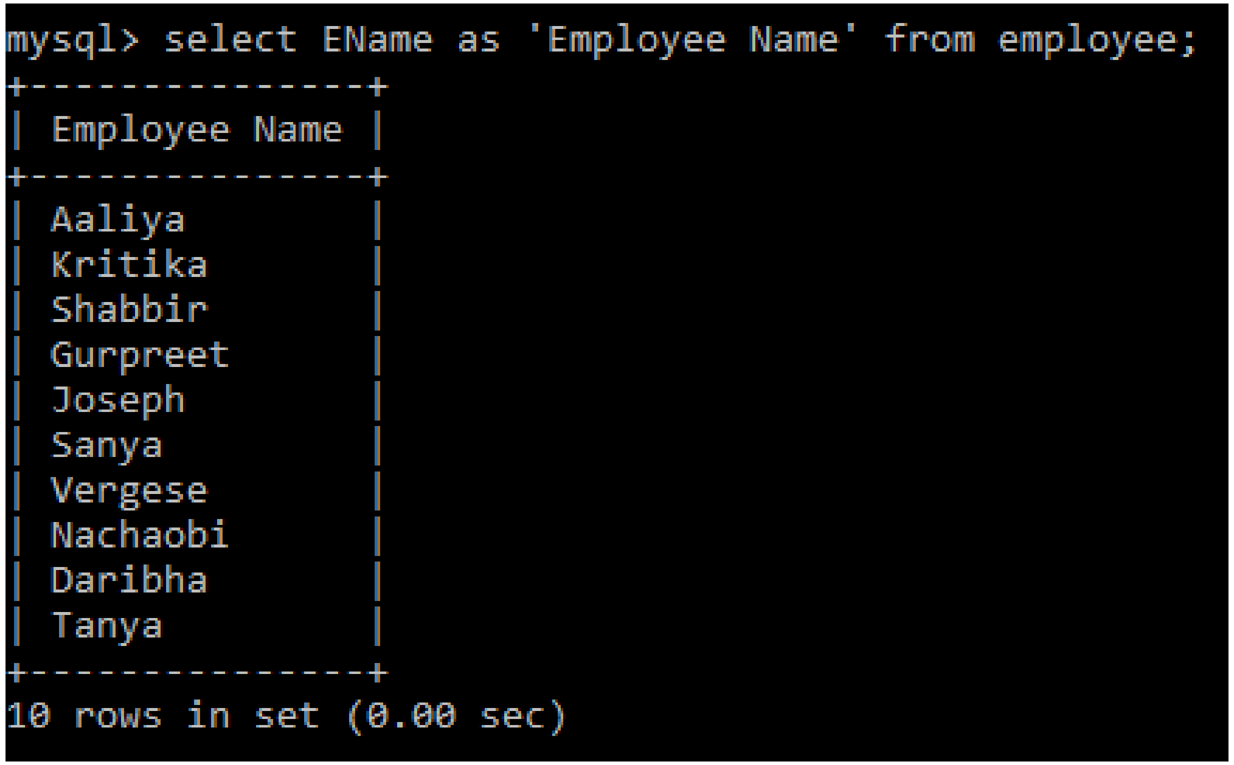
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



```
mysql> select name as Employee Name from Employee;

ERROR 1064 (42000): You have an error in your SQL
syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use n
ear 'Name from Employee' at line 1
```



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

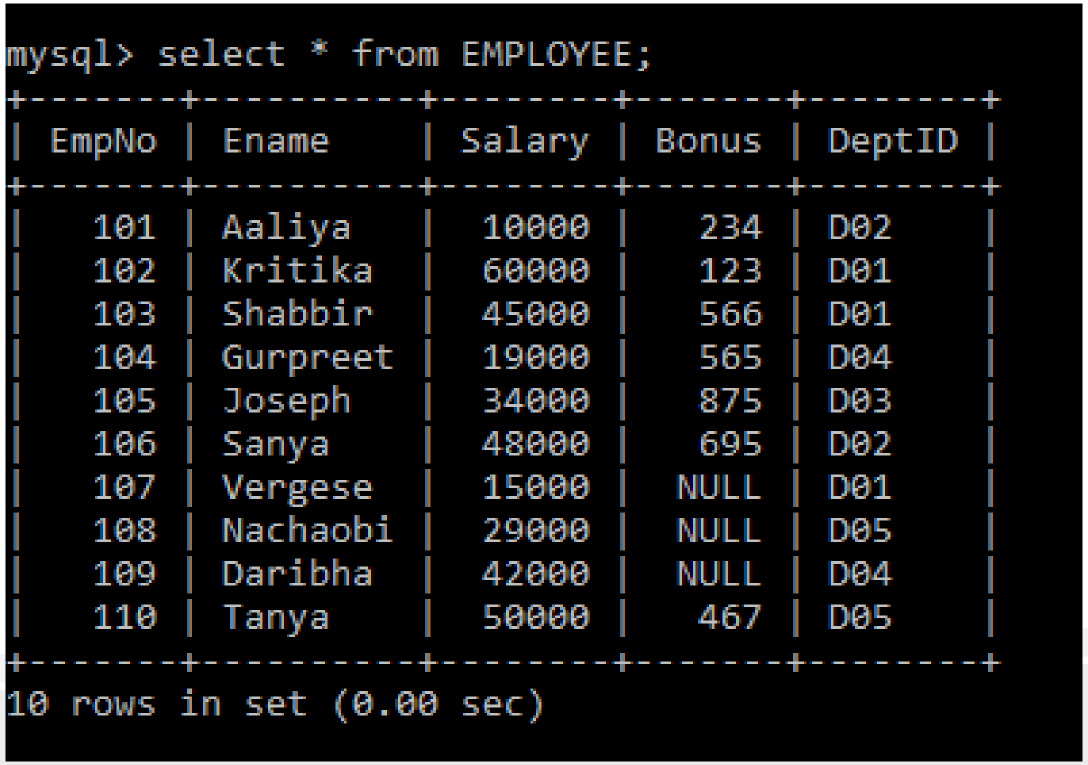


```
mysql> select EName as 'Employee Name' from employee;
+---------------+
| Employee Name |
+---------------+
| Aaliya        |
| Kritika       |
| Shabbir       |
| Gurpreet      |
| Joseph        |
| Sanya         |
| Vergese       |
| Nachaobi      |
| Daribha       |
| Tanya         |
+---------------+
10 rows in set (0.00 sec)
```

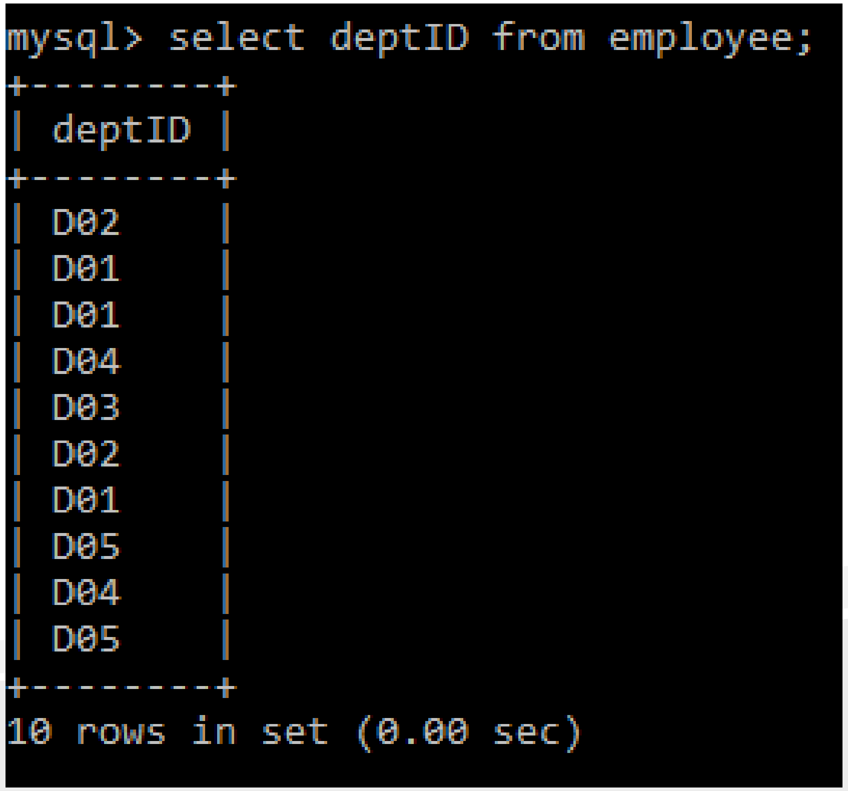
Note:
Renaming will help in displaying the query result, there will be no change in the original relation.

DQL: DISTINCT clause

- By default, SQL shows all the data retrieved through query as output. However, there can be duplicate values.
  For e.g.)



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

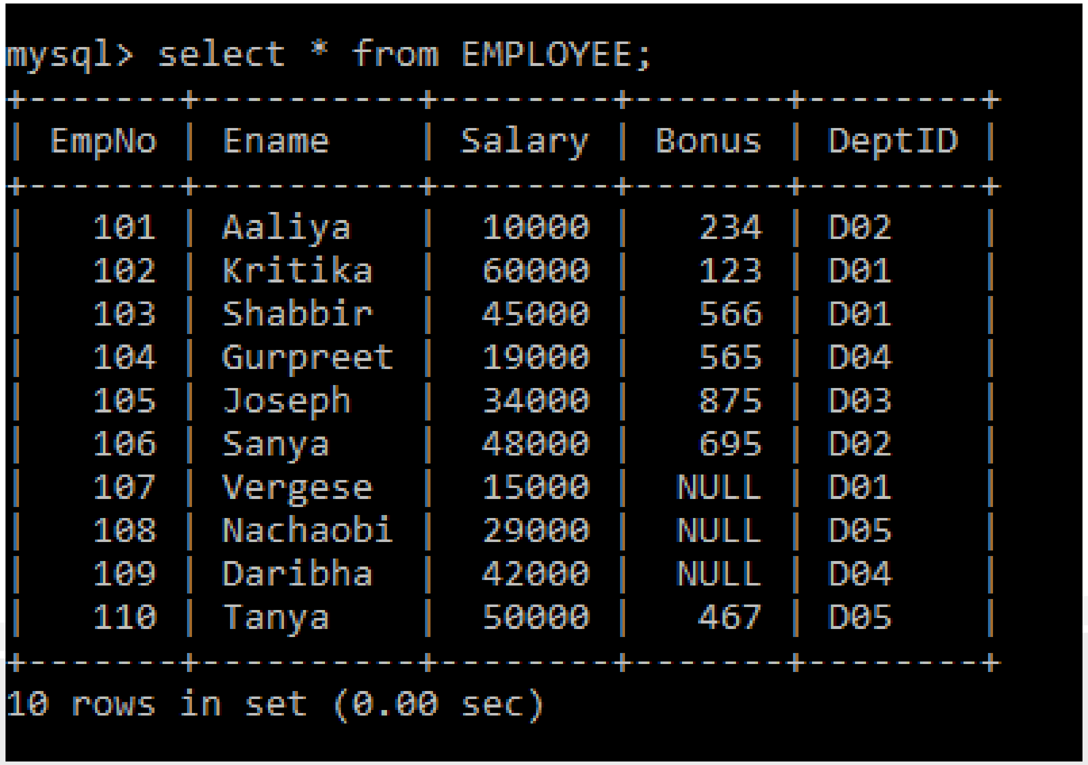


```
mysql> select deptID from employee;
+--------+
| deptID |
+--------+
| D02    |
| D01    |
| D01    |
| D04    |
| D03    |
| D02    |
| D01    |
| D05    |
| D04    |
| D05    |
+--------+
10 rows in set (0.00 sec)
```

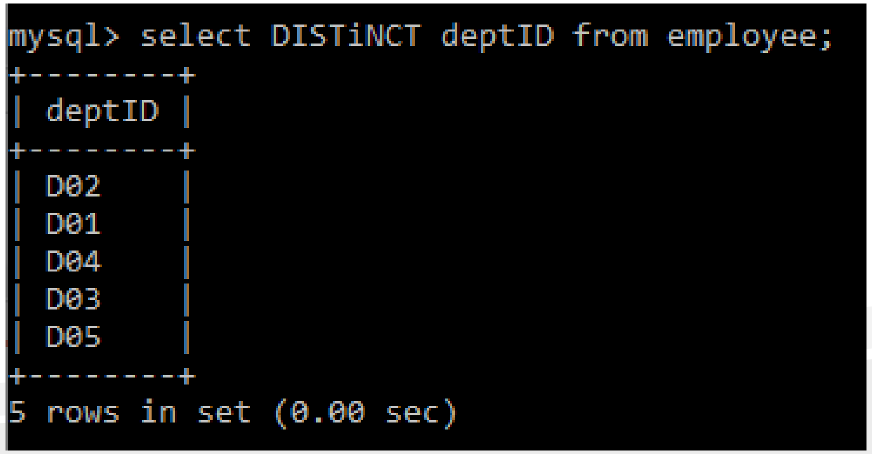
DQL: DISTINCT clause

- The SELECT statement when combined with DISTINCT clause, returns records without repetition (distinct records) and its syntax is

SELECT DISTINCT attribute_name FROM table_name ;



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



```
mysql> select DISTiNCT deptID from employee;
+--------+
| deptID |
+--------+
| D02    |
| D01    |
| D04    |
| D03    |
| D05    |
+--------+
5 rows in set (0.00 sec)
```

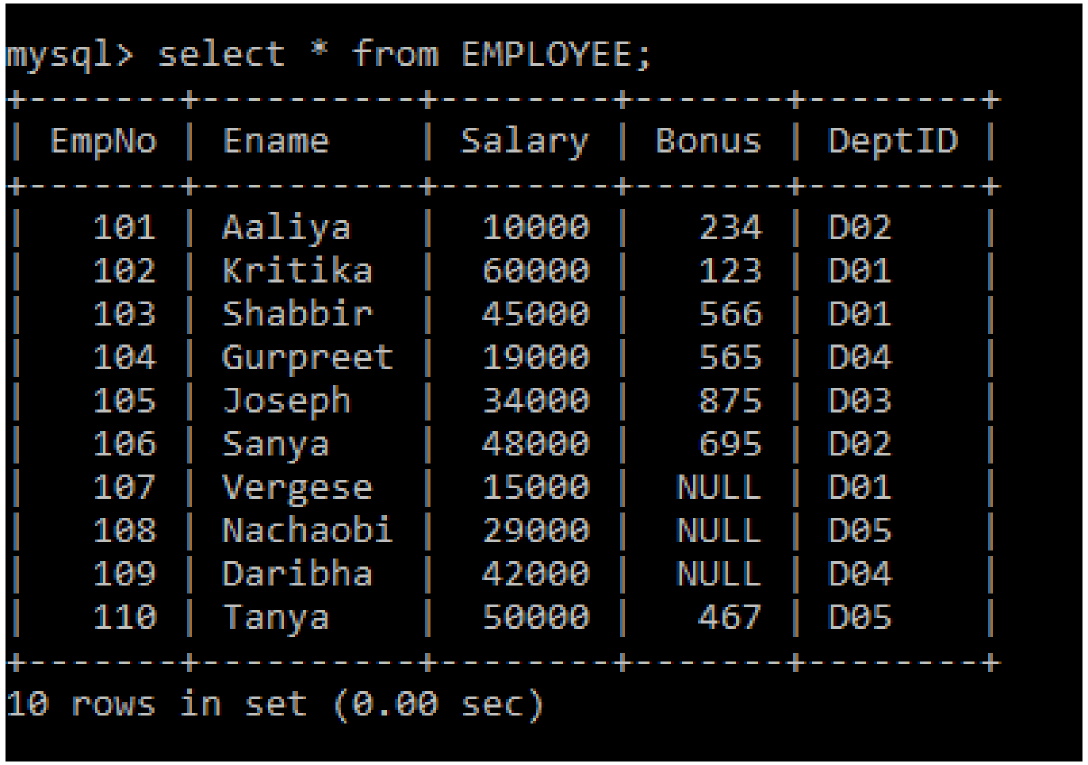
DQL: WHERE clause

- The WHERE clause is used to retrieve data based on some specific condition and its syntax is

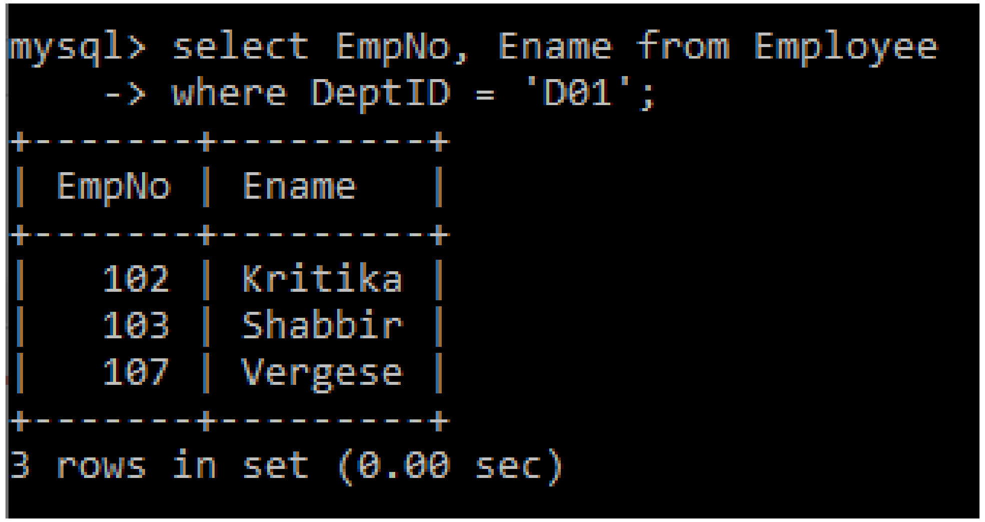
```
SELECT attribute_name[,attribute2,..] FROM table_name WHERE condition;
```

- Let’s suppose, we want to know the Employee number, and name of the Employees who are working in the department D01, then we need to write the following SQL query:

```
SELECT EmpNo, Ename
FROM Employee
WHERE DeptID = ‘D01’;
```



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

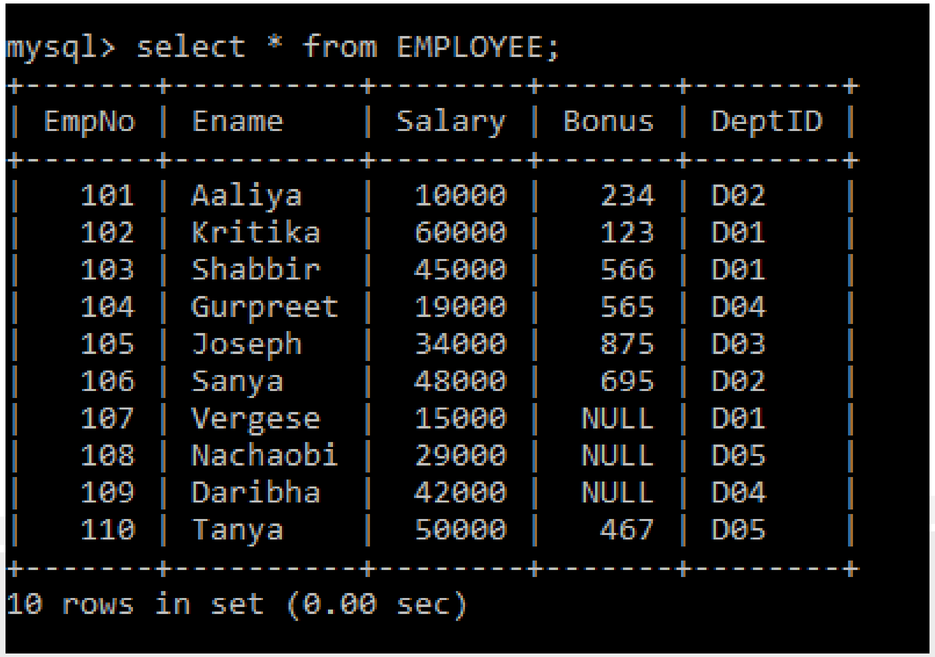


```
mysql> select EmpNo, Ename from Employee
    -> where DeptID = 'D01';
+-------+---------+
| EmpNo | Ename   |
+-------+---------+
|   102 | Kritika |
|   103 | Shabbir |
|   107 | Vergese |
+-------+---------+
3 rows in set (0.00 sec)
```

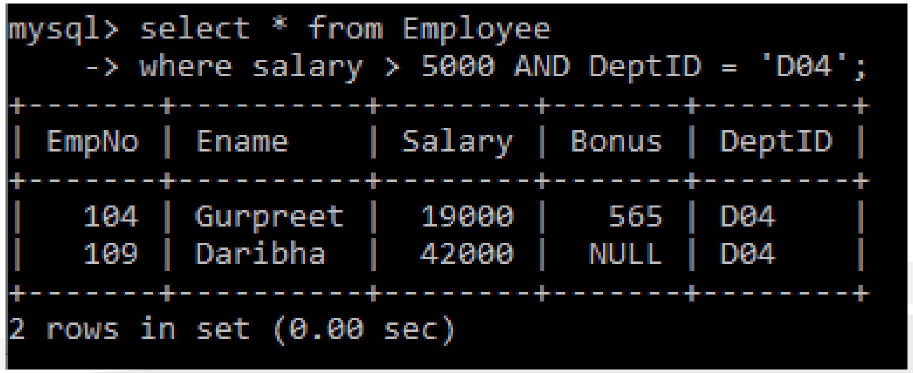
DQL: WHERE clause

- For e.g.) Display all the details of those employees of D04 department who earn more than 5000.

```
SELECT * FROM Employee
WHERE Salary > 5000 AND DeptID = 'D01';
```



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

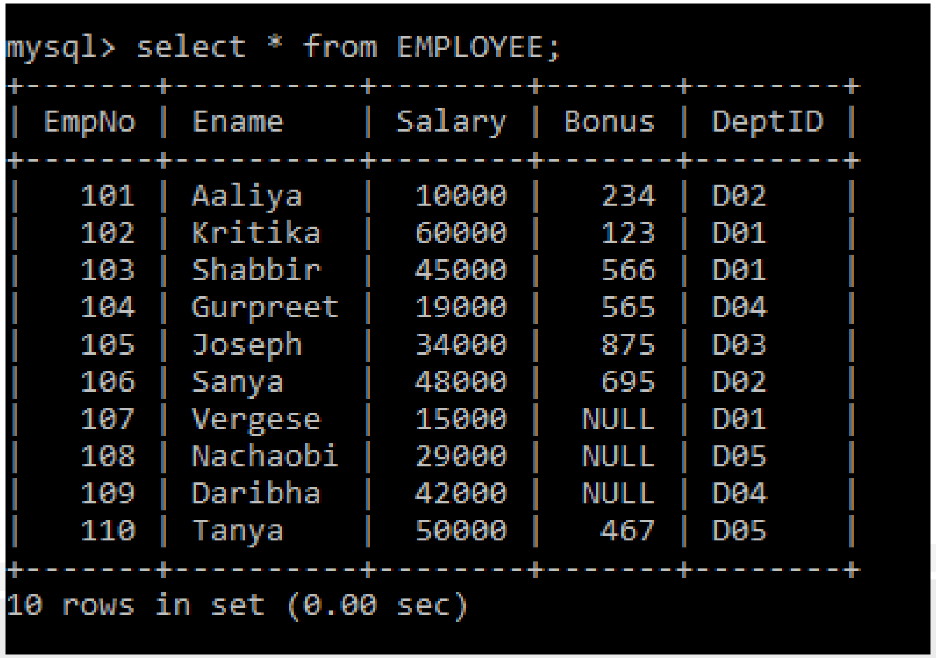


```
mysql> select * from Employee
    -> where salary > 5000 AND DeptID = 'D04';
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   104 | Gurpreet |  19000 |   565 | D04    |
|   109 | Daribha  |  42000 |  NULL | D04    |
+-------+----------+--------+-------+--------+
2 rows in set (0.00 sec)
```

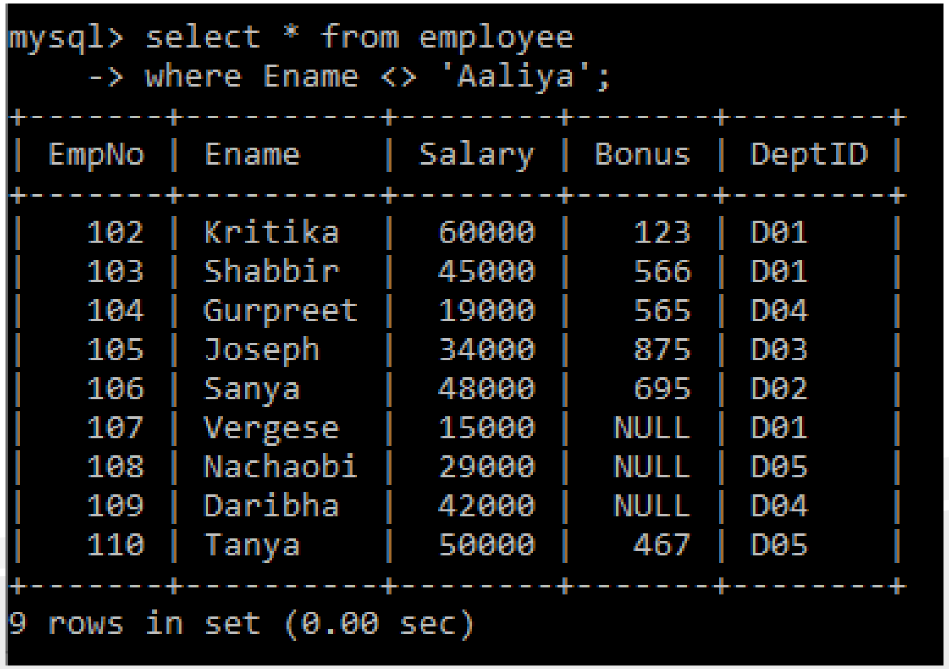
DQL: WHERE clause

- For e.g.) The following query selects records of all the employees except Aaliya.

SELECT * FROM Employee
WHERE Ename <> ‘Aaliya’;



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

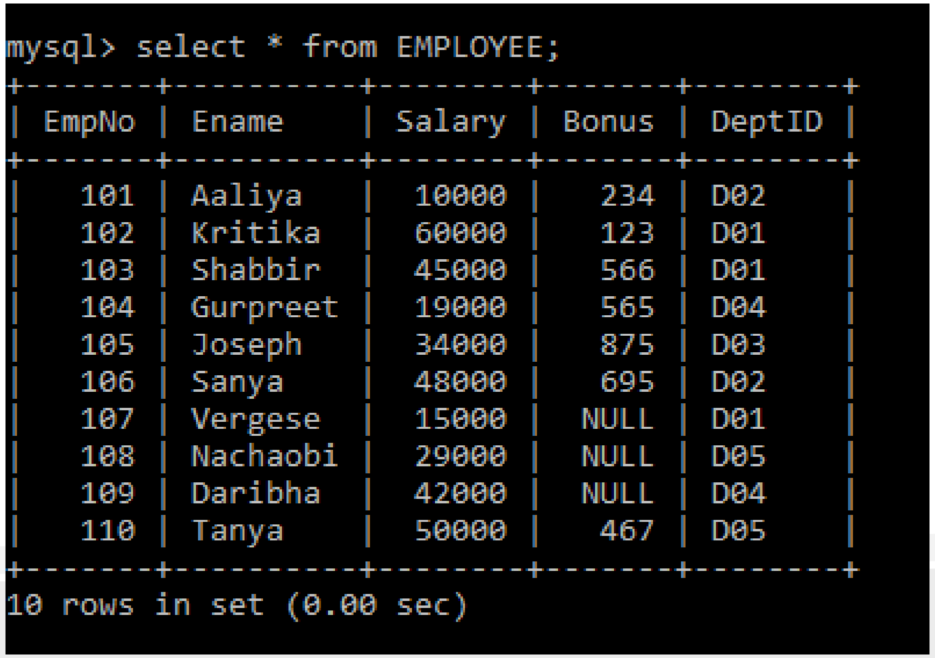


```
mysql> select * from employee
    -> where Ename <> 'Aaliya';
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
9 rows in set (0.00 sec)
```

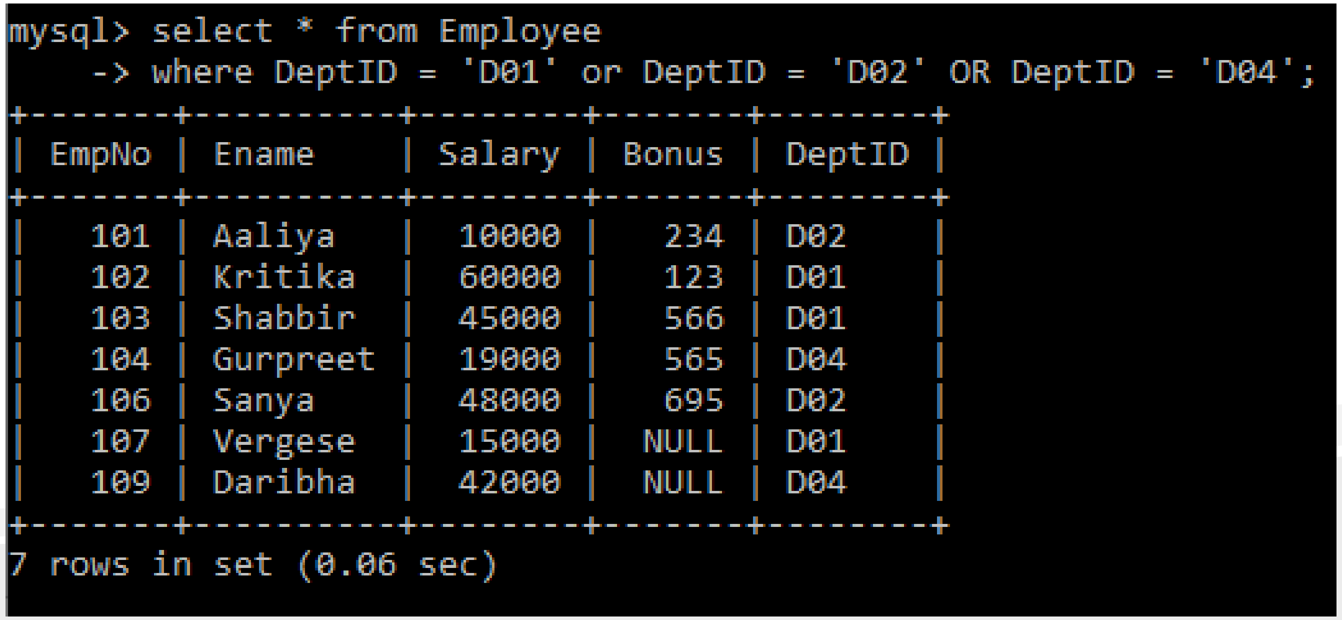
DQL: WHERE clause

- For e.g.) Select details of all the employees who work in departments having deptID = D01, D02 and D04

```
SELECT * FROM Employee
WHERE DeptID = 'D01' OR DeptID = 'D02' OR DeptID = 'D04';
```



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



```
mysql> select * from Employee
    -> where DeptID = 'D01' or DeptID = 'D02' OR DeptID = 'D04';
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   109 | Daribha  |  42000 |  NULL | D04    |
+-------+----------+--------+-------+--------+
7 rows in set (0.06 sec)
```

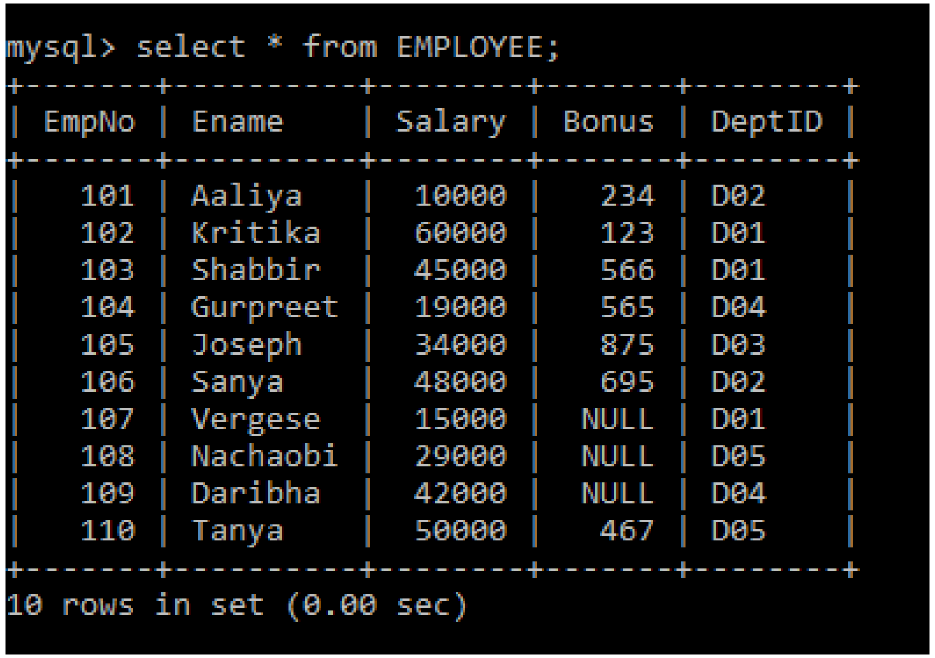
DQL: Membership Operators

- The IN operator compares a value with a set of values and returns the value belongs to that set.
- So the following query

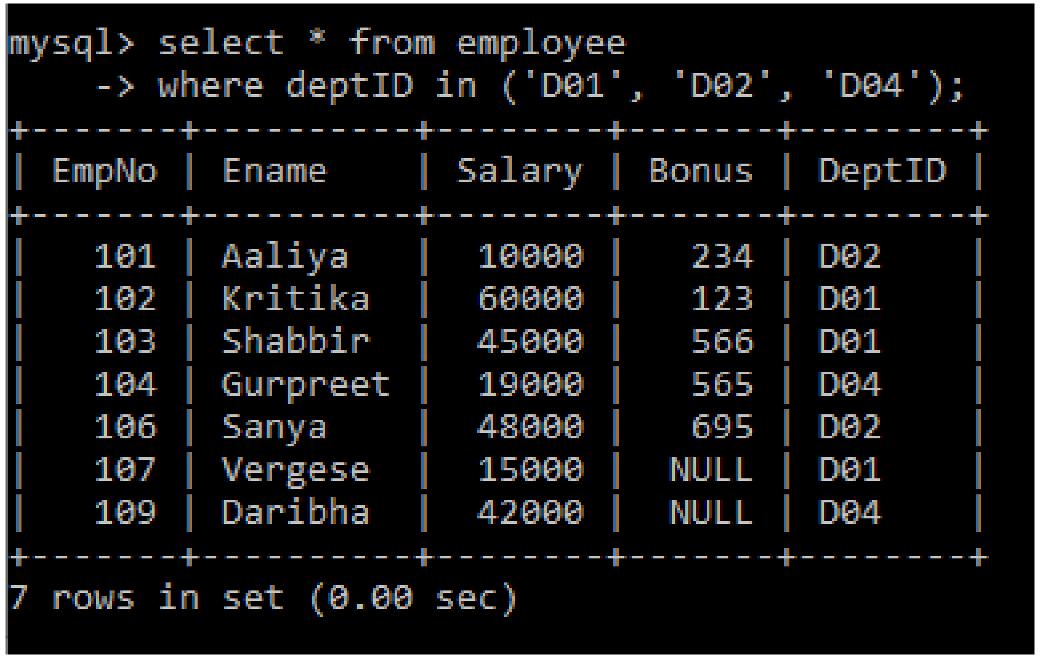
```
SELECT * FROM Employee
WHERE DeptID = ‘D01’ OR DeptID = ‘D02’ OR DeptID = ‘D04’;
```

can be written as:

```
SELECT * FROM Employee
WHERE DeptID IN ( ‘D01’ , ‘D02’, ‘D04’) ;
```



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



```
mysql> select * from employee
    -> where deptID in ('D01', 'D02', 'D04');
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   109 | Daribha  |  42000 |  NULL | D04    |
+-------+----------+--------+-------+--------+
7 rows in set (0.00 sec)
```

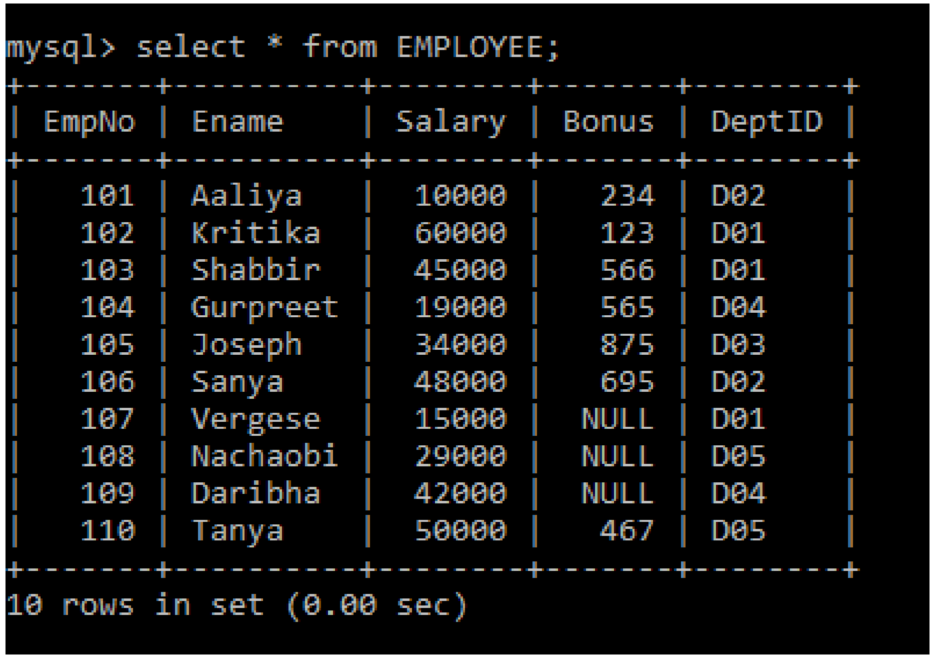
DQL: Membership Operators

- The NOT IN operator compares a value with a set of values and returns the value doesn't belong to that set.
- So the following query

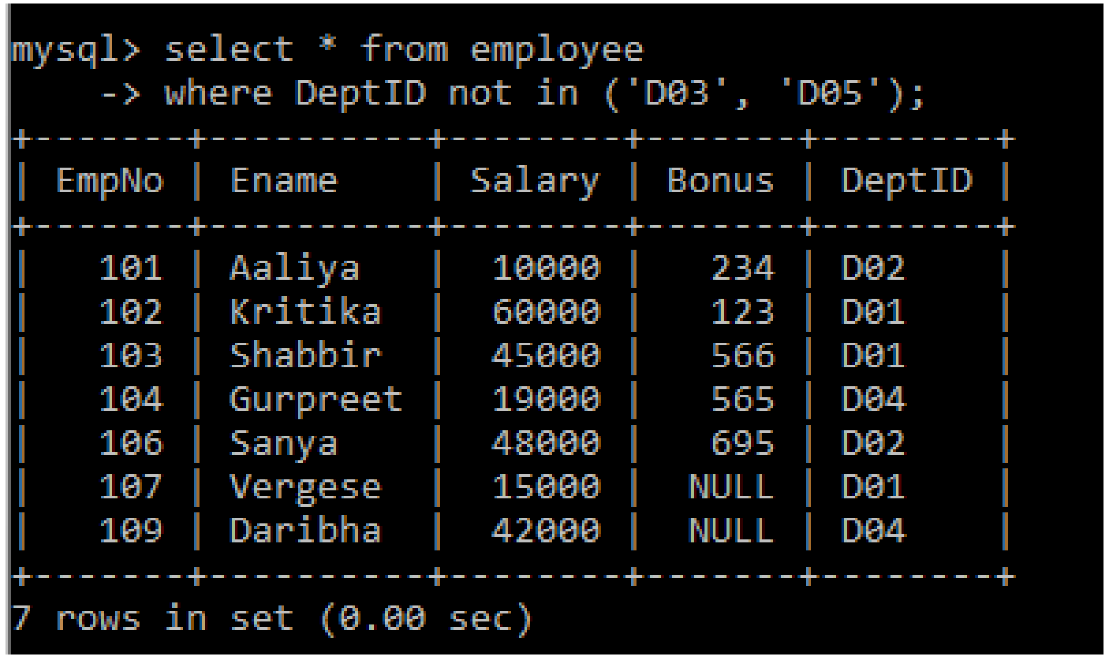
```
SELECT * FROM Employee
WHERE DeptID IN ( ‘D01’ , ‘D02’, ‘D04’) ;
```

can be written as:

```
SELECT * FROM Employee
WHERE DeptID NOT IN ( ‘D03’ , ‘D05’) ;
```



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

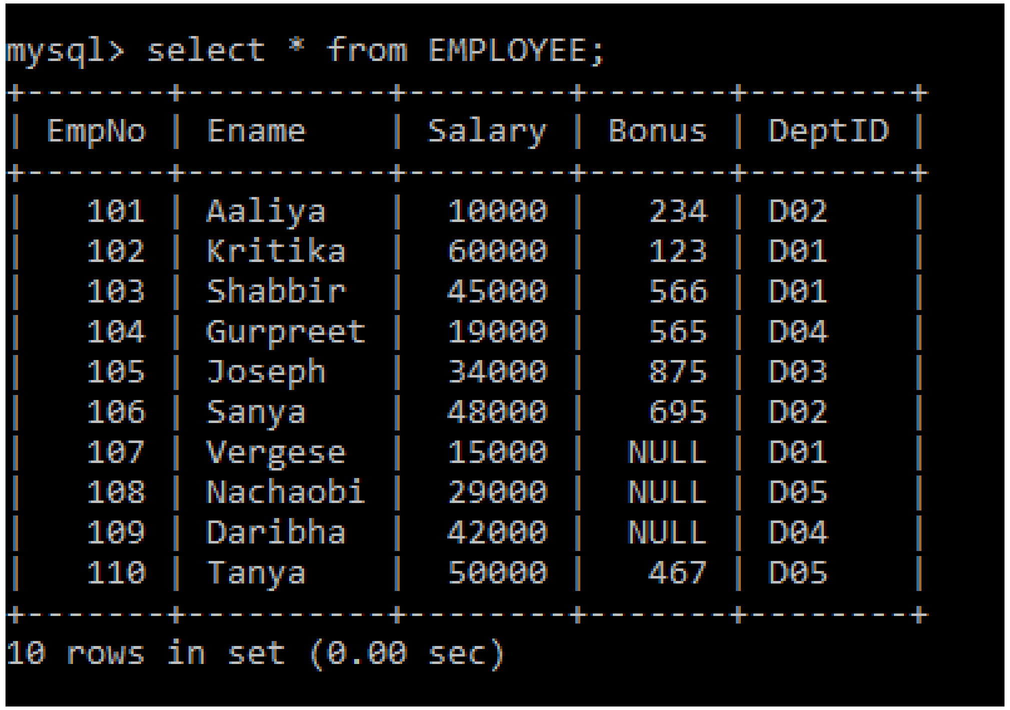


```
mysql> select * from employee
    -> where DeptID not in ('D03', 'D05');
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   109 | Daribha  |  42000 |  NULL | D04    |
+-------+----------+--------+-------+--------+
7 rows in set (0.00 sec)
```

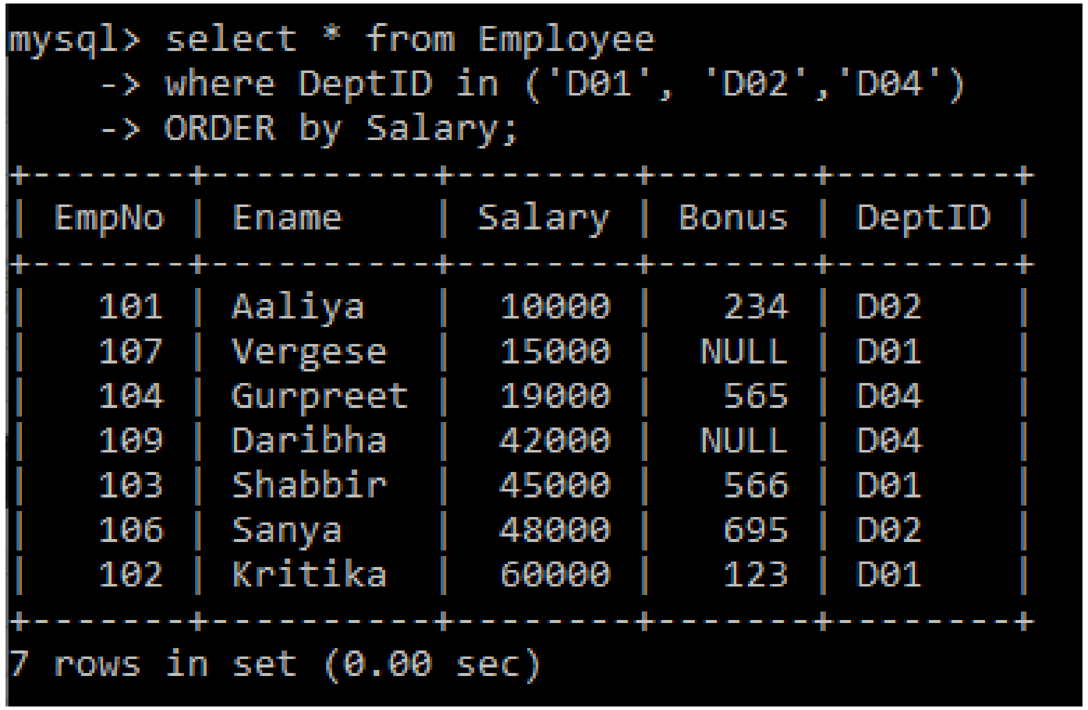
DQL: Order by Clause

- ORDER BY clause is used to display data in an ordered form with respect to a specified column.
- This query selects all the employees from Department D01, D02 or D04 in ascending order of their salaries

```
SELECT * FROM Employee
WHERE DeptID IN ( 'D01' , 'D02', 'D04')
ORDER BY Salary ;
```



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



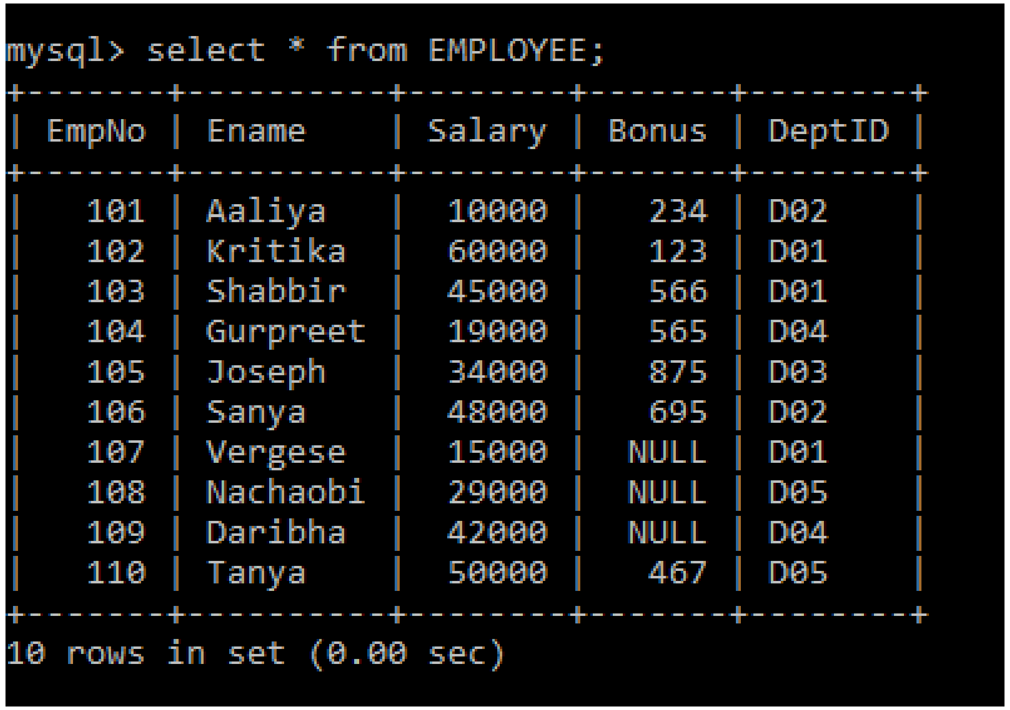
```
mysql> select * from Employee
    -> where DeptID in ('D01', 'D02','D04')
    -> ORDER by Salary;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
+-------+----------+--------+-------+--------+
7 rows in set (0.00 sec)
```

Note:

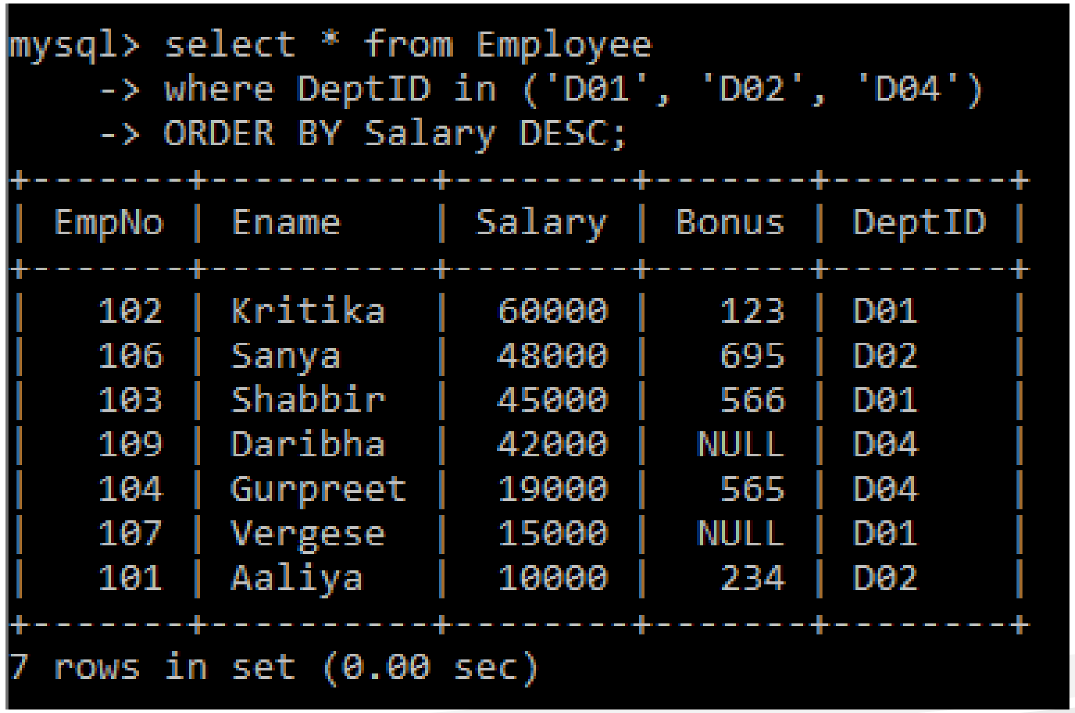
By default, ORDER BY displays records in ascending order of the specified column’s values.

Note:

To display the records in descending order, the DESC (means descending) keyword needs to be written with that column.



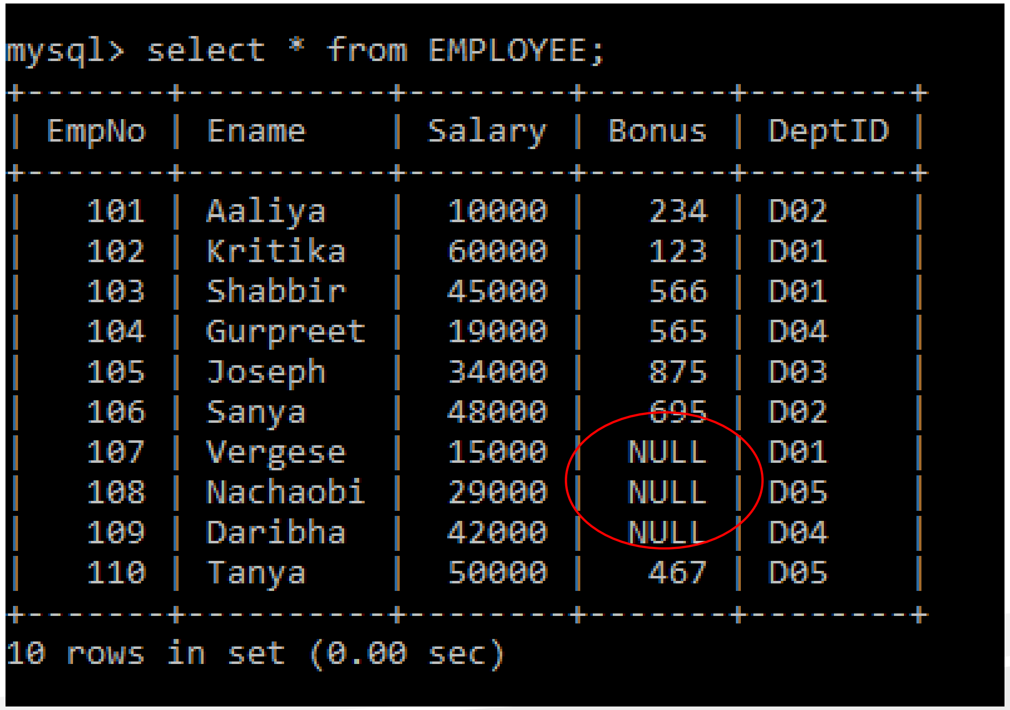
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



```
mysql> select * from Employee
    -> where DeptID in ('D01', 'D02', 'D04')
    -> ORDER BY Salary DESC;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   102 | Kritika  |  60000 |   123 | D01    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   101 | Aaliya   |  10000 |   234 | D02    |
+-------+----------+--------+-------+--------+
7 rows in set (0.00 sec)
```

DQL: Handling NULL Values

- SQL supports a special value called NULL to represent a missing or unknown value.
- For example,
  the Bonus column in the Employee table can have missing value for certain records



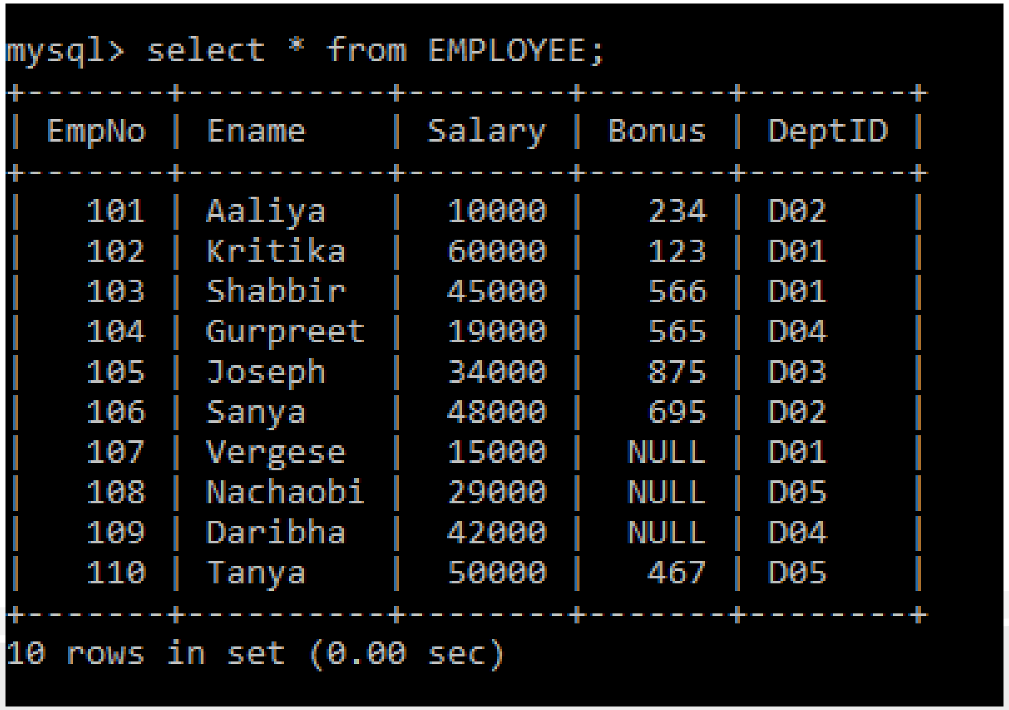
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```

DQL: Handling NULL Values

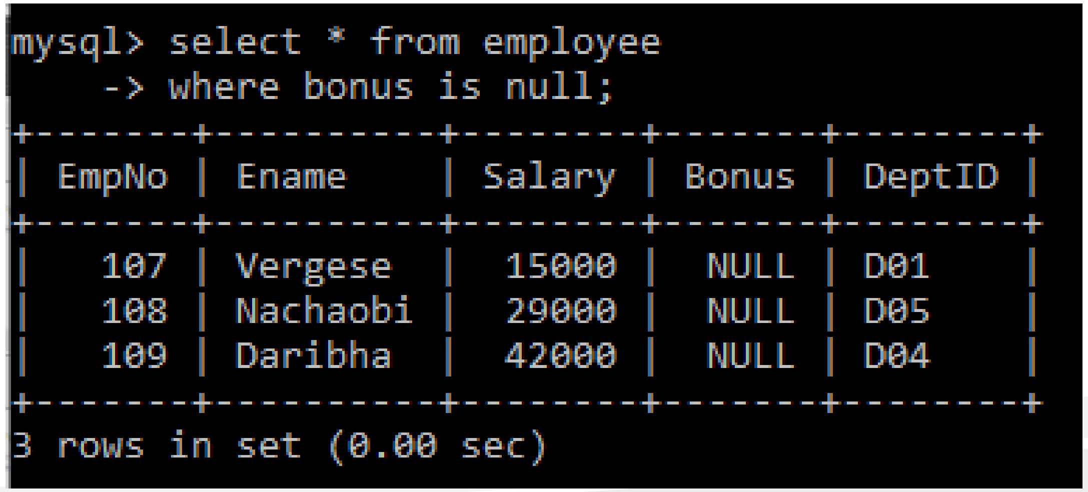
- It is important to note that NULL is different from 0 (zero). Also, any arithmetic operation performed with NULL value gives NULL.
- For example,
  5 + NULL = NULL because NULL is unknown hence the result is also unknown.
- In order to check for NULL value in a column, we use IS NULL operator in condition in where clause

DQL: IS NULL operator

- For example,
  The following query selects details of all the employees who haven't been given a bonus



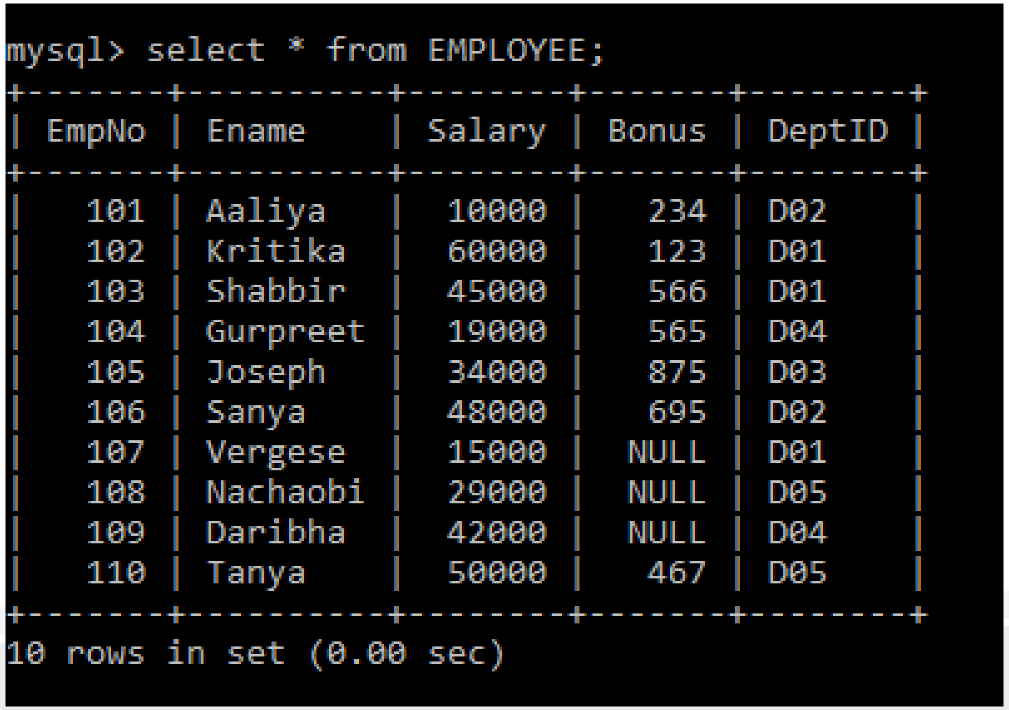
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



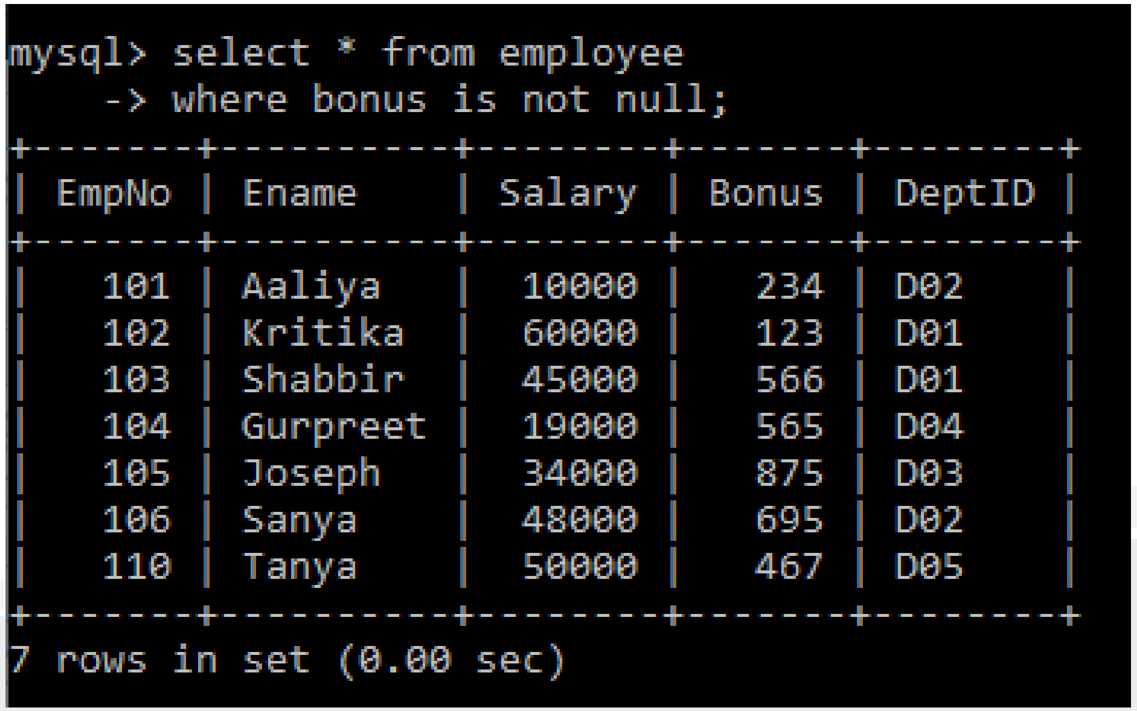
```
mysql> select * from employee
    -> where bonus is null;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
+-------+----------+--------+-------+--------+
3 rows in set (0.00 sec)
```

DQL: IS NOT NULL operator

- For example,
  The following query selects details of all the employees who have been given a bonus



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



```
mysql> select * from employee
    -> where bonus is not null;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
7 rows in set (0.00 sec)
```

DQL: Substring pattern matching

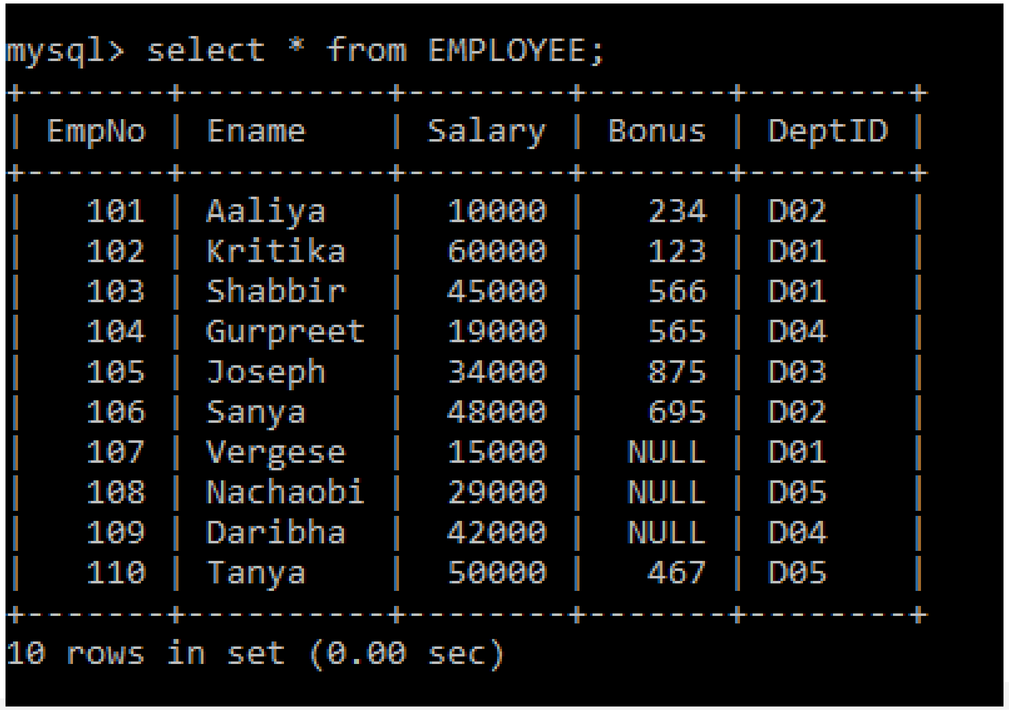
- Many a times we come across situations where we do not want to query by matching exact text or value. Rather, we are interested to find matching of only a few characters or values in column values.
- For example, to find out names starting with “T” or to find out pin codes starting with ‘60’. This is called substring pattern matching.
- We cannot match such patterns using = operator as we are not looking for an exact match.

DQL: LIKE operator

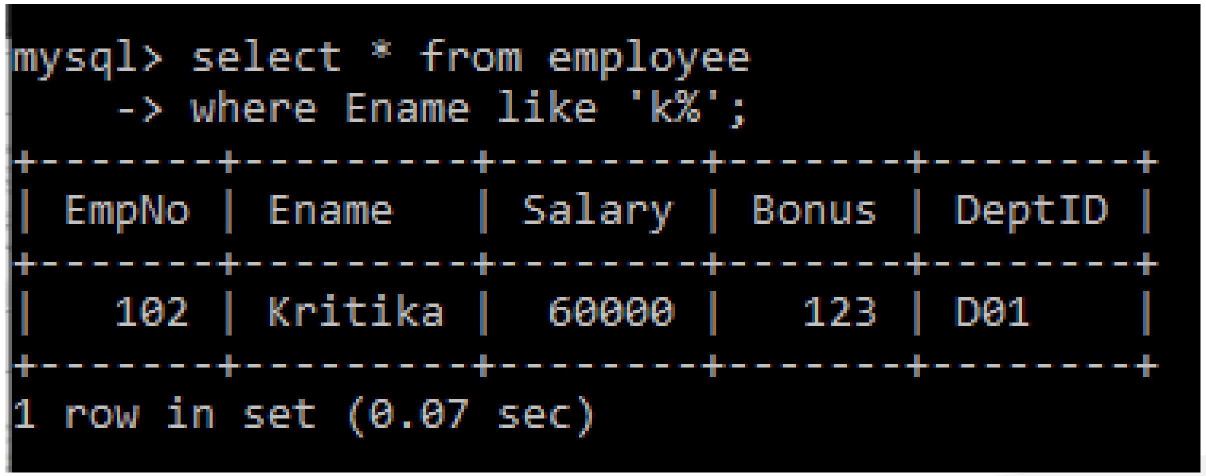
- SQL provides a LIKE operator that can be used with the WHERE clause to search for a specified pattern in a column.
- The LIKE operator makes use of the following two wild card characters:
  - % (per cent)- used to represent zero, one, or multiple characters
  - _ (underscore)- used to represent exactly a single character

DQL: LIKE operator examples

- The following query selects details of all those employees whose name starts with 'K'.



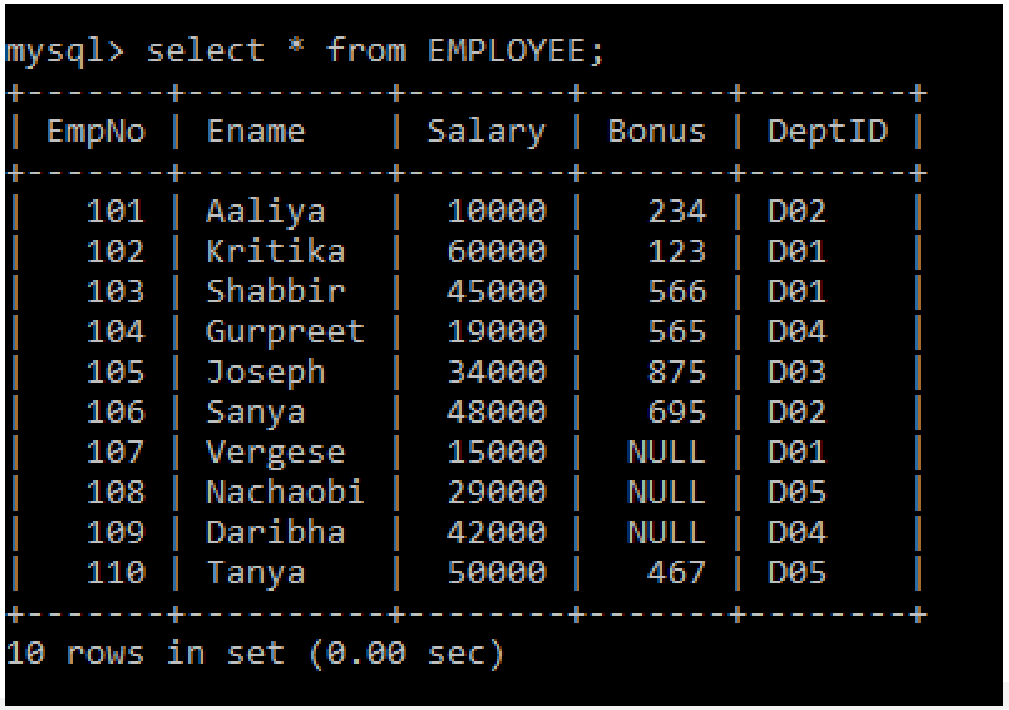
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



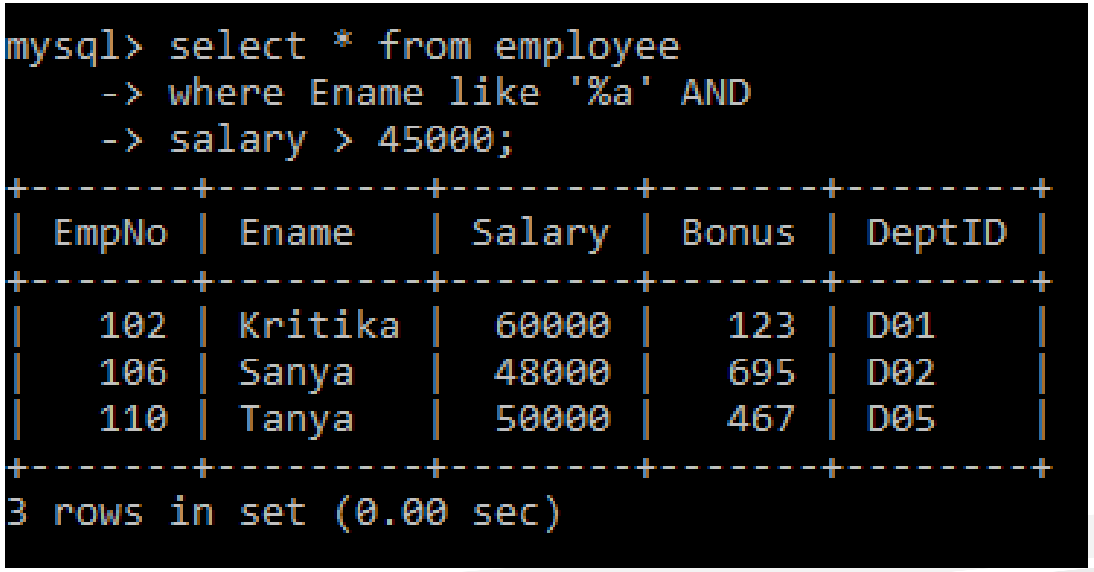
```
mysql> select * from employee
    -> where Ename like 'k%';
+-------+---------+--------+-------+--------+
| EmpNo | Ename   | Salary | Bonus | DeptID |
+-------+---------+--------+-------+--------+
|   102 | Kritika |  60000 |   123 | D01    |
+-------+---------+--------+-------+--------+
1 row in set (0.07 sec)
```

DQL: LIKE operator examples

- The following query selects details of all those employees whose name ends with 'a', and gets a salary more than 45000.



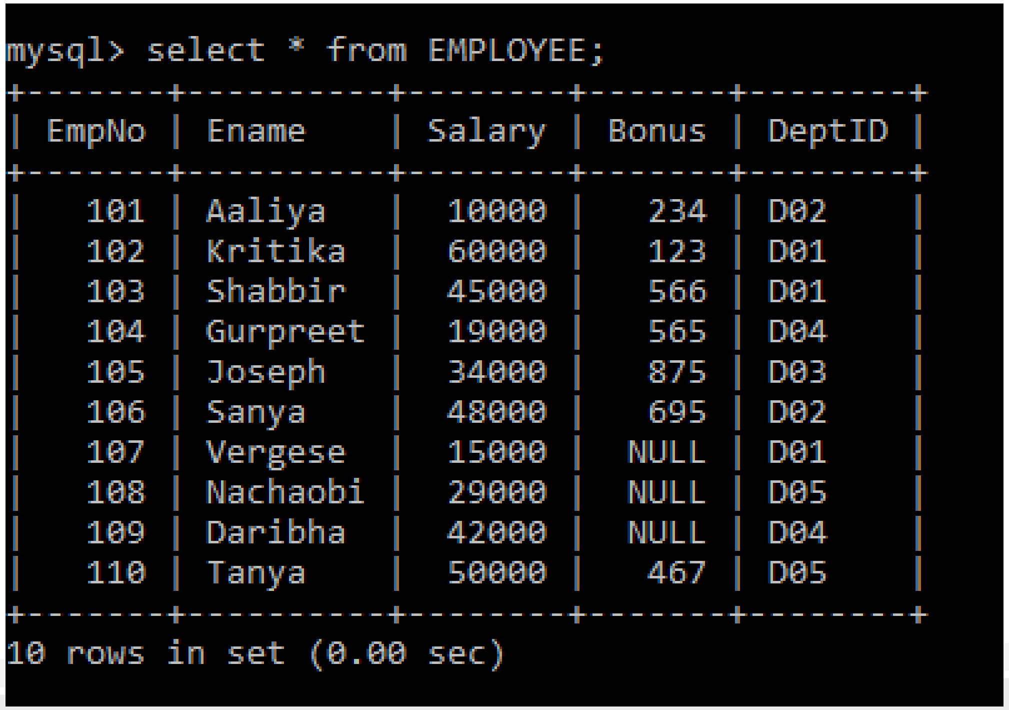
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



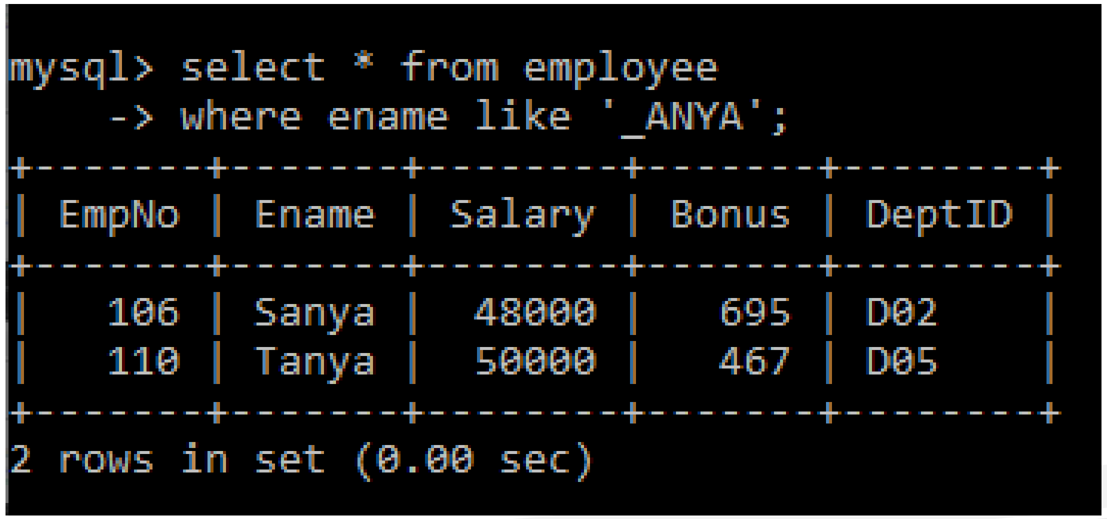
```
mysql> select * from employee
    -> where Ename like '%a' AND
    -> salary > 45000;
+-------+---------+--------+-------+--------+
| EmpNo | Ename   | Salary | Bonus | DeptID |
+-------+---------+--------+-------+--------+
|   102 | Kritika |  60000 |   123 | D01    |
|   106 | Sanya   |  48000 |   695 | D02    |
|   110 | Tanya   |  50000 |   467 | D05    |
+-------+---------+--------+-------+--------+
3 rows in set (0.00 sec)
```

DQL: LIKE operator examples

- The following query selects details of all those employees whose name consists of exactly 5 letters and starts with any letter but has ‘ANYA’ after that.



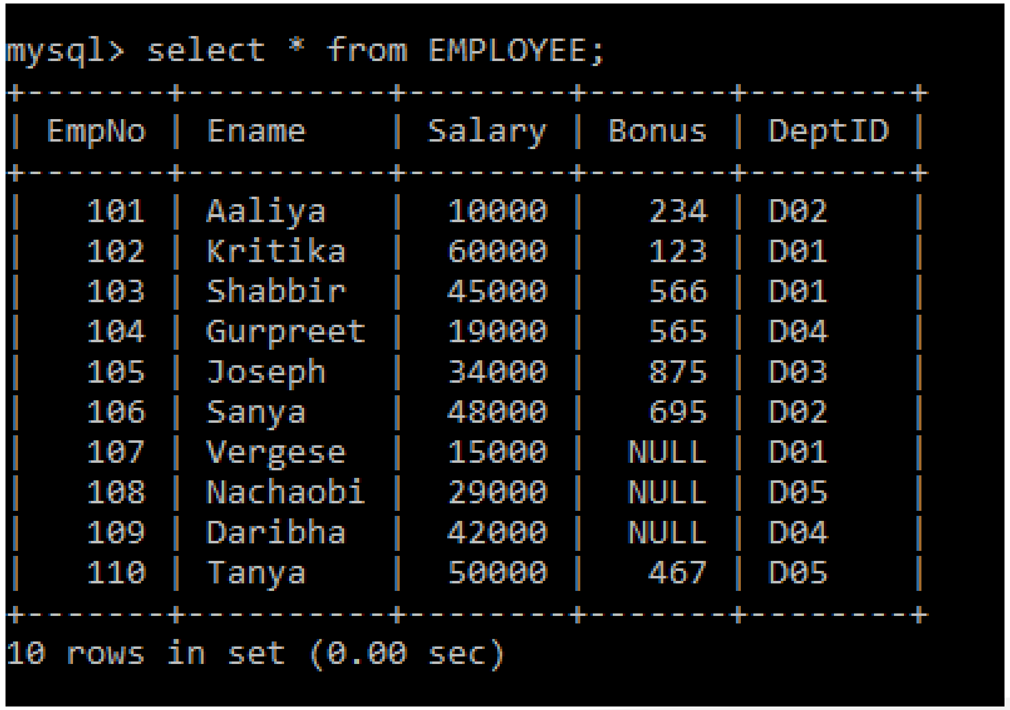
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



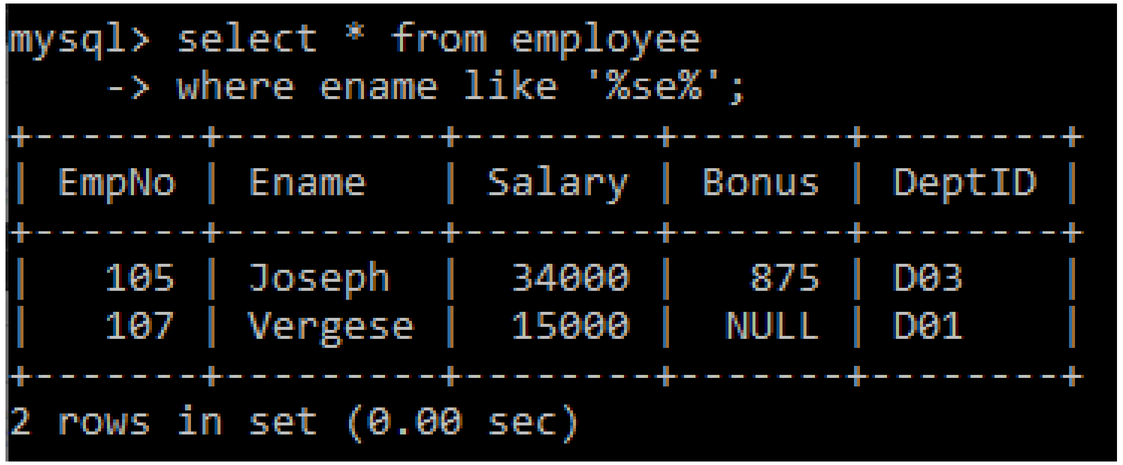
```
mysql> select * from employee
    -> where ename like '_ANYA';
+-------+-------+--------+-------+--------+
| EmpNo | Ename | Salary | Bonus | DeptID |
+-------+-------+--------+-------+--------+
|   106 | Sanya |  48000 |   695 | D02    |
|   110 | Tanya |  50000 |   467 | D05    |
+-------+-------+--------+-------+--------+
2 rows in set (0.00 sec)
```

DQL: LIKE operator examples

- The following query selects names of all employees containing 'se' as a substring in name.



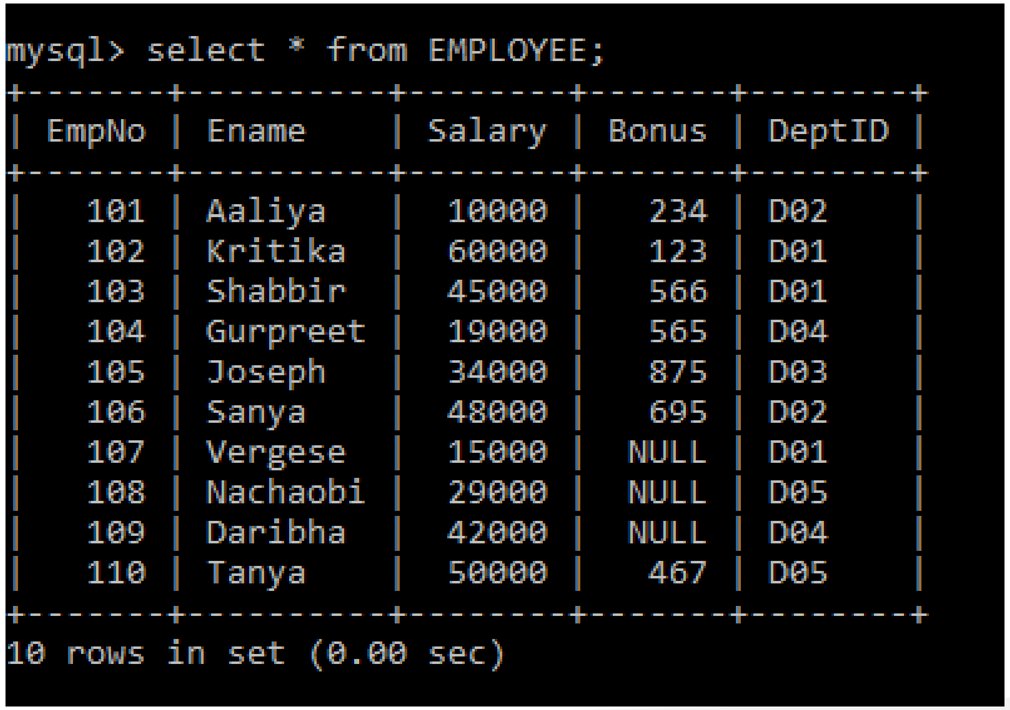
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



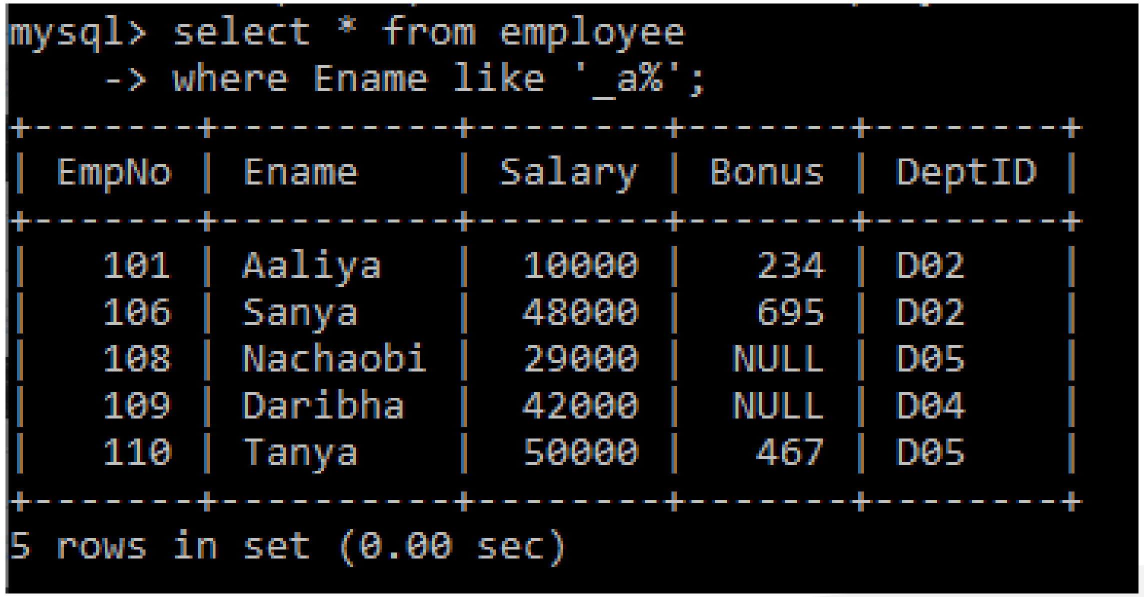
```
mysql> select * from employee
    -> where ename like '%se%';
+-------+---------+--------+-------+--------+
| EmpNo | Ename   | Salary | Bonus | DeptID |
+-------+---------+--------+-------+--------+
|   105 | Joseph  |  34000 |   875 | D03    |
|   107 | Vergese |  15000 |  NULL | D01    |
+-------+---------+--------+-------+--------+
2 rows in set (0.00 sec)
```

DQL: LIKE operator examples

- The following query selects names of all employees containing 'a' as the second character.



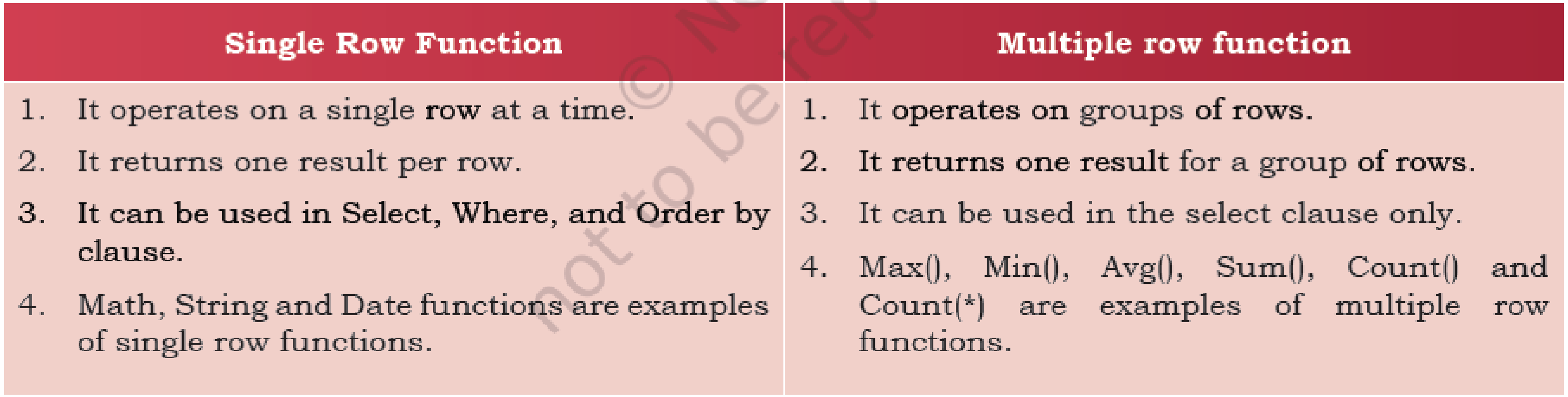
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



```
mysql> select * from employee
    -> where Ename like '_a%';
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
5 rows in set (0.00 sec)
```

DQL: Aggregate Functions

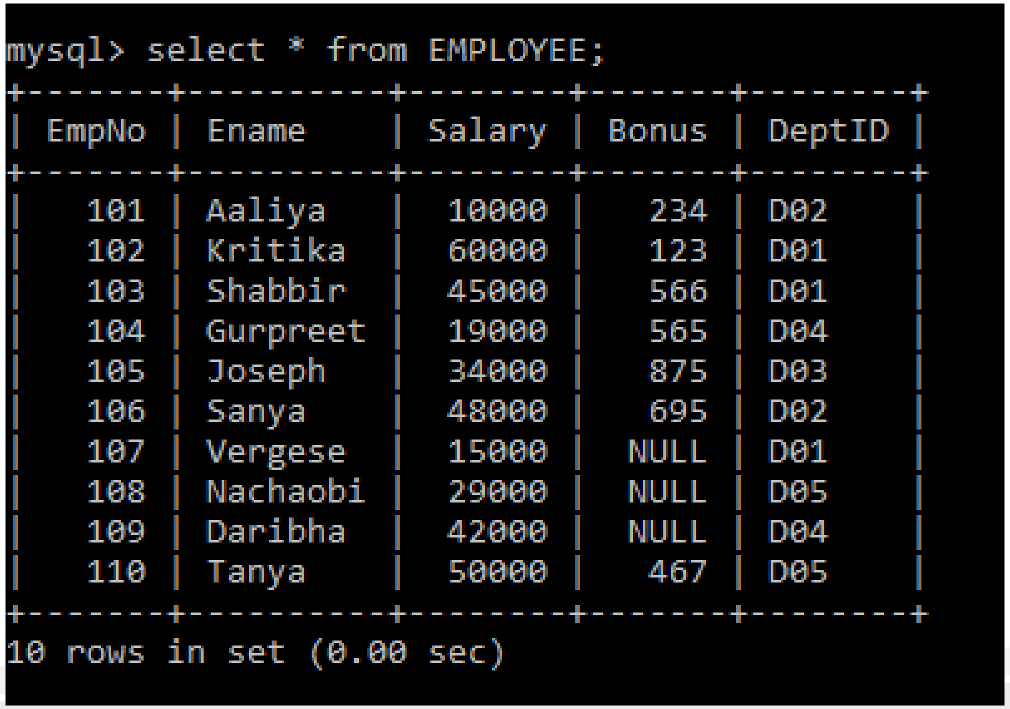
- Aggregate functions are also called Multiple Row functions. These functions work on a set of records as a whole and return a single value for each column of the records on which the function is applied.



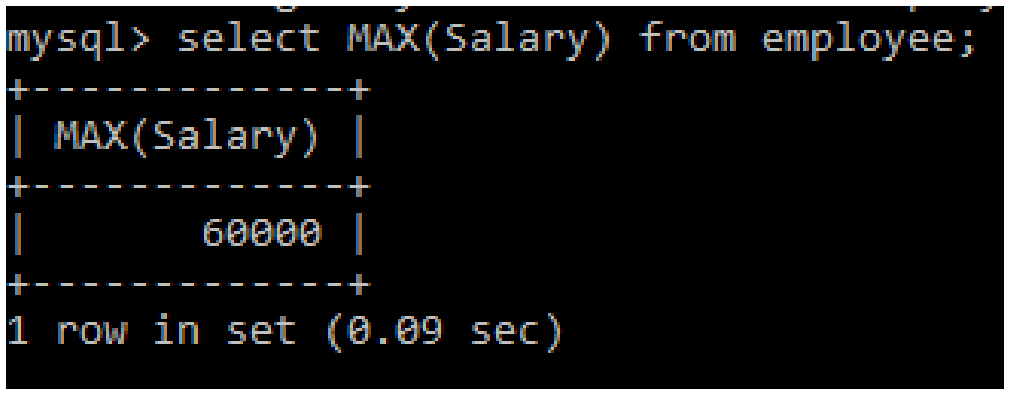
Single Row Function	Multiple row function
1. It operates on a single row at a time.	1. It operates on groups of rows.
2. It returns one result per row.	2. It returns one result for a group of rows.
3. It can be used in Select, Where, and Order by clause.	3. It can be used in the select clause only.
4. Math, String and Date functions are examples of single row functions.	4. Max(), Min(), Avg(), Sum(), Count() and Count(*) are examples of multiple row functions.

DQL: Aggregate Functions: MAX (column)

Function	Description
MAX(Column)	Returns the largest value from the specified column.



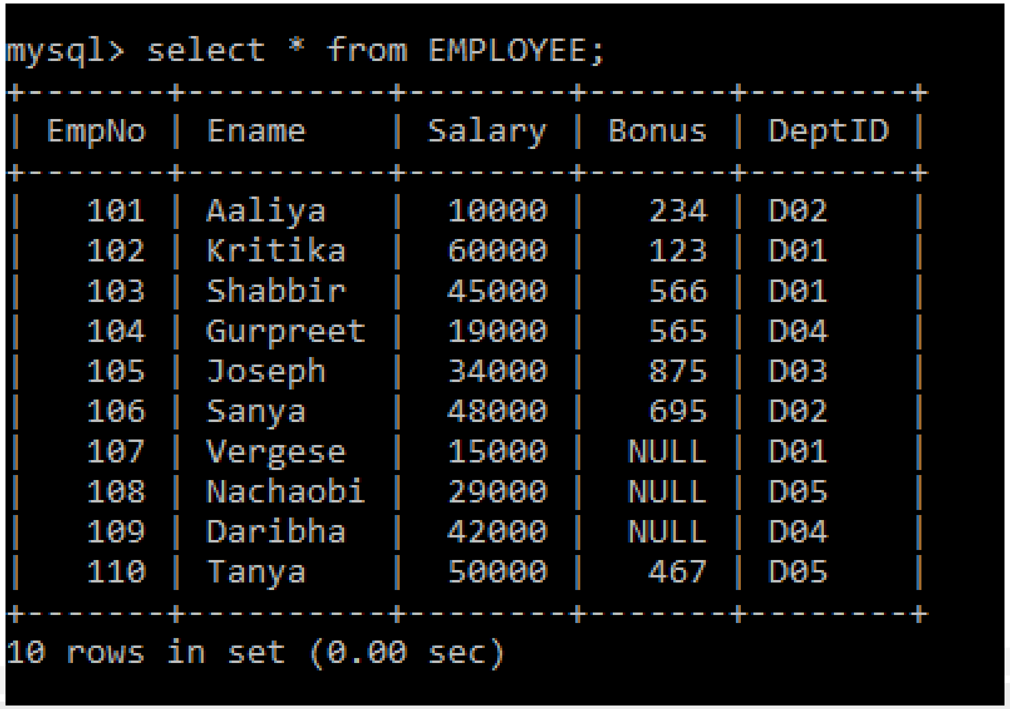
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



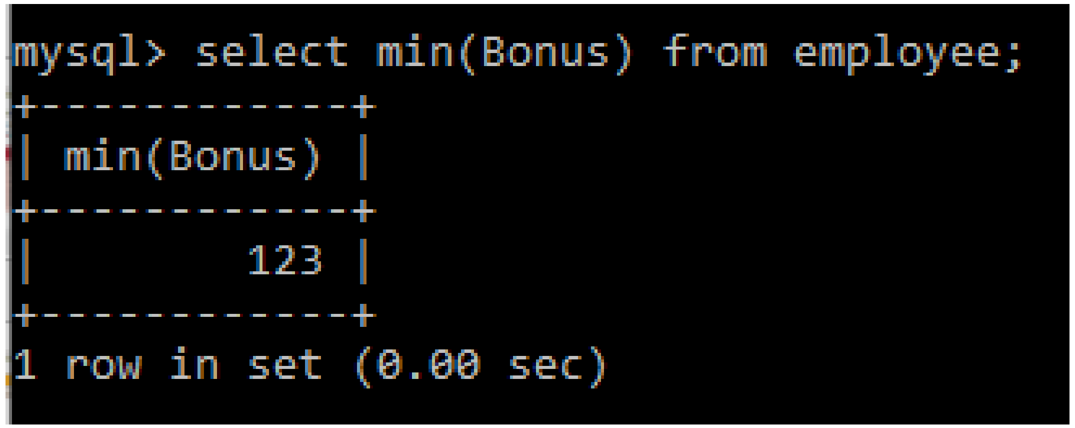
```
mysql> select MAX(Salary) from employee;
+-------------+
| MAX(Salary) |
+-------------+
|       60000 |
+-------------+
1 row in set (0.09 sec)
```

DQL: Aggregate Functions: MIN (column)

Function	Description
MIN(Column)	Returns the smallest value from the specified column.



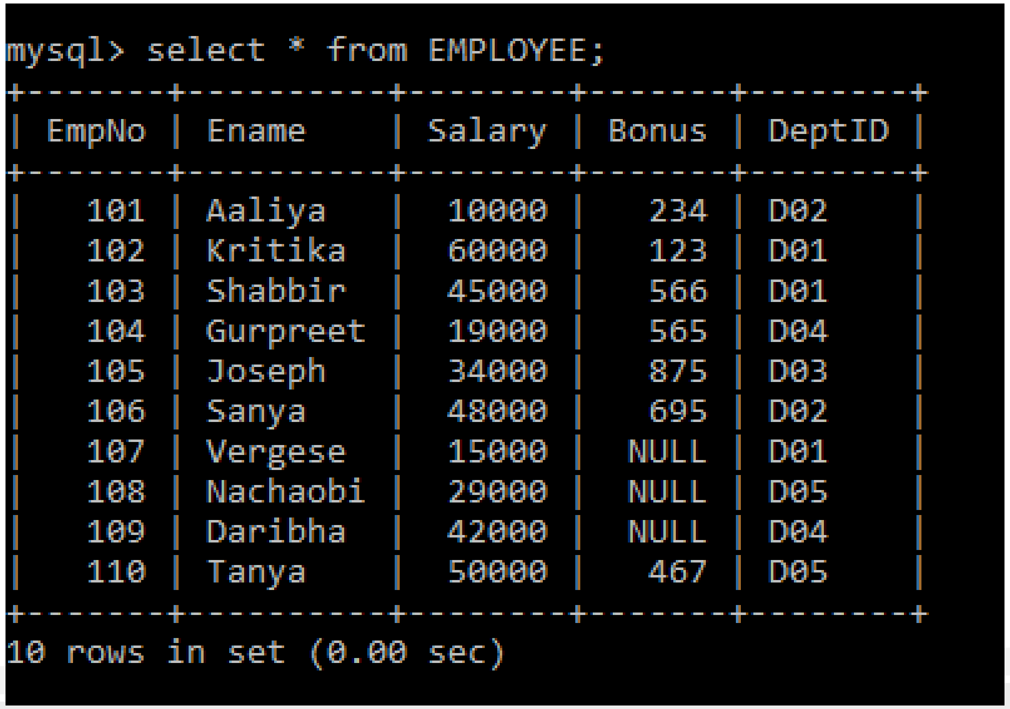
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



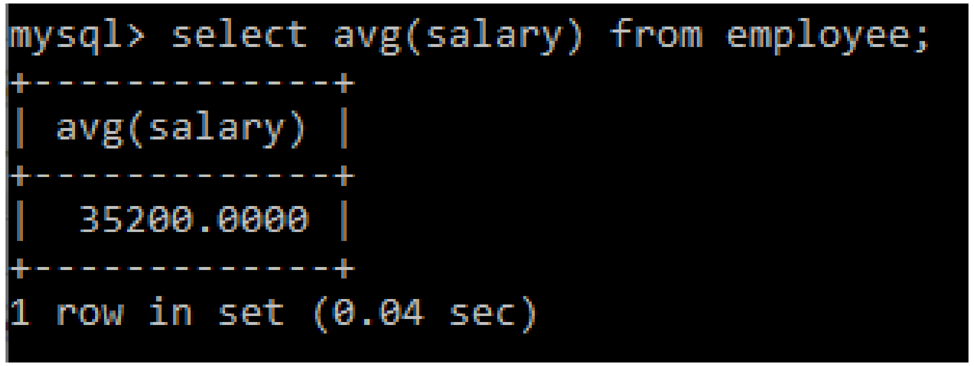
```
mysql> select min(Bonus) from employee;
+------------+
| min(Bonus) |
+------------+
|        123 |
+------------+
1 row in set (0.00 sec)
```

DQL: Aggregate Functions: AVG(column)

Function	Description
AVG(Column)	Returns the average of the values in the specified column.



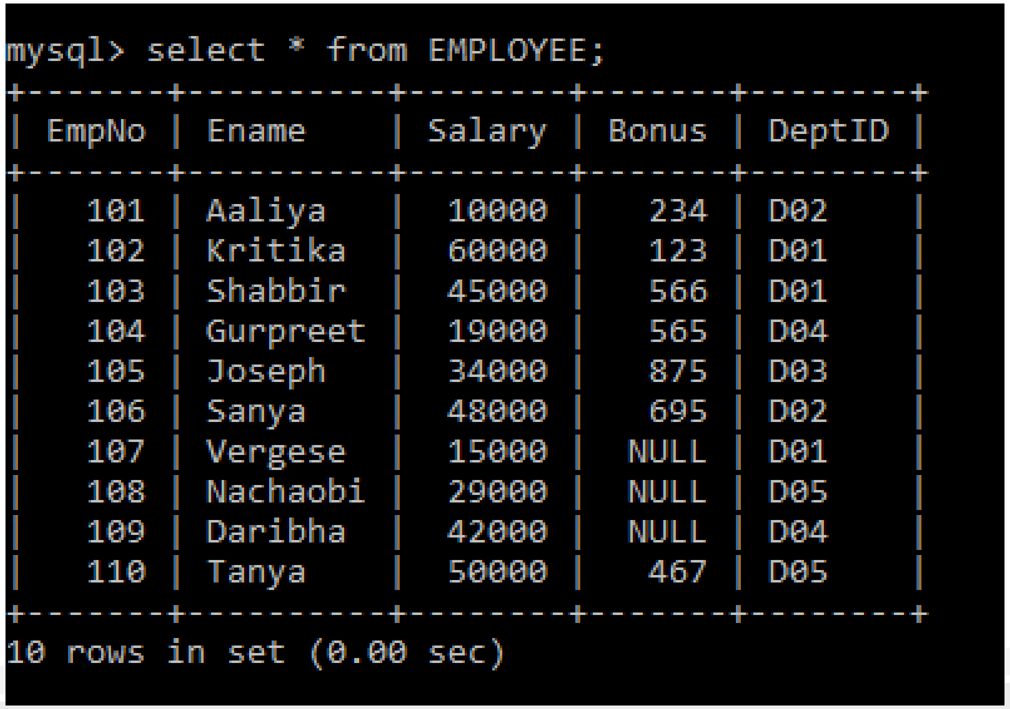
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



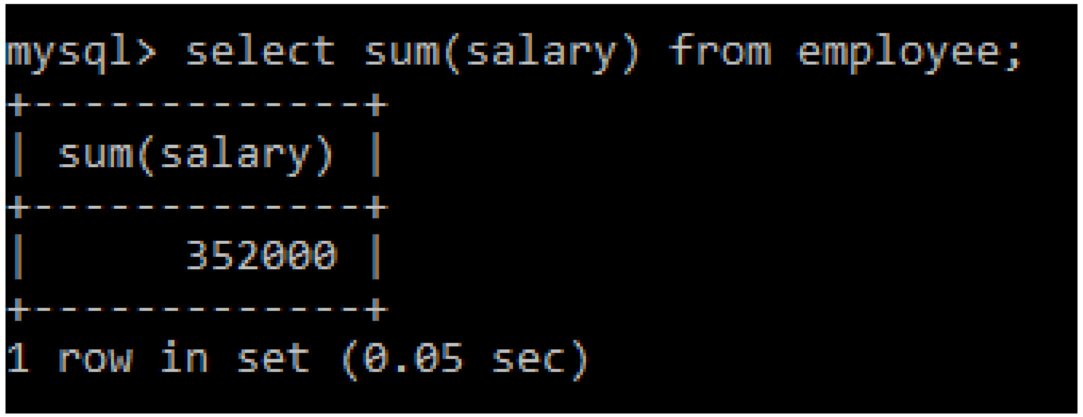
```
mysql> select avg(salary) from employee;
+-------------+
| avg(salary) |
+-------------+
|  35200.0000 |
+-------------+
1 row in set (0.04 sec)
```

DQL: Aggregate Functions: SUM(column)

Function	Description
SUM(Column)	Returns the sum of the values for the specified column.



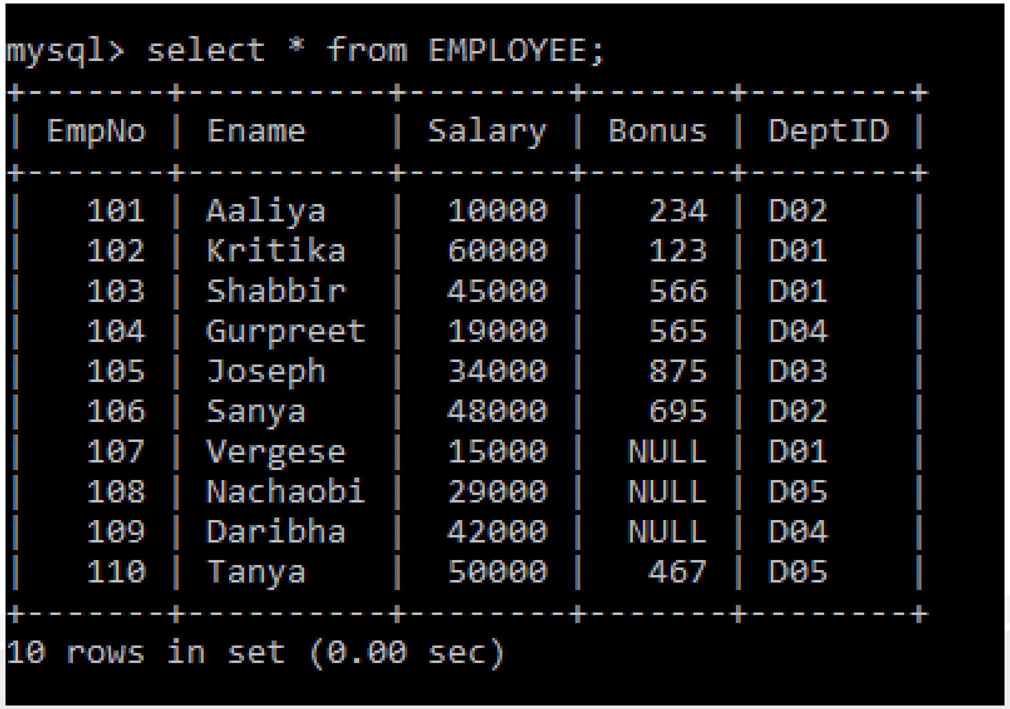
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



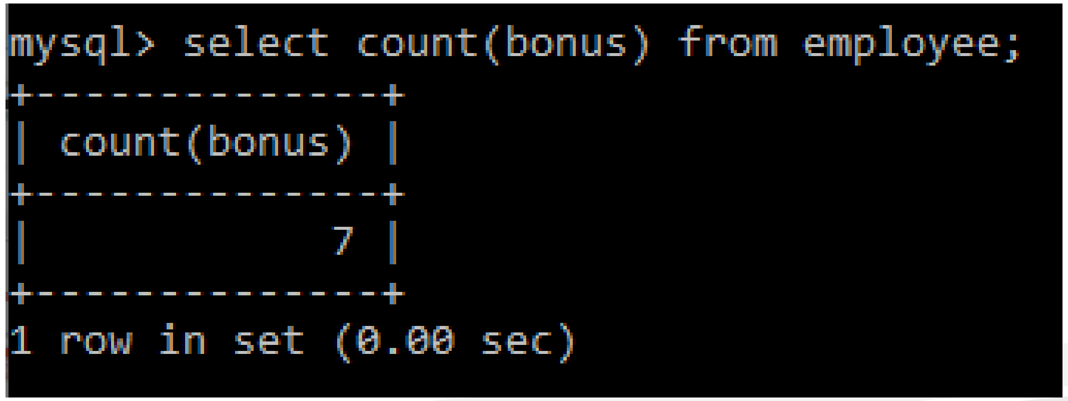
```
mysql> select sum(salary) from employee;
+-------------+
| sum(salary) |
+-------------+
|      352000 |
+-------------+
1 row in set (0.05 sec)
```

DQL: Aggregate Functions: count(column)

Function	Description
count(column)	Returns the number of values in the specified column ignoring the NULL values.



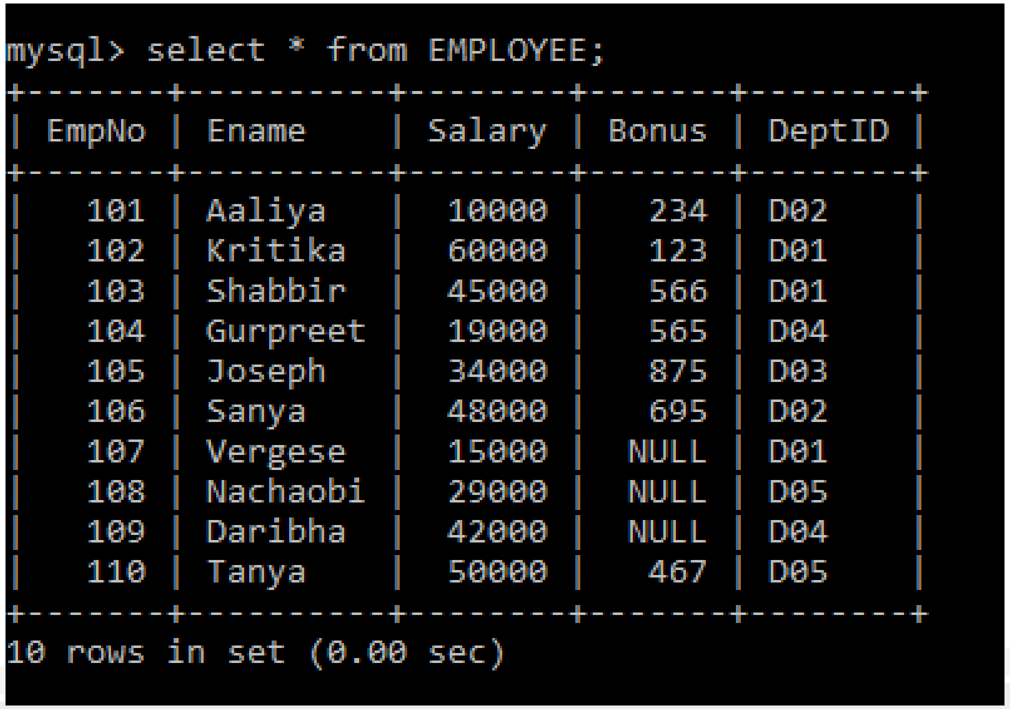
```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



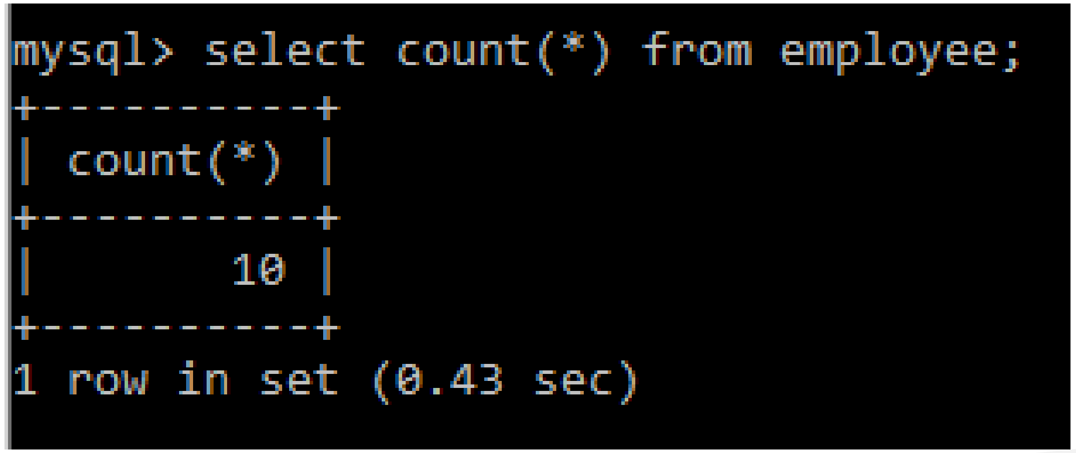
```
mysql> select count(bonus) from employee;
+--------------+
| count(bonus) |
+--------------+
|            7 |
+--------------+
1 row in set (0.00 sec)
```

DQL: Aggregate Functions: SUM(column)

Function	Description
count(*)	Returns the number of records in a table.



```
mysql> select * from EMPLOYEE;
+-------+----------+--------+-------+--------+
| EmpNo | Ename    | Salary | Bonus | DeptID |
+-------+----------+--------+-------+--------+
|   101 | Aaliya   |  10000 |   234 | D02    |
|   102 | Kritika  |  60000 |   123 | D01    |
|   103 | Shabbir  |  45000 |   566 | D01    |
|   104 | Gurpreet |  19000 |   565 | D04    |
|   105 | Joseph   |  34000 |   875 | D03    |
|   106 | Sanya    |  48000 |   695 | D02    |
|   107 | Vergese  |  15000 |  NULL | D01    |
|   108 | Nachaobi |  29000 |  NULL | D05    |
|   109 | Daribha  |  42000 |  NULL | D04    |
|   110 | Tanya    |  50000 |   467 | D05    |
+-------+----------+--------+-------+--------+
10 rows in set (0.00 sec)
```



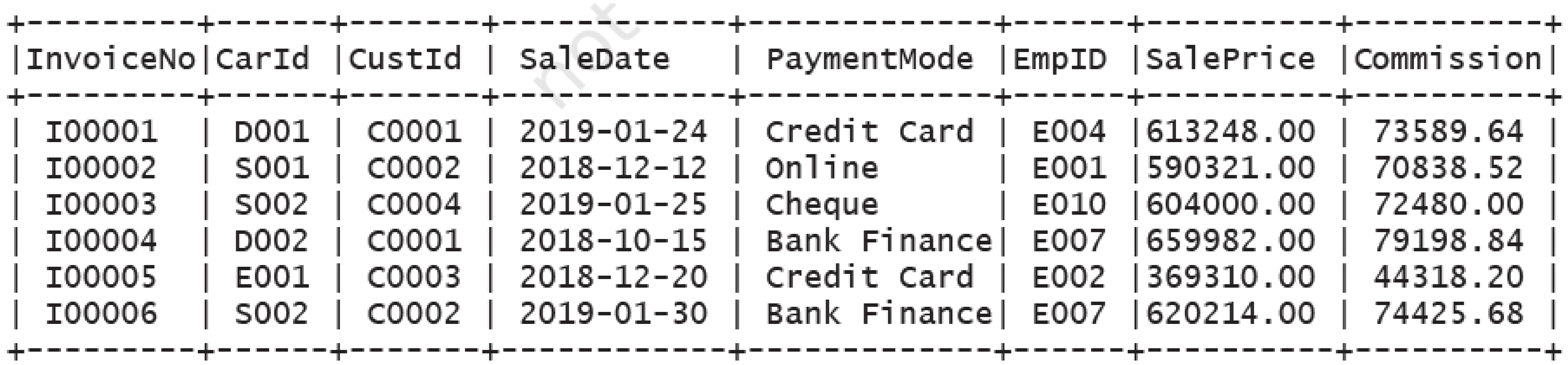
```
mysql> select count(*) from employee;
+----------+
| count(*) |
+----------+
|       10 |
+----------+
1 row in set (0.43 sec)
```

DQL: Group by and Having clause

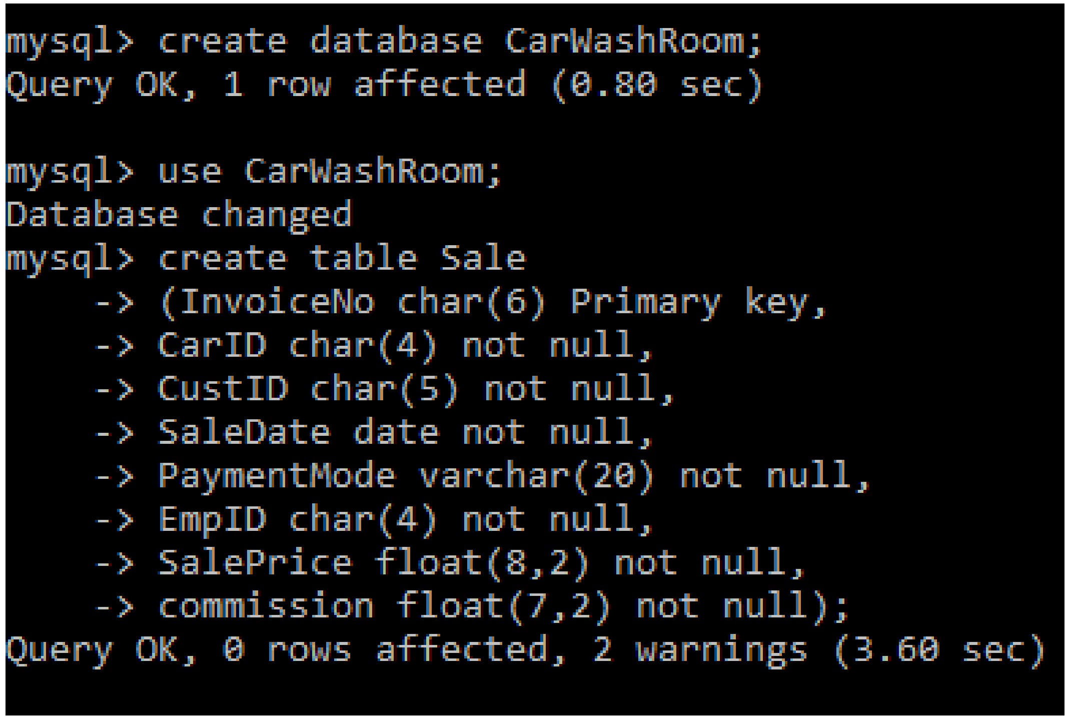
- At times we need to fetch a group of rows on the basis of common values in a column. This can be done using a group by clause.
- It groups the rows together that contains the same values in a specified column. We can use the aggregate functions (COUNT, MAX, MIN, AVG and SUM) to work on the grouped values.
- HAVING Clause in SQL is used to specify conditions on the rows with Group By clause.

DQL: Group by and Having clause

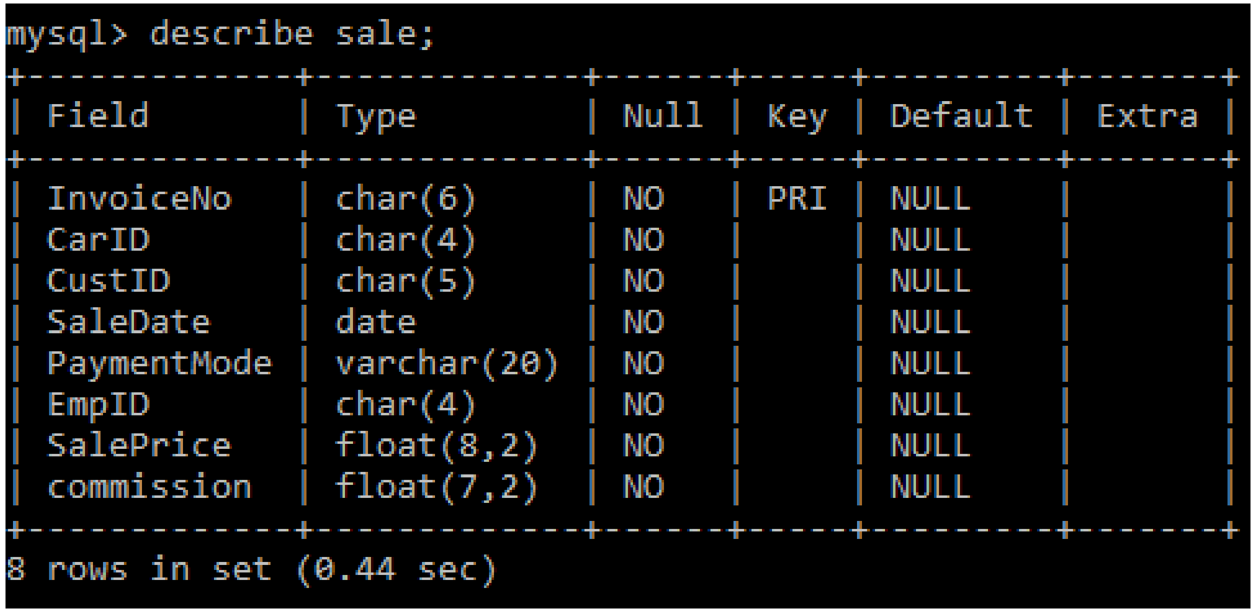
- In order to see the queries based on Group by and having clause, let's create a table 'Sale' for the database 'CarWashRoom' with the below data:



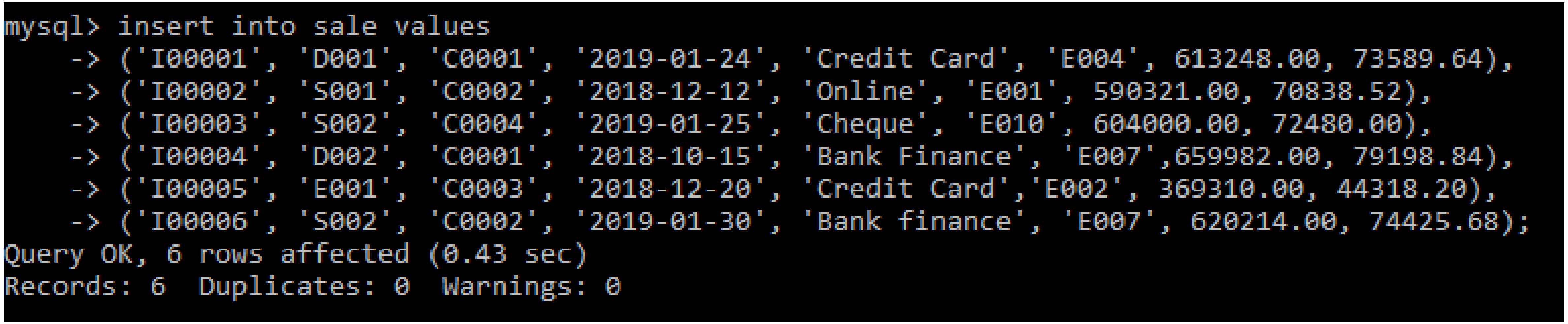
```
+---------+------+-------+-----------+-------------+------+----------+----------+
|InvoiceNo|CarId |CustId | SaleDate  | PaymentMode |EmpID |SalePrice |Commission|
+---------+------+-------+-----------+-------------+------+----------+----------+
| I00001  | D001 | C0001 | 2019-01-24 | Credit Card | E004 |613248.00 | 73589.64 |
| I00002  | S001 | C0002 | 2018-12-12 | Online      | E001 |590321.00 | 70838.52 |
| I00003  | S002 | C0004 | 2019-01-25 | Cheque      | E010 |604000.00 | 72480.00 |
| I00004  | D002 | C0001 | 2018-10-15 | Bank Finance| E007 |659982.00 | 79198.84 |
| I00005  | E001 | C0003 | 2018-12-20 | Credit Card | E002 |369310.00 | 44318.20 |
| I00006  | S002 | C0002 | 2019-01-30 | Bank Finance| E007 |620214.00 | 74425.68 |
+---------+------+-------+-----------+-------------+------+----------+----------+
```



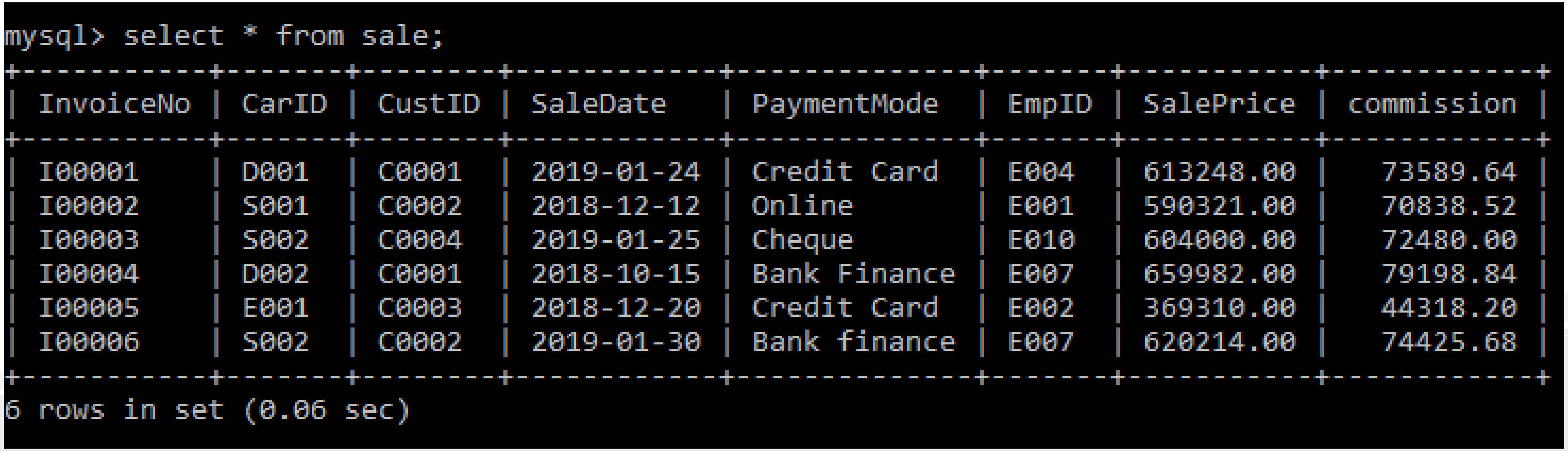
```
mysql> create database CarWashRoom;
Query OK, 1 row affected (0.80 sec)

mysql> use CarWashRoom;
Database changed
mysql> create table Sale
    -> (InvoiceNo char(6) Primary key,
    -> CarID char(4) not null,
    -> CustID char(5) not null,
    -> SaleDate date not null,
    -> PaymentMode varchar(20) not null,
    -> EmpID char(4) not null,
    -> SalePrice float(8,2) not null,
    -> commission float(7,2) not null);
Query OK, 0 rows affected, 2 warnings (3.60 sec)
```



```
mysql> describe sale;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| InvoiceNo   | char(6)     | NO   | PRI | NULL    |       |
| CarID       | char(4)     | NO   |     | NULL    |       |
| CustID      | char(5)     | NO   |     | NULL    |       |
| SaleDate    | date        | NO   |     | NULL    |       |
| PaymentMode | varchar(20) | NO   |     | NULL    |       |
| EmpID       | char(4)     | NO   |     | NULL    |       |
| SalePrice   | float(8,2)  | NO   |     | NULL    |       |
| commission  | float(7,2)  | NO   |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
8 rows in set (0.44 sec)
```



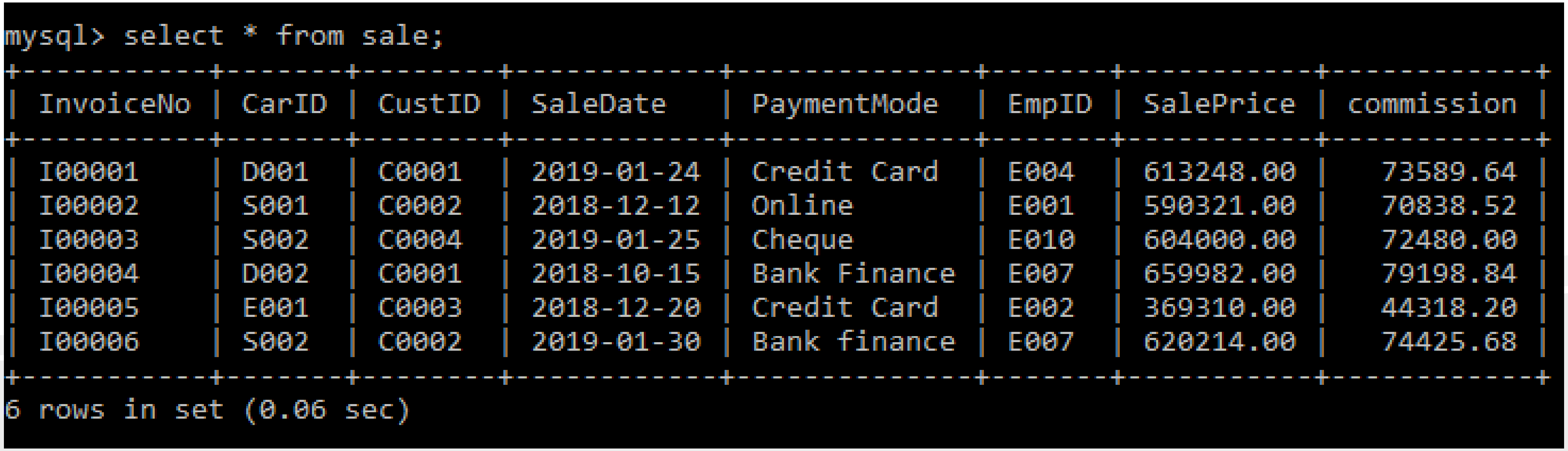
```
mysql> insert into sale values
    -> ('I00001', 'D001', 'C0001', '2019-01-24', 'Credit Card', 'E004', 613248.00, 73589.64),
    -> ('I00002', 'S001', 'C0002', '2018-12-12', 'Online', 'E001', 590321.00, 70838.52),
    -> ('I00003', 'S002', 'C0004', '2019-01-25', 'Cheque', 'E010', 604000.00, 72480.00),
    -> ('I00004', 'D002', 'C0001', '2018-10-15', 'Bank Finance', 'E007',659982.00, 79198.84),
    -> ('I00005', 'E001', 'C0003', '2018-12-20', 'Credit Card','E002', 369310.00, 44318.20),
    -> ('I00006', 'S002', 'C0002', '2019-01-30', 'Bank finance', 'E007', 620214.00, 74425.68);
Query OK, 6 rows affected (0.43 sec)
Records: 6  Duplicates: 0  Warnings: 0
```



```
mysql> select * from sale;
+-----------+-------+--------+------------+--------------+-------+-----------+------------+
| InvoiceNo | CarID | CustID | SaleDate   | PaymentMode  | EmpID | SalePrice | commission |
+-----------+-------+--------+------------+--------------+-------+-----------+------------+
| I00001    | D001  | C0001  | 2019-01-24 | Credit Card  | E004  | 613248.00 |   73589.64 |
| I00002    | S001  | C0002  | 2018-12-12 | Online       | E001  | 590321.00 |   70838.52 |
| I00003    | S002  | C0004  | 2019-01-25 | Cheque       | E010  | 604000.00 |   72480.00 |
| I00004    | D002  | C0001  | 2018-10-15 | Bank Finance | E007  | 659982.00 |   79198.84 |
| I00005    | E001  | C0003  | 2018-12-20 | Credit Card  | E002  | 369310.00 |   44318.20 |
| I00006    | S002  | C0002  | 2019-01-30 | Bank finance | E007  | 620214.00 |   74425.68 |
+-----------+-------+--------+------------+--------------+-------+-----------+------------+
6 rows in set (0.06 sec)
```

DQL: Group by and Having clause

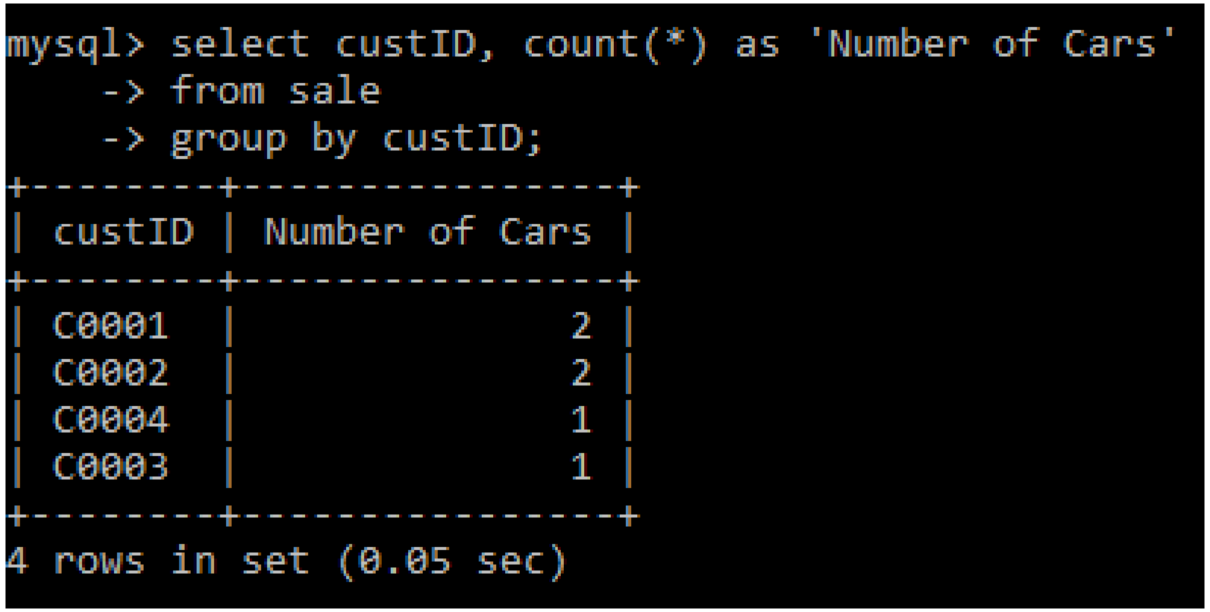
- As you can see CarID, CustID, SaleDate, PaymentMode, EmpID, are the columns that have rows with the same values in it.
- So, Group by clause can be used in these columns
  - to find the number of records of a particular type (column), or
  - to calculate the sum of the price of each car type.



```
mysql> select * from sale;
+-----------+-------+--------+------------+--------------+-------+-----------+------------+
| InvoiceNo | CarID | CustID | SaleDate   | PaymentMode  | EmpID | SalePrice | commission |
+-----------+-------+--------+------------+--------------+-------+-----------+------------+
| I00001    | D001  | C0001  | 2019-01-24 | Credit Card  | E004  | 613248.00 |   73589.64 |
| I00002    | S001  | C0002  | 2018-12-12 | Online       | E001  | 590321.00 |   70838.52 |
| I00003    | S002  | C0004  | 2019-01-25 | Cheque       | E010  | 604000.00 |   72480.00 |
| I00004    | D002  | C0001  | 2018-10-15 | Bank Finance | E007  | 659982.00 |   79198.84 |
| I00005    | E001  | C0003  | 2018-12-20 | Credit Card  | E002  | 369310.00 |   44318.20 |
| I00006    | S002  | C0002  | 2019-01-30 | Bank finance | E007  | 620214.00 |   74425.68 |
+-----------+-------+--------+------------+--------------+-------+-----------+------------+
6 rows in set (0.06 sec)
```

DQL: Group by and Having clause

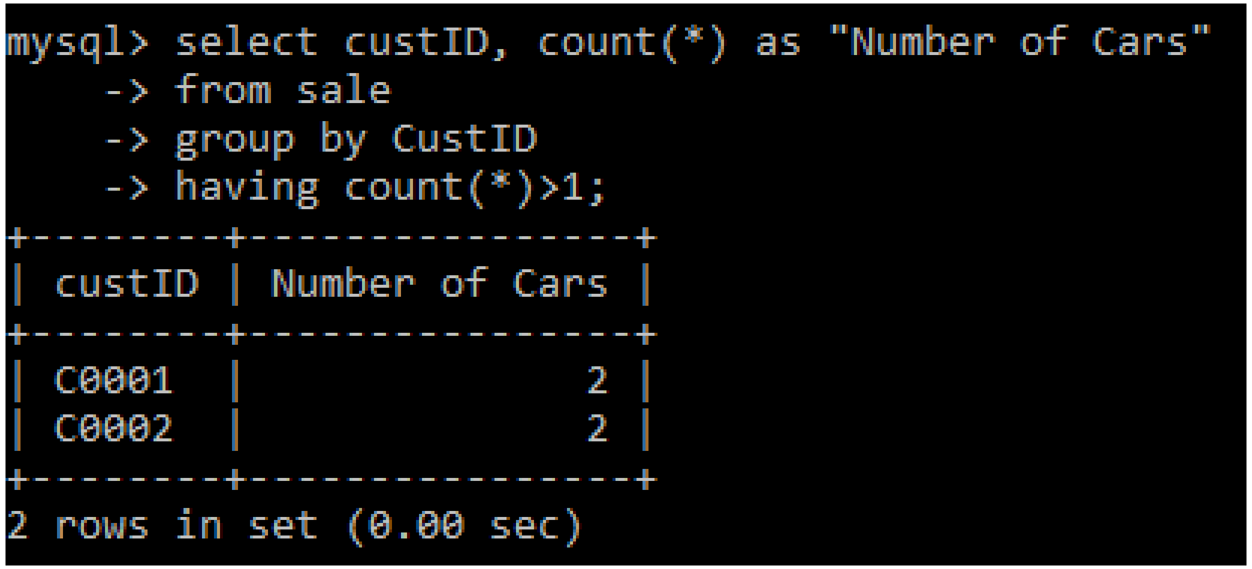
- For e.g.)
  Display the number of Cars purchased by each Customer from SALE table.



```
mysql> select custID, count(*) as 'Number of Cars'
    -> from sale
    -> group by custID;
+--------+----------------+
| custID | Number of Cars |
+--------+----------------+
| C0001  |              2 |
| C0002  |              2 |
| C0004  |              1 |
| C0003  |              1 |
+--------+----------------+
4 rows in set (0.05 sec)
```

DQL: Group by and Having clause

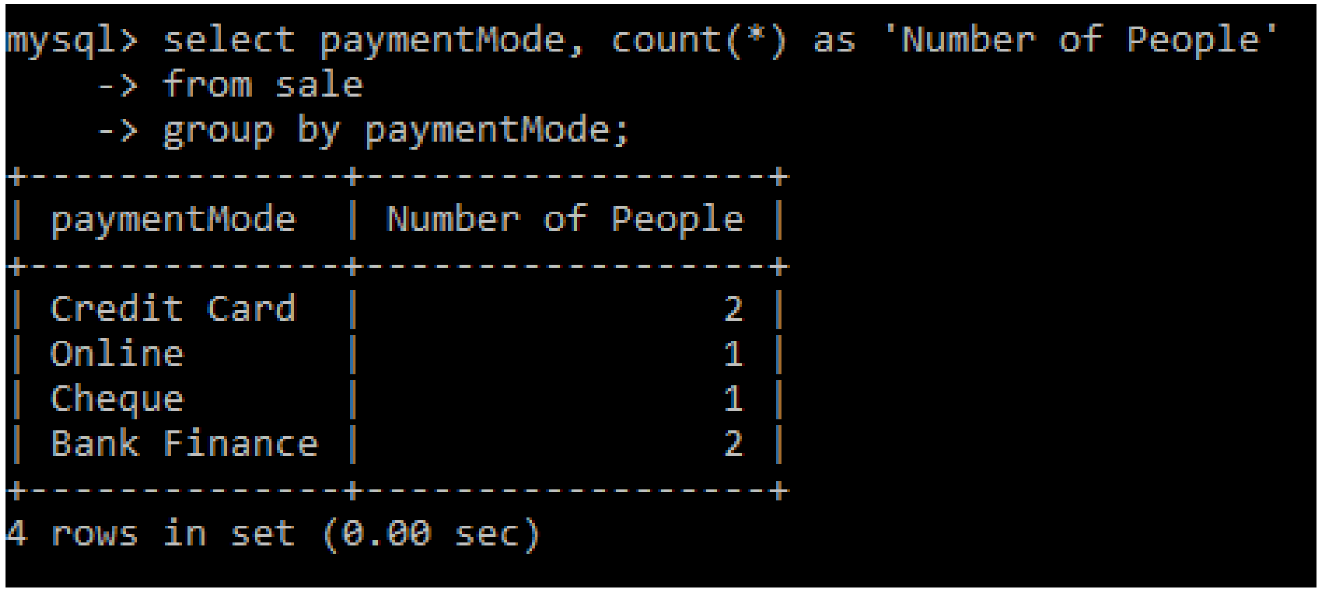
- For e.g.)
  Display the Customer Id and number of cars purchased if the customer purchased more than 1 car from SALE table.



```
mysql> select custID, count(*) as "Number of Cars"
    -> from sale
    -> group by CustID
    -> having count(*)>1;
+--------+-----------------+
| custID | Number of Cars  |
+--------+-----------------+
| C0001  |               2 |
| C0002  |               2 |
+--------+-----------------+
2 rows in set (0.00 sec)
```

DQL: Group by and Having clause

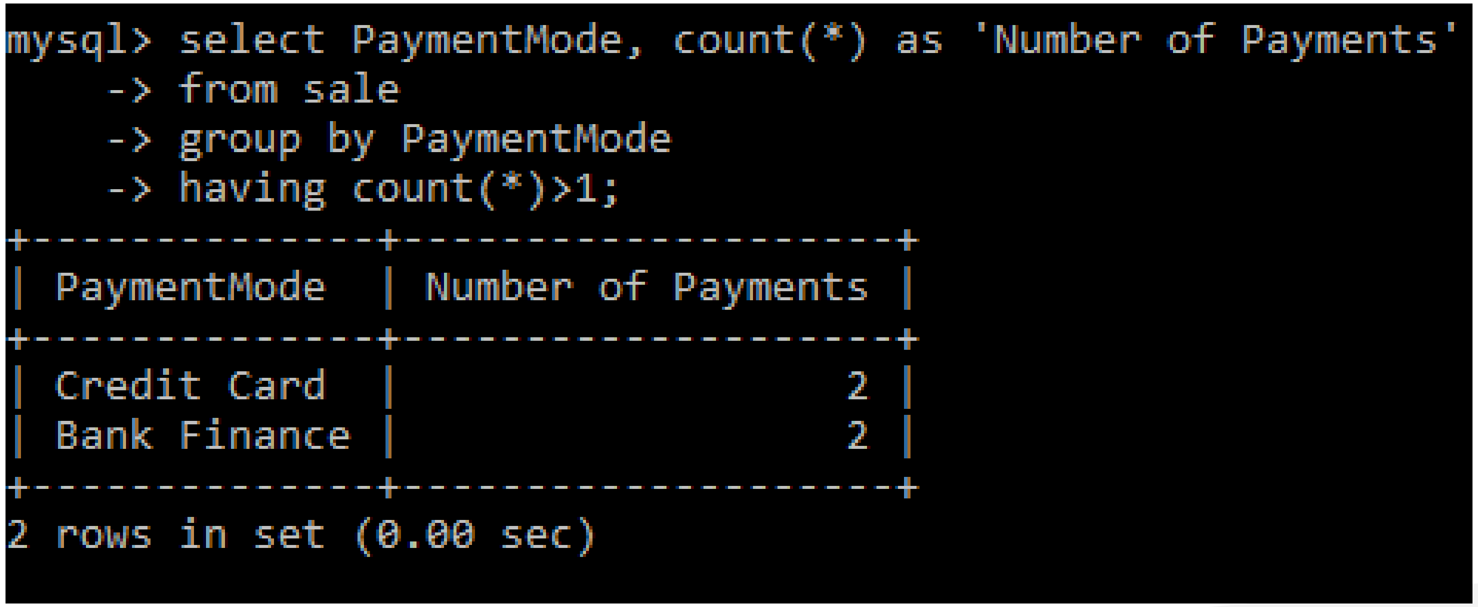
- For e.g.)
  Display the number of people in each category of payment mode from the table SALE.



```
mysql> select paymentMode, count(*) as 'Number of People'
    -> from sale
    -> group by paymentMode;
+--------------+------------------+
| paymentMode  | Number of People |
+--------------+------------------+
| Credit Card  |                2 |
| Online       |                1 |
| Cheque       |                1 |
| Bank Finance |                2 |
+--------------+------------------+
4 rows in set (0.00 sec)
```

DQL: Group by and Having clause

- For e.g.)
  Display the PaymentMode and number of payments made using that mode more than once.



```
mysql> select PaymentMode, count(*) as 'Number of Payments'
    -> from sale
    -> group by PaymentMode
    -> having count(*)>1;
+--------------+--------------------+
| PaymentMode  | Number of Payments |
+--------------+--------------------+
| Credit Card  |                  2 |
| Bank Finance |                  2 |
+--------------+--------------------+
2 rows in set (0.00 sec)
```

Using two relations in a Query

Using two relations in a Query

- Till now, we have written queries in SQL using a single relation only. Now, we will learn to write queries using two relations.
- We will learn:
    1. Cartesian Product on two tables
    2. Join on two tables
        a) Equi-join
        b) Natural join

Cartesian product on two queries

- When we apply Cartesian product on two tables, we get all pairs of rows from the two input relations.
- The degree of the resulting relation is calculated as the sum of the degrees of both the relations under consideration.

  $$\deg(R_1 \times R_2) = \deg(R_1) + \deg(R_2)$$

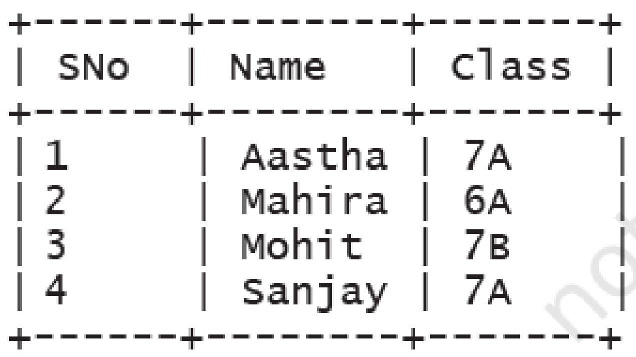
- The cardinality of the resulting relation is calculated as the product of the cardinality of relations on which cartesian product is applied.

  $$\mathrm{card}(R_1 \times R_2) = \mathrm{card}(R_1) * \mathrm{card}(R_2)$$

Cartesian product on two queries

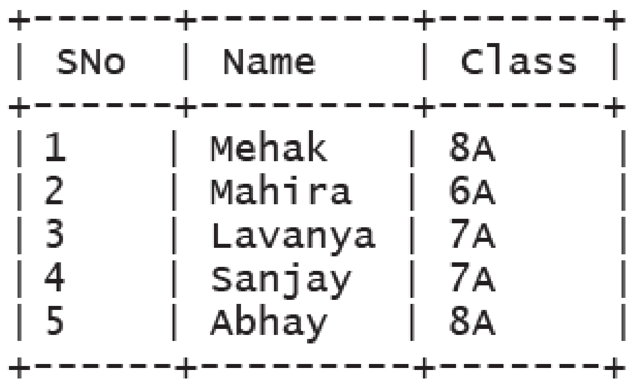
- To run queries on Cartesian product, consider two relations Dance and Music, with the below records, under Database ‘School’:

Table 9.18 DANCE

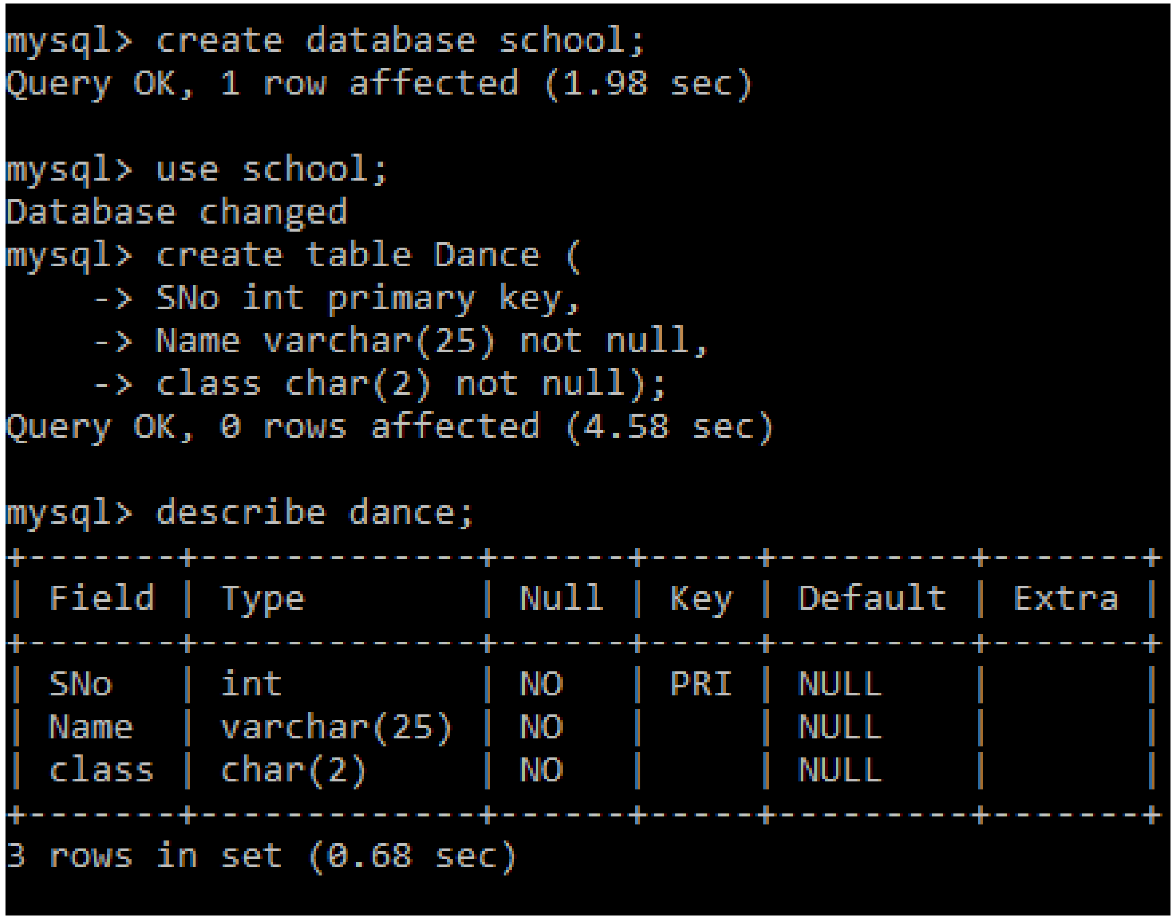


```
+------+--------+-------+
| SNo  | Name   | Class |
+------+--------+-------+
| 1    | Aastha | 7A    |
| 2    | Mahira | 6A    |
| 3    | Mohit  | 7B    |
| 4    | Sanjay | 7A    |
+------+--------+-------+
```

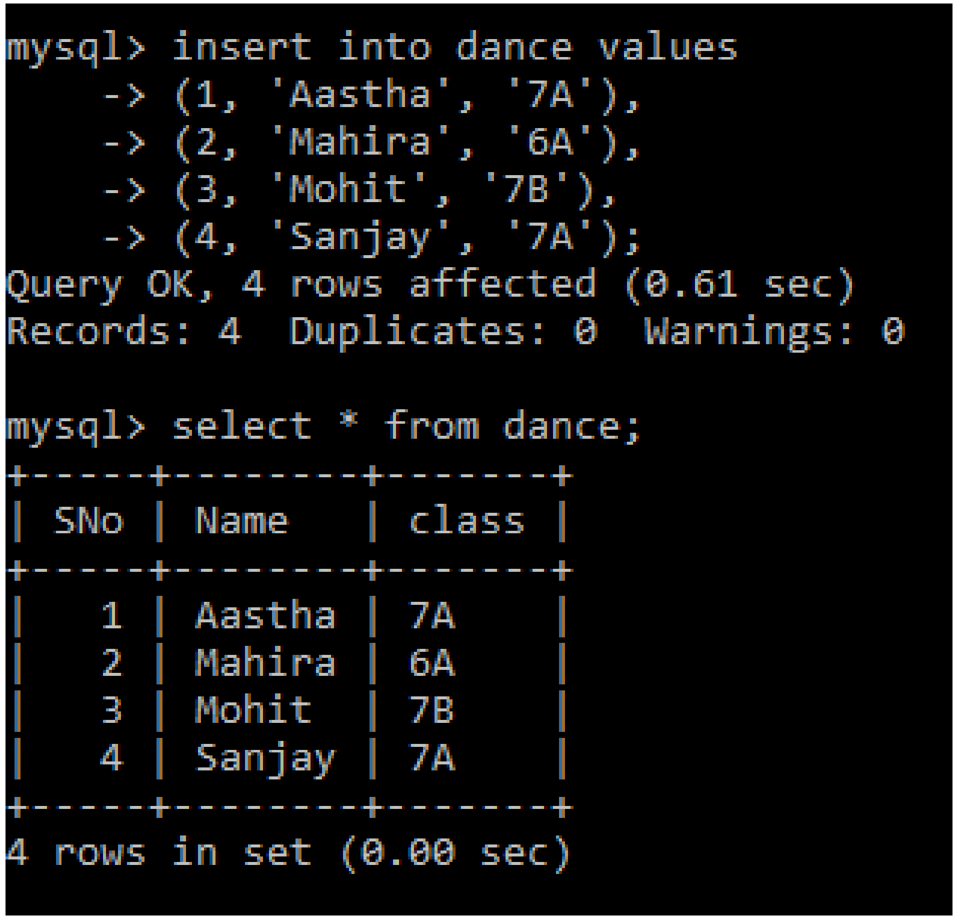
Table 9.19 MUSIC



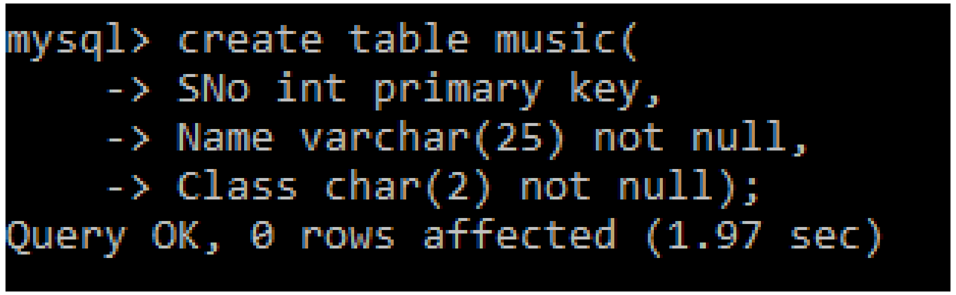
```
+------+---------+-------+
| SNo  | Name    | Class |
+------+---------+-------+
| 1    | Mehak   | 8A    |
| 2    | Mahira  | 6A    |
| 3    | Lavanya | 7A    |
| 4    | Sanjay  | 7A    |
| 5    | Abhay   | 8A    |
+------+---------+-------+
```



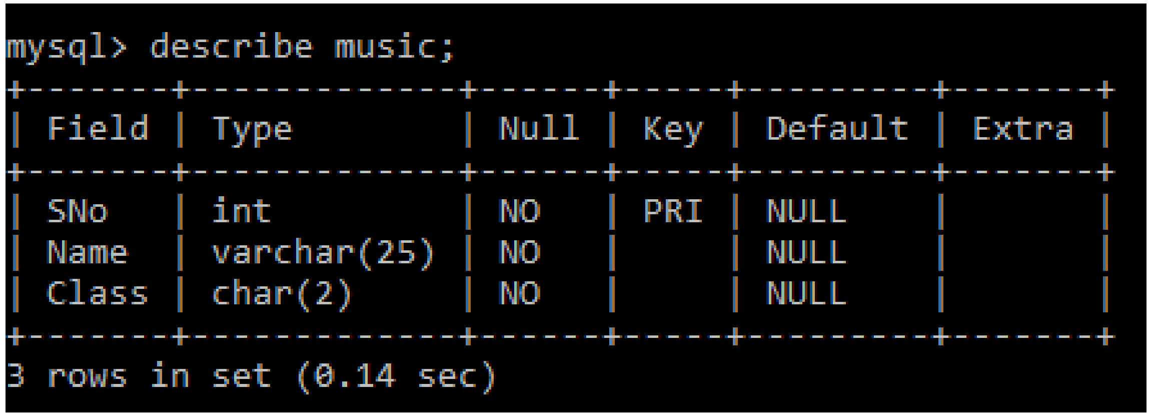
```
mysql> create database school;
Query OK, 1 row affected (1.98 sec)

mysql> use school;
Database changed
mysql> create table Dance (
    -> SNo int primary key,
    -> Name varchar(25) not null,
    -> class char(2) not null);
Query OK, 0 rows affected (4.58 sec)

mysql> describe dance;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| SNo   | int         | NO   | PRI | NULL    |       |
| Name  | varchar(25) | NO   |     | NULL    |       |
| class | char(2)     | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.68 sec)
```



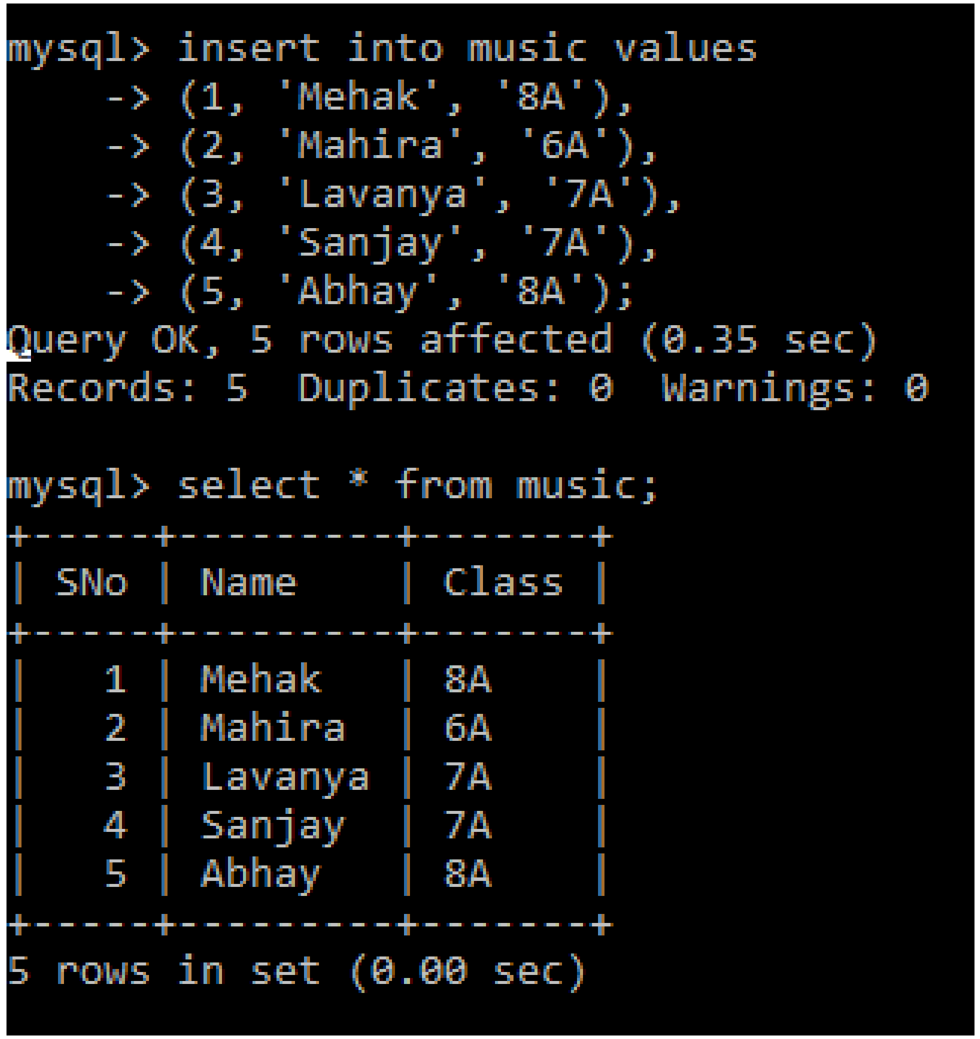
```
mysql> insert into dance values
    -> (1, 'Aastha', '7A'),
    -> (2, 'Mahira', '6A'),
    -> (3, 'Mohit', '7B'),
    -> (4, 'Sanjay', '7A');
Query OK, 4 rows affected (0.61 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> select * from dance;
+-----+--------+-------+
| SNo | Name   | class |
+-----+--------+-------+
|   1 | Aastha | 7A    |
|   2 | Mahira | 6A    |
|   3 | Mohit  | 7B    |
|   4 | Sanjay | 7A    |
+-----+--------+-------+
4 rows in set (0.00 sec)
```



```
mysql> create table music(
    -> SNo int primary key,
    -> Name varchar(25) not null,
    -> Class char(2) not null);
Query OK, 0 rows affected (1.97 sec)
```



```
mysql> describe music;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| SNo   | int         | NO   | PRI | NULL    |       |
| Name  | varchar(25) | NO   |     | NULL    |       |
| Class | char(2)     | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.14 sec)
```



```
mysql> insert into music values
    -> (1, 'Mehak', '8A'),
    -> (2, 'Mahira', '6A'),
    -> (3, 'Lavanya', '7A'),
    -> (4, 'Sanjay', '7A'),
    -> (5, 'Abhay', '8A');
Query OK, 5 rows affected (0.35 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from music;
+-----+---------+-------+
| SNo | Name    | Class |
+-----+---------+-------+
|   1 | Mehak   | 8A    |
|   2 | Mahira  | 6A    |
|   3 | Lavanya | 7A    |
|   4 | Sanjay  | 7A    |
|   5 | Abhay   | 8A    |
+-----+---------+-------+
5 rows in set (0.00 sec)
```

Cartesian product on two queries: example

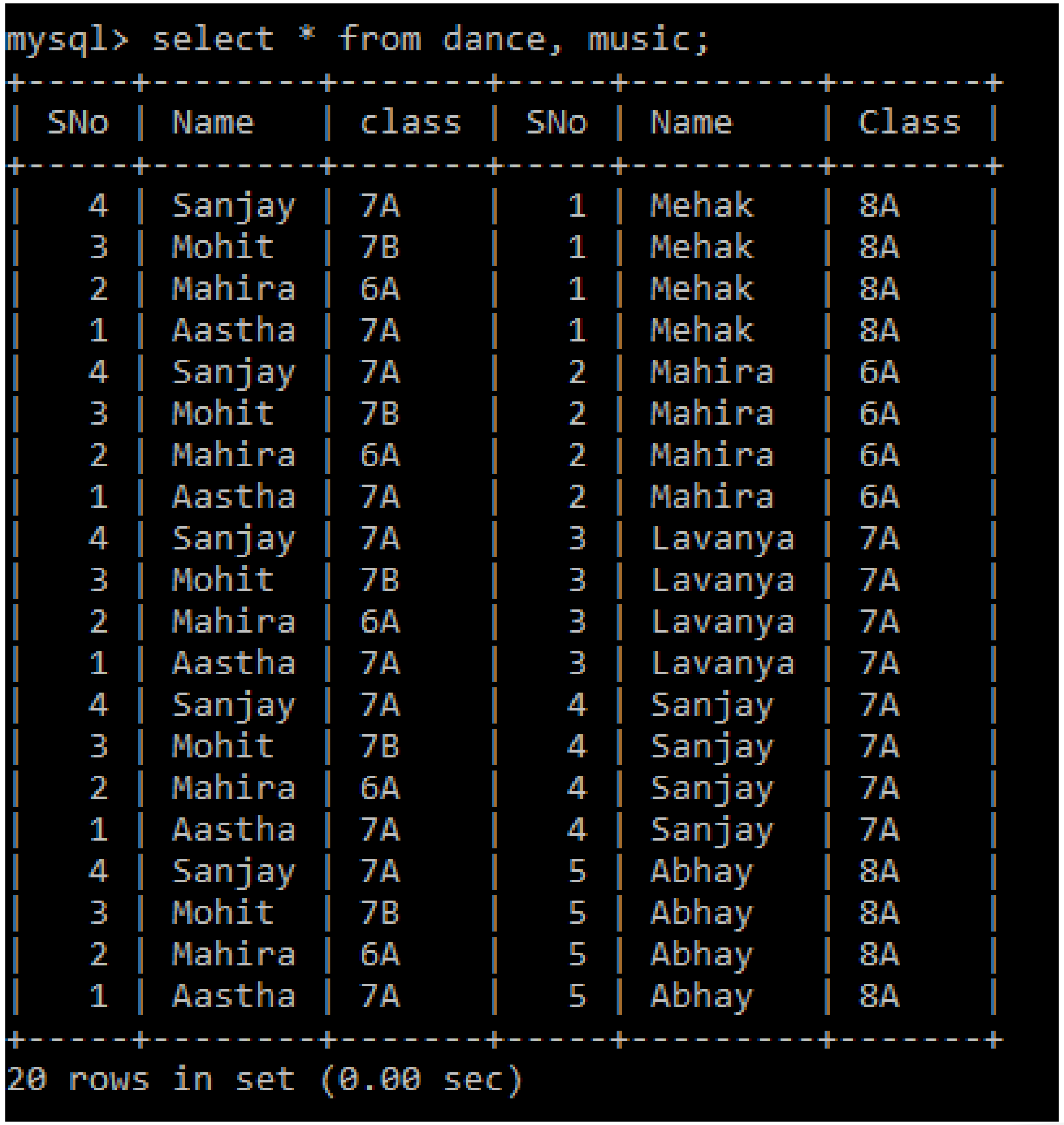
- For e.g.)
  Display all possible combinations of tuples of relations DANCE and MUSIC

```
SELECT * FROM DANCE, MUSIC;
```

Note:

- When more than one table is to be used in a query, then we must specify the table names by separating commas in the FROM clause, as shown in above query.

- On execution of such a query, the DBMS (MySql) will first apply Cartesian product on specified tables to have a single table.



```
mysql> select * from dance, music;
+-----+--------+-------+-----+---------+-------+
| SNo | Name   | class | SNo | Name    | Class |
+-----+--------+-------+-----+---------+-------+
|   4 | Sanjay | 7A    |   1 | Mehak   | 8A    |
|   3 | Mohit  | 7B    |   1 | Mehak   | 8A    |
|   2 | Mahira | 6A    |   1 | Mehak   | 8A    |
|   1 | Aastha | 7A    |   1 | Mehak   | 8A    |
|   4 | Sanjay | 7A    |   2 | Mahira  | 6A    |
|   3 | Mohit  | 7B    |   2 | Mahira  | 6A    |
|   2 | Mahira | 6A    |   2 | Mahira  | 6A    |
|   1 | Aastha | 7A    |   2 | Mahira  | 6A    |
|   4 | Sanjay | 7A    |   3 | Lavanya | 7A    |
|   3 | Mohit  | 7B    |   3 | Lavanya | 7A    |
|   2 | Mahira | 6A    |   3 | Lavanya | 7A    |
|   1 | Aastha | 7A    |   3 | Lavanya | 7A    |
|   4 | Sanjay | 7A    |   4 | Sanjay  | 7A    |
|   3 | Mohit  | 7B    |   4 | Sanjay  | 7A    |
|   2 | Mahira | 6A    |   4 | Sanjay  | 7A    |
|   1 | Aastha | 7A    |   4 | Sanjay  | 7A    |
|   4 | Sanjay | 7A    |   5 | Abhay   | 8A    |
|   3 | Mohit  | 7B    |   5 | Abhay   | 8A    |
|   2 | Mahira | 6A    |   5 | Abhay   | 8A    |
|   1 | Aastha | 7A    |   5 | Abhay   | 8A    |
+-----+--------+-------+-----+---------+-------+
20 rows in set (0.00 sec)
```

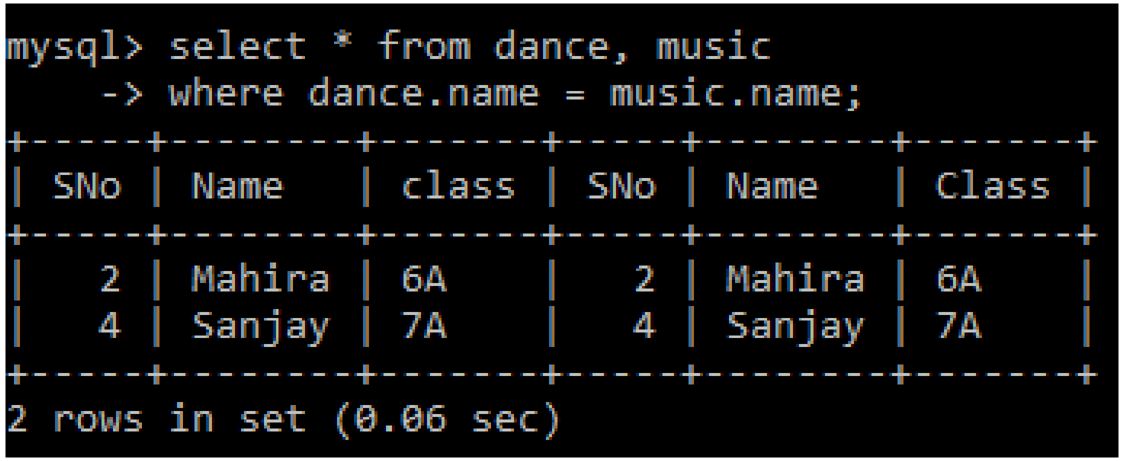
degree of the output table = 6

cardinality of the table = 20

Cartesian product on two queries: example

- For e.g.)
  From the all possible combinations of tuples of relations DANCE and MUSIC display only those rows such that the attribute name in both have the same value.

*SELECT * FROM DANCE, MUSIC*
WHERE DANCE.NAME = MUSIC.NAME;



```
mysql> select * from dance, music
    -> where dance.name = music.name;
+-----+--------+-------+-----+--------+-------+
| SNo | Name   | class | SNo | Name   | Class |
+-----+--------+-------+-----+--------+-------+
|   2 | Mahira | 6A    |   2 | Mahira | 6A    |
|   4 | Sanjay | 7A    |   4 | Sanjay | 7A    |
+-----+--------+-------+-----+--------+-------+
2 rows in set (0.06 sec)
```

Join on two tables

- A 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:
    - Equi Join
    - Natural Join

Join on two tables: Equi-join

- An Equi-join is a simple SQL join condition that uses equal to (=) as a comparison operator for defining a relationship between two tables on the basis of matching values in a specified columns.

- There are two ways to use Equi-join in SQL:

```
SELECT <column1>, column2>,….
FROM <table1>, <table2>
WHERE <table1.column> = <table2.column>;
```

```
SELECT <column1>, column2>,….
FROM <table1> JOIN <table2>
ON <table1.column> = <table2.column>;
```

Join on two tables: Equi-join

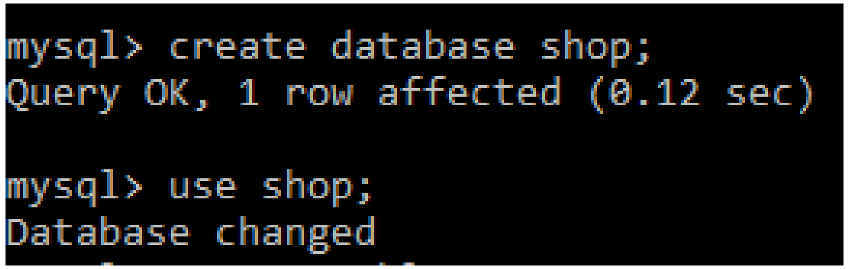
- Let's assume, we have the tables category, and product in database shop with the following data:

Category_ID	Category_name
1	Mobiles
2	Laptop
3	Cameras
4	Gaming Console
5	Earphones

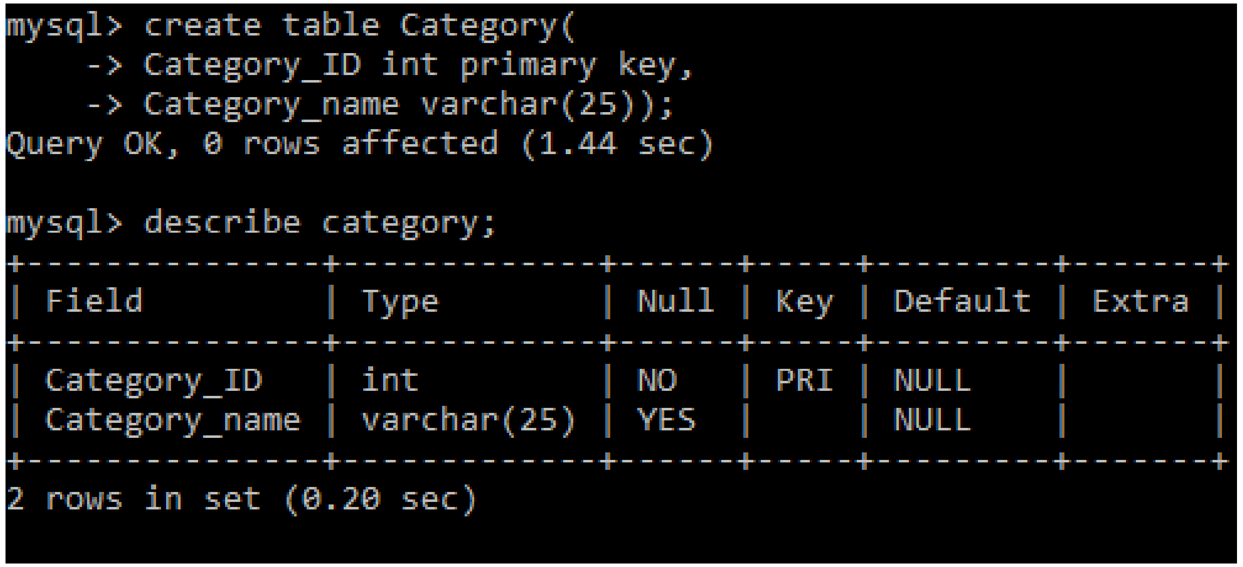
Table: Category

Category_ID	Product_name
1	Xiaomi
1	Vivo
2	Dell
2	Acer
2	HP
5	JBL

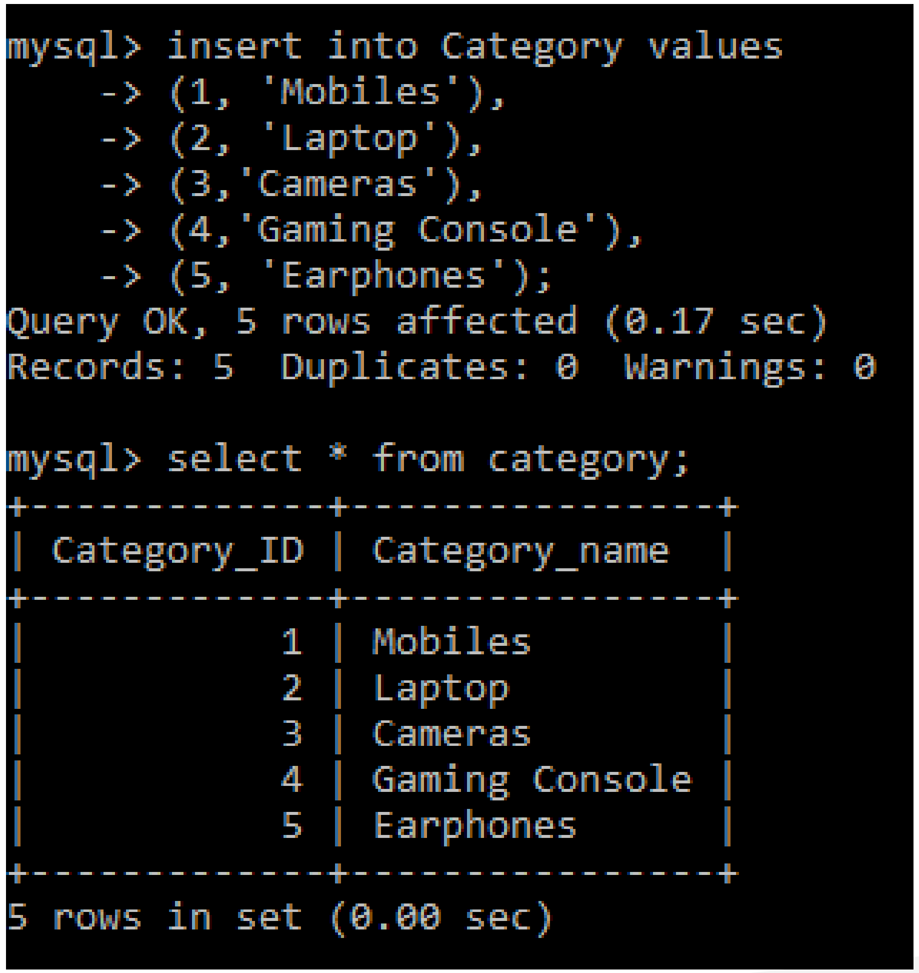
Table: Product



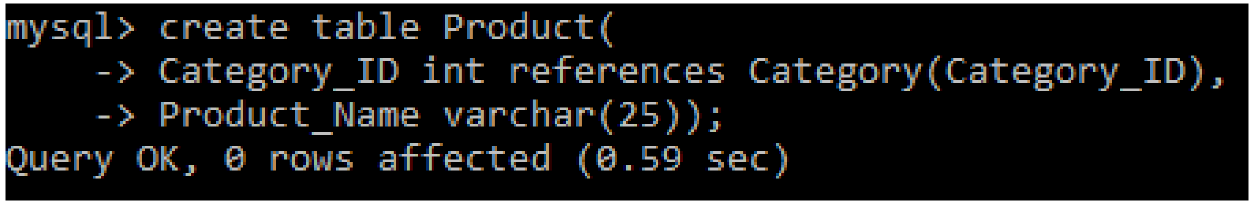
```
mysql> create database shop;
Query OK, 1 row affected (0.12 sec)

mysql> use shop;
Database changed
```



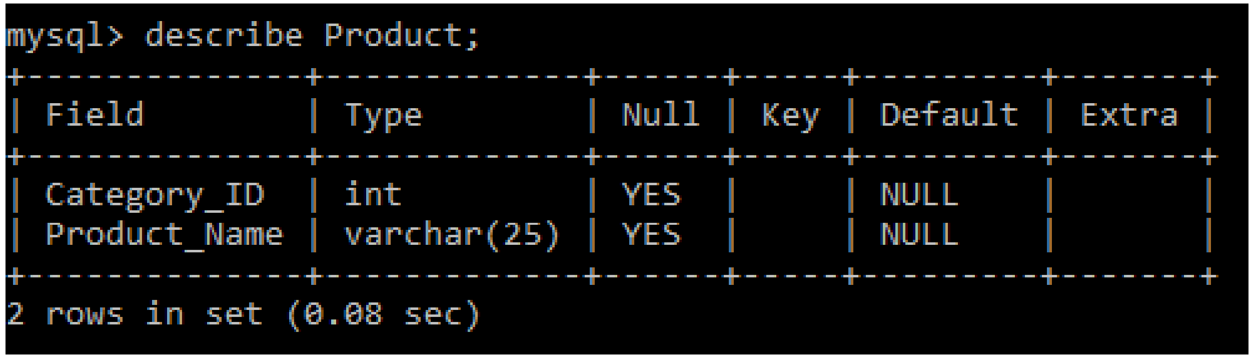
```
mysql> create table Category(
    -> Category_ID int primary key,
    -> Category_name varchar(25));
Query OK, 0 rows affected (1.44 sec)

mysql> describe category;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| Category_ID   | int         | NO   | PRI | NULL    |       |
| Category_name | varchar(25) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
2 rows in set (0.20 sec)
```



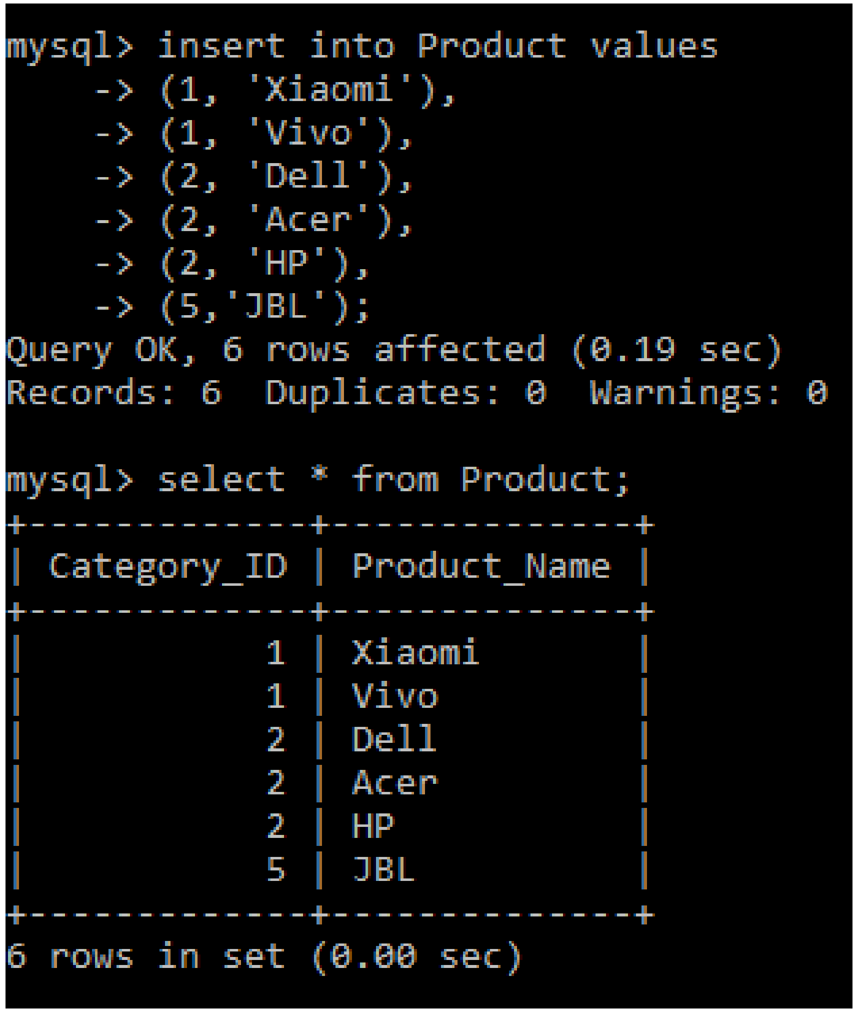
```
mysql> insert into Category values
    -> (1, 'Mobiles'),
    -> (2, 'Laptop'),
    -> (3,'Cameras'),
    -> (4,'Gaming Console'),
    -> (5, 'Earphones');
Query OK, 5 rows affected (0.17 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from category;
+-------------+----------------+
| Category_ID | Category_name  |
+-------------+----------------+
|           1 | Mobiles        |
|           2 | Laptop         |
|           3 | Cameras        |
|           4 | Gaming Console |
|           5 | Earphones      |
+-------------+----------------+
5 rows in set (0.00 sec)
```



```
mysql> create table Product(
    -> Category_ID int references Category(Category_ID),
    -> Product_Name varchar(25));
Query OK, 0 rows affected (0.59 sec)
```



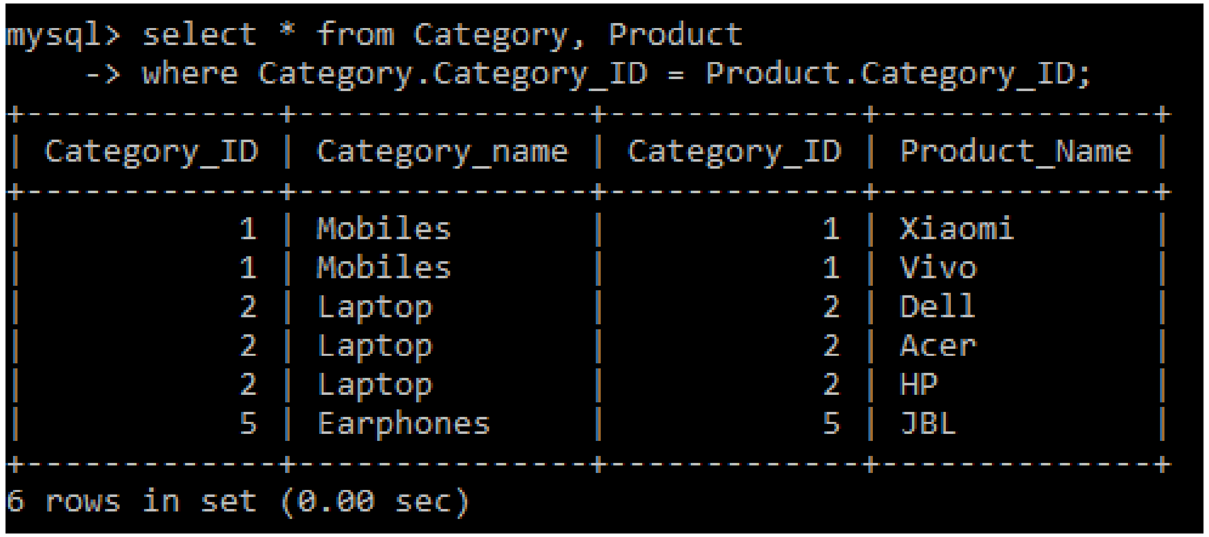
```
mysql> describe Product;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| Category_ID  | int         | YES  |     | NULL    |       |
| Product_Name | varchar(25) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
2 rows in set (0.08 sec)
```



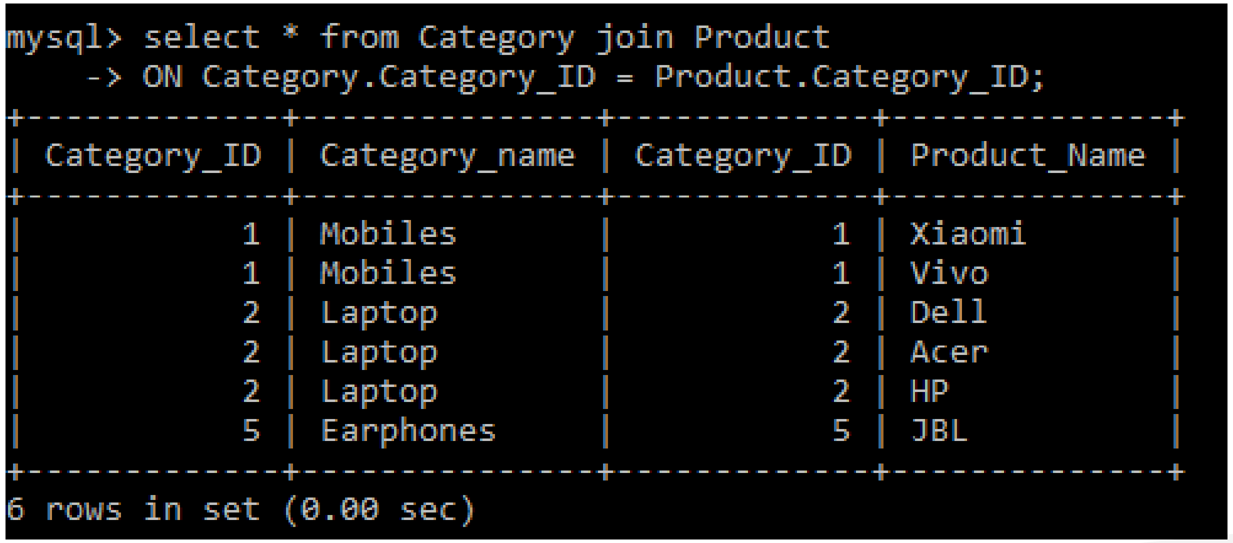
```
mysql> insert into Product values
    -> (1, 'Xiaomi'),
    -> (1, 'Vivo'),
    -> (2, 'Dell'),
    -> (2, 'Acer'),
    -> (2, 'HP'),
    -> (5,'JBL');
Query OK, 6 rows affected (0.19 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> select * from Product;
+-------------+--------------+
| Category_ID | Product_Name |
+-------------+--------------+
|           1 | Xiaomi       |
|           1 | Vivo         |
|           2 | Dell         |
|           2 | Acer         |
|           2 | HP           |
|           5 | JBL          |
+-------------+--------------+
6 rows in set (0.00 sec)
```

Join on two tables: Equi-join

- For e.g.) select all the details from the Category and Product using Equi-join on attribute Category_ID



```
mysql> select * from Category, Product
    -> where Category.Category_ID = Product.Category_ID;
+-------------+---------------+-------------+--------------+
| Category_ID | Category_name | Category_ID | Product_Name |
+-------------+---------------+-------------+--------------+
|           1 | Mobiles       |           1 | Xiaomi       |
|           1 | Mobiles       |           1 | Vivo         |
|           2 | Laptop        |           2 | Dell         |
|           2 | Laptop        |           2 | Acer         |
|           2 | Laptop        |           2 | HP           |
|           5 | Earphones     |           5 | JBL          |
+-------------+---------------+-------------+--------------+
6 rows in set (0.00 sec)
```



```
mysql> select * from Category join Product
    -> ON Category.Category_ID = Product.Category_ID;
+-------------+---------------+-------------+--------------+
| Category_ID | Category_name | Category_ID | Product_Name |
+-------------+---------------+-------------+--------------+
|           1 | Mobiles       |           1 | Xiaomi       |
|           1 | Mobiles       |           1 | Vivo         |
|           2 | Laptop        |           2 | Dell         |
|           2 | Laptop        |           2 | Acer         |
|           2 | Laptop        |           2 | HP           |
|           5 | Earphones     |           5 | JBL          |
+-------------+---------------+-------------+--------------+
6 rows in set (0.00 sec)
```

Join on two tables: Natural Join

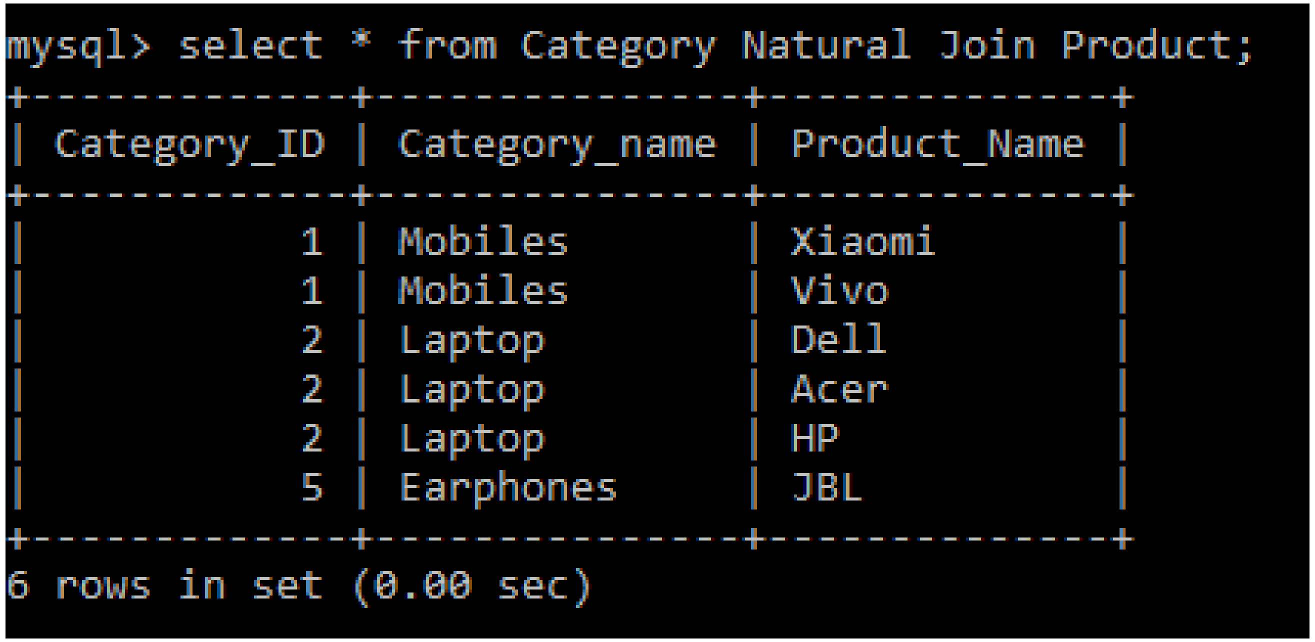
- If you observed the result of previous query, we have a repetitive column Category_ID which has the same exact values. This redundant column provides no extra information.

- In order to remove that we can use Natural Join, which is an extension of Join operation which works similar to join clause but removes the redundant attribute, and its syntax is:

```
SELECT <column1>, column2>,....
FROM <table1> NATURAL JOIN <table2>;
```

Join on two tables: Natural Join

- For e.g.) select all the details from the Category and Product using Natural Join



```
mysql> select * from Category Natural Join Product;
+-------------+---------------+--------------+
| Category_ID | Category_name | Product_Name |
+-------------+---------------+--------------+
|           1 | Mobiles       | Xiaomi       |
|           1 | Mobiles       | Vivo         |
|           2 | Laptop        | Dell         |
|           2 | Laptop        | Acer         |
|           2 | Laptop        | HP           |
|           5 | Earphones     | JBL          |
+-------------+---------------+--------------+
6 rows in set (0.00 sec)
```

Summary

- DDL (Data Definition Language): Create, drop, alter
- DML(Data Manipulation Language): insert into, update, delete
- DQL (Data Query Language):

```
select <coulmn1>, <column2>, ...
from <table_name>
where condition
order by <column_name>
group by <column_name>
having condition;
```