Lab Manual

List of Practicals

1. SQL Statements - 1

(a) Writing basic SQL SELECT Statement.

SELECT Query

SELECT query is used to retrieve the data from database. SELECT query never make any change in the database.

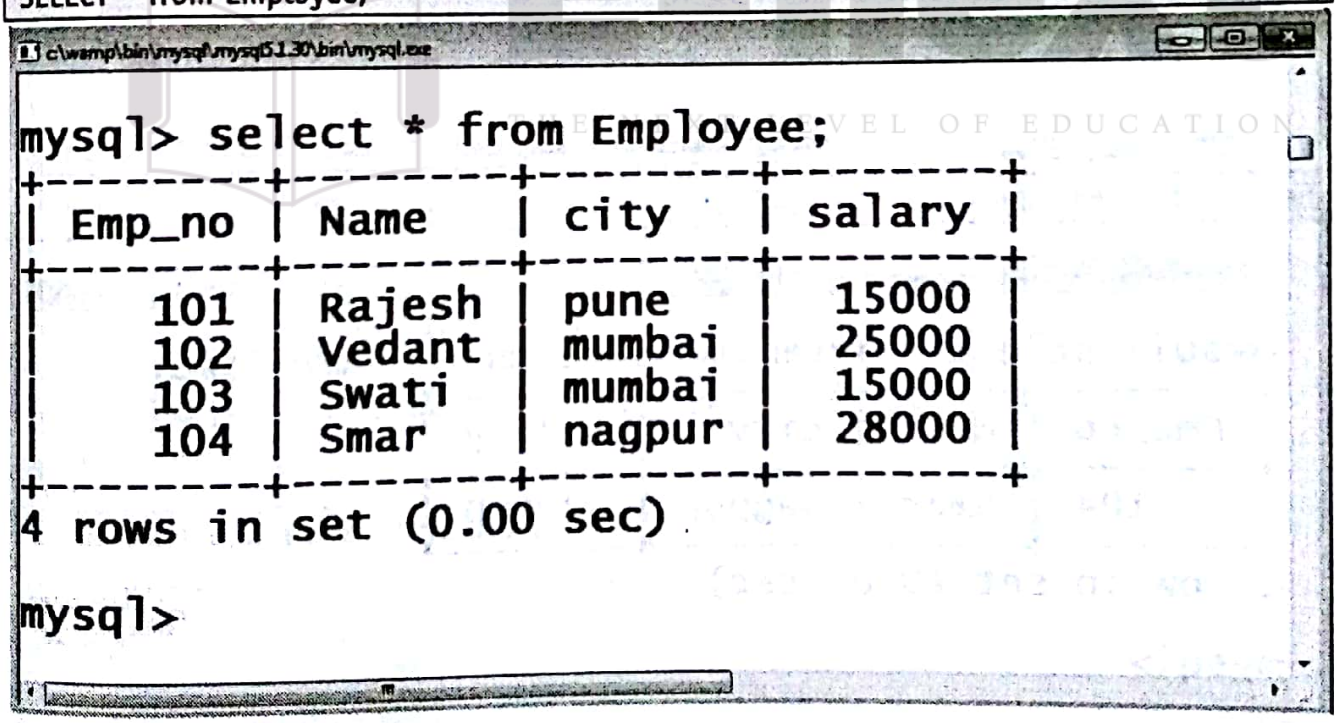
Syntax

```
SELECT column_name1, column_name1... from table_name;
```

If we want to retrieve data from all the columns of a table then instead of writing all the column name just use '*'. The '* 'symbol represent all the columns.

For Example

```
SELECT * from Employee;
```



With SELECT statement different clauses can be used to display the data as per our requirements.

WHERE clause

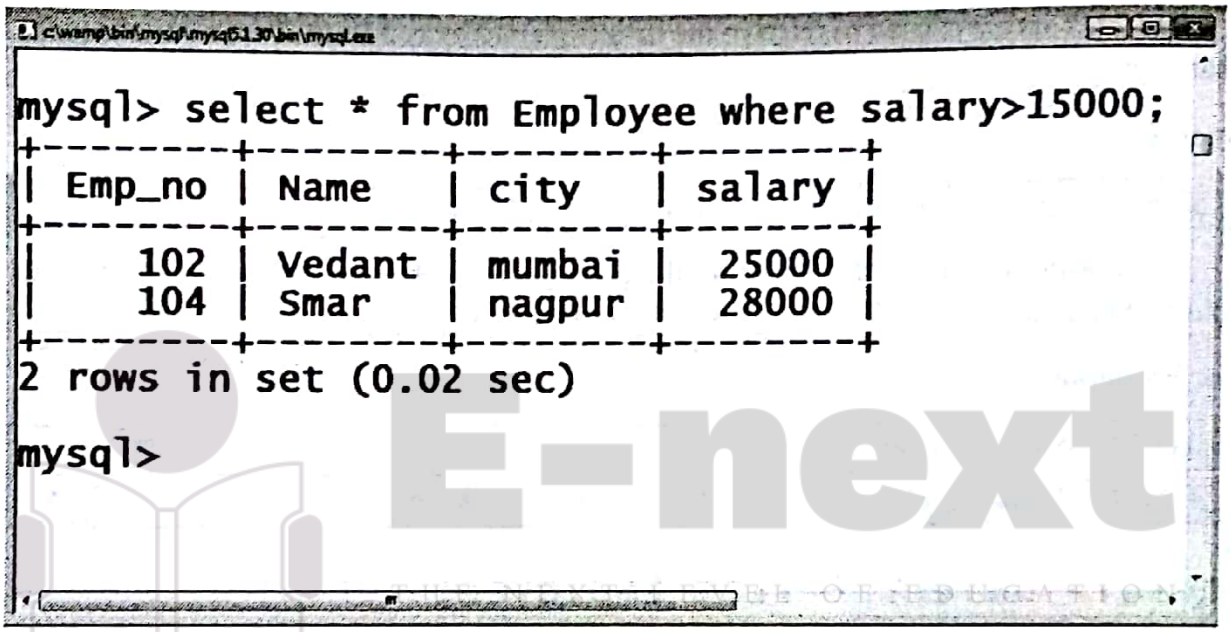
WHERE clause is used to specify condition in SELECT statement while fetching records from the database. The records satisfying the condition given by where clause are retrieved.

Syntax

```
SELECT column_1,columns_2... from table_name where condition;
```

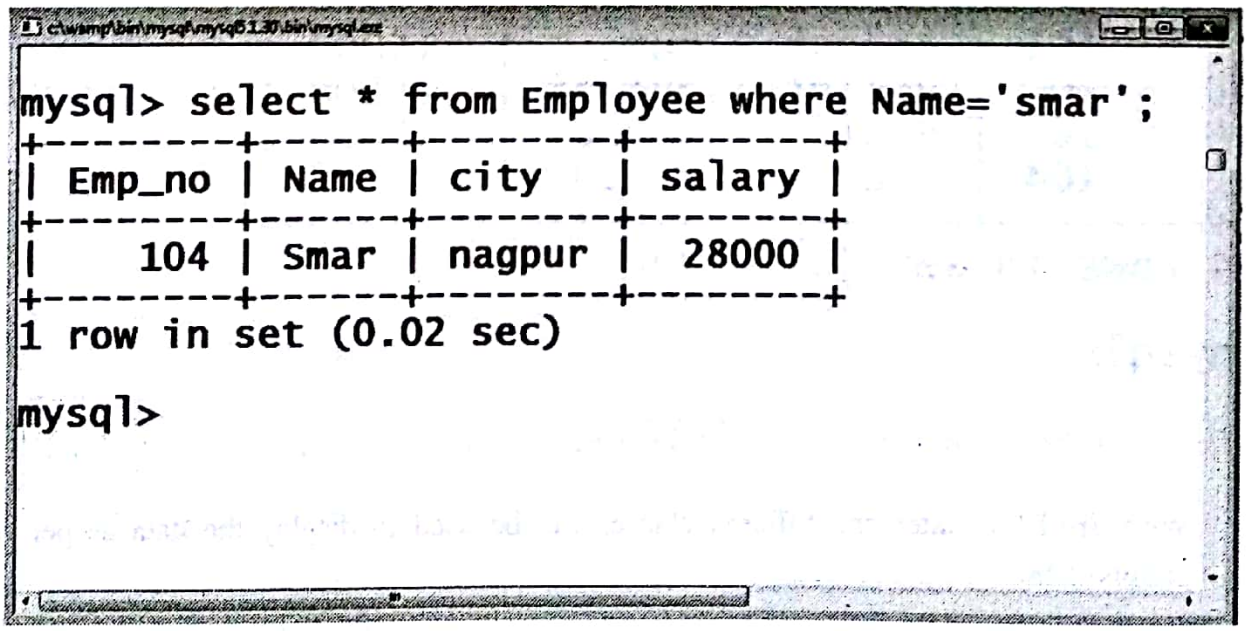
For Example

```
select * from Employee where Salary > 15000;
```



```
mysql> select * from Employee where salary>15000;
+--------+--------+--------+--------+
| Emp_no | Name   | city   | salary |
+--------+--------+--------+--------+
|    102 | Vedant | mumbai |  25000 |
|    104 | Smar   | nagpur |  28000 |
+--------+--------+--------+--------+
2 rows in set (0.02 sec)

mysql>
```

```
Select * from Employee where Name='Smar' ;
```



```
mysql> select * from Employee where Name='smar';
+--------+------+--------+--------+
| Emp_no | Name | city   | salary |
+--------+------+--------+--------+
|    104 | Smar | nagpur |  28000 |
+--------+------+--------+--------+
1 row in set (0.02 sec)

mysql>
```

(b) Restricting and Sorting Data.

DISTINCT clause

This clause is used to avoid selection of duplicate rows.

Consider a situation where duplicate values for Salary column are present in Employee table, and the manager wants to know the salary amount given to the employees. To avoid duplication of salary amount we use distinct keyword as follows:

```
Select distinct(salary) from Employee;
```

```
mysql> select * from Employee;
+--------+--------+--------+--------+
| Emp_no | Name   | city   | salary |
+--------+--------+--------+--------+
|    101 | Rajesh | pune   |  15000 |
|    102 | Vedant | mumbai |  25000 |
|    103 | Swati  | mumbai |  15000 |
|    104 | Smar   | nagpur |  28000 |
+--------+--------+--------+--------+
4 rows in set (0.00 sec)

mysql> select distinct(salary) from Employee;
+--------+
| salary |
+--------+
|  15000 |
|  25000 |
|  28000 |
+--------+
3 rows in set (0.03 sec)
```

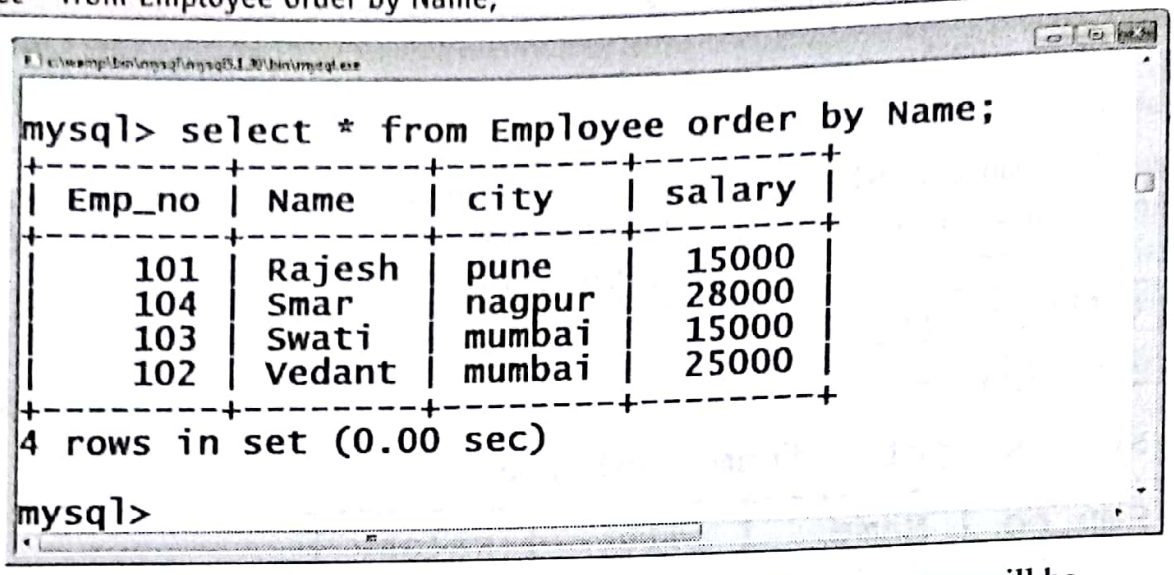
ORDER BY clause

To arrange the displayed rows in ascending or descending order of given column, Order By Clause is used.

For Example

We need to display the employee information as per their joining dates in ascending order, the query will be

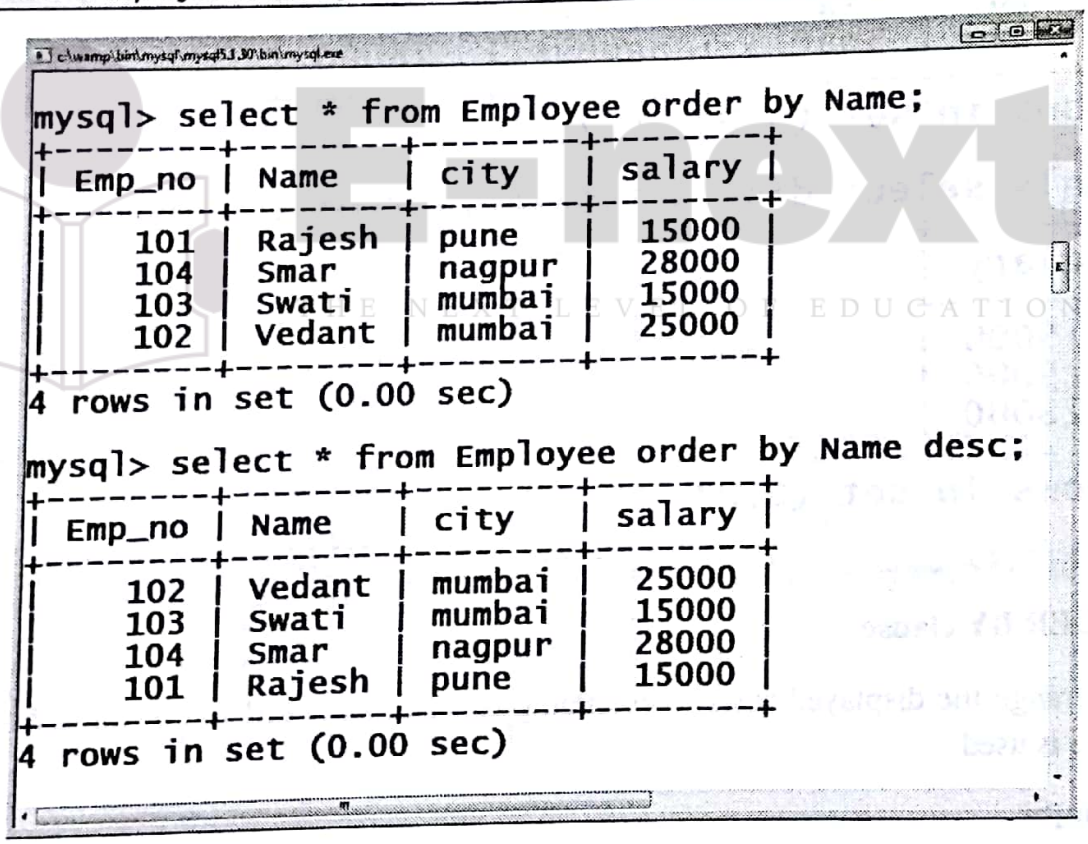
```
Select * from Employee order by Name;
```



```
mysql> select * from Employee order by Name;
+--------+--------+--------+--------+
| Emp_no | Name   | city   | salary |
+--------+--------+--------+--------+
|    101 | Rajesh | pune   |  15000 |
|    104 | Smar   | nagpur |  28000 |
|    103 | Swati  | mumbai |  15000 |
|    102 | Vedant | mumbai |  25000 |
+--------+--------+--------+--------+
4 rows in set (0.00 sec)

mysql>
```

Now to display same information in descending order on job the query will be

```
Select * from Employee order by Name desc;
```



```
mysql> select * from Employee order by Name;
+--------+--------+--------+--------+
| Emp_no | Name   | city   | salary |
+--------+--------+--------+--------+
|    101 | Rajesh | pune   |  15000 |
|    104 | Smar   | nagpur |  28000 |
|    103 | Swati  | mumbai |  15000 |
|    102 | Vedant | mumbai |  25000 |
+--------+--------+--------+--------+
4 rows in set (0.00 sec)

mysql> select * from Employee order by Name desc;
+--------+--------+--------+--------+
| Emp_no | Name   | city   | salary |
+--------+--------+--------+--------+
|    102 | Vedant | mumbai |  25000 |
|    103 | Swati  | mumbai |  15000 |
|    104 | Smar   | nagpur |  28000 |
|    101 | Rajesh | pune   |  15000 |
+--------+--------+--------+--------+
4 rows in set (0.00 sec)
```

(c) Single – Row Functions

Single row functions can be

- **Case Conversion functions :** Accepts character input and returns a character value. Functions under the category are UPPER, LOWER and INITCAP.
- **UPPER :** UPPER function converts a string to upper case.

- **LOWER :** LOWER function converts a string to lower case.
- **INITCAP :** INITCAP function converts only the initial alphabets of a string to upper case.

For example

```
select Emp_no, UPPER(Name),LOWER(city) from Employee;
```

```
mysql> select Emp_no, UPPER(Name),LOWER(city) from Employee;
+--------+-------------+-------------+
| Emp_no | UPPER(Name) | LOWER(city) |
+--------+-------------+-------------+
|    101 | RAJESH      | pune        |
|    102 | VEDANT      | mumbai      |
|    103 | SWATI       | mumbai      |
|    104 | SMAR        | nagpur      |
+--------+-------------+-------------+
4 rows in set (0.00 sec)

mysql>
```

- **Character functions :** Accepts character input and returns number or character value. Character functions are as follows:
- **CONCAT :** CONCAT function concatenates two string values.

For example

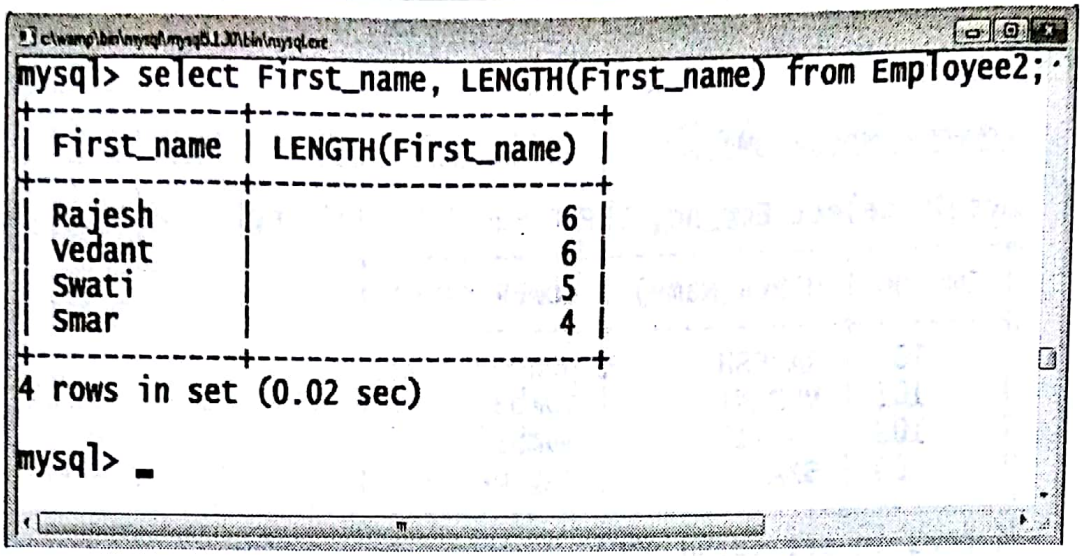
```
select CONCAT(First_name,Last_name) from Employee2;
```

```
mysql> select * from Employee2;
+--------+------------+-----------+-------+--------+
| Emp_no | First_name | Last_name | city  | salary |
+--------+------------+-----------+-------+--------+
|    101 | Rajesh     | powar     | pune  |  15000 |
|    102 | Vedant     | jadhav    | mumbai|  25000 |
|    103 | Swati      | patil     | mumbai|  15000 |
|    104 | Smar       | sawant    | nagpur|  28000 |
+--------+------------+-----------+-------+--------+
4 rows in set (0.00 sec)

mysql> select CONCAT(First_name,Last_name) from Employee2;
+-----------------------------+
| CONCAT(First_name,Last_name) |
+-----------------------------+
| Rajeshpowar                 |
| Vedantjadhav                |
| Swatipatil                  |
| Smarsawant                  |
+-----------------------------+
```

- **LENGTH :** LENGTH function returns the length of the input string

For example

```
Select First_name, LENGTH(First_name) from Employee2;
```

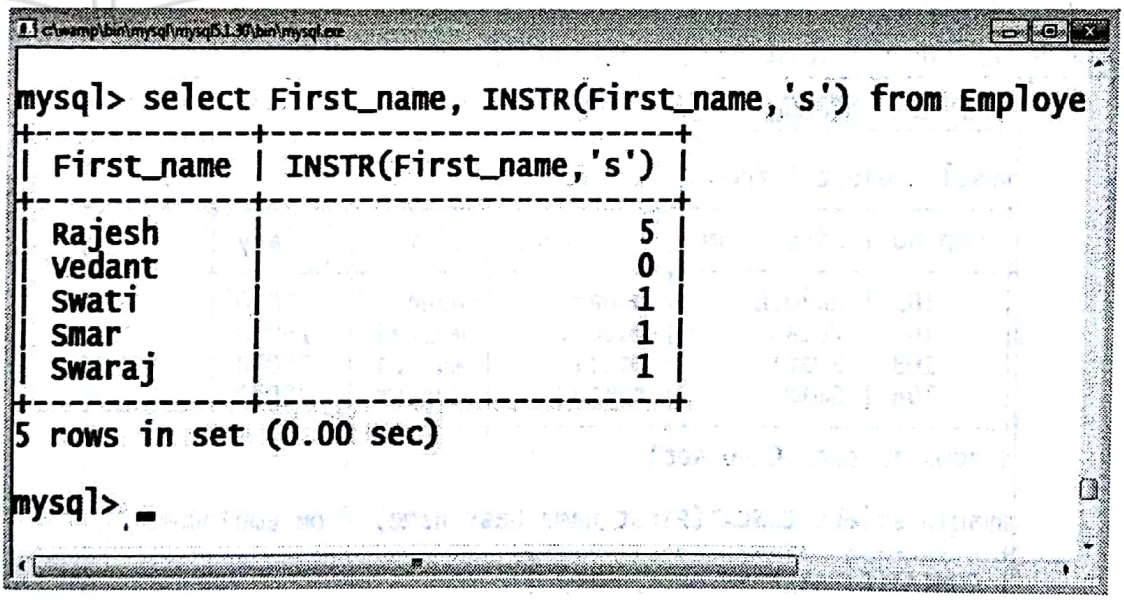


```
mysql> select First_name, LENGTH(First_name) from Employee2;
+------------+--------------------+
| First_name | LENGTH(First_name) |
+------------+--------------------+
| Rajesh     |                  6 |
| Vedant     |                  6 |
| Swati      |                  5 |
| Smar       |                  4 |
+------------+--------------------+
4 rows in set (0.02 sec)

mysql>
```

- **SUBSTR :** SUBSTR function returns a portion of a string from a given start point to an end point.
- **INSTR :** INSTR function returns numeric position of a character or a string in a given string.

For example

```
select First_name, INSTER(First_name, 's') from Employee2;
```

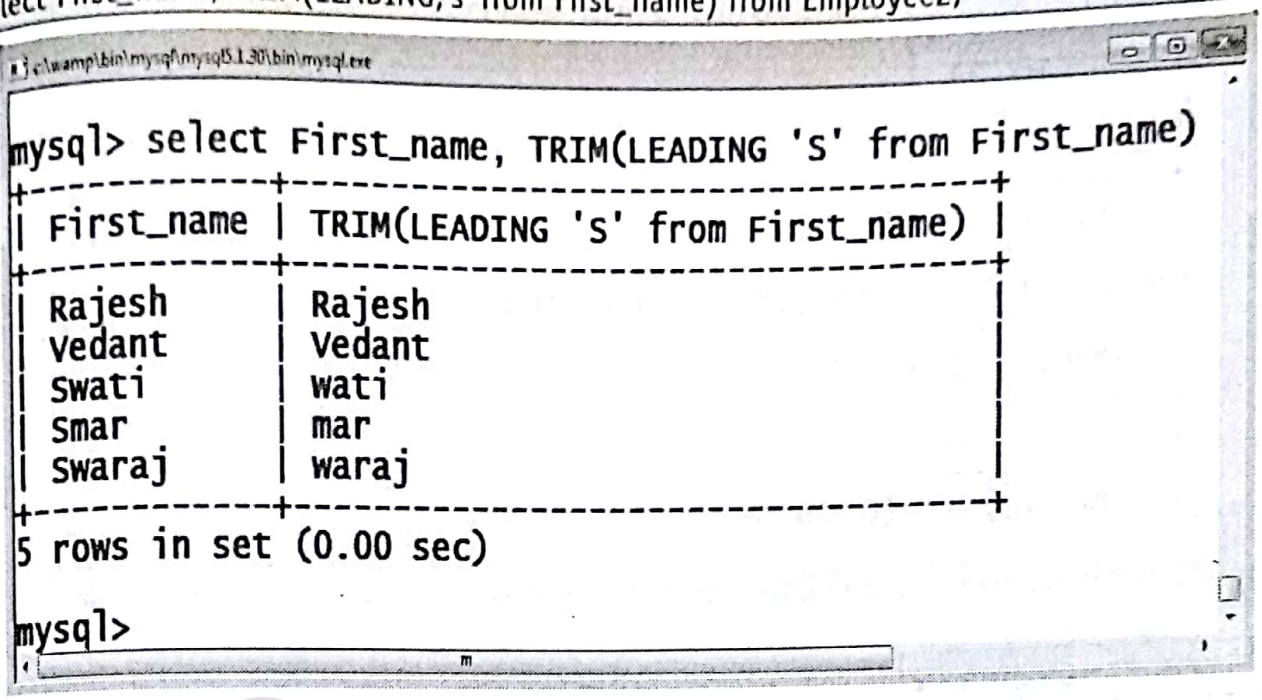


```
mysql> select First_name, INSTR(First_name,'s') from Employe
+------------+----------------------+
| First_name | INSTR(First_name,'s') |
+------------+----------------------+
| Rajesh     |                    5 |
| Vedant     |                    0 |
| Swati      |                    1 |
| Smar       |                    1 |
| Swaraj     |                    1 |
+------------+----------------------+
5 rows in set (0.00 sec)

mysql>
```

- LPAD and RPAD functions pad the given string upto a specific length with a given character.
- **TRIM :** TRIM function trims the string input from the start or end.

For example

```
select First_name, TRIM(LEADING,'s' from First_name) from Employee2;
```

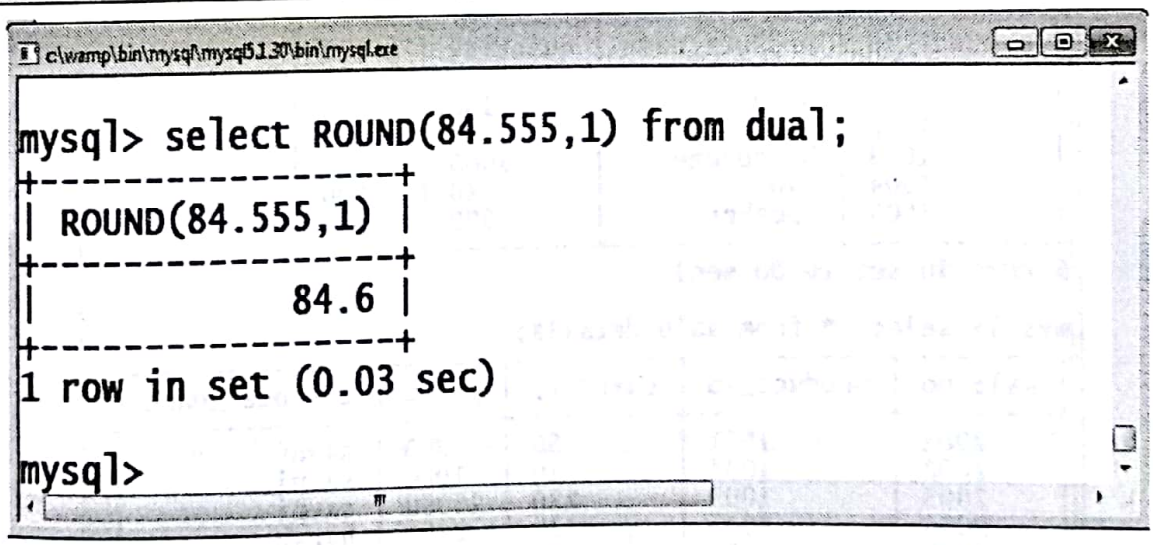


```
mysql> select First_name, TRIM(LEADING 's' from First_name)
+------------+-----------------------------------+
| First_name | TRIM(LEADING 's' from First_name) |
+------------+-----------------------------------+
| Rajesh     | Rajesh                            |
| Vedant     | Vedant                            |
| Swati      | wati                              |
| Smar       | mar                               |
| Swaraj     | waraj                             |
+------------+-----------------------------------+
5 rows in set (0.00 sec)

mysql>
```

- **REPLACE :** REPLACE function replaces characters from the input string with a given character.
- **Number functions :** Accepts numeric input and returns numeric values. Functions under the category are ROUND, TRUNC, and MOD.
- ROUND and TRUNC functions are used to round and truncate the number value.

For example

```
SELECT ROUND(84.555,1) from dual;
```



```
mysql> select ROUND(84.555,1) from dual;
+-----------------+
| ROUND(84.555,1) |
+-----------------+
|            84.6 |
+-----------------+
1 row in set (0.03 sec)

mysql>
```

- MOD is used to return the remainder of the division operation between two numbers.

For example

```
SELECT MOD(10,2) from dual;
```

```
mysql> SELECT MOD(10,2) from dual;
+-----------+
| MOD(10,2) |
+-----------+
|         0 |
+-----------+
1 row in set (0.00 sec)

mysql> SELECT MOD(25,2) from dual;
+-----------+
| MOD(25,2) |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)
```

2. SQL Statements - 2

(a) Displaying Data from Multiple Tables.

To understand joins consider following two tables.

For example

```
select * from product_details;
```

```
mysql> select * from product_details;
+------------+--------------+----------+-------+
| product_id | product_name | quantity | price |
+------------+--------------+----------+-------+
|       1001 | pendrive     |      100 |   900 |
|       1002 | harddisk     |      200 |  4000 |
|       1003 | headphone    |     1000 | 15000 |
|       1004 | DVD          |       20 |  1000 |
|       1005 | speaker      |      600 |  2400 |
+------------+--------------+----------+-------+
5 rows in set (0.00 sec)

mysql> select * from sale_details;
+---------+------------+----------+-------+---------------+
| sale_no | product_id | quantity | price | customer_name |
+---------+------------+----------+-------+---------------+
|    2001 |       1001 |       50 |   900 | savni         |
|    2002 |       1004 |       10 |  1000 | savni         |
|    2003 |       1003 |      120 | 15000 | savni         |
|    2004 |       1005 |      420 |  2400 | harsh         |
|    2005 |       1002 |       40 |  4000 | Akash         |
+---------+------------+----------+-------+---------------+
5 rows in set (0.02 sec)

mysql>
```

Inner Join (Equi Join)

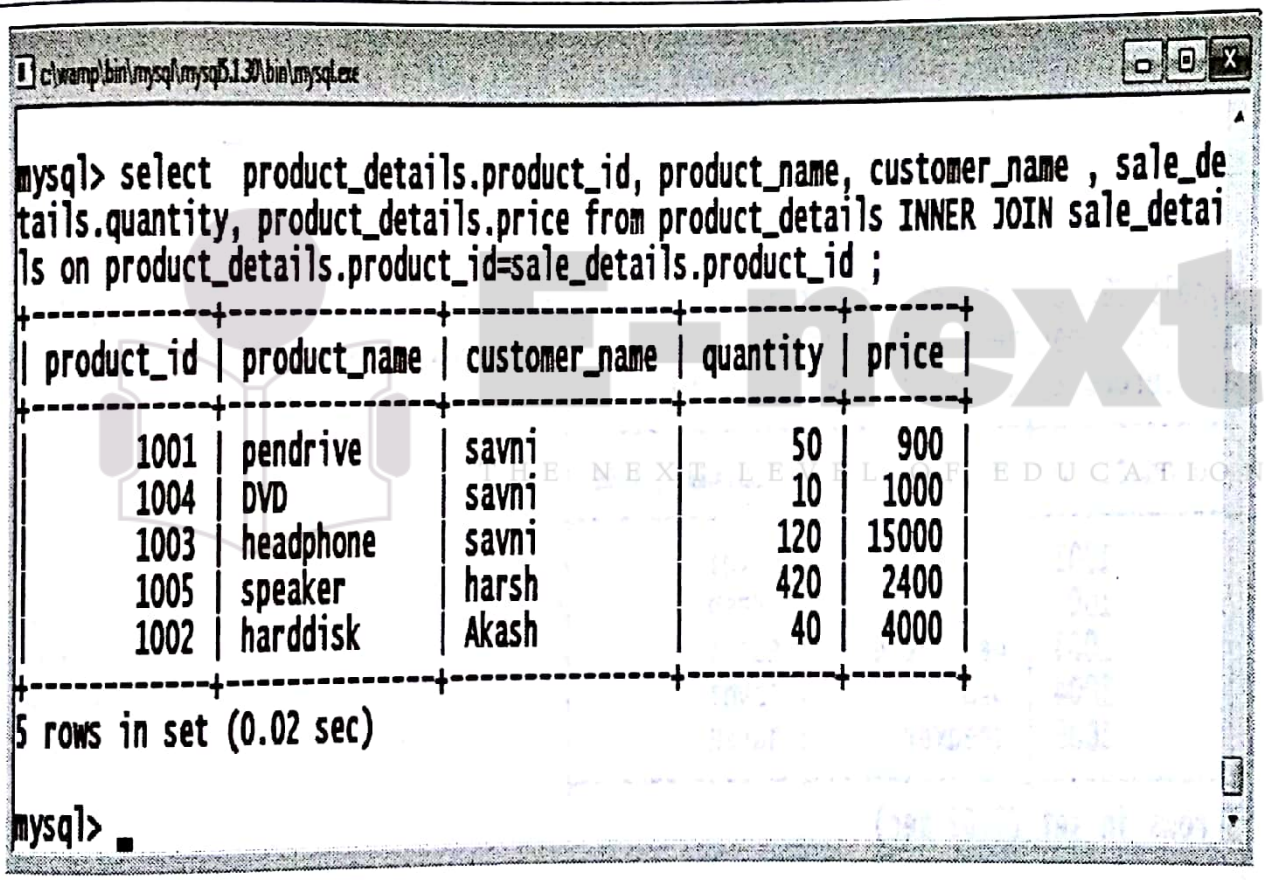
The INNER JOIN is used to display records that have matching values in both tables.

Syntax

```
Select column_name_list from table_1
INNER JOIN table_2
where table_1.column_name = table_2.column_name
```

For Example

```
select  product_details.product_id, product_name, customer_name , sale_details.quantity,
product_details.price from product_details INNER JOIN sale_details on
product_details.product_id=sale_details.product_id ;
```



```
c:\wamp\bin\mysql\mysql5.1.30\bin\mysql.exe

mysql> select  product_details.product_id, product_name, customer_name , sale_de
tails.quantity, product_details.price from product_details INNER JOIN sale_detai
ls on product_details.product_id=sale_details.product_id ;
+------------+--------------+---------------+----------+-------+
| product_id | product_name | customer_name | quantity | price |
+------------+--------------+---------------+----------+-------+
|       1001 | pendrive     | savni         |       50 |   900 |
|       1004 | DVD          | savni         |       10 |  1000 |
|       1003 | headphone    | savni         |      120 | 15000 |
|       1005 | speaker      | harsh         |      420 |  2400 |
|       1002 | harddisk     | Akash         |       40 |  4000 |
+------------+--------------+---------------+----------+-------+
5 rows in set (0.02 sec)

mysql>
```

Outer Join

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

(i) Left Outer Join (ii) Right Outer Join

(i) Left Outer Join

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. Null values are shown at the place of right table values.

Syntax

```
SELECT column-name-list
from table-name1
LEFT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

For Example

```
SELECT product_details.product_id, product_details.product_name,
sale_details.customer_name FROM product_details LEFT OUTER JOIN sale_details ON
sale_details.product_id =product_details.product_id;
```

```
c:\wamp\bin\mysql\mysql5.1.30\bin\mysql.exe
mysql> SELECT product_details.product_id, product_details.product_name, sale_det
ails.customer_name FROM product_details LEFT OUTER JOIN sale_details ON sale_det
ails.product_id =product_details.product_id;
+------------+--------------+---------------+
| product_id | product_name | customer_name |
+------------+--------------+---------------+
|       1001 | pendrive     | savni         |
|       1002 | harddisk     | Akash         |
|       1003 | headphone    | savni         |
|       1004 | DVD          | savni         |
|       1005 | speaker      | harsh         |
+------------+--------------+---------------+
5 rows in set (0.02 sec)

mysql>
```

(ii) Right Outer Join

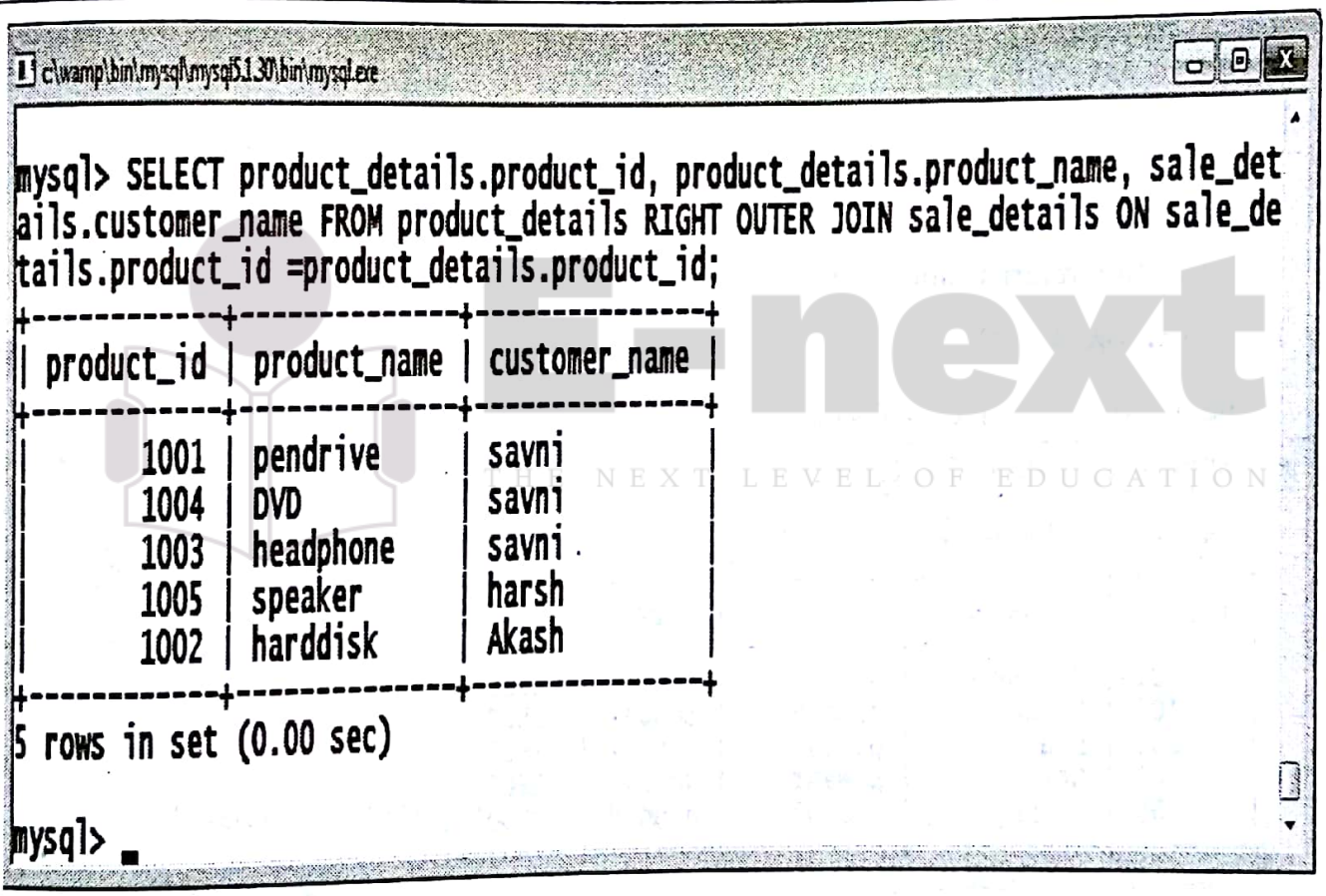
Returns all rows from the right table even if there are no matches in the left table. Null values are shown at the place of left table values.

Syntax

```
select column-name-list
from table-name1
RIGHT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

For Example

```
SELECT product_details.product_id, product_details.product_name,
sale_details.customer_name FROM product_details RIGHT OUTER JOIN sale_details ON
sale_details.product_id =product_details.product_id;
```



```
c:\wamp\bin\mysql\mysql5.1.30\bin\mysql.exe

mysql> SELECT product_details.product_id, product_details.product_name, sale_det
ails.customer_name FROM product_details RIGHT OUTER JOIN sale_details ON sale_de
tails.product_id =product_details.product_id;
+------------+--------------+---------------+
| product_id | product_name | customer_name |
+------------+--------------+---------------+
|       1001 | pendrive     | savni         |
|       1004 | DVD          | savni         |
|       1003 | headphone    | savni         |
|       1005 | speaker      | harsh         |
|       1002 | harddisk     | Akash         |
+------------+--------------+---------------+
5 rows in set (0.00 sec)

mysql>
```

(b) Aggregating Data Using Group Functions.

Count()

This function returns total number of values of specified column of the table.

For Example

Write a query to retrieve count of employees who join in 2015 year.

```
select count(Emp_no) from Employee2 where First_name like 'swa%';
```

```
mysql>  select * from Employee2;
+--------+------------+----------+--------+--------+
| Emp_no | First_name | Last_name | city   | salary |
+--------+------------+----------+--------+--------+
|    101 | Rajesh     | powar    | pune   |  15000 |
|    102 | Vedant     | jadhav   | mumbai |  25000 |
|    103 | Swati      | patil    | mumbai |  15000 |
|    104 | Smar       | sawant   | nagpur |  28000 |
|    105 | Swaraj     | sawant   | nagpur |  20000 |
+--------+------------+----------+--------+--------+
5 rows in set (0.00 sec)

mysql>  select count(Emp_no)  from Employee2 where First_name  like 'swa%';
+---------------+
| count(Emp_no) |
+---------------+
|             2 |
+---------------+
1 row in set (0.00 sec)
```

Sum()

This function returns sum of all the values of specified column of the table.

For Example

Write a query to to find total salary amount paid to all the employees.

```
select Sum(Salary)  from Employee2;
```

```
+--------+------------+----------+--------+--------+
| Emp_no | First_name | Last_name | city   | salary |
+--------+------------+----------+--------+--------+
|    101 | Rajesh     | powar    | pune   |  15000 |
|    102 | Vedant     | jadhav   | mumbai |  25000 |
|    103 | Swati      | patil    | mumbai |  15000 |
|    104 | Smar       | sawant   | nagpur |  28000 |
|    105 | Swaraj     | sawant   | nagpur |  20000 |
+--------+------------+----------+--------+--------+
5 rows in set (0.00 sec)

mysql>  select sum(salary) from Employee2;
+-------------+
| sum(salary) |
+-------------+
|      103000 |
+-------------+
1 row in set (0.00 sec)

mysql>
```

Min()

This function returns smallest value from specified column of the table.

For Example

Write a query to retrieve record of employee who gets least salary.

```
Select * from Employee2 where Salary =(Select min(Salary) from Employee2);
```

```
c:\wamp\bin\mysql\mysql5.1.30\bin\mysql.exe
mysql>  select * from Employee2;
+--------+------------+-----------+--------+--------+
| Emp_no | First_name | Last_name | city   | salary |
+--------+------------+-----------+--------+--------+
|    101 | Rajesh     | powar     | pune   |  15000 |
|    102 | Vedant     | jadhav    | mumbai |  25000 |
|    103 | Swati      | patil     | mumbai |  15000 |
|    104 | Smar       | sawant    | nagpur |  28000 |
|    105 | Swaraj     | sawant    | nagpur |  20000 |
+--------+------------+-----------+--------+--------+
5 rows in set (0.00 sec)

mysql> Select * from Employee2 where Salary=(Select min(Salary) from Employee2);

+--------+------------+-----------+--------+--------+
| Emp_no | First_name | Last_name | city   | salary |
+--------+------------+-----------+--------+--------+
|    101 | Rajesh     | powar     | pune   |  15000 |
|    103 | Swati      | patil     | mumbai |  15000 |
+--------+------------+-----------+--------+--------+
2 rows in set (0.00 sec)
```

Max()

This function returns greatest value from specified column of the table.

For Example

Write a query to retrieve record of employee who gets maximum salary.

```
Select * from Employee2 where Salary=(Select max(Salary) from Employee2);
```

```
mysql>  select * from Employee2;
+--------+------------+-----------+--------+--------+
| Emp_no | First_name | Last_name | city   | salary |
+--------+------------+-----------+--------+--------+
|    101 | Rajesh     | powar     | pune   |  15000 |
|    102 | Vedant     | jadhav    | mumbai |  25000 |
|    103 | Swati      | patil     | mumbai |  15000 |
|    104 | Smar       | sawant    | nagpur |  28000 |
|    105 | Swaraj     | sawant    | nagpur |  20000 |
+--------+------------+-----------+--------+--------+
5 rows in set (0.00 sec)

mysql>
mysql> Select * from Employee2 where Salary=(Select max(Salary) from Employee2);

+--------+------------+-----------+--------+--------+
| Emp_no | First_name | Last_name | city   | salary |
+--------+------------+-----------+--------+--------+
|    104 | Smar       | sawant    | nagpur |  28000 |
+--------+------------+-----------+--------+--------+
1 row in set (0.00 sec)

mysql>
```

Avg()

The following SQL statement finds the average salary of all employee:

For example

```
select Avg(salary) from Employee2;
```

```
mysql>  select * from Employee2;
+--------+------------+-----------+--------+--------+
| Emp_no | First_name | Last_name | city   | salary |
+--------+------------+-----------+--------+--------+
|    101 | Rajesh     | powar     | pune   |  15000 |
|    102 | Vedant     | jadhav    | mumbai |  25000 |
|    103 | Swati      | patil     | mumbai |  15000 |
|    104 | Smar       | sawant    | nagpur |  28000 |
|    105 | Swaraj     | sawant    | nagpur |  20000 |
+--------+------------+-----------+--------+--------+
5 rows in set (0.00 sec)

mysql> select Avg(salary) from Employee2;
+-------------+
| Avg(salary) |
+-------------+
|  20600.0000 |
+-------------+
1 row in set (0.00 sec)

mysql>
```

(c) Sub queries

Sub-Queries

Writing a query inside another query is known as nested query or subquery. The inner query get executed first, then the output on inner query is given as input to outer query.

Consider the previous emp table

Example

To display records of employees working in SMITH's department

```
Select * from emp where deptno =
(select deptno from emp where ename = 'SMITH');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7876	ADAMS	CLERK	7788	01/12/1983	1100		20

3. Manipulating Data

Data Manipulation Language (DML)

- The Data Manipulation Language (DML) is used for accessing and manipulating data in a database. It allows users to access, insert, update, and delete data from the database.
  - To insert record into the table – **INSERT**
  - To access or read records from table – **SELECT**
  - Update the records in table – **UPDATE**
  - Delete the records from the table – **DELETE**

(a) Using INSERT STATEMENT

1 Inserting record into tables

After creation of table the insert command is used to insert one or more records in the table.

Insert query has different forms.

Format 1 : Inserting a single row of data into a table

Syntax

```
Insert into table_name[(column_name1, column_name2,...)]
values (value1, value2....);
```

For example

1. Insert 5 records in Employee table

```
Insert into Employee values(101,'Rajesh', '1995-11-02',12000),(102,'Swati', '2015-01-
07',20000), (103,'Vedant','1999-05-03',30000), (104,'Vedika','2005-02-
10',52000),(105,'Ankita', '201603-01',8000);
```

After inserting 5 records in table the table will look like as follows:

```
c:\wamp\bin\mysql\mysql5.1.30\bin\mysql.exe

mysql> Insert into Employee values(101,'Rajesh', '1995-11-02',12000);
Query OK, 1 row affected (0.02 sec)

mysql> Insert into Employee values(102,'Swati', '2015-01-07',20000);
Query OK, 1 row affected (0.00 sec)

mysql> Insert into Employee values(103,'Vedant','1999-05-03',30000);
Query OK, 1 row affected (0.00 sec)

mysql> Insert into Employee values(104,'Vedika','2005-02-10',52000);
Query OK, 1 row affected (0.00 sec)

mysql> Insert into Employee values(105,'Ankita', '201603-01',8000);
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         103 | Vedant        | 1999-05-03   |  30000 |
|         104 | Vedika        | 2005-02-10   |  52000 |
|         105 | Ankita        | 0000-00-00   |   8000 |
+-------------+---------------+--------------+--------+
5 rows in set (0.00 sec)

mysql>
```

2. Consider we do not have value for salary while inserting a new record in Employee table. Then the query will be

```
Insert into Employee(Employee_no, Employee_name, Joining_date)
values(106,'Ankur','2017-03-15');
```

It will set salary value for the employee as NULL.

```
mysql> Insert into Employee(Employee_no, Employee_name, Joining_date) values(106
,'Ankur','2017-03-15');
Query OK, 1 row affected (0.02 sec)

mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         103 | Vedant        | 1999-05-03   |  30000 |
|         104 | Vedika        | 2005-02-10   |  52000 |
|         105 | Ankita        | 0000-00-00   |   8000 |
|         106 | Ankur         | 2017-03-15   |   NULL |
+-------------+---------------+--------------+--------+
6 rows in set (0.00 sec)

mysql>
```

Format 2 : Inserting data into a table from another table

Syntax

```
Insert into table_name select column_name1, columne_name2 from table_name;
```

Insert records in newEmployee1 table same as in Employee table.

For Example

```
Insert into newEmployee1   select * from Employee;
```

```
mysql> select * from newEmployee1;
Empty set (0.00 sec)

mysql> Insert into newEmployee1 select * from Employee;
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> select * from newEmployee1;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         103 | Vedant        | 1999-05-03   |  30000 |
|         104 | Vedika        | 2005-02-10   |  52000 |
|         105 | Ankita        | 0000-00-00   |   8000 |
|         106 | Ankur         | 2017-03-15   |   NULL |
+-------------+---------------+--------------+--------+
6 rows in set (0.00 sec)

mysql>
```

(b) Using DELETE STATEMENT

Delete

As per requirement, the records from existing table can be removed using delete command. Delete command can have 'WHERE' clause optionally.

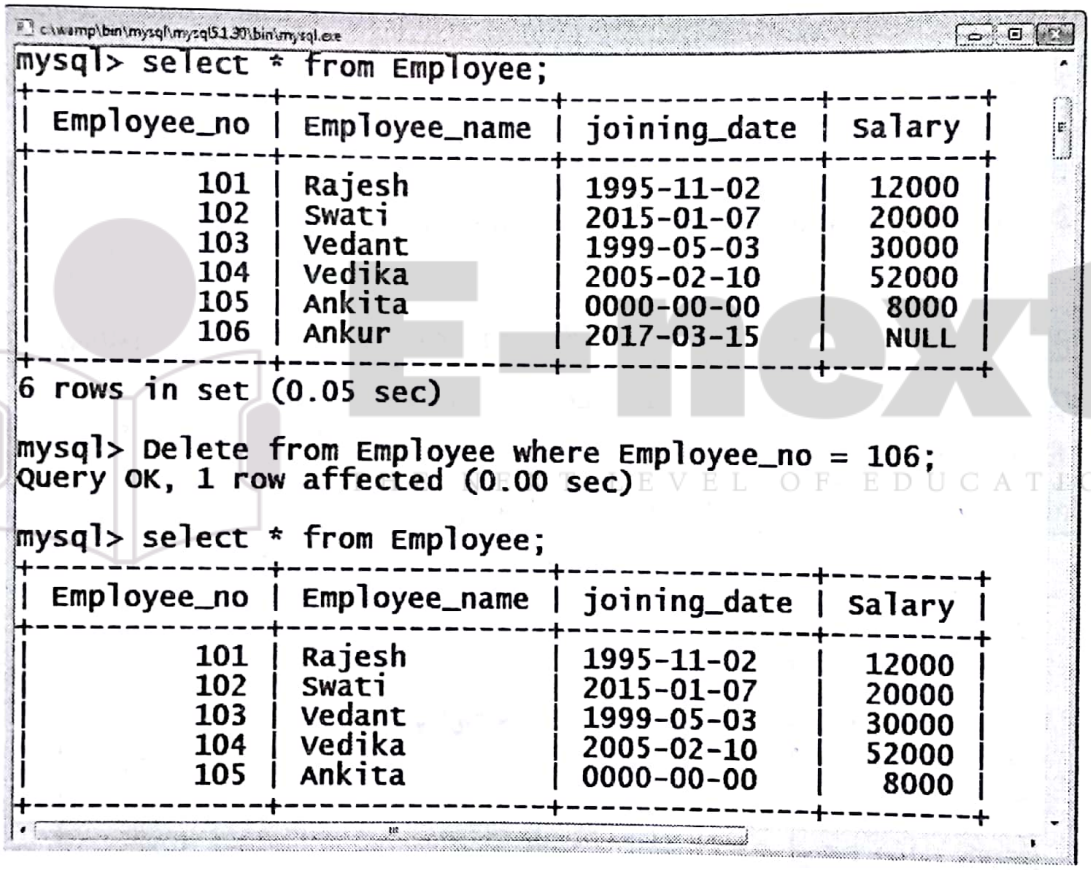
Syntax

```
Delete from table_name;
```

For Example

Write a query to remove record of Employee_no 106 from Employee table,

```
Delete from Employee where Employee_no = 106;
```



```
mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         103 | Vedant        | 1999-05-03   |  30000 |
|         104 | Vedika        | 2005-02-10   |  52000 |
|         105 | Ankita        | 0000-00-00   |   8000 |
|         106 | Ankur         | 2017-03-15   |   NULL |
+-------------+---------------+--------------+--------+
6 rows in set (0.05 sec)

mysql> Delete from Employee where Employee_no = 106;
Query OK, 1 row affected (0.00 sec)

mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         103 | Vedant        | 1999-05-03   |  30000 |
|         104 | Vedika        | 2005-02-10   |  52000 |
|         105 | Ankita        | 0000-00-00   |   8000 |
+-------------+---------------+--------------+--------+
```

(c) Using UPDATE STATEMENT

Update

To make changes in the database 'update' command is used. The update command consists of 'set' clause and an optional 'where' clause'. **'WHERE'** clause is used to make changes in specific records.

Syntax

```
Update table_name set column_name = new_value [where condition];
```

For Example

Update Employee set Salary=50000 where Employee_no=105;

```
mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         103 | Vedant        | 1999-05-03   |  30000 |
|         104 | Vedika        | 2005-02-10   |  52000 |
|         105 | Ankita        | 0000-00-00   |   8000 |
+-------------+---------------+--------------+--------+
5 rows in set (0.00 sec)

mysql> Update Employee set Salary=50000 where Employee_no=105;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         103 | Vedant        | 1999-05-03   |  30000 |
|         104 | Vedika        | 2005-02-10   |  52000 |
|         105 | Ankita        | 0000-00-00   |  50000 |
+-------------+---------------+--------------+--------+
```

Update Employee set joining_date='2015-03-15' where Employee_no=105;

```
mysql> Update Employee set joining_date='2015-03-15' where Employee_no=105;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         103 | Vedant        | 1999-05-03   |  30000 |
|         104 | Vedika        | 2005-02-10   |  52000 |
|         105 | Ankita        | 2015-03-15   |  50000 |
+-------------+---------------+--------------+--------+
5 rows in set (0.00 sec)

mysql>
```

4. Creating and Managing Tables

(a) Creating and Managing Tables

Creating Table

The **CREATE TABLE** statement is used to create table in database.

Syntax

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

Example

Following command creates a table Employee having four columns : Employee_no, Employee_name, Joining _date, and Salary.

```
CREATE Table Employee (
Employee_no integer(3),
Employee_name varchar(20),
 joining_date date ,
Salary integer(6));
```

1. Creating New Table from Existing Table

Consider the existing table Employee

Creating new table same as of existing table.

Syntax

```
Create table table_name as select * from existing_table_name;
```

For Example

```
Create table newEmployee1
As
select * from Employee;
```

The newly created table newEmployee1 will include all the fields and records as in Employee table.

Creating new table having specific fields but all the records from existing table.

Syntax

```
Create table table_name as select field_1,field_2... from existing_table_name;
```

For Example

```
Create table newEmployee2
As
select Employee_no, Employee_name from Employee;
```

After executing this command the newly created table will contain two fields such as Employee_no and Employee_name , and also contain all the corresponding records as in Employee table.

Creating new table having specific records but all the fields from existing table.

Syntax

```
Create table table_name as select * from existing_table_name where condition
```

For Example

```
Create table newEmployee3
As
select * from Employee
where Salary > 80000;
```

The newly created table will have same structure as of employee table, but it will contain records of only those Employees who got Salary above 80000 Rs.

Creating new table having no records but all the fields from existing table.

That means copying only structure of existing table

Syntax

```
Create table table_name as select * from existing_table_name where false condition
```

For Example

```
Create table newEmployee4
As
select * from Employee
where 1=2;
```

Here 1=2 is the false condition

The newly created table will have same structure as of existing table, but it will not copy any records from.

2. Modifying Table

ALTER TABLE query is used to modify structure of a table which is already exists in the database. We can add, delete or modify column.

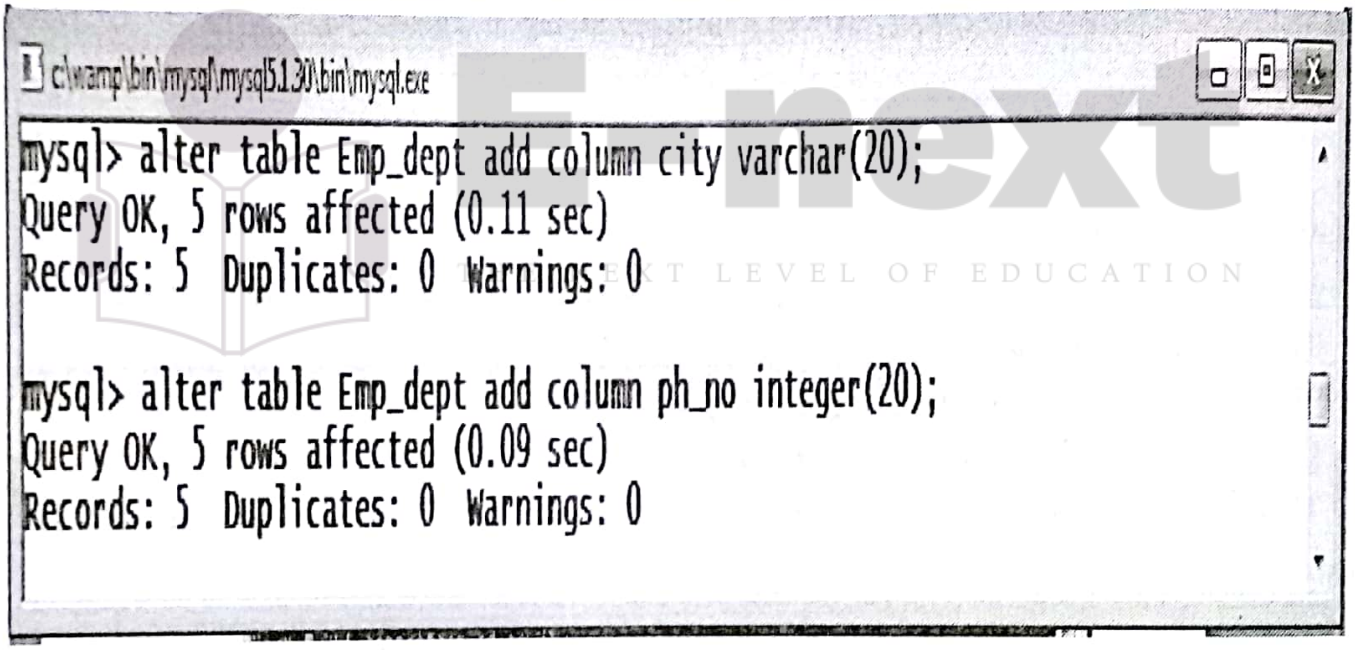
Adding New Column in a Table

Syntax

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

For Example

```
ALTER TABLE Emp_dept ADD column city varchar(20);
```



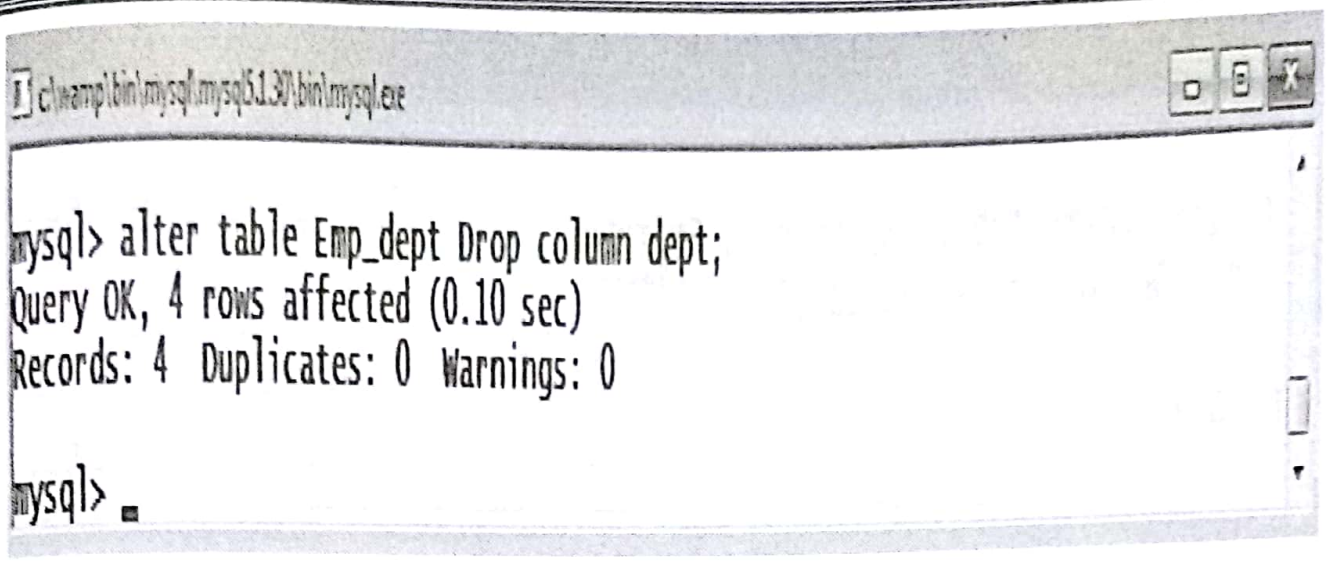
Dropping Column from Table

Syntax

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

For Example:

```
ALTER TABLE Emp_dept DROP COLUMN dept;
```



Modifying Column of a Table

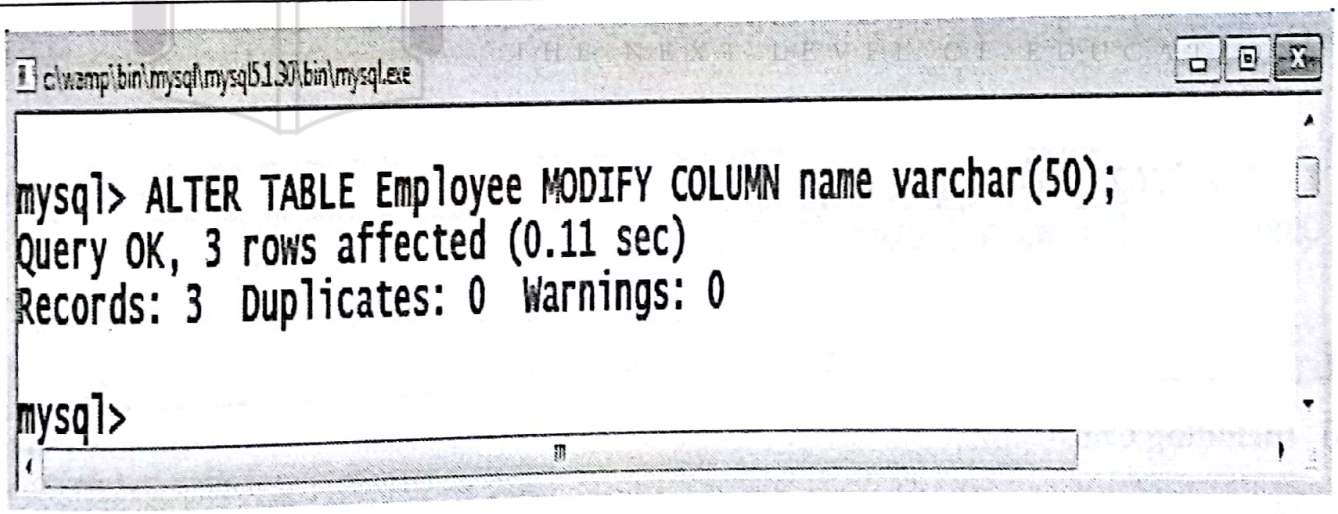
Here we are changing the data type and size of column roll_number.

Syntax

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

For Example

```
ALTER TABLE Employee modify column Employee_no varchar(4);
```



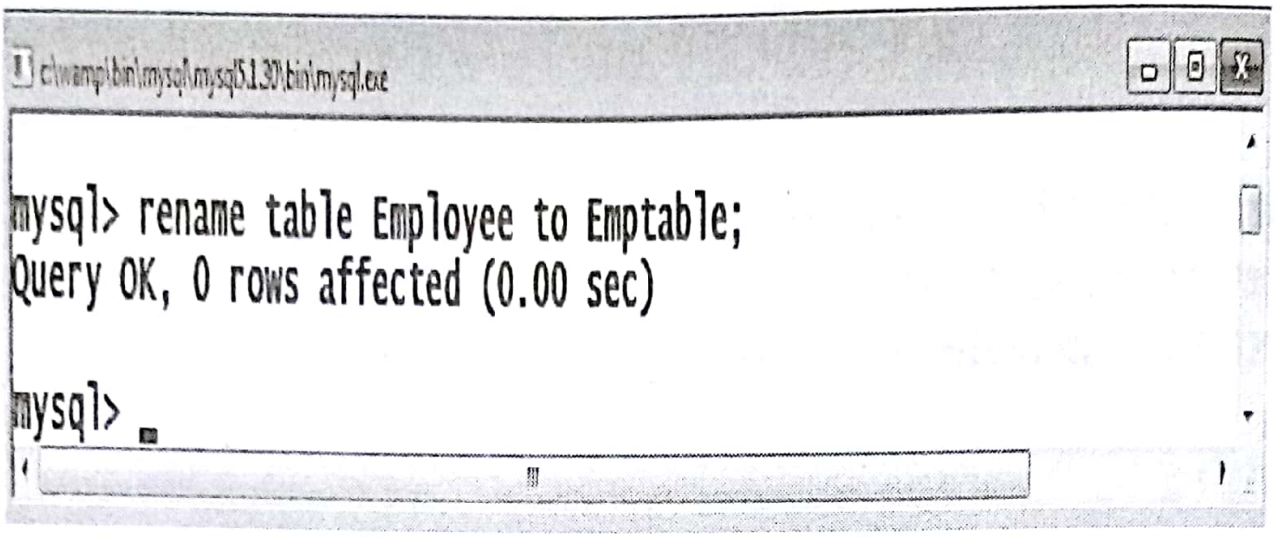
3. Renaming Table

Syntax

```
rename table current_table_name to new_table_name;
```

For Example

```
rename table Employee to Emptable;
```



4. Deleting Table

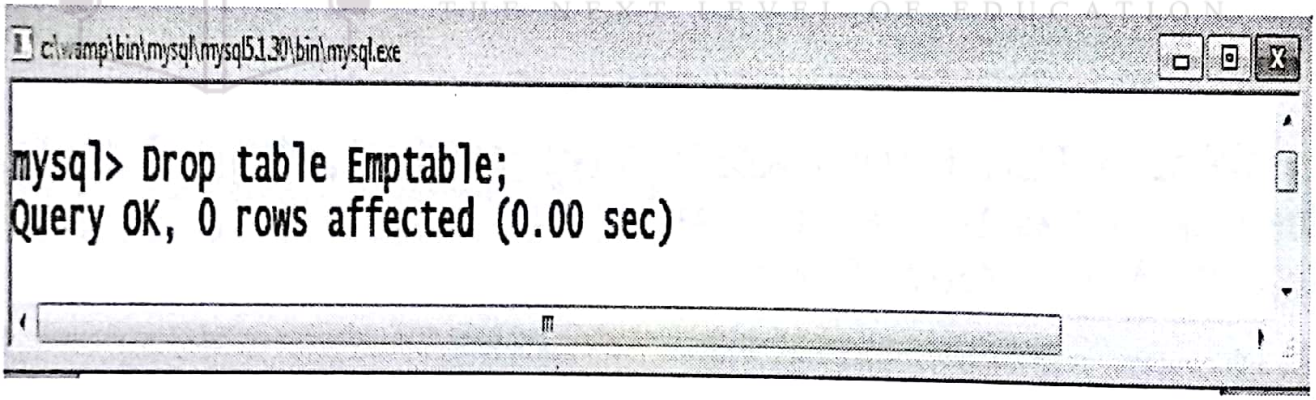
DROP TABLE query is used to delete table.

Syntax

```
drop table table_name;
```

For Example

```
Drop table Emptable;
```



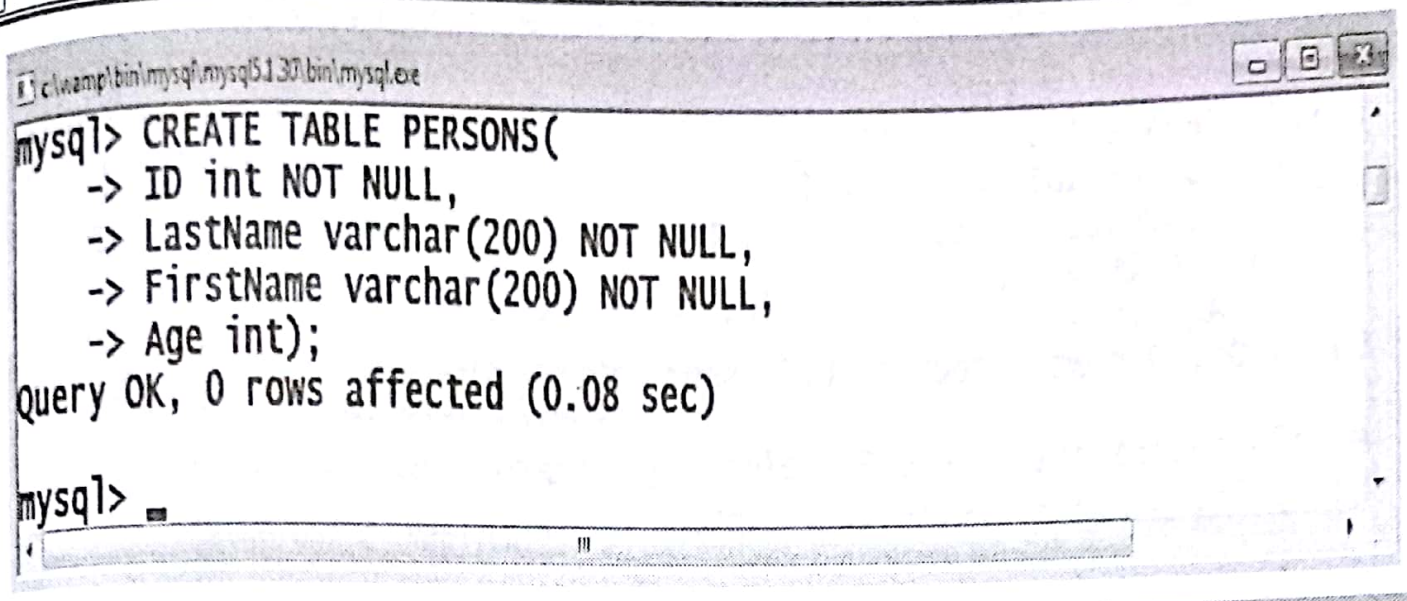
(b) Including Constraints

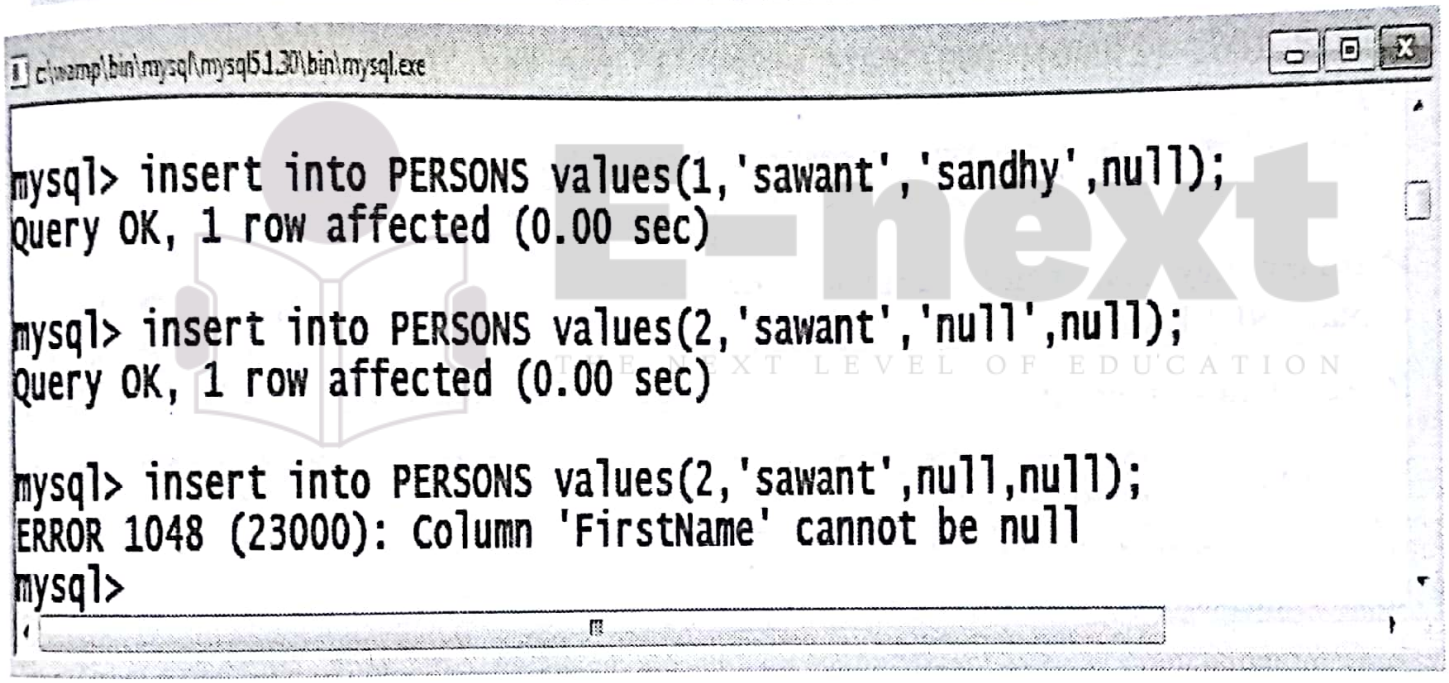
NOT NULL constraints

The NOT NULL constraint enforces a column to NOT accept NULL values

For example

```
Create table Persons(ID integer NOT NULL, LastName varchar(50) NOT NULL, FirstName varchar(50) NOT NULL, Age integer(10));
```



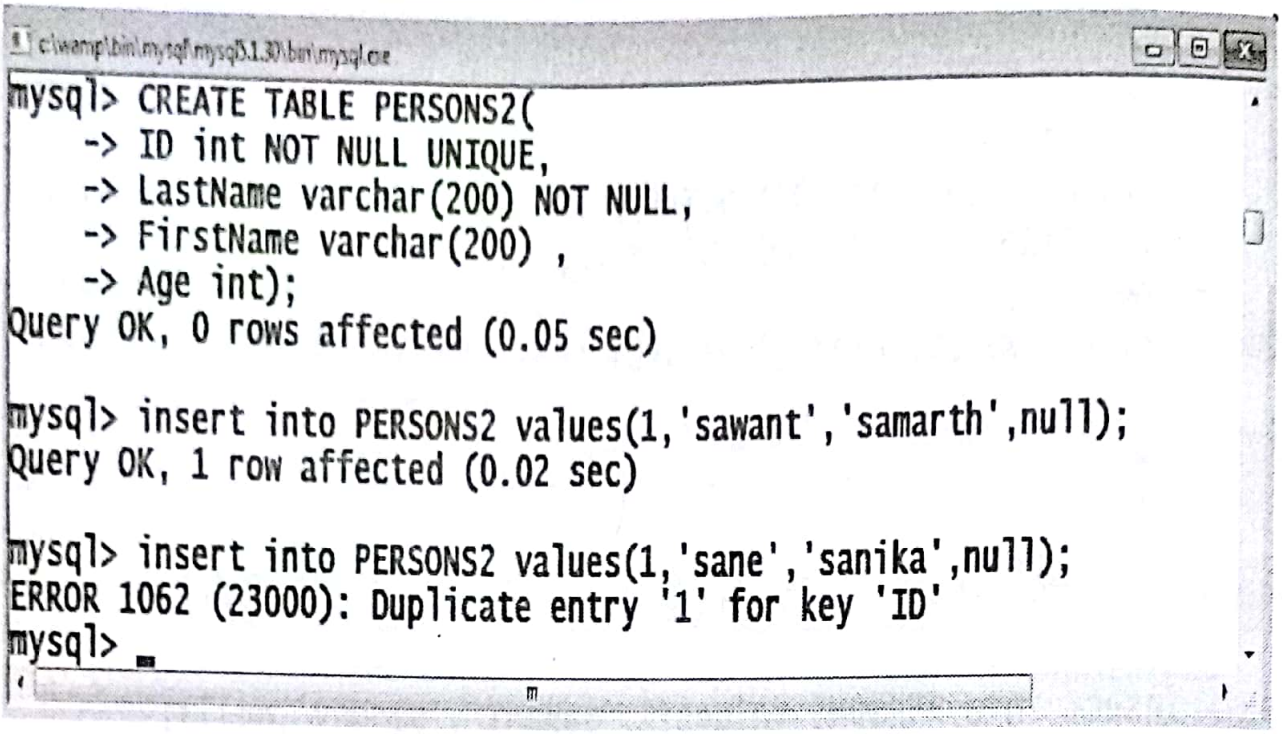


Unique constraints

The UNIQUE constraint ensures that all values in a column are different.

For example

```
Create table Persons(ID integer NOT NULL UNIQUE, LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL, Age integer(10));
```

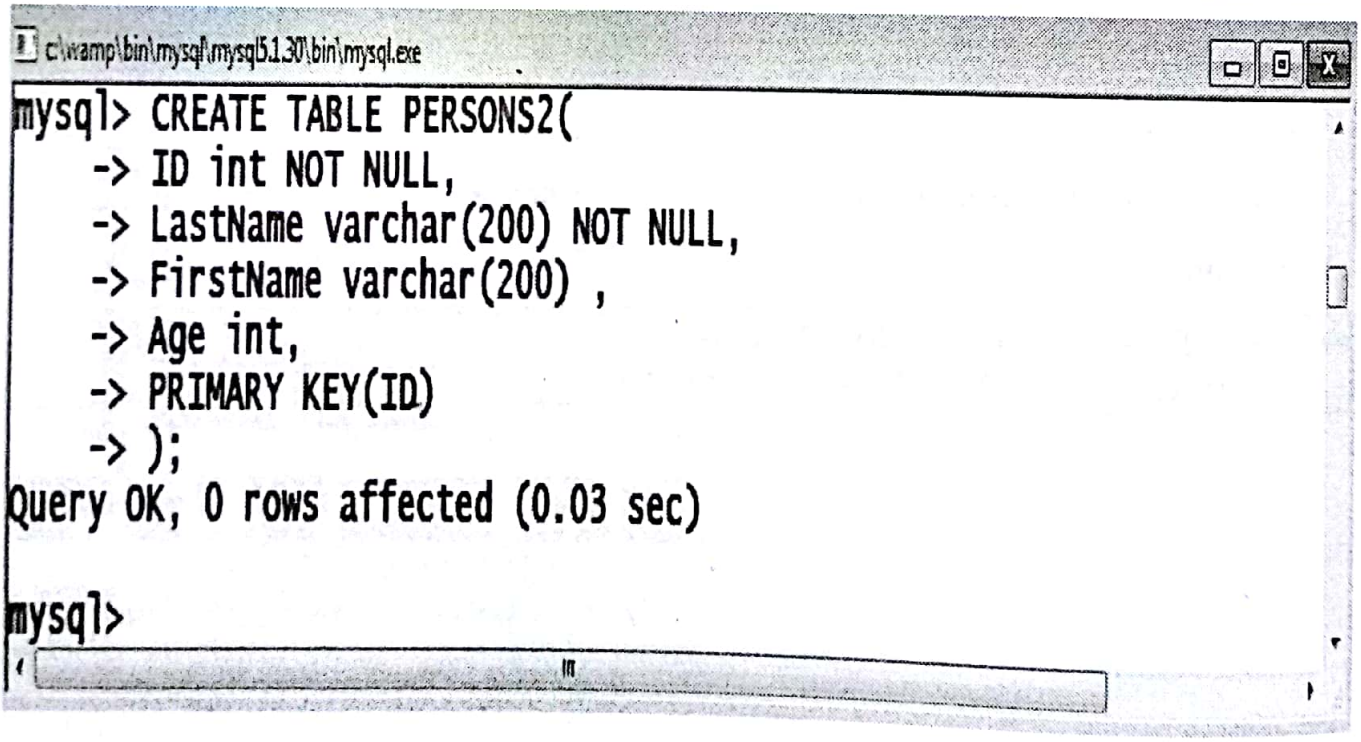


Primary key constraints: Primary keys must contain UNIQUE values, and cannot contain NULL values

Primary key on create table

For example

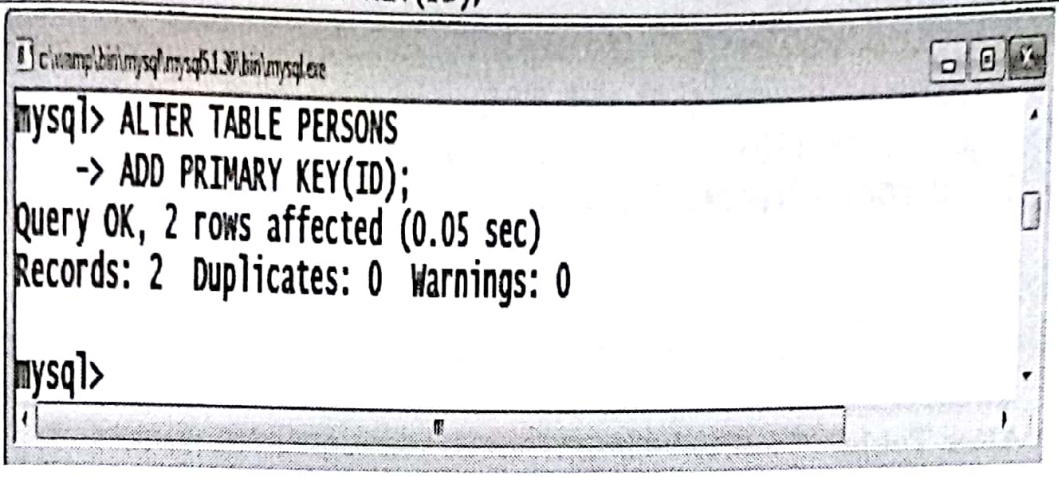
```
Create table Persons(ID integer NOT NULL UNIQUE, LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL, Age integer(10) PRIMARY KEY(ID));
```



Primary key on Alter table

For example

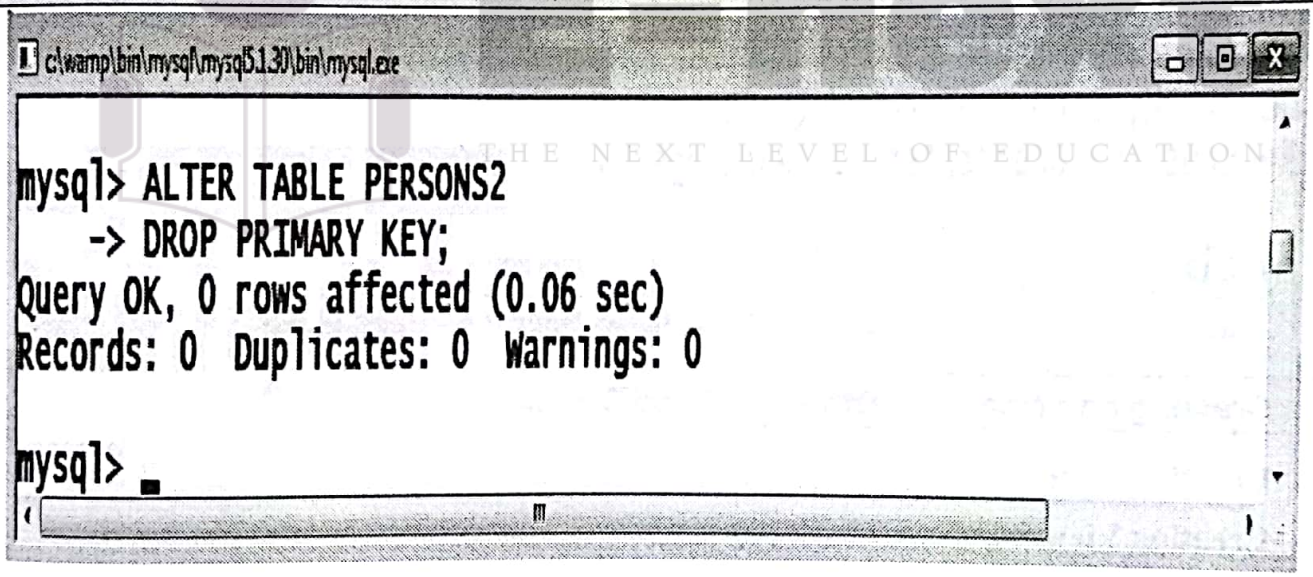
```
Alter table Person2 ADD PRIMARY KEY(ID);
```



Drop primary key constraints:

For example

```
Alter table Person2 DROP PRIMARY KEY;
```



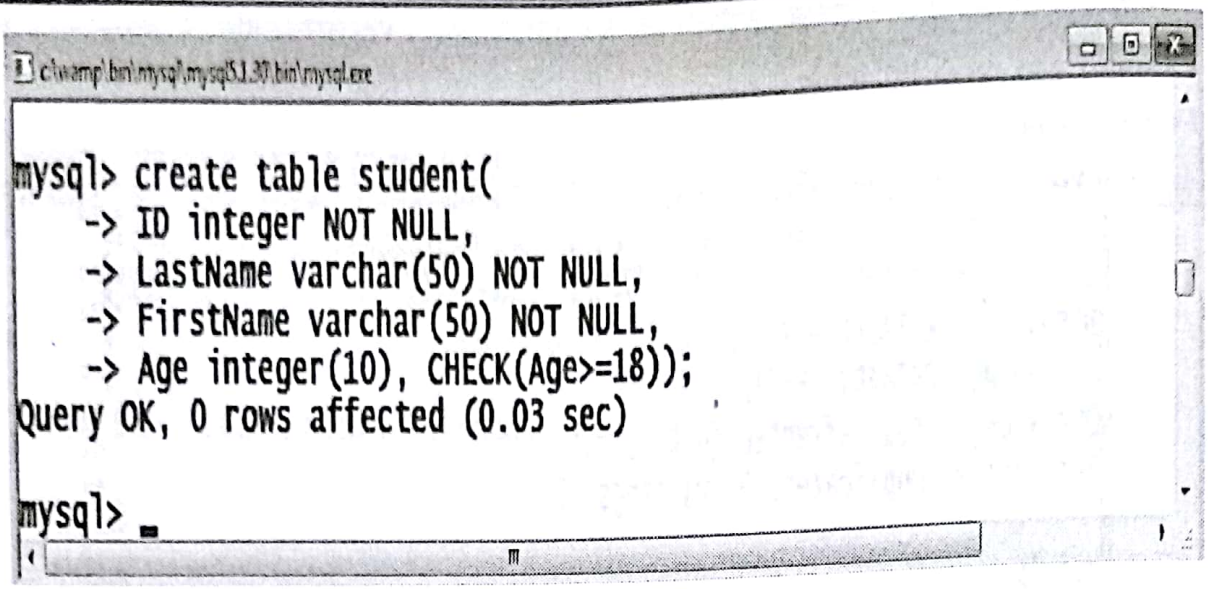
CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

CHECK on create table

For example

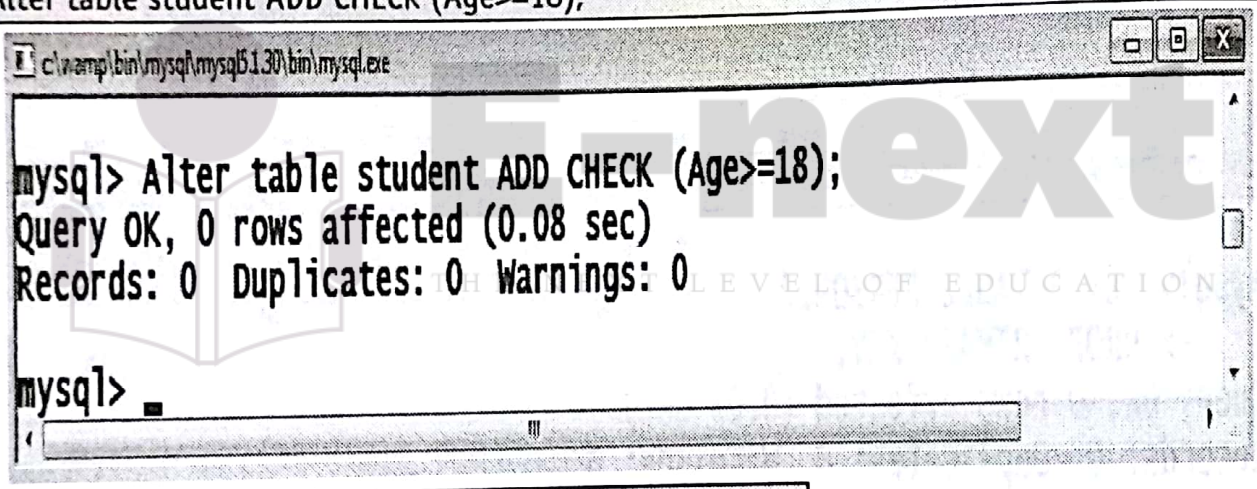
```
create table student(ID integer NOT NULL, LastName varchar(50) NOT NULL, FirstName
varchar(50) NOT NULL, Age integer(10) CHECK(Age>=18));
```



CHECK on Alter Table

For example

```
Alter table student ADD CHECK (Age>=18);
```



5. Creating and Managing Other Database Objects

(a) Creating Views

Creating View

Consider we have existing table as Employee (Employee_no, Employee_name, joining_date, Salary).

1. Creating view having all records and fields from existing table

Syntax

```
CREATE or replace VIEW view_name
AS
SELECT column1, column2, ...
FROM table_name
```

```
WHERE condition;
```

For Example

```
Create or replace view Emp_view1
As
select * from Employee;
```

This statement will create view having all the fields as in Employee table.

2. Creating view having specific fields but all the records from existing table

Syntax

```
Create or replace view view_name
As
select field_1, field_2... from existing_table_name;
```

For Example

```
Create or replace view Emp_view2
As
select Employee_no, Employee_name from Employee;
```

This statement will create view having specific fields from Employee table.

3. Creating new view having specific records but all the fields from existing table

Syntax

```
Create or replace view view_name
As
select * from existing_table_name
where condition;
```

For Example

```
Create or replace view Emp_view3
As
select * from Employee
where Salary > 80000;
```

(b) **Other Database Objects**

An index is a pointer to data in a table. An index in a database is similar to the alphabetical index of a book present at the end of book.

Indexes can be created or dropped with no effect on the data.

Creating Index

CREATE INDEX statement is used to create an index. In this statement we have to mention name of the index, the table and column, and whether the index is in ascending or descending order.

There are different types of indexes.

Syntax

```
CREATE [UNIQUE] INDEX index_name
ON table_name (column_name1,[column_name2, column_name3,...]);
```

For Example

1. CREATE INDEX emp_ind1 on Employee(Employee_name);
2. CREATE INDEX emp_ind2 ON Employee(Employee_no, Employee_name);
3. CREATE UNIQUE INDEX emp_ind3 on Employee(Employee_name);

Displaying Index : To display index information regarding table following query is used.

Syntax

```
Show index from table_name;
```

For Example

```
Show index from Employee;
```

```
mysql> create index emp_ind1 on Employee(name);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> create index emp_ind2 on Employee(no,name);
ERROR 1072 (42000): Key column 'no' doesn't exist in table
mysql> create index emp_ind2 on Employee(id,name);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> create index emp_ind3 on Employee(name);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> Show index from Employee;
+----------+------------+----------+--------------+-------------+---
---------+----------+--------+------+------------+---------+
| Table    | Non_unique | Key_name | Seq_in_index | Column_name | Co
rdinality | Sub_part | Packed | Null | Index_type | Comment |
+----------+------------+----------+--------------+-------------+---
---------+----------+--------+------+------------+---------+
| Employee |          1 | emp_ind1 |            1 | name        | A
     NULL |     NULL | NULL   | YES  | BTREE      |         |
| Employee |          1 | emp_ind2 |            1 | id          | A
     NULL |     NULL | NULL   | YES  | BTREE      |         |
| Employee |          1 | emp_ind2 |            2 | name        | A
     NULL |     NULL | NULL   | YES  | BTREE      |         |
| Employee |          1 | emp_ind3 |            1 | name        | A
     NULL |     NULL | NULL   | YES  | BTREE      |         |
+----------+------------+----------+--------------+-------------+---
---------+----------+--------+------+------------+---------+
4 rows in set (0.03 sec)
```

Dropping Index

To drop index of a table following query is used.

Syntax

```
Drop index index_name on table_name;
```

For Example

Drop index emp_ind2 on Employee;

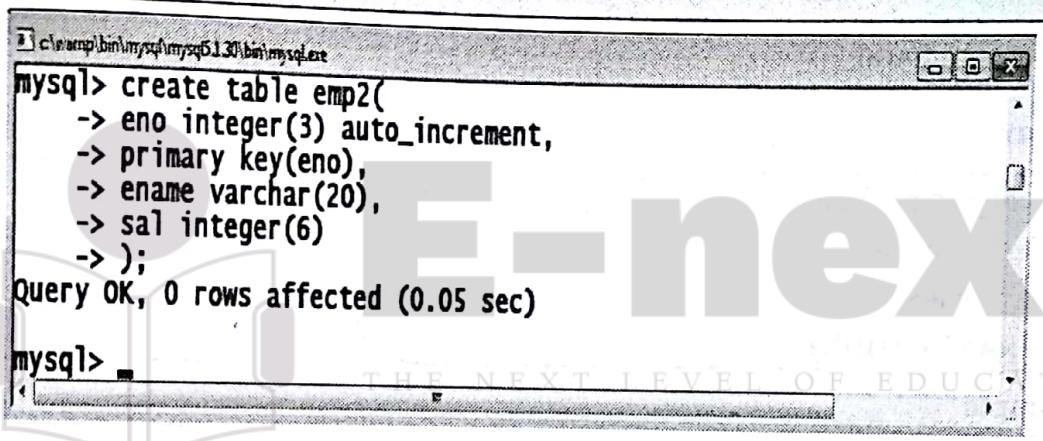
Sequence

Sequence : A sequence is a set of integers. Sequences are generated in order as per requirement. Sequences are used to create unique values for the rows.

Creating sequence

For example

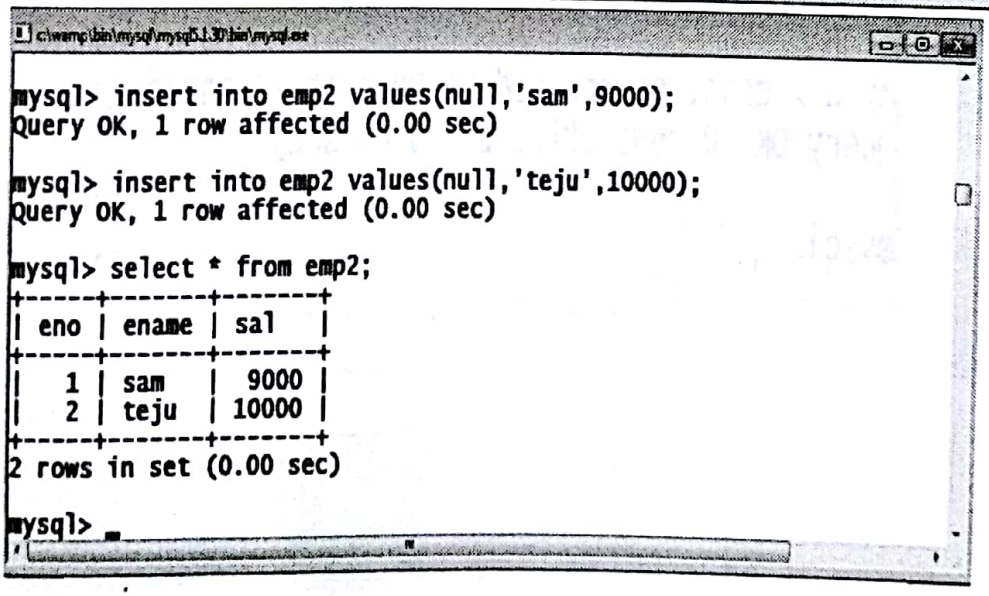
```
create table emp2( eno integer(3) auto_increment, primary key(eno), ename
varchar(20),sal integer(6));
```



Now insert records in the table

For example

```
insert into emp2 values(null,'sam',9000);
```



Here the column eno has auto increment values.

Synonym

Synonyms are basically alternative names for a table, view, sequence, procedure, stored function, package etc. Synonyms provide both data independence and location transparency.

Creating Synonym in Oracle

Create synonym e for emp;

(b) Controlling User Access

- **Grant :** to allow specified users to perform specified tasks.
- **Revoke :** to cancel previously granted or denied permissions.

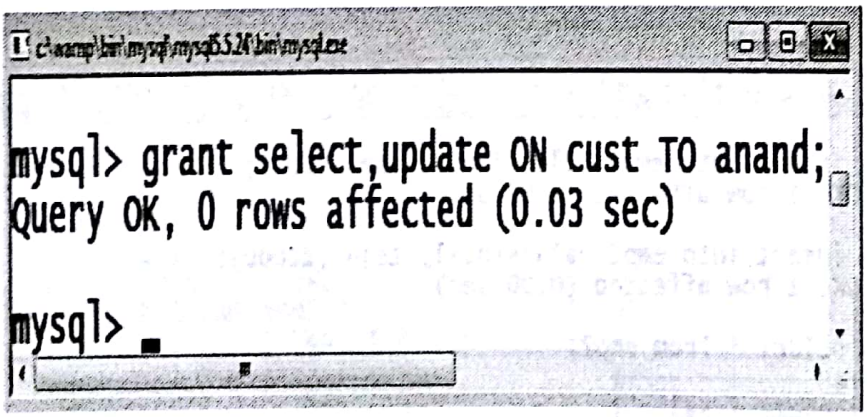
Grant command

Syntax

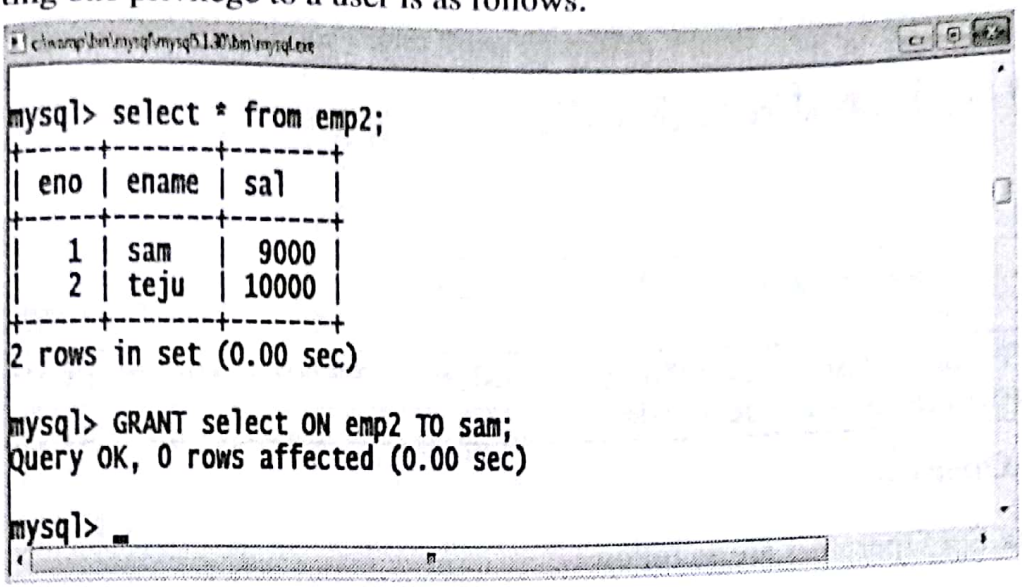
```
GRANT <object privileges>
ON <object_name>
TO <User_Name>
[WITH GRANT OPTION]
```

Example

```
GRANT ALL
ON cust TO anand
WITH GRANT OPTION;
```

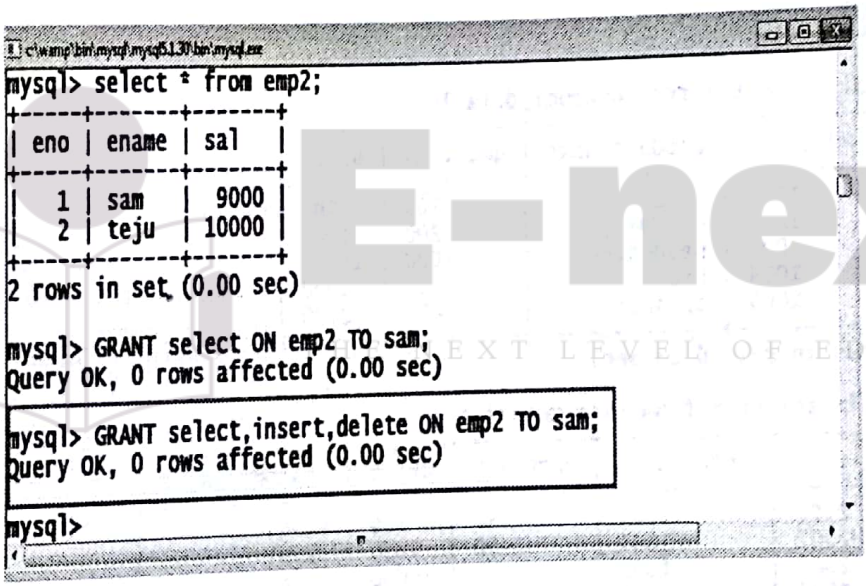


Granting one privilege to a user is as follows:



```
mysql> select * from emp2;
+-----+-------+-------+
| eno | ename | sal   |
+-----+-------+-------+
|   1 | sam   |  9000 |
|   2 | teju  | 10000 |
+-----+-------+-------+
2 rows in set (0.00 sec)

mysql> GRANT select ON emp2 TO sam;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Granting multiple privileges to a user is as follows:



```
mysql> select * from emp2;
+-----+-------+-------+
| eno | ename | sal   |
+-----+-------+-------+
|   1 | sam   |  9000 |
|   2 | teju  | 10000 |
+-----+-------+-------+
2 rows in set (0.00 sec)

mysql> GRANT select ON emp2 TO sam;
Query OK, 0 rows affected (0.00 sec)

mysql> GRANT select,insert,delete ON emp2 TO sam;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Revoke command

Syntax

```
REVOKE <Object_Privileges>
ON <Object_Name>
FROM <User_Name>
```

Example

```
REVOKE  select insert,delete
ON emp2
FROM sam
```

```
mysql> REVOKE select,insert,delete ON emp2 FROM sam;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

6. Using SET operators, Date/Time Functions, GROUP BY clause(advanced features) and advanced subqueries

(a) Using SET Operators

– The different Set Operators are as follows

- Union
- Union All
- Intersect
- Minus

Consider following two tables product_details and Sale_details :

```
mysql> select * from product_details;
+------------+--------------+----------+-------+
| product_id | product_name | quantity | price |
+------------+--------------+----------+-------+
|       1001 | pendrive     |      100 |   900 |
|       1002 | harddisk     |      200 |  4000 |
|       1003 | headphone    |     1000 | 15000 |
|       1004 | DVD          |       20 |  1000 |
|       1005 | speaker      |      600 |  2400 |
+------------+--------------+----------+-------+
5 rows in set (0.03 sec)

mysql> select * from sale_details;
+---------+------------+----------+-------+---------------+
| sale_no | product_id | quantity | price | customer_name |
+---------+------------+----------+-------+---------------+
|    2001 |       1001 |       50 |   900 | savni         |
|    2002 |       1004 |       10 |  1000 | savni         |
|    2003 |       1003 |      120 | 15000 | savni         |
|    2004 |       1005 |      420 |  2400 | harsh         |
|    2005 |       1002 |       40 |  4000 | Akash         |
+---------+------------+----------+-------+---------------+
5 rows in set (0.03 sec)
```

Union

The union operator returns all distinct rows selected by either query

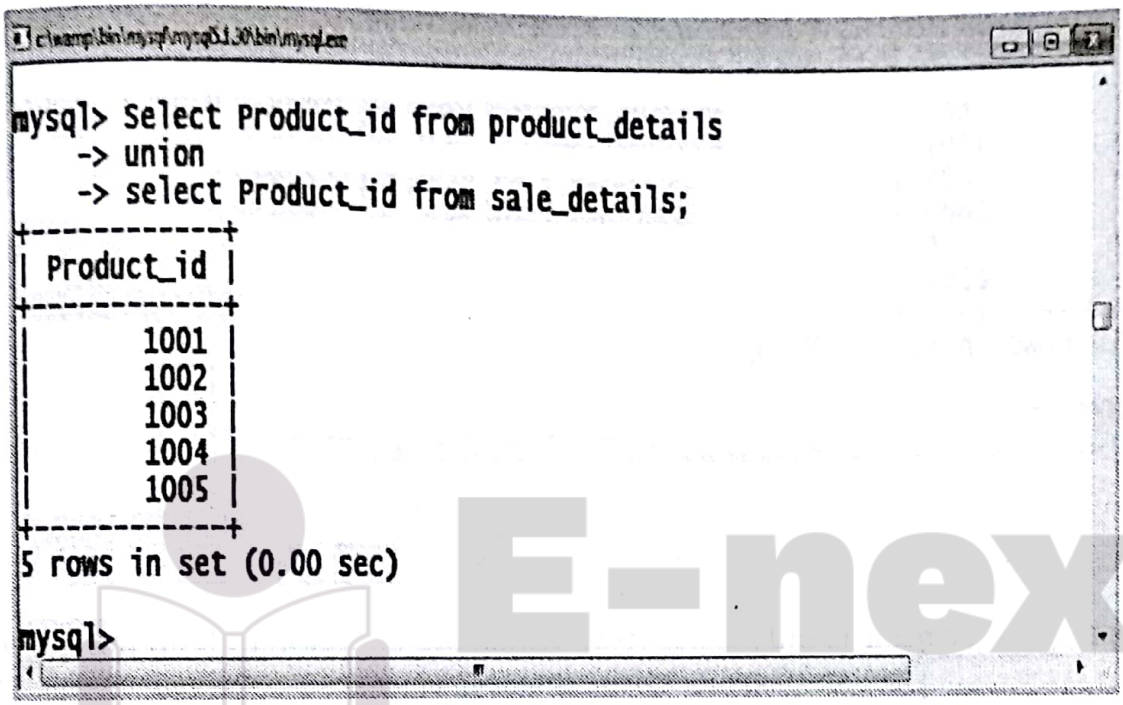
Syntax

```
Select column_name from table_1
Union
Select column_name from table_2
```

For Example

Write a query to retrieve ids of all the products even if they were sale or present in the storage room.

```
Select Product_id from product_details
union
select Product_id from sale_details;
```



```
mysql> Select Product_id from product_details
    -> union
    -> select Product_id from sale_details;
+------------+
| Product_id |
+------------+
|       1001 |
|       1002 |
|       1003 |
|       1004 |
|       1005 |
+------------+
5 rows in set (0.00 sec)

mysql>
```

Union All

The Union All operator returns all rows selected by either query including duplicates.

Syntax

```
Select column_name from table_1
Union all
Select column_name from table_2
```

For Example

Write a query to retrieve ids of all the products even if they were sold or present in the storage room.

```
Select Product_id from product_details
Union all
select Product_id from sale_details;
```

```
mysql> Select Product_id from product_details
    -> Union all
    -> select Product_id from sale_details;
+------------+
| Product_id |
+------------+
|       1001 |
|       1002 |
|       1003 |
|       1004 |
|       1005 |
|       1001 |
|       1004 |
|       1003 |
|       1005 |
|       1002 |
+------------+
10 rows in set (0.00 sec)

mysql>
```

Intersect

The intersect operator returns only those rows which are common to both the queries

Syntax

```
Select column_name from table_1
intersect
Select column_name from table_2
```

For Example

Write a query to retrieve ids of all the sold products.

```
Select Product_id from product_details
intersect
select Product_id from sale_details;
```

Minus

Minus operator displays the rows which are present in the first query but absent in the second query, with no duplicates and data is arranged in ascending order by default.

Syntax

```
Select column_name from table_1
minus
Select column_name from table_2
```

For Example

```
Select Product_id from product_details
minus
select Product_id from sale_details;
```

(b) Datetime Functions

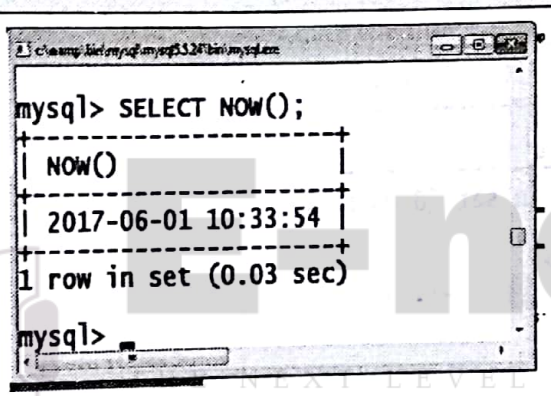
Now():

NOW() returns the current date and time.

Example

The following SELECT statement:

```
SELECT NOW();
```

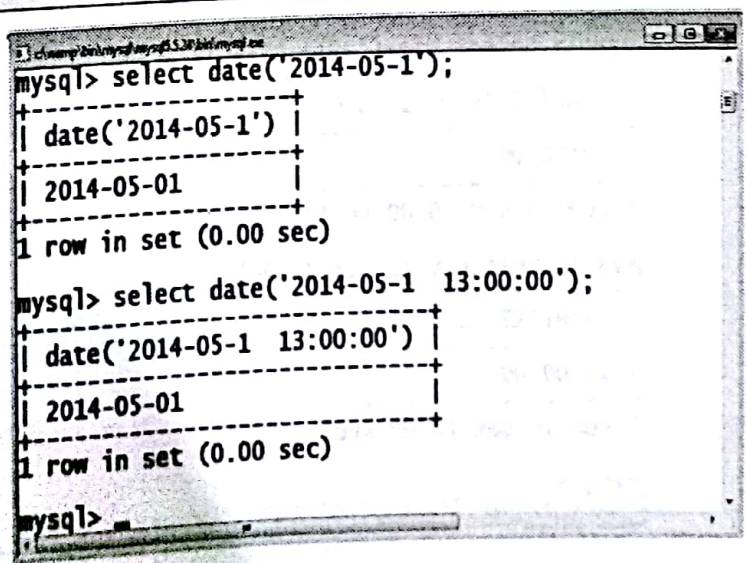


Date()

Extracts the date part of a date or date/time expression

Example

```
Select date('2014-5-1');
Select date('2014-5-1 11:00:00.0');
```

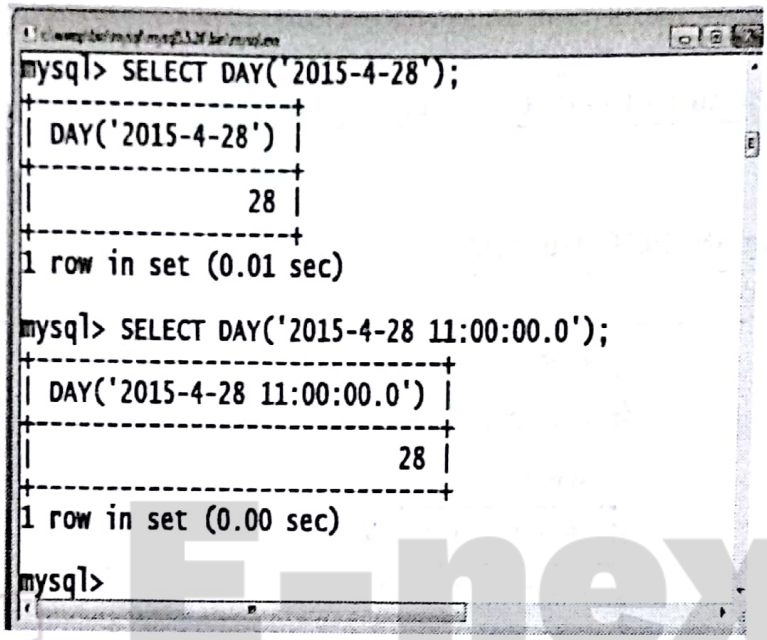


Day()

In SQL Server (Transact-SQL), the DAY function returns the day of the month (a number from 1 to 31) given a date value.

Example

```
Select day('2014-4-28 11:00:00.0');
```



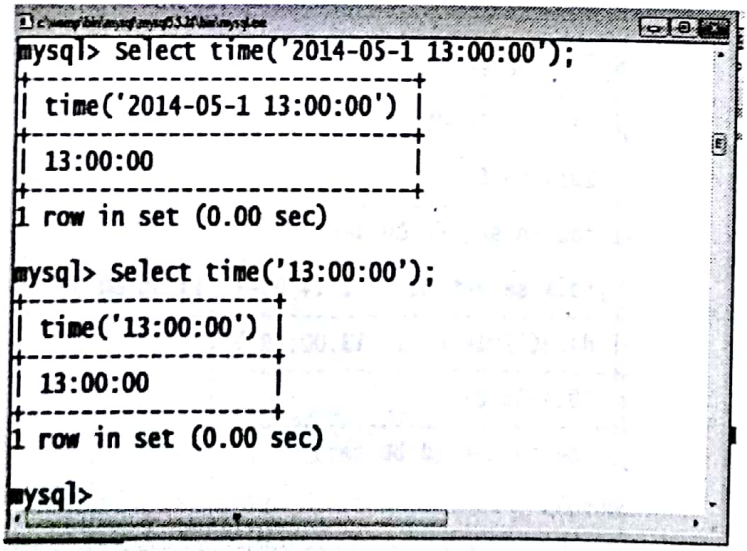
```
mysql> SELECT DAY('2015-4-28');
+------------------+
| DAY('2015-4-28') |
+------------------+
|               28 |
+------------------+
1 row in set (0.01 sec)

mysql> SELECT DAY('2015-4-28 11:00:00.0');
+-----------------------------+
| DAY('2015-4-28 11:00:00.0') |
+-----------------------------+
|                          28 |
+-----------------------------+
1 row in set (0.00 sec)

mysql>
```

Time()

The time function returns the time.

Example

```
Select time('11:00:00.0');
Or      Select time('2017-3-1 11:00:00.0');
```



```
mysql> Select time('2014-05-1 13:00:00');
+----------------------------+
| time('2014-05-1 13:00:00') |
+----------------------------+
| 13:00:00                   |
+----------------------------+
1 row in set (0.00 sec)

mysql> Select time('13:00:00');
+-----------------+
| time('13:00:00') |
+-----------------+
| 13:00:00        |
+-----------------+
1 row in set (0.00 sec)

mysql>
```

(c) **Enhancement to the GROUP BY Clause**

Group by clause

The GROUP BY clause is used in collaboration with the SELECT statement. It helps to arrange similar data into groups. It is also used with SQL functions to group the result from one or more tables.

Syntax

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
```

For example

Display sum of salaries department wise.

```
Select deptno,sum(sal) from emp group by deptno;
```

Output

DEPTNO	SUM(SAL)
30	9400
20	10875
10	8750

(d) **Advanced Subqueries**

In

select sub query:

syntax

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

For example

```
SQL> SELECT *
  FROM Employee
```

```
WHERE Employee_no IN (SELECT Employee_no
    FROM Emp_dept
    WHERE Employee_dept='computer');
```

```
mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         103 | Vedant        | 1999-05-03   |  30000 |
|         104 | Vedika        | 2005-02-10   |  52000 |
|         105 | Ankita        | 2015-03-15   |  50000 |
+-------------+---------------+--------------+--------+
5 rows in set (0.00 sec)

mysql> select * from Emp_dept;
+-------------+---------------+
| Employee_no | Employee_dept |
+-------------+---------------+
|         101 | computer      |
|         102 | computer      |
|         103 | civil         |
|         104 | ETC           |
|         105 | Electrical    |
+-------------+---------------+
5 rows in set (0.00 sec)

mysql> select * from Employee where Employee_no IN(select Employee_no from Emp_d
ept where Employee_dept='computer');
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
+-------------+---------------+--------------+--------+
2 rows in set (0.03 sec)

mysql>
```

Update sub query

Syntax

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

For example:

```
UPDATE Employee
  SET SALARY = SALARY * 0.50
  WHERE Employee_no IN (SELECT Employee_no FROM Emp_dept
    WHERE Employee_dept='ETC');
```

```
mysql> UPDATE Employee
    ->     SET SALARY = SALARY * 0.50
    ->     WHERE Employee_no IN (SELECT Employee_no FROM Emp_dept
    ->         WHERE Employee_dept='ETC');
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from Employee;
+-------------+---------------+-------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+-------------+--------+
|         101 | Rajesh        | 1995-11-02  |  12000 |
|         102 | Swati         | 2015-01-07  |  20000 |
|         103 | Vedant        | 1999-05-03  |  30000 |
|         104 | Vedika        | 2005-02-10  |  26000 |
|         105 | Ankita        | 2015-03-15  |  50000 |
+-------------+---------------+-------------+--------+
5 rows in set (0.00 sec)

mysql>
```

Delete sub query

```
DELETE FROM TABLE_NAME
[WHERE OPERATOR [VALUE]
  IN (SELECT COLUMN_NAME
  FROM TABLE_NAME)
  [WHERE)]
```

For example

```
DELETE FROM Employee
WHERE Employee_no IN (SELECT Employee_no FROM Emp_dept
WHERE Employee_dept='civil');
```

```
mysql> DELETE FROM Employee
    -> WHERE Employee_no IN (SELECT Employee_no FROM Emp_dept
    -> WHERE Employee_dept='civil');
Query OK, 1 row affected (0.00 sec)

mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         104 | Vedika        | 2005-02-10   |  26000 |
|         105 | Ankita        | 2015-03-15   |  50000 |
+-------------+---------------+--------------+--------+
4 rows in set (0.00 sec)

mysql>
```

Insert sub query

For example

```
SQL> Insert into Employee
Select * from Emp_dept
  WHERE Employee_no IN (SELECT Employee_no FROM Emp_dept);
```

```
mysql> Insert into Employee
    -> Select * from Emp_dept
    ->   WHERE Employee_no IN (SELECT Employee_no FROM Emp_dept);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         105 | Electrical    | NULL         |   NULL |
|         104 | ETC           | NULL         |   NULL |
|         103 | civil         | NULL         |   NULL |
|         102 | computer      | NULL         |   NULL |
|         101 | computer      | NULL         |   NULL |
+-------------+---------------+--------------+--------+
5 rows in set (0.00 sec)
```

Update sub query

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
 NOT IN (SELECT COLUMN_NAME
  FROM TABLE_NAME)
  [ WHERE) ]
```

For example:

```
UPDATE Employee
SET SALARY = SALARY * 0.50
WHERE Employee_no not IN (SELECT Employee_no FROM Emp_dept
      WHERE Employee_dept='ETC');
```

```
mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |  12000 |
|         102 | Swati         | 2015-01-07   |  20000 |
|         104 | Vedika        | 2005-02-10   |  26000 |
|         105 | Ankita        | 2015-03-15   |  50000 |
+-------------+---------------+--------------+--------+
4 rows in set (0.00 sec)

mysql> UPDATE Employee
    -> SET SALARY = SALARY * 0.50
    -> WHERE Employee_no not IN (SELECT Employee_no FROM Emp_dept
    ->          WHERE Employee_dept='ETC');
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |   6000 |
|         102 | Swati         | 2015-01-07   |  10000 |
|         104 | Vedika        | 2005-02-10   |  26000 |
|         105 | Ankita        | 2015-03-15   |  25000 |
+-------------+---------------+--------------+--------+
4 rows in set (0.00 sec)
```

Delete sub query

Syntax

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
 IN (SELECT COLUMN_NAME
  FROM TABLE_NAME)
  [ WHERE) ]
```

For example

```
DELETE FROM Employee
      WHERE Employee_no not IN (SELECT Employee_no FROM Emp_dept WHERE
Employee_ dept='civil');
```

```
mysql> select * from Employee;
+-------------+---------------+--------------+--------+
| Employee_no | Employee_name | joining_date | Salary |
+-------------+---------------+--------------+--------+
|         101 | Rajesh        | 1995-11-02   |   6000 |
|         102 | Swati         | 2015-01-07   |  10000 |
|         104 | Vedika        | 2005-02-10   |  26000 |
|         105 | Ankita        | 2015-03-15   |  25000 |
+-------------+---------------+--------------+--------+
4 rows in set (0.00 sec)

mysql> DELETE FROM Employee
    ->          WHERE Employee_no not IN (SELECT Employee_no FROM Emp_dept WHERE
 Employee_dept='civil');
Query OK, 4 rows affected (0.00 sec)

mysql> select * from Employee;
Empty set (0.00 sec)

mysql>
```

Some

SOME compare a value to each value in a list or results from a query and evaluate to true if the result of an inner query contains at least one row. SOME must match at least one row in the subquery and must be preceded by comparison operators. Suppose using greater than (>) with SOME means greater than at least one value same as it if we use less than (<) with SOME means less than at least one value.

Syntax

```
SELECT [column_name... | expression1 ]
FROM [table_name]
WHERE expression2 comparison_operator {ALL | ANY | SOME} ( subquery )
```

For example

```
SELECT * from Employee
WHERE no=SOME( SELECT no FROM Emp_dept
WHERE dept='computer');
```

```
mysql> select * from Employee;
+------+---------+--------+
| no   | name    | city   |
+------+---------+--------+
|    1 | sanjay  | pune   |
|    2 | ram     | sangli |
|    3 | samarth | mumbai |
+------+---------+--------+
3 rows in set (0.00 sec)

mysql> select * from Emp_dept;
+------+------------+
| no   | dept       |
+------+------------+
|    1 | computer   |
|    2 | civil      |
|    3 | ETC        |
|    4 | Electrical |
+------+------------+
4 rows in set (0.00 sec)

mysql> SELECT * from Employee
    -> WHERE no=SOME(  SELECT no FROM Emp_dept
    -> WHERE dept='computer');
+------+--------+------+
| no   | name   | city |
+------+--------+------+
|    1 | sanjay | pune |
+------+--------+------+
1 row in set (0.00 sec)

mysql> _
```


For example

```
SELECT * from emp2
                WHERE salary < SOME (  SELECT salary FROM emp1
                WHERE city='pune');
SELECT * from emp2
                WHERE salary <=SOME (  SELECT salary FROM emp1
                WHERE city='pune');
SELECT * from emp2
                WHERE salary >=SOME (  SELECT salary FROM emp1
                WHERE city='pune');
```

```
mysql> Select * from emp2 where salary<some(select salary from
    -> emp1 where city='pune');
+------+--------+-------+--------+
| id   | name   | city  | salary |
+------+--------+-------+--------+
|    1 | Ramesh | Delhi |   2000 |
+------+--------+-------+--------+
1 row in set (0.00 sec)

mysql> Select * from emp2 where salary<=some(select salary from
    -> emp1 where city='pune');
+------+--------+-------+--------+
| id   | name   | city  | salary |
+------+--------+-------+--------+
|    3 | komal  | pune  |  10000 |
|    2 | kajal  | pune  |  10000 |
|    1 | Ramesh | Delhi |   2000 |
+------+--------+-------+--------+
3 rows in set (0.00 sec)

mysql> Select * from emp2 where salary>=some(select salary from
    -> emp1 where city='pune');
+------+-------+------+--------+
| id   | name  | city | salary |
+------+-------+------+--------+
|    3 | komal | pune |  10000 |
|    2 | kajal | pune |  10000 |
+------+-------+------+--------+
2 rows in set (0.00 sec)

mysql>
```


All

ALL is used to select all records of a SELECT STATEMENT. It compares a value to every value in a list or results from a query. The ALL must be preceded by the comparison operators and evaluates to TRUE if the query returns no rows. For example, ALL means greater than every value, means greater than the maximum value. Suppose ALL (1, 2, 3) means greater than 3.

Syntax

```
SELECT [column_name... | expression1 ]
FROM [table_name]
WHERE expression2 comparison_operator {ALL | ANY | SOME} ( subquery )
```

```
mysql> Select * from emp2 where salary<all(select salary from
    -> emp1 where city='pune');
+------+--------+-------+--------+
| id   | name   | city  | salary |
+------+--------+-------+--------+
|    1 | Ramesh | Delhi |   2000 |
+------+--------+-------+--------+
1 row in set (0.01 sec)

mysql> Select * from emp2 where salary<=all(select salary from
    -> emp1 where city='pune');
+------+--------+-------+--------+
| id   | name   | city  | salary |
+------+--------+-------+--------+
|    3 | komal  | pune  |  10000 |
|    2 | kajal  | pune  |  10000 |
|    1 | Ramesh | Delhi |   2000 |
+------+--------+-------+--------+
3 rows in set (0.00 sec)

mysql> Select * from emp2 where salary>=all(select salary from
    -> emp1 where city='pune');
+------+-------+------+--------+
| id   | name  | city | salary |
+------+-------+------+--------+
|    3 | komal | pune |  10000 |
|    2 | kajal | pune |  10000 |
+------+-------+------+--------+
2 rows in set (0.00 sec)

mysql>
```


7. PL/SQL Basics

(a) Declaring Variables

Question : Displaying sum of two numbers

```
Set serveroutput on;
declare
v_x number(5);
v_y number(5);
v_sum number(5);
BEGIN
v_x := 10;
v_y := 20;
v_sum := v_x+ v_ y;
dbms_output.put_line('Sum is'|| v_sum);
end;
/
---------------------------------------------------------------------------
----------------
```

OUTPUT

```
Sum is 30
P/L SQL procedure successfully completed.
```

(b) Writing Executable Statements

Question : Displaying salary of a specific employee

```
Set serveroutput on;
declare
var_sal number(6);
var_emp_no number(4);
BEGIN
var_emp_no:='7369';
SELECT sal
INTO var_sal
FROM emp
WHERE empno = var_emp_no;
dbms_output.put_line(var_sal);
end;
/
---------------------------------------------------------------------------
---------------
```

OUTPUT

```
800
PL/SQL procedure successfully completed.
```

(c) Interacting with the Oracle Server

```
set serveroutput on;
declare
var_x number(5);
var_ y number(5);
var_sum number(10);
var_avg number(10);
begin
var_x:=&var_x;
var_y:=&var_y;
dbms_output.put_line('Sum ='|| var_x+var_y);
dbms_output.put_line('average ='|| var_x/var_y);
end;
/
---------------------------------------------------------------------------
```

OUTPUT

```
Enter value for var_x: 2
old   7: var_x:=&var_x;
new   7: var_x:=2;
Enter value for var_y: 2
old   8: var_y:=&var_y;
Sum=4
average=2
PL/SQL procedure successfully completed.
```

(d) Writing Control Structures

(i) Using if Statement

```
Set serveroutput on;
DECLARE
str1 VARCHAR(12);
srt2 VARCHAR(12);
BEGIN
str1 := 'TECHMAX';
str2 := 'TECHMAX';
IF str1 LIKE str2 THEN
DBMS_OUTPUT.PUT_LINE(str1 || 'is same like ' || str2);
END IF;
END;
/
```

OUTPUT:

```
TECHMAX is same like TECHMAX
PL/SQL procedure successfully completed.

----------
```

(ii) Using if –then-else statements

```
Set serveroutput on;
declare
x number(10);
y number(10);
begin
```

```
x:=7782;
select sal into y from emp where empno=x;
if y>4500
then update emp set sal=(sal+2500) where empno=x;
elsif y>3500
then
update emp set sal=(sal+1500) where empno=x;
else
update emp set sal=(sal+500) where empno=x;
end if;
end;
/
```

OUTPUT

```
SQL> select * from emp where empno=7782;

    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
--------- ---------- --------- --------- --------- --------- ----------
     7782 CLARK      MANAGER          7839 09-JUN-81    5348         10

SQL> select * from emp where empno=7782;

    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
--------- ---------- --------- --------- --------- --------- ----------
     7782 CLARK      MANAGER          7839 09-JUN-81    5159         10

PL/SQL procedure successfully completed.
```

(iii) Nested if –then-else statements

```
Set serveroutput on;
DECLARE
percent NUMERIC;
BEGIN
percent := '83';
IF percent >= 75 THEN
DBMS_OUTPUT.PUT_LINE('DISINCTION');
ELSIF percent >= 60 AND percent <75 THEN
DBMS_OUTPUT.PUT_LINE('FIRST CLASS');
ELSIF percent >= 50 AND percent<60 THEN
```

```
DBMS_OUTPUT.PUT_LINE('SECOND CLASS ');
ELSIF percent >= '40' AND percent <50 THEN
DBMS_OUTPUT.PUT_LINE('PASS CLASS ');
ELSE
DBMS_OUTPUT.PUT_LINE('FAIL..... ');
END IF;
END;
/
```

OUTPUT

```
DISINCTION

PL/SQL procedure successfully completed.
```

(iv) Using CASE - WHEN statement

```
SQL>set serveroutput on;
DECLARE
grade CHAR(1);
BEGIN
grade := 'A';
CASE grade
WHEN 'O' THEN DBMS_OUTPUT.PUT_LINE('Outstanding');
WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Good');
WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Satisfactory');
WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Fail');
ELSE DBMS_OUTPUT.PUT_LINE('Invalid grade');
END CASE;
END;
/
```

OUTPUT

```
Excellent

PL/SQL procedure successfully completed.
```

(v) Using While Loop

```
SQL>set serveroutput on;
DECLARE
```

```
grade CHAR(1);
BEGIN
grade := 'A';
CASE grade
WHEN 'O' THEN DBMS_OUTPUT.PUT_LINE('Outstanding');
WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Good');
WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Satisfactory');
WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Fail');
ELSE DBMS_OUTPUT.PUT_LINE('Invalid grade');
END CASE;
END;
/
```

OUTPUT

```
Excellent

PL/SQL procedure successfully completed.
```

(vi) Using For Loop

```
Set serveroutput on;
DECLARE
ans NUMBER(4);
BEGIN
ans := 5;
FOR num IN 1..10 LOOP
DBMS_OUTPUT.PUT_LINE('5 x' || num ||'='|| ans);
ans := ans + 5;
END LOOP;
END;
/
```

OUTPUT

```
5 x1=5
5 x2=10
5 x3=15
5 x4=20
5 x5=25
5 x6=30
```

```
5 x7=35
5 x8=40
5 x9=45
5 x10=50

PL/SQL procedure successfully completed.
```

(vii) Using Continue –When in Loop

```
SQL> BEGIN
FOR num IN 1 .. 10 LOOP
CONTINUE WHEN MOD(num,2) != 0;
DBMS_OUTPUT.PUT_LINE(num);
END LOOP;
END;
/
```

OUTPUT

```
2
4
6
8
10

PL/SQL procedure successfully completed.
```

8. Composite data types, cursors and exceptions.

(a) Working with Composite Data Types

```
SQL> DECLARE
TYPE t_type IS TABLE OF NUMBER(10)
INDEX BY BINARY_INTEGER;
var_tabt_type;
var_indx NUMBER;
BEGIN
<< loading >>
FOR i IN 1 .. 7 LOOP
var_tab(i) := i;
END LOOP loading;
var_tab.DELETE(5);
```

```
var_indx := var_tab.FIRST;
<< displaying >>
 WHILE var_indx IS NOT NULL LOOP
DBMS_OUTPUT.PUT_LINE('Displaying....' || var_tab(var_indx));
var_indx := var_tab.NEXT(var_indx);
END LOOP displaying;
END;
 /
```

OUTPUT

```
Displaying....1
Displaying....2
Displaying....3
Displaying....4
Displaying....6
Displaying....7
PL/SQL procedure successfully completed.
```

(b) Writing Explicit Cursors

(i) Using explicit cursors

```
SQL> DECLARE
var_emp_no char(4);
var_emp_namevarchar(10);
var_emp_sal number(8,2);
var_emp_addrvarchar(10);
cursor c_emp is
select emp_no, emp_name, emp_sal, emp_addr from emp;
BEGIN
open c_emp;
loop
fetch c_emp into var_emp_no,var_emp_name,var_emp_sal,var_emp_addr;
exit when c_emp%notfound;
var_new_sal := var_emp_sal +5000;
insert into emp
values(var_emp_no,var_emp_name,var_new_sal,var_emp_addr);
end loop;
close c_emp;
END;
/
```

OUTPUT

```
SQL> select * from emp;

    EMPNO   ENAME    SAL       ADDR
--------- ---------- --------- ---------- --------- ---
     4012      CLARK       9000    MUMBAI

     PL/SQL procedure successfully completed.
```

(ii) Using loops in cursor

```
DECLARE
cursor c1 is select * from emp where sal>2500 and job!='president';
x emp%rowtype;
BEGIN
open c1;
LOOP
fetch c1 into x;
DBMS_OUTPUT.PUT_LINE(x.ename);
exit when(c1%notfound);
if(x.job='managers' and x.sal>2500)
then update emp set sal=sal+50;
elsif(x.job='analyst' and x.sal>2500)
then update emp set sal=sal+70;
end if;
end loop;
close c1;
END;
/
```

OUTPUT

```
JONES
BLAKE
SCOTT
KING
FORD
FORD

PL/SQL procedure successfully completed.
```

(c) Handling Exceptions

```
SQL> CREATE OR REPLACE PROCEDURE new_dept (d_no IN
CHAR, d_name IN VARCHAR2)
IS
BEGIN
INSERT INTO department (dept_no, dept_name) VALUES ( d_no, d_name );
EXCEPTION
WHEN DUP_VAL_ON_INDEX THEN
raise_application_error (-25001,'Department No. already existing');
WHEN OTHERS THEN
raise_application_error (-25002,'Error inserting new Department.');
END;
/
```

OUTPUT

Case 1

```
exec new_dept;
Department No. already existing.
```

Case 2

```
Exec new _dept;
Error inserting new department.
```

9. Procedures and Functions

(a) Creating Procedures

Question 1 : Create a procedure that adds the details(empno,ename,job,sal) of newest employee from table Emp into a table newperson having structure as (eno,firstname,lastname,design,salary).The last name of a person is to be passed to the procedure as read only value.

```
Create or replace procedure newperson(lname varchar2)
As
Id emp.empno%type;
Fnameemp.ename%type;
Des emp.job%type;
Salary emp.sal%type;
Begin
Select empno,ename,job,sal into id,fname,des,salary
From emp
```

```
Where empno=(select max(empno) from emp);
Insert into person(eno,firstname,lastname,design,salary)
Values(id,fname,lname,des,salary);
End newperson;
/
```

OUTPUT

```
Exec newperson;
Select * from person;

   ENO  FIRSTNAME   LASTNAME     DESIGN .   SALARY
--------- ---------- --------- ---------- --------- ----------

    4012     JHONNY    DEP          HEAD        4023
```

Question 2 : Create a stored procedure namely raise that receives the employee number and raise in salary as parameter. It then raises the salary of that employee in the table emp.

```
Create or replace procedure raise_emp(emp1  integer,increase  float)
Is
Currentsalary float;
Salary exception;
Begin
Select sal into currentsalary from emp where empno=emp1;
If currentsalary is null then
Raise  salary_exception;
Else
Update emp
Set sal=sal+increase
Where empno=emp1;
End if;
Exception
When no_data_found then
Insert into audit Values(emp1,'unknow employee');
When salary_exception then
Insert into audit values(emp1,'nullsalary');
End;
/
```

OUTPUT

```
Exec raise;
Select * from emp;
   EMPNO    ENAME     SAL      ADDR
---------- ---- ------ --------- ---------- --------- --------- ------------------------------
    4012      JAY      13000    NEW YORK

Select * from audit;

   EMPNO    SAL
---------- ---------- ------
    4012     nullsalary
```

Question 3 : Create a stored procedure that provides the details of customername and city from table customers having structure (cid,name,city,status,credit).To obtain these details the customer number is passed to the procedure.

```
Create table cust01(cidinteger,name varchar2(20),status varchar2(1),credit integer);

Insert into cust01 values(101,'aa','s1',1000);
Insert into cust01 values(102,'bb','s2',1001);
Insert into cust 01values(103,'cc','s3',1002);

Create or replace procedure getcustomer
( custid in customer.cid%type,
Custname out cutomer.cname%type,
Custcity out customer.city%type)
Is
Begin
Select cname,city into custname,custcity
From customers
Where cid=custid;
End getcustomer;
```

OUTPUT

```
exec getcustomer;

CNAME          CITY
-----------------------
GOHIL          LAS VEGAS
```

(b) Creating Functions

Question : Create a function that return doubled contributed amount (contamt) from table donation if it is more than 10 o.w. it returns tripled contributed amount for a given id no.

```
Create or replace function func_cal1
(vidnodonation.idno%type) return number
Is
vamtdonation.contamt%type;
vreturndonation.contamt%type;
Begin
Select contamt into vamt
From donation
Where idno=vidno;
If vamt>10 then
return vreturn;
else
vreturn := vamt *3;
return vreturn;
end if;
end func_calc1;
```

OUTPUT

```
Function created.
Case 1:
2000

Case 2:
3000
```

(c) Managing Subprograms

```
Declare
X number:=2;
procedure abc is
Begin
Insert into temp values(1,'in abc');
End;
Procedure pqr is
Begin
Insert into temp  values(2,'in pqr');
End;
```

```
Begin
Insert into temp values(x,'in main');
abc;
pqr;
End;
/
```

(d) Creating Packages

```
Create or replace package emp_salas
Procedure find_sal(e_no number);


CREATE OR REPLACE PACKAGE BODY emp_sal AS
PROCEDURE find_sal(e_no number) IS
e_sal number(10);
BEGIN
SELECT emp_sal INTO e_sal
FROM customers
WHERE emp_no = e_no;
dbms_output.put_line('Salary: '|| e_sal);
END find_sal;
END emp_sal;
/
```


OUTPUT

```
Package body created.
```

10. Creating Database Triggers

(i) Trigger

```
create or replace trigger xyz
before insert on file_A
for each row
when(new.fir!=10)
begin
insert into file_B values(:new.fir,'record');
end;
/
```

OUTPUT

```
Insert into file_ A  values(12,'hello');
```

(ii) Trigger

```
create or replace trigger upd_trig
before update of sal on emp
for each row
begin
if :new.sal<:old.sal then
raise_application_error(-20001,'salary can not be reduced');
end if;
end;
/
```

OUTPUT

```
First of all create a copy of emp table and then use this query
Update emp set sal=sal-1000 where sal>4000;
```

(iii) Trigger

```
create or replace trigger psal
before delete or insert or update on emp
for each row
when(new.empno>0)
declare saldiff number(8,2);
begin
saldiff:= :new.sal-:old.sal;
dbms_output.put_line(:old.sal);
dbms_output.put_line(:new.sal);
dbms_output.put_line(:new.sal-:old.sal);
end;
/
```

OUTPUT

```
Update emp set sal=sal-1000 where sal>4000;
Pl/SQL Procedure Successfully  executed
```