SQL

1. Table Name: PERSONS

P_ID	LASTNAME	FIRSTNAME	ADDRESS	CITY
1	HANSEN	OLA	TIMOTEIVN 10	SANDNES
2	SVENDSON	TOVE	BORGVN23	SANDNES
3	PETTERSEN	KARI	STORGT20	STAVANGER

a. Write a SQL statement to create a table Persons and insert the values.
b. To select the content of the columns named LASTNAME AND FIRSTNAME from the PERSONS table.
c. To select the all columns from the PERSONS table.
d. To select only the persons living in the city "SANDNES" from the table PERSONS.
e. To select only PERSONS with first name "TOVE" And last name "SVENDSON"
f. To select only Persons with first name "TOVE" OR "OLA".

2. Table Name: EMPLOYEE

E_ID	SALARY	BONUS
101	2000	200
102	3000	150
103	1000	100
104	2050	125
105	3500	200

a. To find total salary + Bonus from Employee table.
b. Using select statement Subtraction (-) operator used in SALARY & BONUS table.

3. Answer the a) & b) question on the basic of the following tables SHOPPE and ACCESSORIES.

TABLE: SHOPPE

ID	SNAME	AREA
S01	ABC COMPUTERONICS	CP
S02	ALL INFOTECH MEDIA	GK II
S03	TECH SHOPPE	CP
S04	GEEKS TECNO SOFT	Nehru Place
S05	HITECH TECH STORE	Nehru Place

TABLE: ACCESSORIES

NO	NAME	PRICE	ID
A01	Mother Board	12000	S01
A02	Hard Disk	5000	S01
A03	Keyboard	500	S02
A04	Mouse	300	S01
A05	Mother Board	13000	S02
A06	Keyboard	400	S03

A07	LCD	6000	S04
A08	LCD	5500	S05
A09	Mouse	350	S05
A10	Hard Disk	4500	S03

a) Write the SQL queries.

i) To display Name & Price of all the Accessories in ascending order of their price.

ii) To display ID AND SNAME of all Shoppe located in Nehru Place.

iii)To display Minimum and Maximum Price of each Name of Accessories.

iv) To display Name, Price of all Accessories and their respective SNAME, Where they are available.

b) Write the output of the following SQL commands.

i) SELECT DISTINCT NAME FROM ACCESSORIES WHERE PRICE>=5000;

ii) SELECT AREA, COUNT (*) FROM SHOPPE GROUP BY AREA;

iii) SELECT COUNT (DISTINCT AREA) FROM SHOPPE;

iv) SELECT NAME, PRICE*0.05 DISCOUNT FROM ACCESSORIES WHERE ID IN('S02', 'S03');

4. TABLE: ITEMS

CODE	INAME	QTY	PRICE	COMPANY	TCODE
1001	DIGITAL PAD 121	120	11000	XENITA	T01
1006	LED SCREEN 40	70	38000	SANTORA	T02
1004	CAR GPS SYSTEM	50	2150	GEOKNOW	T01
1003	DIGITAL CAMERA 12X	160	8000	DIGICLICK	T02
1005	PEN DRIVE 32 GB	600	1200	STOREHOME	T03

TABLE: TRADERS

TCODE	TNAME	CITY
T01	ELECTRONIC SALES	MUMBAI
T03	BUSY STORE CORP	DELHI
T02	DISP HOUSE INC	CHENNAI

a) To display the details of all the items in ascending order of item names (i.e. INAME)

b) To display item name and price of all those items, whose price is in the range of 10000 and 22000 (both values inclusive)

c) To display the number of items, which are traded by each trader. The expected output of this query should be:

T01	2
T02	2
T03	1

d) To display the price, item name and quantity of those items, which have quantity more than 150.

e) To display the names of those traders who are either from DELHI or from MUMBAI.

f) To display the name of companies and the name of the items in descending order of company names.

g) The expected output of these query should be:

i)

MAX(price)	MIN(price
38000	1200

ii)

AMOUNT
107500

iii)

TCODE
T01
T02
T03

iv)

INAME	TNAME
CAR GPS SYSTEM	ELECTRONIC SALES
LED SCREEN 40	DISP HOUSE INC

5. TABLE NAME: APPLICANTS

NO	NAME	FEE	GENDER	C_ID	JOINYEAR
1012	AMANDEEP	30000	M	A01	2012
1102	AVISHA	25000	F	A02	2009
1103	EKANT	30000	M	A02	2011
1049	ARUN	30000	M	A03	2009
1025	AMBER	40000	M	A02	2011
1106	ELA	40000	F	A05	2010
1017	NIKITA	35000	F	A03	2012
1108	ARLUNA	30000	F	A03	2012
2109	SHAKTI	35000	M	A04	2011
1101	KIRAT	25000	M	A01	2012

TABLE NAME: COURSES

C_ID	COURSE
A01	FASHION DESIGN
A02	NETWORKING
A03	HOTEL MANAGEMENT
A04	EVENT MANAGEMENT
A05	OFFICE MANAGEMENT

a) To display NAME, FEE, GENDER, JOINYEAR about the APPLICANTS, who have joined before 2010.

b) To display the names of applicants, who are paying FEE more than 30000.

c) To display the names of all applicants in ascending order of their JOINYEAR.

d) To display the year and the total number of applicants joined in each year from the table APPLICANTS.

e) To display the C_ID and the number of applicants registered in the course from the APPLICANTS table.

f) To display the applicant's name with their respective courses name from the tables applicants and courses.

g) Give the SQL statements of the outputs:

i)

NAME	JOINYEAR
AVISHA	2009

ii)

MIN(JOINYEAR)
2009

iii)

AVG(FEE)
31666.666

iv)

SUM(FEE)	C_ID
55000	A01

NOTES:

The SQL DELETE Statement:

The DELETE statement is used to delete existing records in a table.

Syntax: DELETE FROM table_name WHERE condition;

The SQL WHERE Clause

The WHERE clause is used to filter records. The WHERE clause is used to extract only those records that fulfill a specified condition.

Syntax:

SELECT column1, column2, ...
FROM table_name
WHERE condition;

The SQL SELECT DISTINCT Statement: The SELECT DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

Syntax:

SELECT DISTINCT column1, column2, ...
FROM table_name;

The SQL SELECT Statement: The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

Syntax: SELECT * FROM table_name;

SQL

6. TABLE: CUSTOMER

CID	CNAME	GENDER	SID	AREA
1001	R SHARMA	FEMALE	101	NORTH
1002	M R TIWARY	MALE	102	SOUTH
1003	M K KHAN	MALE	103	EAST
1004	A K SINGH	MALE	102	EAST
1005	S SEN	FEMALE	101	WEST
1006	R DUBEY	MALE	104	NORTH
1007	M AGARWAL	FEMALE	104	NORTH
1008	S DAS	FEMALE	103	SOUTH
1009	R K PATIL	MALE	102	NORTH
1010	N KRISHNA MURTY	MALE	102	SOUTH

TABLE: ONLINESHOP

SID	SHOP
101	MY BUY
102	ECO BUY
103	JUST SHOPPING
104	SHOPPING EASY

a) To display CNAME, AREA of all female customers from CUSTOMER table.

b) To display the details of all the CUSTOMERS in ascending order of CNAME within SID.

c) To display the total number of customers for each AREA from CUSTOMER table.

d) To display CNAME and CORRESPONDING SHOP from CUSTOMER table and ONLINESHOP table.

e) Write the code to see the output:

COUNT(*)	GENDER
4	FEMALE
6	MALE

f) Write the code to see the output:

COUNT(*)
4

g) SELECT CNAME FROM CUSTOMER WHERE CNAME LIKE ‘L%’; [Write the output]

h) SELECT DISTINCT AREA FROM CUSTOMER; [Write the output]

NOTES

UPDATE: The UPDATE statement is used to modify the existing records in a table.
Syntax: UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;

AND, OR and NOT Operators: The WHERE clause can be combined with AND, OR, and NOT operators.

The AND & OR operators are used to filter records based on more than one condition:
The AND operator displays a record if all the conditions separated by AND are TRUE.
The OR operator displays a record if any of the conditions separated by OR is TRUE.
The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax:

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

OR Syntax:

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

NOT Syntax:

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

7. TABLE: CARDEN

CCODE	CARNAME	MAKE	COLOR	CAPACITY	CHARGES
501	A-STAR	SUZUKI	RED	3	14
503	INDIGO	TATA	SILVER	3	12
502	INNOVA	TOYOTA	WHITE	7	15
509	SX4	SUZUKI	SILVER	4	14
510	C-CLASS	MERECEDES	RED	4	35

TABLE: CUSTOMER

CODE	CNAME	CCODE
1001	HAMANT SAHU	501
1002	RAJ LAL	509
1003	FEROZA SHAH	503
1004	KETAN DHAL	502

a)
i) To display the name of all the SILVER colored cars.
ii) To display name of Car, Make and sitting Capacity of cars in descending order of their sitting CAPACITY.
iii) To display the highest charges at which a vehicle can be hired from CARDEN.
iv) To display the CUSTOMER name and the corresponding name of the CARS hired by them.
b) Give the output of following SQL queries:
i) SELECT COUNT(DISTINCT MAKE) FROM CARDEN;
ii) SELECT MAX(CHARGES), MIN(CHARGES) FROM CARDEN;
iii) SELECT COUNT(*) MAKE FROM CARDEN;
iv) SELECT CARNAME FROM CARDEN WHERE CAPACITY=4;

SQL

SQL RENAME TABLE: SQL RENAME TABLE syntax is used to change the name of a table. Sometimes, we choose non-meaningful name for the table. So it is required to be changed.
Syntax: ALTER TABLE table_name RENAME TO new_table_name;

SQL COPY TABLE: If you want to copy a SQL table into another table in the same SQL server database, it is possible by using the select statement.
Syntax: SELECT * INTO <destination_table> FROM <source_table> ;

SQL ALTER TABLE DROP Column Syntax:
ALTER TABLE table_name DROP COLUMN column_name;

SQL ALTER TABLE Add Column Syntax:
ALTER TABLE table_name ADD column_name datatype;

SQL SELECT FIRST: The SQL first() function is used to return the first value of the selected column. Syntax: SELECT FIRST(column_name) FROM table_name;[The SELECT FIRST statement is only supported by MS Access]

SQL SELECT LAST: The last() function is used to return the last value of the specified column. Syntax: SELECT LAST (column_name) FROM table_name;[;[The SELECT LAST statement is only supported by MS Access]

SQL SELECT SUM Syntax: SELECT SUM (expression) FROM tables WHERE conditions;

8. Create a table to store information about weather observation stations. [No duplicate ID fields allowed]

```
CREATE TABLE STATION
(ID INTEGER PRIMARY KEY,
CITY CHAR(20),
STATE CHAR(2),
LAT_N REAL,
LONG_W REAL);
```

Populate the table STATION with a few rows:

```
INSERT INTO STATION VALUES (13, 'Phoenix', 'AZ', 33, 112);
INSERT INTO STATION VALUES (44, 'Denver', 'CO', 40, 105);
INSERT INTO STATION VALUES (66, 'Caribou', 'ME', 47, 68);
```

1.i) Display the all stations.
ii) Query to select Northern stations (Northern latitude > 39.7).

iii) Query to select only ID, CITY, and STATE columns.

2. Create another table to store normalized temperature and precipitation data:

```
-- ID field must match some STATION table ID(so name and location will be known).
-- Allowable ranges will be enforced for other values.
-- No duplicate ID and MONTH combinations.
-- Temperature is in degrees Fahrenheit.
-- Rainfall is in inches.
CREATE TABLE STATS
(ID INTEGER REFERENCES STATION(ID),
MONTH INTEGER CHECK (MONTH BETWEEN 1 AND 12),
TEMP_F REAL CHECK (TEMP_F BETWEEN -80 AND 150),
RAIN_I REAL CHECK (RAIN_I BETWEEN 0 AND 100),
PRIMARY KEY (ID, MONTH));

INSERT INTO STATS VALUES (13, 1, 57.4, 0.31);
INSERT INTO STATS VALUES (13, 7, 91.7, 5.15);
INSERT INTO STATS VALUES (44, 1, 27.3, 0.18);
INSERT INTO STATS VALUES (44, 7, 74.8, 2.11);
INSERT INTO STATS VALUES (66, 1, 6.7, 2.10);
INSERT INTO STATS VALUES (66, 7, 65.8, 4.52);
```

i) Display the all values of STATS table.

ii) Query to look at table STATS, picking up location information by joining with table STATION on the ID column.

iii) Query to look at the table STATS, ordered by month and greatest rainfall, with columns rearranged.

iv) Query to look at temperatures for July from table STATS, lowest temperatures first, picking up city name and latitude by joining with table STATION on the ID column.

v) Query to show MAX and MIN temperatures as well as average rainfall for each station.

vi) Query (with sub query) to show stations with year-round average temperature above 50 degrees.

vii) Create a view (derived table or persistent query) to convert Fahrenheit to Celsius and inches to centimeters.

viii) Query to look at table STATS in a metric light (through the new view).

ix) Another metric query restricted to January below-freezing (0 Celsius) data, sorted on rainfall.

x) Update all rows of table STATS to compensate for faulty rain gauges known to read 0.01 inches low.

SQL

Create a database using the CREATE DATABASE command:

CREATE DATABASE database_name;

Insert data multiple rows, use a comma to separate each row, like this:

INSERT INTO table_name
VALUES
(value_1, value_2, value_3),
(value_1, value_2, value_3),
(value_1, value_2, value_3),
(value_1, value_2, value_3);

SQL PRIMARY KEY: Primary keys must contain UNIQUE values, and cannot contain NULL values.

SQL PRIMARY KEY on CREATE TABLE EXMAPLE:

MySQL:	SQL Server / Oracle / MS Access
CREATE TABLE Persons (ID int NOT NULL, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int, PRIMARY KEY (ID));	CREATE TABLE Persons (ID int NOT NULL PRIMARY KEY, LastName varchar(255) NOT NULL, FirstName varchar(255), Age int);

9. Create table Worker.

WORKER_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
001	NIHARIKA	ARORA	20000	2013-02-25 09:00:00	HR
002	AYUSHI	GURONDIA	5000	2015-02-10 09:00:00	ADMIN
003	PRIYANSHA	CHOUKSEY	25000	2014-05-16 09:00:00	HR
004	APARNA	DESHPANDE	8000	2016-12-20 09:00:00	ADMIN
005	SHAFALI	JAIN	21000	2015-08-29 09:00:00	ADMIN
006	SUCHITA	JOSHI	20000	2017-02-12 09:00:00	ACCOUNT
007	SHUBHI	MISHRA	15000	2018-03-23 09:00:00	ADMIN
008	DEVYANI	PATIDAR	18000	2014-05-02 09:00:00	ACCOUNT

SQL

1. i) Query to create the Table

```
CREATE TABLE Worker (WORKER_ID INT NOT NULL PRIMARY KEY
AUTO_INCREMENT, FIRST_NAME CHAR(25), LAST_NAME CHAR(25),SALARY
INT(15),JOINING_DATE DATETIME,DEPARTMENT CHAR(25));
```

ii) Query to insert values into the Table Worker.

iii) Create table **Bonus**

WORKER_REF_ID	BONUS_DATE	BONUS_AMOUNT
1	2015-04-20 00:00:00	5000
2	2015-08-11 00:00:00	3000
3	2015-04-20 00:00:00	4000
1	2015-04-20 00:00:00	4500
2	2015-08-11 00:00:00	3500

```
CREATE TABLE Bonus( WORKER_REF_ID INT, BONUS_DATE DATETIME,
BONUS_AMOUNT INT(10), FOREIGN KEY (WORKER_REF_ID) REFERENCES
WORKER(WORKER_ID) ON DELETE CASCADE);
```

iv)Query to insert values into table Bonus.

v) Create table **Title**

WORKER_REF_ID	WORKER_TITLE	AFFECTED_FROM
1	Manager	2016-02-20 00:00:00
2	Executive	2016-06-11 00:00:00
8	Executive	2016-06-11 00:00:00
5	Manager	2016-06-11 00:00:00
4	Asst. Manager	2016-06-11 00:00:00
7	Executive	2016-06-11 00:00:00
6	Lead	2016-06-11 00:00:00
3	Lead	2016-06-11 00:00:00

```
CREATE TABLE Title (WORKER_REF_ID INT, WORKER_TITLE CHAR(25),
AFFECTED_FROM DATETIME, FOREIGN KEY (WORKER_REF_ID) REFERENCES
Worker(WORKER_ID) ON DELETE CASCADE);
```

vi) Query to insert values into table Title.

2. i) Write an SQL query for fetching “FIRST_NAME” from the WORKER table using <WORKER_NAME> as alias.

ii) What is an SQL Query for fetching the “FIRST_NAME” from WORKER table in upper case?

iii) What is an SQL query for fetching the unique values of the column DEPARTMENT from the WORKER table?

iv) Write an SQL query for printing the first three characters of the column FIRST_NAME.

v) What is an SQL query for finding the position of the alphabet (‘A’) in the FIRST_NAME column of Ayushi.

vi) What is an SQL Query for printing the FIRST_NAME from Worker Table after the removal of white spaces from right side.

vii) Write an SQL Query for printing the DEPARTMENT from Worker Table after the removal of white spaces from the left side.
viii) What is an SQL query for fetching the unique values from the DEPARTMENT column and thus printing is the length?
ix) Write an SQL query for printing the FIRST_NAME after replacing ‘A’ with ‘a’.
x) What is an SQL query for printing the FIRST_NAME and LAST_NAME into a column named COMPLETE_NAME? (A space char should be used)
xi) What is an SQL query for printing all details of the worker table which ordered by FIRST_NAME ascending?
xii) Write an SQL query for printing the all details of the worker table which ordered by FIRST_NAME ascending and the DEPARTMENT in descending
xiii) What is an SQL query to print the details of the workers ‘NIHARIKA’ and ‘PRIYANSHA’.
xiv) What is an SQL query printing all details of workers excluding the first names of ‘NIHARIKA’ and ‘PRIYANSHA’
xv) Write an SQL query for printing the details of DEPARTMENT name as “Admin”.
xvi) What is an SQL query for printing the details of workers whose FIRST_NAME Contains ‘A’?
xvii) What is an SQL Query for printing the FIRST_NAME of workers whose name ends with ‘A’?
xviii) What is an SQL Query for printing the details of the workers whose FIRST_NAME ends with ‘H’ and contains six alphabets?
xix) Write an SQL Query for printing the details of workers whose SALARY lies between 10000 and 20000.
xx) Write an SQL Query for printing the details of workers who joined inFeb’2014
xxi) Write an SQL Query for fetching the count of workers in DEPARTMENT with ‘Admin’.
xxii) Write an SQL Query for fetching the details of workers with Salaries >= 5000 and <= 10000.
xxiii) What is an SQL Query for fetching the no. of workers in each department in descending order?
xxiv) What is an SQL Query for printing the details of workers who are also managers?
xxv) Write an SQL Query for fetching the details of duplicate records in some fields.
xxvi) What is an SQL Query for only showing odd rows?
xxvii) What is an SQL Query for only showing even rows?
xxviii) Write an SQL Query for cloning a new table from another table.
xxix) Write an SQL query to show the top n (say 10) records of a table.
xxx) Write an SQL query to determine the 5th highest salary from a table.

3.

i) Write an SQL query to fetch the list of employees with the same salary.
ii) Write an SQL query to show the second highest salary from a table.
iii) Write an SQL query to show one row twice in results from a table.
iv) Write an SQL query to fetch the first 50% records from a table.

v) Write an SQL query to fetch the departments that have less than five people in it.
vi) Write an SQL query to show the last record from a table.
vii) Write an SQL query to fetch the first row of a table.
viii) Write an SQL query to print the name of employees having the highest salary in each department.

Drop Column Syntax:

MYSQL	Oracle and SQL Server
ALTER TABLE "table_name" DROP "column_name";	ALTER TABLE "table_name" DROP COLUMN "column_name";

Rename Column Syntax:

MySQL	Oracle
ALTER TABLE "table_name" Change "column 1" "column 2" ["Data Type"];	ALTER TABLE "table_name" RENAME COLUMN "column 1" TO "column 2";

MYSQL NUMERIC FUNCTIONS

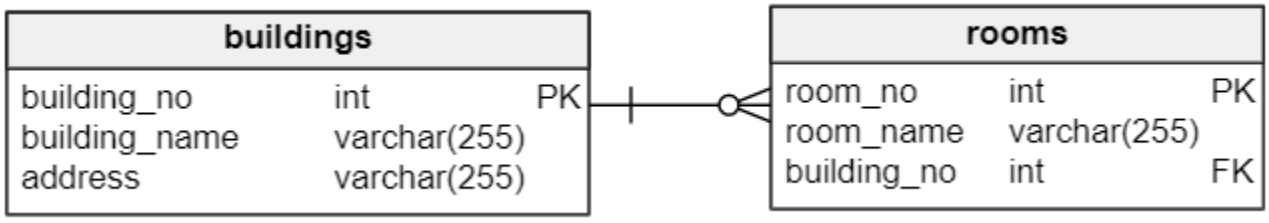
ABS() - SELECT ABS(-243.5);	CEIL() - SELECT CEIL(35.75);
ACOS() - SELECT ACOS(0.5);	CEILING() - SELECT CEILING(25.75);
ATAN() - SELECT ATAN(.25);	COS() - SELECT COS(1);
ATAN2()- SELECT ATAN2(0.4, 2);	COT() - SELECT COT(6);
DEGREES() - SELECT DEGREES(2.5);	GREATEST()- SELECT GREATEST(30, 120, 20, 81, 205);
EXP() - SELECT EXP(3);	FLOOR() - SELECT FLOOR(45.95);
DIV() - SELECT 200 DIV 2;	LEAST()-SELECT LEAST(8, 102, 74, 81, 275);
LN() (logarithm) -SELECT LN(10);	LOG()-SELECT LOG(4, 10);
LOG10() - SELECT LOG10(2);	MOD()-SELECT MOD(28, 3);
PI() - SELECT PI();	POW()-SELECT POW(5, 3);
POWER()-SELECT POWER(7, 3);	RADIANS() - SELECT RADIANS(180);
RAND()-SELECT RAND();	ROUND()-SELECT ROUND(135.375, 2);
SIGN() - SELECT SIGN(-78);	SQRT()-SELECT SQRT(25);
TAN() - SELECT TAN(90);	TRUNCATE()- SELECT TRUNCATE(7235.37589, 2);
ASIN() - SELECT ASIN(0.5);	CONV()- SELECT CONV('a',16,2);

10. Table: Student

ID	STD_ID	NAME	MARKS
1	3	ABHI	99
2	5	GEETHASRI	89
3	6	RAHIM	49
4	9	RAM	69
5	1	RAHUL	87
6	1	RAHUL	96
7	1	RAHUL	96
8	9	RAM	96
9	9	RAM	96

1. Query To Find Second Highest Marks Of A Student?
2. Query To Find Duplicate Rows In Table?
3. What Is The Query To Fetch First Record From Student Table?
4. What Is The Query To Fetch Last Record From The Student Table?
5. What Is Query to Display First 4 Records from Student Table?
6. What Is Query to Display Last 3 Records from Student Table?
7. What Is Query To Display Nth Record From Student Table?
8. How to Get 3 Highest Marks from Student Table?
9. How to Display Odd Rows in Student Table?
10. How to Display Even Rows in Student Table?
11. How Can I Create Table With Same Structure Of Student Table?
12. Select All Records from Student Table Whose Name Is ‘abhi’ and ‘geethasri’.

MySQL 5.7 ON DELETE CASCADE: Deleting Data from Multiple Related Tables:



1. Create the buildings table:

CREATE TABLE buildings (building_no INT PRIMARY KEY AUTO_INCREMENT, building_name VARCHAR(255) NOT NULL, address VARCHAR(255) NOT NULL);

2. Create the rooms table:

CREATE TABLE rooms (room_no INT PRIMARY KEY AUTO_INCREMENT, room_name VARCHAR(255) NOT NULL, building_no INT NOT NULL, FOREIGN KEY (building_no) REFERENCES buildings (building_no) ON DELETE CASCADE);

3. Insert data into the buildings table:

INSERT INTO buildings(building_name,address) VALUES('ACME Headquaters','3950 North 1st Street CA 95134'), ('ACME Sales','5000 North 1st Street CA 95134');

4. SELECT * FROM buildings;

5. Insert data into the rooms table:

INSERT INTO rooms(room_name,building_no) VALUES('Amazon',1),('War Room',1), ('Office of CEO',1),('Marketing',2),('Showroom',2);

6. SELECT * FROM rooms;

Delete the building with building no. 2:

DELETE FROM buildings WHERE building_no = 2;

SELECT * FROM rooms;

Note: ON DELETE CASCADE means that if the parent record is deleted, any child records are also deleted.

Primary Key: A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

FOREIGN KEY: A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

……..

11. TABLE: EMPLOYEE

ECODE	NAME	DESIGN	SGRADE	DOJ	DOB
101	ABDUL AHMED	EXECUTIVE	S03	23-MAR-2003	13-JAN-1980
102	RAVI CHANDER	HEAD-IT	S02	12-FEB-2010	22-JUL-1987
103	JOHN KEN	RECEPTIONIST	S03	24-JUN-2009	24-FEB-1983
105	NAZAR AMEEN	GM	S02	11-AUG-2006	03-MAR-1984
108	PRIYAM SEN	CEO	S01	29-DEC-2004	19-JAN-1982

[Notes: Aggregate Functions are all about: Performing calculations on multiple rows Of a single column of a table And returning a single value. The ISO standard defines five (5) aggregate functions namely; 1) COUNT 2) SUM 3) AVG 4) MIN 5) MAX]

SQL

TABLE: SALGRADE

SGRADE	SALARY	HRA
S01	56000	18000
S02	32000	12000
S03	24000	8000

a) i) To display the details of all the employee in descending order of DOJ.
ii) To display NAME and DESIGN of those Employees, whose SGRADE is either S02 or S03.
iii) To display the content of all the EMPLOYEEs, whose DOJ is between '09-FEB-2006' and '08-AUG-2009'.
iv) To add a new row in the EMPLOYEE table with the following:
109, 'HARISH ROY', 'HEAD-IT', 'S02', '09-SEP-2007', '21-APR-1983'.
b) Give the output of the following SQL queries:
i) SELECT COUNT(SGRADE), SGRADE FROM EMPLOYEE GROUP BY SGRADE;
ii) SELECT MIN(DOB), MAX(DOJ) FROM EMPLOYEE;
iii) SELECT NAME, SALARY FROM EMPLOYEE E, SALGRADE S WHERE E.SGRADE =S.SGRADE AND E.ECODE<103;
iv) SELECT SGRADE, SALARY + HRA FROM SALGRADE WHERE SGRADE='S02';

12. TABLE: STORE

ITEMNO	ITEM	SCODE	OTY	RATE	LASTBUY
2005	SHARPENER CLASSIC	23	60	8	31-JUN-09
2003	BALLS	22	50	25	01-FEB-10
2002	GEL PEN PREMIUM	21	150	12	24-FEB-10
2006	GEL PEN CLASSIC	21	250	20	11-MAR-09
2001	ERASER SMALL	22	220	6	19-JAN-09
2004	ERASER BIG	22	110	8	02-DEC-09
2009	BALL PEN 0.5	21	180	18	03-NOV-09

TABLE: SUPPLIERS

SCODES	SNAME
21	PREMIUM STATIONERS
23	SOFT PLASTICS
22	TERA SUPPLY

A) i) To display details of all the items in the store table in ascending order of Last-Buy.
ii) To display ItemNo and Item name of those items from STORE table, whose Rate is more than 15;
iii) To display the details of those items whose Supplier code (Scode) is 22 or Quantity in Store (Qty) is more than 110 from the table STORE.
iv) To display minimum rate of items for each Supplier individually as per Scode from the table STORE.

b) Give the output of the following SQL queries:

i) SELECT COUNT(DISTINCT SCODE) FROM STORE;

ii) SELECT RATE * QTY FROM STORE WHERE ITEMNO=2004;

iii) SELECT ITEM, SNAME FROM STORE S, SUPPLIERS P WHERE S.SCODE=P.SCODE AND ITEMNO=2006;

iv) SELECT MAX(LASTBUY) FROM STORE;

13. TABLE: STUDENT

SCODE	NAME	AGE	STRCDE	POINTS	GRADE
101	AMIT	16	1	6	NULL
102	ARJUN	13	3	4	NULL
103	ZAHEER	14	2	1	NULL
105	GAGAN	15	5	2	NULL
108	KUMAR	13	6	8	NULL
109	RAJESH	17	5	8	NULL
110	NAVEEN	13	3	9	NULL
113	AJAY	16	2	3	NULL
115	KAPIL	14	3	2	NULL
120	GURDEEP	15	2	6	NULL

TABLE: STREAM

STRCDE	STRNAME
1	SCIENCE +COMP
2	SCIENCE + BIO
3	SCIENCE + ECO
4	COMMERCE + MATHS
5	COMMERCE + SOCIO
6	ARTS + MATHS
7	ARTS + SOCIO

a) To display the name of stream in alphabetical order from table STREAM.

b) To display the number of students whose POINTS are more than 5.

c) To update GRADE to ‘A’ for all those students, who are getting more than 8 as POINTS.

d) ARTS+ MATHS stream is no more available. Make necessary change in table STREAM.

e) SELECT SUM(POINTS) FROM STUDENT WHERE AGE>14;

f) SELECT STRCDE, MAX(POINTS) FROM STUDENT GROUP BY STRCDE HAVING SCODE BETWEEN 105 AND 130;

g) SELECT AVG(AGE) FROM STUDENT WHERE SCODE IN(102,105,110,120);

h) SELECT COUNT(STRNAME) FROM STREAM WHERE STRNAME LIKE “SCI%”;

SQL

14. Consider the following tables GARMENT and FABRIC. Write SQL commands for the statements (a) to (d) and give outputs for SQL queries (e) to (h).

TABLE: GARMENT

GCODE	DESCRIPTION	PRICE	FCODE	READYDATE
10023	PENCIL SKIRT	1150	F03	19-DEC-08
10001	FORMAL SHIRT	1250	F01	12-JAN-08
10012	INFORMAL SHIRT	1550	F02	06-JUN-08
10024	BABY TOP	750	F03	07-APR-07
10090	TULIP SKIRT	850	F02	31-MAR-07
10019	EVENING GOWN	850	F03	06-JUN-08
10009	INFORMAL PANT	1500	F02	20-OCT-08
10007	FORMAL PANT	1350	F01	09-MAR-08
10020	FROCK	850	F04	09-SEP-07
10089	SLACKS	750	F03	20-OCT-08

TABLE: FABRIC

FCODE	TYPE
F04	POLYSTER
F02	COTTON
F03	SILK
F01	TERELENE

a) To display GCODE and DESCRIPTION of each GARMENT in descending order of GCODE.

b) To display the details of all the GARMENTs, which have READYDATE in between 08-DEC-07 and 16-JUN-08 (inclusive of both dates).

c) To display the average PRICE of all the GARMENTS, which are made up of FABRIC with FCODE as F03.

d)To display FABRIC wise highest and lowest price of GARMENTS from GARMENT table. (Display FCODE of each GARMENT along with highest and lowest price).

e) SELECT SUM(PRICE) FROM GARMENT WHERE FCODE='F01';

f) SELECT DESCRITION, TYPE FROM GARMENT, FABRIC WHERE GARMENT.FCODE =FABRIC.FCODE AND GARMENT.PRICE>=1260;

g) SELECT MAX(FCODE) FROM FABRIC;

h) SELECT COUNT (DISTINCT PRICE) FROM GARMENT;

15. TABLE: SENDER

SENDERID	SENDERNAME	SENDERADDRESS	SENDERCITY
ND01	R JAIN	2. ABC APPTS	NEW DELHI
MU02	H SINHA	12. NEWTOWN	MUMBAI
MU15	S JHA	27/A, PARK STREET	MUMBAI
ND50	T PRASAD	122-K,SDA	NEW DELHI

SQL

TABLE: RECIPIENT

RECID	SENDERID	RECNAME	RECADDRESS	RECCITY
KO05	ND01	R BAJPAYEE	5, CENTRAL AVENUE	KOLKATA
ND08	MU02	S MAHAJAN	116,A VIHAR	NEW DELHI
MU19	ND01	H SINGH	2A, ANDHERI EAST	MUMBAI
MU32	MU15	P K SWAMY	B5,C S TERMINUS	MUMBAI
ND48	ND50	S TRIPATHI	13, B1 D, MAYUR VIHAR	NEW DELHI

a) To display the names of all senders from MUMBAI.

b) To display the RECID, SENDERNAME, SENDERADDRESS, RECNAME, RECADDRESS for every RECIPIENT.

c) To display RECIPIENT details in ascending order of RECNAME.

d) To display number of Recipients from each city.

e) Write SQL code to see the output:

DISTINCT SENDERCITY
NEW DELHI
MUMBAI

f) Write SQL code to see the output:

SENDERNAME	RECNAME
R JAIN	H SINGH
S JHA	P K SWAMY

g) Write SQL code to see the output:

RECNAME	RECADDRESS
S MAHAJAN	116, A VIHAR
S TRIPATHI	13,B1D, MAYUR VIHAR

MySQL HAVING CLAUSE:

MySQL HAVING Clause is used with GROUP BY clause. It always returns the rows where condition is TRUE. Syntax: SELECT expression1, expression2, ... expression_n, aggregate_function (expression) FROM tables [WHERE conditions] GROUP BY expression1, expression2, ... expression_n HAVING condition;

MySQL SUBQUERIES:

A subquery is a query in a query. It is also called an inner query or a nested query. A subquery can be used anywhere an expression is allowed. It is a query expression enclosed in parentheses. Subqueries can be used with SELECT, INSERT, UPDATE, or DELETE statements.

```
mysql> SELECT * FROM Cars;
+----+------------+--------+
| Id | Name       | Cost   |
+----+------------+--------+
|  1 | Audi       |  52642 |
|  2 | Mercedes   |  57127 |
```

```
|  3 | Skoda      |   9000 |
|  4 | Volvo      |  29000 |
|  5 | Bentley    | 350000 |
|  6 | Citroen    |  21000 |
|  7 | Hummer     |  41400 |
|  8 | Volkswagen |  21600 |
+----+------------+--------+
```

```
mysql> SELECT * FROM Customers; SELECT * FROM Reservations;
+------------+-------------+
| CustomerId | Name        |
+------------+-------------+
|          1 | Paul Novak  |
|          2 | Terry Neils |
|          3 | Jack Fonda  |
|          4 | Tom Willis  |
+------------+-------------+
4 rows in set (0.00 sec)

+----+------------+------------+
| Id | CustomerId | Day        |
+----+------------+------------+
|  1 |          1 | 2009-11-22 |
|  2 |          2 | 2009-11-28 |
|  3 |          2 | 2009-11-29 |
|  4 |          1 | 2009-11-29 |
|  5 |          3 | 2009-12-02 |
+----+------------+------------+
5 rows in set (0.00 sec)
```

Subquery with the INSERT statement: To create a copy of the Cars table. Into another table called Cars2. We will create a subquery for this.

mysql> CREATE TABLE Cars2(Id INT NOT NULL PRIMARY KEY, Name VARCHAR(50) NOT NULL, Cost INT NOT NULL);

We create a new Cars2 table with the same columns and datatypes as the Cars table. To find out how a table was created, we can use the SHOW CREATE TABLE statement.

mysql> INSERT INTO Cars2 SELECT * FROM Cars;

This is a simple subquery. We insert all rows from the Cars table into the Cars2 table.

```
mysql> SELECT * FROM Cars2;
+----+------------+--------+
| Id | Name       | Cost   |
+----+------------+--------+
|  1 | Audi       |  52642 |
|  2 | Mercedes   |  57127 |
|  3 | Skoda      |   9000 |
|  4 | Volvo      |  29000 |
|  5 | Bentley    | 350000 |
|  6 | Citroen    |  21000 |
|  7 | Hummer     |  41400 |
|  8 | Volkswagen |  21600 |
+----+------------+--------+
```

The data was copied to a new Cars2 table.

Scalar subqueries: A scalar subquery returns a single value.

mysql> SELECT Name FROM Customers WHERE CustomerId=(SELECT CustomerId FROM Reservations WHERE Id=5);

```
+------------+
| Name       |
+------------+
| Jack Fonda |
+------------+
```

Table subqueries: A table subquery returns a result table of zero or more rows.

mysql> SELECT Name FROM Customers WHERE CustomerId IN (SELECT DISTINCT CustomerId FROM Reservations);

```
+-------------+
| Name        |
+-------------+
| Paul Novak  |
| Terry Neils |
| Jack Fonda  |
+-------------+
```

The above query returns the names of the customers, who made some reservations. The inner query returns customer Ids from the Reservations table. We use the IN predicate to select those names of customers, who have their CustomerId returned from the inner select query. The previous subquery can be rewritten using SQL join.

mysql> SELECT DISTINCT Name FROM Customers JOIN Reservations ON Customers.CustomerId=Reservations.CustomerId;

```
+-------------+
| Name        |
+-------------+
| Paul Novak  |
| Terry Neils |
| Jack Fonda  |
```

Correlated subqueries: A correlated subquery is a subquery that uses values from the outer query in its WHERE clause. The subquery is evaluated once for each row processed by the outer query.

mysql> SELECT Name FROM Cars WHERE Cost < (SELECT AVG(Cost) FROM Cars);

```
+------------+
| Name       |
+------------+
| Audi       |
| Mercedes   |
| Skoda      |
| Volvo      |
| Citroen    |
| Hummer     |
| Volkswagen |
+------------+
```

Subqueries with EXISTS, NOT EXISTS: If a subquery returns any values, then the predicate EXISTS returns TRUE, and NOT EXISTS FALSE.

mysql> SELECT Name FROM Customers WHERE EXISTS (SELECT * FROM Reservations WHERE Customers.CustomerId=Reservations.CustomerId);

```
+-------------+
| Name        |
+-------------+
| Paul Novak  |
| Terry Neils |
| Jack Fonda  |
+-------------+
```

In the above SQL statement we select all customers' names, which have an entry in the Reservations table.

mysql> SELECT Name FROM Customers WHERE NOT EXISTS (SELECT * FROM Reservations WHERE Customers.CustomerId=Reservations.CustomerId);

```
+------------+
| Name       |
+------------+
| Tom Willis |
+------------+
```

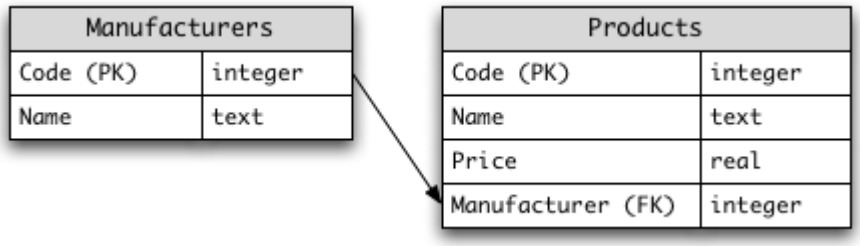
In this query, we return all customers that do not have an entry in the Reservations table. Both SQL queries are correlated queries.

SQL Aliases Notes: SQL aliases are used to give a table, or a column in a table, a temporary name. Aliases are often used to make column names more readable. An alias only exists for the duration of the query.

Alias Column Syntax: SELECT column_name AS alias_name FROM table_name;

Alias Table Syntax: SELECT column_name(s) FROM table_name AS alias_name;

16. TABLE SCHEMA FIGURE:



1. Create a table Manufactures and Products. And Insert the values:

INSERT INTO Manufacturers(Code,Name) VALUES(1,'Sony');
INSERT INTO Manufacturers(Code,Name) VALUES(2,'Creative Labs');
INSERT INTO Manufacturers(Code,Name) VALUES(3,'Hewlett-Packard');
INSERT INTO Manufacturers(Code,Name) VALUES(4,'Iomega');
INSERT INTO Manufacturers(Code,Name) VALUES(5,'Fujitsu');
INSERT INTO Manufacturers(Code,Name) VALUES(6,'Winchester');
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(1,'Hard drive',240,5);

SQL

```
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(2,'Memory',120,6);
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(3,'ZIP drive',150,4);
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(4,'Floppy disk',5,6);
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(5,'Monitor',240,1);
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(6,'DVD drive',180,2);
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(7,'CD drive',90,2);
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(8,'Printer',270,3);
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(9,'Toner cartridge',66,3);
INSERT INTO Products(Code,Name,Price,Manufacturer) VALUES(10,'DVD burner',180,2);
```

TABLE CREATION CODE [THIS CODE IS NOT PROVIDE IN EXAM HALL]

TABLE - Manufacturers	TABLE- Products
CREATE TABLE Manufacturers (Code INTEGER, Name VARCHAR(255) NOT NULL, PRIMARY KEY (Code));	CREATE TABLE Products (Code INTEGER, Name VARCHAR(255) NOT NULL , Price DECIMAL NOT NULL , Manufacturer INTEGER NOT NULL, PRIMARY KEY (Code), FOREIGN KEY (Manufacturer) REFERENCES Manufacturers(Code));

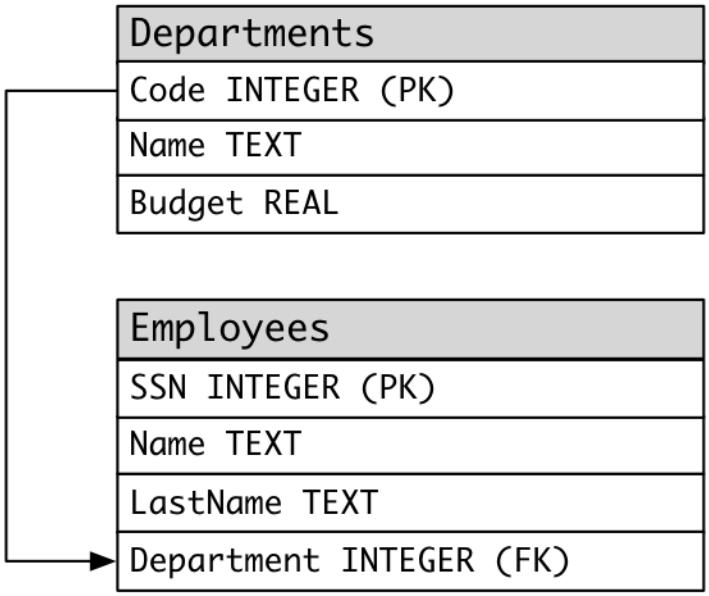
1.1 Select the names of all the products in the store.
1.2 Select the names and the prices of all the products in the store.
1.3 Select the name of the products with a price less than or equal to $200.
1.4 Select all the products with a price between $60 and $120.
1.5 Select the name and price in cents (i.e., the price must be multiplied by 100).
1.6 Compute the average price of all the products.
1.7 Compute the average price of all products with manufacturer code equal to 2.
1.8 Compute the number of products with a price larger than or equal to $180.
1.9 Select the name and price of all products with a price larger than or equal to $180, and sort first by price (in descending order), and then by name (in ascending order).
1.10 Select all the data from the products, including all the data for each product's manufacturer.
1.11 Select the product name, price, and manufacturer name of all the products.
1.12 Select the average price of each manufacturer's products, showing only the manufacturer's code.
1.13 Select the average price of each manufacturer's products, showing the manufacturer's name.
1.14 Select the names of manufacturer whose products have an average price larger than or equal to $150.
1.15 Select the name and price of the cheapest product.

SQL

1.16 Select the name of each manufacturer along with the name and price of its most expensive product.
1.17 Add a new product: Loudspeakers, $70, manufacturer 2.
1.18 Update the name of product 8 to "Laser Printer".
1.19 Apply a 10% discount to all products.
1.20 Apply a 10% discount to all products with a price larger than or equal to $120.

17.

TABLE CODE [NOT PROVIDE IN EXAM]
CREATE TABLE Departments (Code INTEGER PRIMARY KEY, Name varchar(255) NOT NULL , Budget decimal NOT NULL);
CREATE TABLE Employees (SSN INTEGER PRIMARY KEY, Name varchar(255) NOT NULL , LastName varchar(255) NOT NULL , Department INTEGER NOT NULL , foreign key (department) references Departments(Code));



1. Create Table and insert the values.

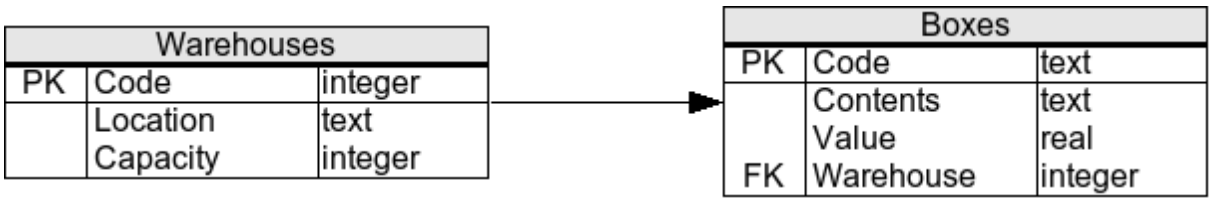
```
INSERT INTO Departments(Code,Name,Budget) VALUES(14,'IT',65000);
INSERT INTO Departments(Code,Name,Budget) VALUES(37,'Accounting',15000);
INSERT INTO Departments(Code,Name,Budget) VALUES(59,'Human Resources',240000);
INSERT INTO Departments(Code,Name,Budget) VALUES(77,'Research',55000);

INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('123234877','Michael','Rogers',14);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('152934485','Anand','Manikutty',14);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('222364883','Carol','Smith',37);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('326587417','Joe','Stevens',37);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('332154719','Mary-Anne','Foster',14);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('332569843','George','ODonnell',77);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('546523478','John','Doe',59);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('631231482','David','Smith',77);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('654873219','Zacary','Efron',59);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('745685214','Eric','Goldsmith',59);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('845657245','Elizabeth','Doe',14);
INSERT INTO Employees(SSN,Name,LastName,Department) VALUES('845657246','Kumar','Swamy',14);
```

SQL

2. Solve the SQL queries:
2.1 Select the last name of all employees.
2.2 Select the last name of all employees, without duplicates.
2.3 Select all the data of employees whose last name is "Smith".
2.4 Select all the data of employees whose last name is "Smith" or "Doe".
2.5 Select all the data of employees that work in department 14.
2.6 Select all the data of employees that work in department 37 or department 77.
2.7 Select all the data of employees whose last name begins with an "S".
2.8 Select the sum of all the departments' budgets.
2.9 Select the number of employees in each department (you only need to show the department code and the number of employees).
2.10 Select all the data of employees, including each employee's department's data.
2.11 Select the name and last name of each employee, along with the name and budget of the employee's department.
2.12 Select the name and last name of employees working for departments with a budget greater than $60,000.
2.13 Select the departments with a budget larger than the average budget of all the departments.
2.14 Select the names of departments with more than two employees.
2.15 Very Important - Select the name and last name of employees working for departments with second lowest budget.
2.16 Add a new department called "Quality Assurance", with a budget of $40,000 and departmental code 11. And Add an employee called "Mary Moore" in that department, with SSN 847-21-9811.
2.17 Reduce the budget of all departments by 10%.
2.18 Reassign all employees from the Research department (code 77) to the IT department (code 14).
2.19 Delete from the table all employees in the IT department (code 14).
2.20 Delete from the table all employees who work in departments with a budget greater than or equal to $60,000.
2.21 Delete from the table all employees.

18.



1. CREATE TABLE AND INSERT ALL VALUES.
2. TABLE: WAREHOUES VALUES INSERT CODE:
INSERT INTO Warehouses(Code,Location,Capacity) VALUES(1,'Chicago',3);

```
INSERT INTO Warehouses(Code,Location,Capacity) VALUES(2,'Chicago',4);
INSERT INTO Warehouses(Code,Location,Capacity) VALUES(3,'New York',7);
INSERT INTO Warehouses(Code,Location,Capacity) VALUES(4,'Los Angeles',2);
INSERT INTO Warehouses(Code,Location,Capacity) VALUES(5,'San Francisco',8);
```

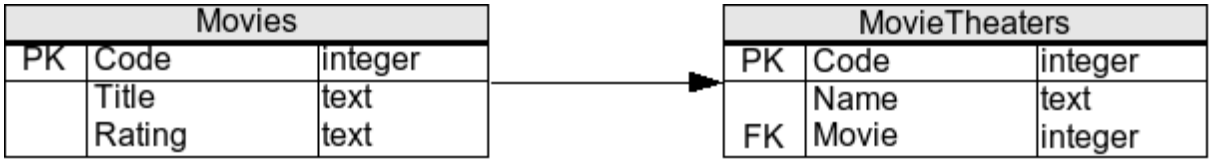
3. TABLE: BOXES VALUES INSERT CODE:

```
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('0MN7','Rocks',180,3);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('4H8P','Rocks',250,1);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('4RT3','Scissors',190,4);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('7G3H','Rocks',200,1);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('8JN6','Papers',75,1);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('8Y6U','Papers',50,3);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('9J6F','Papers',175,2);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('LL08','Rocks',140,4);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('P0H6','Scissors',125,1);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('P2T6','Scissors',150,2);
INSERT INTO Boxes(Code,Contents,Value,Warehouse) VALUES('TU55','Papers',90,5);
```

3.1 Select all warehouses.
3.2 Select all boxes with a value larger than $150.
3.3 Select all distinct contents in all the boxes.
3.4 Select the average value of all the boxes.
3.5 Select the warehouse code and the average value of the boxes in each warehouse.
3.6 Same as previous exercise, but select only those warehouses where the average value of the boxes is greater than 150.
3.7 Select the code of each box, along with the name of the city the box is located in.
3.8 Select the warehouse codes, along with the number of boxes in each warehouse.
Optionally, take into account that some warehouses are empty (i.e., the box count should show up as zero, instead of omitting the warehouse from the result).
3.9 Select the codes of all warehouses that are saturated (a warehouse is saturated if the number of boxes in it is larger than the warehouse's capacity).
3.10 Select the codes of all the boxes located in Chicago.
3.11 Create a new warehouse in New York with a capacity for 3 boxes.
3.12 Create a new box, with code "H5RT", containing "Papers" with a value of $200, and located in warehouse 2.
3.13 Reduce the value of all boxes by 15%.
3.14 Remove all boxes with a value lower than $100.
3.15 Add Index for column "Warehouse" in table "boxes"
3.16 Print all the existing indexes
3.17 Remove (drop) the index you added just

SQL

19. 1. Create table Movies and Movie Theaters using schema figure.



2. Insert values into table Movies

```
INSERT INTO Movies(Code,Title,Rating) VALUES(1,'Citizen Kane','PG');
INSERT INTO Movies(Code,Title,Rating) VALUES(2,'Singin" in the Rain','G');
INSERT INTO Movies(Code,Title,Rating) VALUES(3,'The Wizard of Oz','G');
INSERT INTO Movies(Code,Title,Rating) VALUES(4,'The Quiet Man',NULL);
INSERT INTO Movies(Code,Title,Rating) VALUES(5,'North by Northwest',NULL);
INSERT INTO Movies(Code,Title,Rating) VALUES(6,'The Last Tango in Paris','NC-17');
INSERT INTO Movies(Code,Title,Rating) VALUES(7,'Some Like it Hot','PG-13');
INSERT INTO Movies(Code,Title,Rating) VALUES(8,'A Night at the Opera',NULL);
```

3. Insert values into table Movie Theaters

```
INSERT INTO MovieTheaters(Code,Name,Movie) VALUES(1,'Odeon',5);
INSERT INTO MovieTheaters(Code,Name,Movie) VALUES(2,'Imperial',1);
INSERT INTO MovieTheaters(Code,Name,Movie) VALUES(3,'Majestic',NULL);
INSERT INTO MovieTheaters(Code,Name,Movie) VALUES(4,'Royale',6);
INSERT INTO MovieTheaters(Code,Name,Movie) VALUES(5,'Paraiso',3);
INSERT INTO MovieTheaters(Code,Name,Movie) VALUES(6,'Nickelodeon',NULL);
```

4. Solve this Questions:

4.1 Select the title of all movies.

4.2 Show all the distinct ratings in the database.

4.3 Show all unrated movies.

4.4 Select all movie theaters that are not currently showing a movie.

4.5 Select all data from all movie theaters and, additionally, the data from the movie that is being shown in the theater (if one is being shown).

4.6 Select all data from all movies and, if that movie is being shown in a theater, show the data from the theater.

4.7 Show the titles of movies not currently being shown in any theaters.

4.8 Add the unrated movie "One, Two, Three".

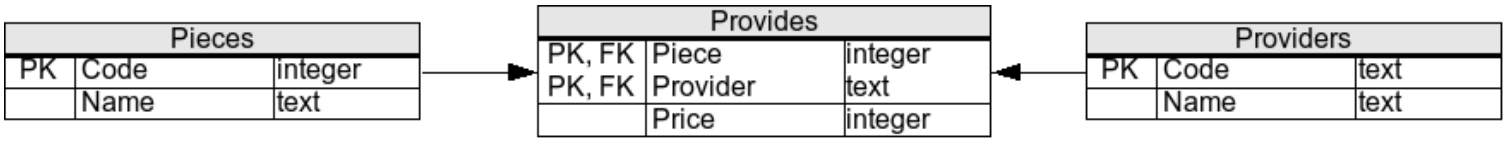
4.9 Set the rating of all unrated movies to "G".

4.10 Remove movie theaters projecting movies rated "NC-17".

[**Notes:** The MySQL IS NOT NULL condition is used to test for a NOT NULL value in a SELECT, INSERT, UPDATE, or DELETE statement. NULL is not a data type - this means it is not recognized as an "int", "date" or any other defined data type. Arithmetic operations involving NULL always return NULL for example, 69 + NULL = NULL. All aggregate functions affect only rows that do not have NULL values.]

SQL

20. 1. Create tables Pieces, Provides and Providers using schema figure



2. TABLE CODE [NOT PROVIDE IN EXAM]

CREATE TABLE Pieces (Code INTEGER PRIMARY KEY NOT NULL, Name TEXT NOT NULL);	CREATE TABLE Providers (Code VARCHAR(40) PRIMARY KEY NOT NULL, Name TEXT NOT NULL);
CREATE TABLE Provides (Piece INTEGER, FOREIGN KEY (Piece) REFERENCES Pieces(Code), Provider VARCHAR(40), FOREIGN KEY (Provider) REFERENCES Providers(Code), Price INTEGER NOT NULL, PRIMARY KEY(Piece, Provider));	**alternative one for SQLite** CREATE TABLE Provides (Piece INTEGER, Provider VARCHAR(40), Price INTEGER NOT NULL, PRIMARY KEY(Piece, Provider));

3. Insert values Providers and pieces tables:

```
INSERT INTO Providers(Code, Name) VALUES('HAL','Clarke Enterprises');
INSERT INTO Providers(Code, Name) VALUES('RBT','Susan Calvin Corp.');
INSERT INTO Providers(Code, Name) VALUES('TNBC','Skellington Supplies');
INSERT INTO Pieces(Code, Name) VALUES(1,'Sprocket');
INSERT INTO Pieces(Code, Name) VALUES(2,'Screw');
INSERT INTO Pieces(Code, Name) VALUES(3,'Nut');
INSERT INTO Pieces(Code, Name) VALUES(4,'Bolt');
```

4. Insert values Provides Table:

```
INSERT INTO Provides(Piece, Provider, Price) VALUES(1,'HAL',10);
INSERT INTO Provides(Piece, Provider, Price) VALUES(1,'RBT',15);
INSERT INTO Provides(Piece, Provider, Price) VALUES(2,'HAL',20);
INSERT INTO Provides(Piece, Provider, Price) VALUES(2,'RBT',15);
INSERT INTO Provides(Piece, Provider, Price) VALUES(2,'TNBC',14);
INSERT INTO Provides(Piece, Provider, Price) VALUES(3,'RBT',50);
INSERT INTO Provides(Piece, Provider, Price) VALUES(3,'TNBC',45);
INSERT INTO Provides(Piece, Provider, Price) VALUES(4,'HAL',5);
INSERT INTO Provides(Piece, Provider, Price) VALUES(4,'RBT',7);
```

5. Solve the questions:

5.1 Select the name of all the pieces.

5.2 Select all the providers' data.

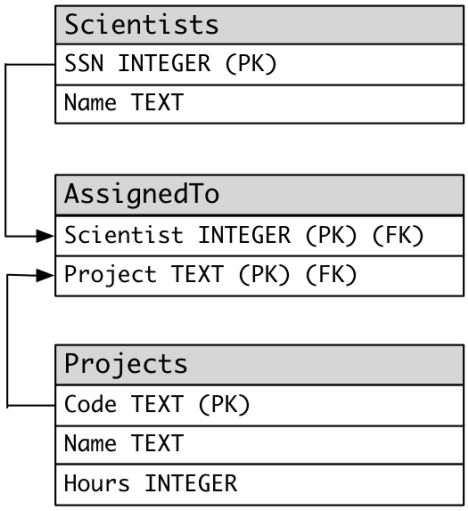
5.3 Obtain the average price of each piece (show only the piece code and the average price).

5.4 Obtain the names of all providers who supply piece 1.

5.5 Select the name of pieces provided by provider with code "HAL".
5.6 For each piece, find the most expensive offering of that piece and include the piece name, provider name, and price (note that there could be two providers who supply the same piece at the most expensive price).
5.7 Add an entry to the database to indicate that "Skellington Supplies" (code "TNBC") will provide sprockets (code "1") for 7 cents each.
5.8 Increase all prices by one cent.
5.9 Update the database to reflect that "Susan Calvin Corp." (code "RBT") will not supply bolts (code 4).
5.10 Update the database to reflect that "Susan Calvin Corp." (code "RBT") will not supply any pieces (the provider should still remain in the database).

21. 1. Create tables using schema figure.

```
INSERT INTO Scientists(SSN,Name)
  VALUES(123234877,'Michael Rogers'),
    (152934485,'Anand Manikutty'),
    (222364883, 'Carol Smith'),
    (326587417,'Joe Stevens'),
    (332154719,'Mary-Anne Foster'),
    (332569843,'George ODonnell'),
    (546523478,'John Doe'),
    (631231482,'David Smith'),
    (654873219,'Zacary Efron'),
    (745685214,'Eric Goldsmith'),
    (845657245,'Elizabeth Doe'),
    (845657246,'Kumar Swamy');
```



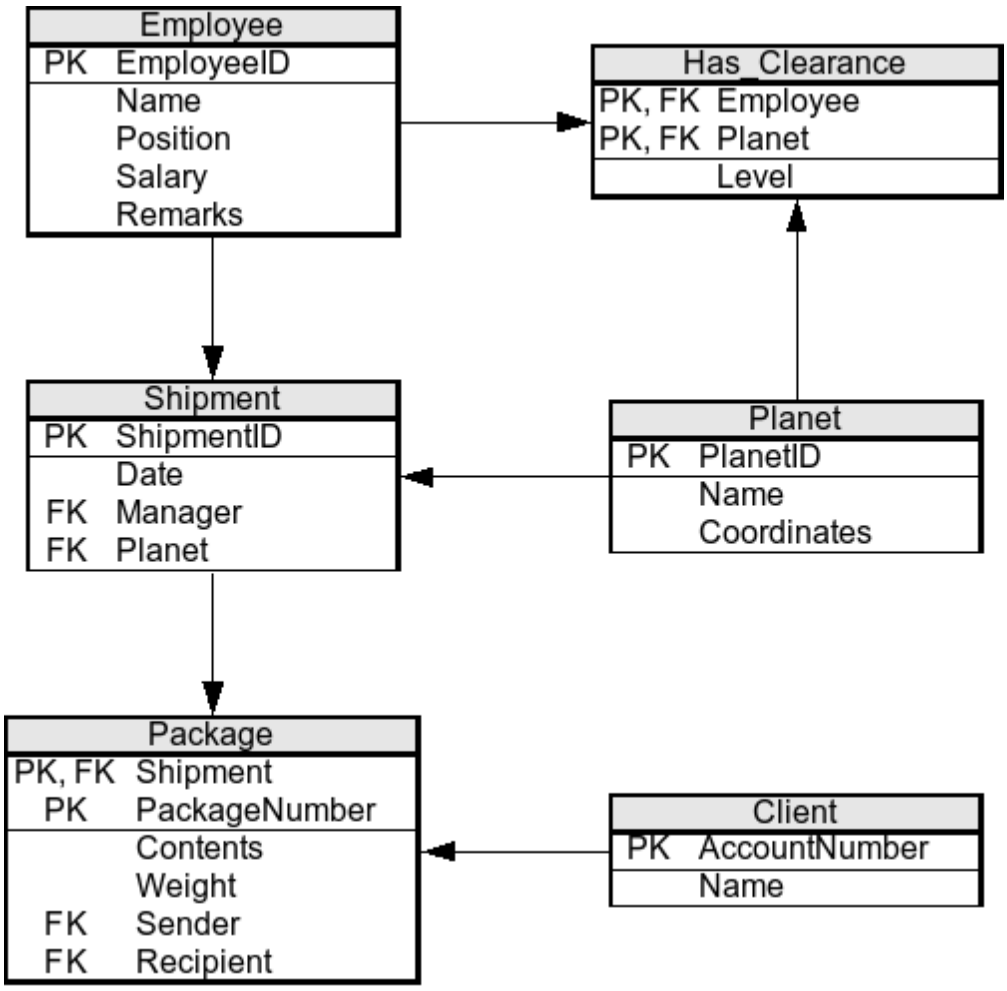

```
INSERT INTO Projects ( Code,Name,Hours)
VALUES ('AeH1','Winds: Studying Bernoullis
Principle', 156),
('AeH2','Aerodynamics and Bridge Design',189),
('AeH3','Aerodynamics and Gas Mileage', 256),
('AeH4','Aerodynamics and Ice Hockey', 789),
('AeH5','Aerodynamics of a Football', 98),
('AeH6','Aerodynamics of Air Hockey',89),
('Ast1','A Matter of Time',112),
('Ast2','A Puzzling Parallax', 299),
('Ast3','Build Your Own Telescope', 6546),
('Bte1','Juicy: Extracting Apple Juice with Pectinase',
321),
('Bte2','A Magnetic Primer Designer', 9684),
('Bte3','Bacterial Transformation Efficiency', 321),
('Che1','A Silver-Cleaning Battery', 545),
('Che2','A Soluble Separation Solution', 778);
```

```
INSERT INTO AssignedTo ( Scientist, Project)
VALUES (123234877,'AeH1'),
(152934485,'AeH3'),
(222364883,'Ast3'),
(326587417,'Ast3'),
(332154719,'Bte1'),
(546523478,'Che1'),
(631231482,'Ast3'),
(654873219,'Che1'),
(745685214,'AeH3'),
(845657245,'Ast1'),
(845657246,'Ast2'),
(332569843,'AeH4');
```

SQL

21.1. List all the scientists' names, their projects' names, and the hours worked by that scientist on each project, in alphabetical order of project name, then scientist name.
21.2 Select the project names which are not assigned yet

22. 1. Create tables using schema figure.



INSERT INTO Client VALUES(1, 'Zapp Brannigan');
INSERT INTO Client VALUES(2, "Al Gore's Head");
INSERT INTO Client VALUES(3, 'Barbados Slim');
INSERT INTO Client VALUES(4, 'Ogden Wernstrom');
INSERT INTO Client VALUES(5, 'Leo Wong');
INSERT INTO Client VALUES(6, 'Lrrr');
INSERT INTO Client VALUES(7, 'John Zoidberg');
INSERT INTO Client VALUES(8, 'John Zoidfarb');
INSERT INTO Client VALUES(9, 'Morbo');
INSERT INTO Client VALUES(10, 'Judge John Whitey');
INSERT INTO Client VALUES(11, 'Calculon');

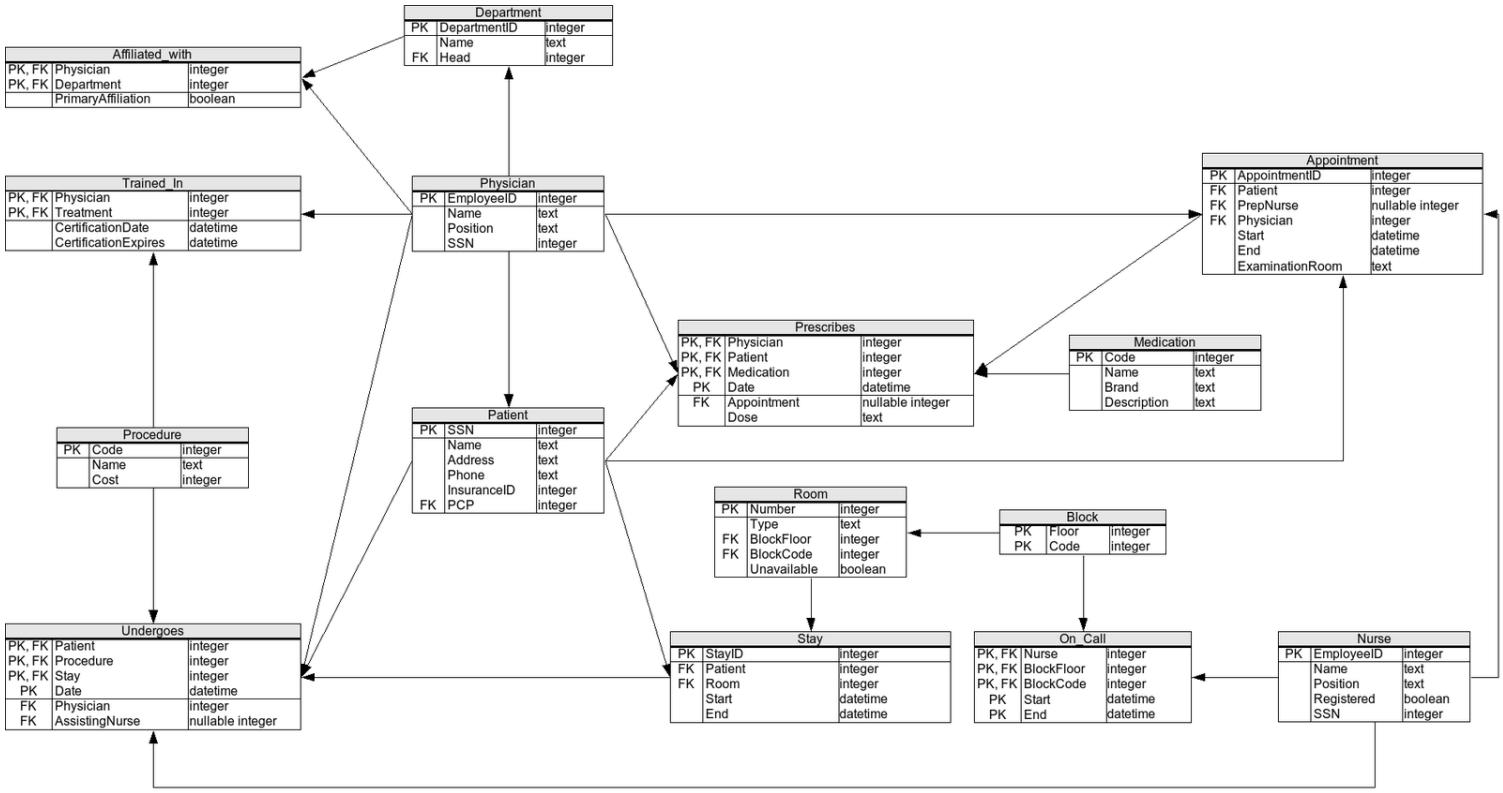
```
INSERT INTO Employee VALUES(1, 'Phillip J. Fry', 'Delivery boy', 7500.0, 'Not to be confused with the Philip J. Fry from Hovering Squid World 97a');
INSERT INTO Employee VALUES(2, 'Turanga Leela', 'Captain', 10000.0, NULL);
INSERT INTO Employee VALUES(3, 'Bender Bending Rodriguez', 'Robot', 7500.0, NULL);
INSERT INTO Employee VALUES(4, 'Hubert J. Farnsworth', 'CEO', 20000.0, NULL);
INSERT INTO Employee VALUES(5, 'John A. Zoidberg', 'Physician', 25.0, NULL);
INSERT INTO Employee VALUES(6, 'Amy Wong', 'Intern', 5000.0, NULL);
INSERT INTO Employee VALUES(7, 'Hermes Conrad', 'Bureaucrat', 10000.0, NULL);
INSERT INTO Employee VALUES(8, 'Scruffy Scruffington', 'Janitor', 5000.0, NULL);
INSERT INTO Planet VALUES(1, 'Omicron Persei 8', 89475345.3545);
INSERT INTO Planet VALUES(2, 'Decapod X', 65498463216.3466);
INSERT INTO Planet VALUES(3, 'Mars', 32435021.65468);
INSERT INTO Planet VALUES(4, 'Omega III', 98432121.5464);
INSERT INTO Planet VALUES(5, 'Tarantulon VI', 849842198.354654);
INSERT INTO Planet VALUES(6, 'Cannibalon', 654321987.21654);
INSERT INTO Planet VALUES(7, 'DogDoo VII', 65498721354.688);
INSERT INTO Planet VALUES(8, 'Nintenduu 64', 6543219894.1654);
INSERT INTO Planet VALUES(9, 'Amazonia', 65432135979.6547);
INSERT INTO Has_Clearance VALUES(1, 1, 2);
INSERT INTO Has_Clearance VALUES(1, 2, 3);
INSERT INTO Has_Clearance VALUES(2, 3, 2);
INSERT INTO Has_Clearance VALUES(2, 4, 4);
INSERT INTO Has_Clearance VALUES(3, 5, 2);
INSERT INTO Has_Clearance VALUES(3, 6, 4);
INSERT INTO Has_Clearance VALUES(4, 7, 1);
INSERT INTO Shipment VALUES(1, '3004/05/11', 1, 1);
INSERT INTO Shipment VALUES(2, '3004/05/11', 1, 2);
INSERT INTO Shipment VALUES(3, NULL, 2, 3);
INSERT INTO Shipment VALUES(4, NULL, 2, 4);
INSERT INTO Shipment VALUES(5, NULL, 7, 5);
INSERT INTO Package VALUES(1, 1, 'Undeclared', 1.5, 1, 2);
INSERT INTO Package VALUES(2, 1, 'Undeclared', 10.0, 2, 3);
INSERT INTO Package VALUES(2, 2, 'A bucket of krill', 2.0, 8, 7);
INSERT INTO Package VALUES(3, 1, 'Undeclared', 15.0, 3, 4);
INSERT INTO Package VALUES(3, 2, 'Undeclared', 3.0, 5, 1);
INSERT INTO Package VALUES(3, 3, 'Undeclared', 7.0, 2, 3);
INSERT INTO Package VALUES(4, 1, 'Undeclared', 5.0, 4, 5);
INSERT INTO Package VALUES(4, 2, 'Undeclared', 27.0, 1, 2);
INSERT INTO Package VALUES(5, 1, 'Undeclared', 100.0, 5, 1);
```

22.1 Who received a 1.5kg package? The result is "Al Gore's Head".

22..2 What is the total weight of all the packages that he sent?

SQL

23. Create tables using schema figure



INSERT INTO Physician VALUES(1,'John Dorian','Staff Internist',111111111);
INSERT INTO Physician VALUES(2,'Elliot Reid','Attending Physician',222222222);
INSERT INTO Physician VALUES(3,'Christopher Turk','Surgical Attending Physician',333333333);
INSERT INTO Physician VALUES(4,'Percival Cox','Senior Attending Physician',444444444);
INSERT INTO Physician VALUES(5,'Bob Kelso','Head Chief of Medicine',555555555);
INSERT INTO Physician VALUES(6,'Todd Quinlan','Surgical Attending Physician',666666666);
INSERT INTO Physician VALUES(7,'John Wen','Surgical Attending Physician',777777777);
INSERT INTO Physician VALUES(8,'Keith Dudemeister','MD Resident',888888888);
INSERT INTO Physician VALUES(9,'Molly Clock','Attending Psychiatrist',999999999);
INSERT INTO Department VALUES(1,'General Medicine',4);
INSERT INTO Department VALUES(2,'Surgery',7);
INSERT INTO Department VALUES(3,'Psychiatry',9);
INSERT INTO Affiliated_With VALUES(1,1,1);
INSERT INTO Affiliated_With VALUES(2,1,1);
INSERT INTO Affiliated_With VALUES(3,1,0);
INSERT INTO Affiliated_With VALUES(3,2,1);
INSERT INTO Affiliated_With VALUES(4,1,1);
INSERT INTO Affiliated_With VALUES(5,1,1);
INSERT INTO Affiliated_With VALUES(6,2,1);
INSERT INTO Affiliated_With VALUES(7,1,0);
INSERT INTO Affiliated_With VALUES(7,2,1);

SQL

```
INSERT INTO Affiliated_With VALUES(8,1,1);
INSERT INTO Affiliated_With VALUES(9,3,1);
INSERT INTO Procedures VALUES(1,'Reverse Rhinopodoplasty',1500.0);
INSERT INTO Procedures VALUES(2,'Obtuse Pyloric Recombobulation',3750.0);
INSERT INTO Procedures VALUES(3,'Folded Demiophtalmectomy',4500.0);
INSERT INTO Procedures VALUES(4,'Complete Walletectomy',10000.0);
INSERT INTO Procedures VALUES(5,'Obfuscated Dermogastrotomy',4899.0);
INSERT INTO Procedures VALUES(6,'Reversible Pancreomyoplasty',5600.0);
INSERT INTO Procedures VALUES(7,'Follicular Demiectomy',25.0);
INSERT INTO Patient VALUES(100000001,'John Smith','42 Foobar Lane','555-0256',68476213,1);
INSERT INTO Patient VALUES(100000002,'Grace Ritchie','37 Snafu Drive','555-0512',36546321,2);
INSERT INTO Patient VALUES(100000003,'Random J. Patient','101 Omgbbq Street','555-1204',65465421,2);
INSERT INTO Patient VALUES(100000004,'Dennis Doe','1100 Foobaz Avenue','555-2048',68421879,3);
INSERT INTO Nurse VALUES(101,'Carla Espinosa','Head Nurse',1,111111110);
INSERT INTO Nurse VALUES(102,'Laverne Roberts','Nurse',1,222222220);
INSERT INTO Nurse VALUES(103,'Paul Flowers','Nurse',0,333333330);
INSERT INTO Appointment VALUES(13216584,100000001,101,1,'2008-04-24 10:00','2008-04-24 11:00','A');
INSERT INTO Appointment VALUES(26548913,100000002,101,2,'2008-04-24 10:00','2008-04-24 11:00','B');
INSERT INTO Appointment VALUES(36549879,100000001,102,1,'2008-04-25 10:00','2008-04-25 11:00','A');
INSERT INTO Appointment VALUES(46846589,100000004,103,4,'2008-04-25 10:00','2008-04-25 11:00','B');
INSERT INTO Appointment VALUES(59871321,100000004,NULL,4,'2008-04-26 10:00','2008-04-26 11:00','C');
INSERT INTO Appointment VALUES(69879231,100000003,103,2,'2008-04-26 11:00','2008-04-26 12:00','C');
INSERT INTO Appointment VALUES(76983231,100000001,NULL,3,'2008-04-26 12:00','2008-04-26 13:00','C');
INSERT INTO Appointment VALUES(86213939,100000004,102,9,'2008-04-27 10:00','2008-04-21 11:00','A');
INSERT INTO Appointment VALUES(93216548,100000002,101,2,'2008-04-27 10:00','2008-04-27 11:00','B');
INSERT INTO Medication VALUES(1,'Procrastin-X','X','N/A');
INSERT INTO Medication VALUES(2,'Thesisin','Foo Labs','N/A');
INSERT INTO Medication VALUES(3,'Awakin','Bar Laboratories','N/A');
INSERT INTO Medication VALUES(4,'Crescavitin','Baz Industries','N/A');
INSERT INTO Medication VALUES(5,'Melioraurin','Snafu Pharmaceuticals','N/A');
INSERT INTO Prescribes VALUES(1,100000001,1,'2008-04-24 10:47',13216584,'5');
INSERT INTO Prescribes VALUES(9,100000004,2,'2008-04-27 10:53',86213939,'10');
INSERT INTO Prescribes VALUES(9,100000004,2,'2008-04-30 16:53',NULL,'5');
INSERT INTO Block VALUES(1,1);
INSERT INTO Block VALUES(1,2);
INSERT INTO Block VALUES(1,3);
INSERT INTO Block VALUES(2,1);
INSERT INTO Block VALUES(2,2);
INSERT INTO Block VALUES(2,3);
INSERT INTO Block VALUES(3,1);
INSERT INTO Block VALUES(3,2);
INSERT INTO Block VALUES(3,3);
INSERT INTO Block VALUES(4,1);
INSERT INTO Block VALUES(4,2);
INSERT INTO Block VALUES(4,3);
```

```
INSERT INTO Room VALUES(101,'Single',1,1,0);
INSERT INTO Room VALUES(102,'Single',1,1,0);
INSERT INTO Room VALUES(103,'Single',1,1,0);
INSERT INTO Room VALUES(111,'Single',1,2,0);
INSERT INTO Room VALUES(112,'Single',1,2,1);
INSERT INTO Room VALUES(113,'Single',1,2,0);
INSERT INTO Room VALUES(121,'Single',1,3,0);
INSERT INTO Room VALUES(122,'Single',1,3,0);
INSERT INTO Room VALUES(123,'Single',1,3,0);
INSERT INTO Room VALUES(201,'Single',2,1,1);
INSERT INTO Room VALUES(202,'Single',2,1,0);
INSERT INTO Room VALUES(203,'Single',2,1,0);
INSERT INTO Room VALUES(211,'Single',2,2,0);
INSERT INTO Room VALUES(212,'Single',2,2,0);
INSERT INTO Room VALUES(213,'Single',2,2,1);
INSERT INTO Room VALUES(221,'Single',2,3,0);
INSERT INTO Room VALUES(222,'Single',2,3,0);
INSERT INTO Room VALUES(223,'Single',2,3,0);
INSERT INTO Room VALUES(301,'Single',3,1,0);
INSERT INTO Room VALUES(302,'Single',3,1,1);
INSERT INTO Room VALUES(303,'Single',3,1,0);
INSERT INTO Room VALUES(311,'Single',3,2,0);
INSERT INTO Room VALUES(312,'Single',3,2,0);
INSERT INTO Room VALUES(313,'Single',3,2,0);
INSERT INTO Room VALUES(321,'Single',3,3,1);
INSERT INTO Room VALUES(322,'Single',3,3,0);
INSERT INTO Room VALUES(323,'Single',3,3,0);
INSERT INTO Room VALUES(401,'Single',4,1,0);
INSERT INTO Room VALUES(402,'Single',4,1,1);
INSERT INTO Room VALUES(403,'Single',4,1,0);
INSERT INTO Room VALUES(411,'Single',4,2,0);
INSERT INTO Room VALUES(412,'Single',4,2,0);
INSERT INTO Room VALUES(413,'Single',4,2,0);
INSERT INTO Room VALUES(421,'Single',4,3,1);
INSERT INTO Room VALUES(422,'Single',4,3,0);
INSERT INTO Room VALUES(423,'Single',4,3,0);
INSERT INTO On_Call VALUES(101,1,1,'2008-11-04 11:00','2008-11-04 19:00');
INSERT INTO On_Call VALUES(101,1,2,'2008-11-04 11:00','2008-11-04 19:00');
INSERT INTO On_Call VALUES(102,1,3,'2008-11-04 11:00','2008-11-04 19:00');
INSERT INTO On_Call VALUES(103,1,1,'2008-11-04 19:00','2008-11-05 03:00');
INSERT INTO On_Call VALUES(103,1,2,'2008-11-04 19:00','2008-11-05 03:00');
INSERT INTO On_Call VALUES(103,1,3,'2008-11-04 19:00','2008-11-05 03:00');
INSERT INTO Stay VALUES(3215,100000001,111,'2008-05-01','2008-05-04');
INSERT INTO Stay VALUES(3216,100000003,123,'2008-05-03','2008-05-14');
INSERT INTO Stay VALUES(3217,100000004,112,'2008-05-02','2008-05-03');
```

```
INSERT INTO Undergoes VALUES(100000001,6,3215,'2008-05-02',3,101);
INSERT INTO Undergoes VALUES(100000001,2,3215,'2008-05-03',7,101);
INSERT INTO Undergoes VALUES(100000004,1,3217,'2008-05-07',3,102);
INSERT INTO Undergoes VALUES(100000004,5,3217,'2008-05-09',6,NULL);
INSERT INTO Undergoes VALUES(100000001,7,3217,'2008-05-10',7,101);
INSERT INTO Undergoes VALUES(100000004,4,3217,'2008-05-13',3,103);
INSERT INTO Trained_In VALUES(3,1,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(3,2,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(3,5,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(3,6,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(3,7,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(6,2,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(6,5,'2007-01-01','2007-12-31');
INSERT INTO Trained_In VALUES(6,6,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(7,1,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(7,2,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(7,3,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(7,4,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(7,5,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(7,6,'2008-01-01','2008-12-31');
INSERT INTO Trained_In VALUES(7,7,'2008-01-01','2008-12-31');
```

23.1 Obtain the names of all physicians that have performed a medical procedure they have never been certified to perform.

23.2 Same as the previous query, but include the following information in the results: Physician name, name of procedure, date when the procedure was carried out, name of the patient the procedure was carried out on.

23.3 Obtain the names of all physicians that have performed a medical procedure that they are certified to perform, but such that the procedure was done at a date (Undergoes Date) after the physician's certification expired (Trained In. Certification Expires).

23.4 Same as the previous query, but include the following information in the results: Physician name, name of procedure, date when the procedure was carried out, name of the patient the procedure was carried out on, and date when the certification expired.

23.5 Obtain the information for appointments where a patient met with a physician other than his/her primary care physician. Show the following information: Patient name, physician name, nurse name (if any), start and end time of appointment, examination room, and the name of the patient's primary care physician.

23.6 The Patient field in Undergoes is redundant, since we can obtain it from the Stay table. There are no constraints in force to prevent inconsistencies between these two tables. More specifically, the Undergoes table may include a row where the patient ID does not match the one we would

obtain from the Stay table through the Undergoes. Stay foreign key. Select all rows from Undergoes that exhibit this inconsistency.

23.7 Obtain the names of all the nurses who have ever been on call for room 123.

23.8 The hospital has several examination rooms where appointments take place. Obtain the number of appointments that have taken place in each examination room.

23.9 Obtain the names of all patients (also include, for each patient, the name of the patient's primary care physician), such that \emph{all} the following are true:

The patient has been prescribed some medication by his/her primary care physician.

The patient has undergone a procedure with a cost larger that $5,000

The patient has had at least two appointment where the nurse who prepped the appointment was a registered nurse. The patient's primary care physician is not the head of any department.

24. TABLE: HOSPITAL

NO	NAME	AGE	DEPARTMENT	DATEOFADM	CHARGES	SEX
1	SANDEEP	65	SURGERY	23/02/98	300	M
2	RAVINA	24	ORTHOPAEDIC	20/01/98	200	F
3	KARAN	45	ORTHOPAEDIC	19/02/98	200	M
4	TARUN	12	SURGERY	01/01/98	300	M
5	ZUBIN	36	ENT	12/01/98	250	M
6	KETAKI	16	ENT	24/02/98	300	F
7	ANKITA	29	CARDIOLOGY	22/02/98	800	F
8	ZAREEN	45	GYNAECOLOGY	22/02/98	300	F
9	KUSH	19	CARDIOLOGY	13/01/98	800	M
10	SHAILYA	31	NUCLEAR MEDICINE	19/02/98	400	M

a) To show all information about the patients of cardiology department.

b) To list the name of female patients, who are in orthopedic department.

c) To list names of all patients with their date of admission in ascending order.

d) To display patient's name, charges, age for male patients only.

e) To count the number of patients with age >20.

f) To insert a new row in the hospital table with the following data:

{11, 'MUSTAFA', 37, 'ENT', '25/02/98', 250, 'M'}

g) Give the output of following SQL statements:

I) SELECT COUNT (DISTINCT CHARGES) FROM HOSPITAL;

II) SELECT MIN (AGE) FROM HOSPITAL WHERE SEX= 'M';

III) SELECT SUM(CHARGES) FROM HOSPITAL WHERE SEX= 'F';

IV) SELECT AVG(CHARGES) FROM HOSPITAL WHERE DATEOFADM < '12/02/98';

25. TABLE: INTERIORS

NO	ITEMNAME	TYPE	DATEOFSTOCK	PRICE	DISCOUNT
1	RED ROSE	DOUBLE BED	23/02/02	32000	15
2	SOFT TOUCH	BABY COT	20/01/02	9000	10
3	JERRY'S HOME	BABY COT	19/02/02	8500	10
4	ROUGH WOOD	OFFICE TABLE	01/01/02	20000	20
5	COMFORT ZONE	DOUBLE BED	12/01/02	15000	20
6	JERRY LOOK	BABY COT	24/02/02	7000	19
7	LION KING	OFFICE TABLE	20/02/02	16000	20
8	ROYAL TOGER	SOFA	22/02/02	30000	25
9	PARK SITTING	SOFA	13/12/01	9000	15
10	DINE PARADISE	DINING TABLE	19/02/02	11000	15
11	WHITE WOOD	DOUBLE BED	23/03/03	20000	20
12	JAMES 007	SOFA	20/02/03	15000	15
13	TOM LOOK	BABY COT	21/02/03	7000	10

a) To show all information about the sofa from the INTERIORS table.
b) To list the ITEMNAME, which are priced at more than 10000 from the INTERIORS table.
c) To list ITEMNAME and TYPE of those items, in which DATEOFSTOCK is before 22/01/02 from the INTERIORS table in descending order of ITEMNAME.
d) To display ITEMNAME and DATEOFSTOCK of those items, in which the discount percentage is more than 15 from INTERIORS table.
e) To count the number of items, whose type if DOUBLE BED from INTERIORS table.
f) To insert a new row in the INTERIORS table with the following data
{14, 'TRUE INDIAN', 'OFFICE TABLE', '28/03/03', 15000, 20}
g) Give the output of following SQL statements:
I) SELECT COUNT (DISTINCT TYPE) FROM INTERIORS;
II) SELECT AVG(DISCOUNT) FROM INTERIORS WHERE TYPE = 'BABY COT';
III) SELECT SUM(PRICE) FROM INTERIORS WHERE DATEOFSTOCK< '12/02/02';

26. TABLE: BOOKS

BOOK_ID	BOOK_NAME	AUTHOR_NAME	PUBLISHERS	PRICE	TYPE	QTY
F0001	THE TEARS	WILLIAM HOPKINS	FIRST PUBL	750	FICTION	10
F0002	THUNDERBOLTS	ANNA ROBERTS	FIRST PUBL	700	FICTION	5
T0001	MY FIRST C++	BRAIN & BROOKE	EPB	250	TEXT	10
T0002	C++ BRAINWORKS	A.W. ROSSAINE	TDH	325	TEXT	5
C0001	FAST COOK	LATA KAPOOR	EPB	350	COOKERY	8

TABLE: ISSUED

BOOK_ID	QUANTITY_ISSUED
F0001	3
T0001	1
C0001	5

a) To show book name, author name and price of books of EPB publishers.
b) To list the names from books of Fiction type.
c) To display the names and price of the books in descending order of their price.
d) To increase the price of all books of first publishers by 50.
e) To display the BOOK_ID, BOOK_NAME and QUANTITY_ISSUED for all books which have been issued. (The query will require contents from both the tables.)
f) To insert a new row in the table ISSUED having the following data 'F0002',4;
g) Give the output of the following queries based on the above tables:
I) SELECT COUNT (DISTINCT PUBLISHERS) FROM BOOKS;
II) SELECT SUM(PRICE) FROM BOOKS WHERE QTY>5;
III) SELECT BOOK_NAME, AUTHOR_NAME FROM BOOKS WHERE PRICE<500;
IV) SELECT COUNT (*) FROM BOOKS;

27. TABLE: STUDENT

SNO	NAME	STREAM	FEES	AGE	SEX
1	ARUN KUMAR	COMPUTER	750.00	17	M
2	DIVYA JENEJA	COMPUTER	750.00	18	F
3	KESHAR MEHRA	BIOLOGY	500.00	16	M
4	HARISH SINGH	ENG. DR	350.00	18	M
5	PRACHI	ECONOMICS	300.00	19	F
6	NISHA ARORA	COMPUTER	750.00	15	F
7	DEEPAK KUMAR	ECONOMICS	300.00	16	M
8	SARIKA VASWANI	BIOLOGY	500.00	15	F

a) List the name of all the student, who have taken stream as computer.
b) To count the number of female students.
c) To display the number of students, stream wise.
d) To insert a new row in the STUDENT table: 9, 'KARISHMA', 'ECONOMICS',300,18, 'F';
e) To display a report, listing NAME, STREAM, SEX and STIPEND, where stipend is 20% of fees.
f) To display all the records in sorted order of name.
g) Give the output of the following SQL statement based on Student table:
I) SELECT AVG(FEES) FROM STUDENT WHERE STREAM= 'COMPUTER';
II) SELECT MAX(AGE) FROM STUDENT;
III) SELECT COUNT (DISTINCT STREAM) FROM STUDENT;
IV) SELECT SUM(FEES) FROM STUDENT GROUP BY STREAM;

28. TABLE: FAMILY

NO	NAME	FEMALEMEMBERS	MALEMEMBERS	INCOME	OCCUPATION
1	MISHRA	3	2	7000	SERVICE
2	GUPTA	4	1	50000	BUSINESS
3	KHAN	6	3	8000	MIXED
4	CHADDHA	2	2	25000	BUSINESS
5	YADAV	7	2	20000	MIXED
6	JOSHI	3	2	14000	SERVICE
7	MAURYA	6	3	5000	FARMING
8	RAO	5	2	10000	SERVICE

a) To select all the information of family, whose Occupation is Service.
b) To list the name of family, where female members are more than 3.
c) To list all names of family with income in ascending order
d) To display family's name, male members and occupation of business family.
e) To count the number of family, whose income is less than 10,000.
f) To insert a new record in the FAMILY table with the following data:
9, "D Souza", 2, 1, 15000, "Service"
g) Give the output of the following SQL commands:
i) SELECT MIN (DISTINCT INCOME) FROM FAMILY;
ii) SELECT MIN(FEMALEMEMBERS) FROM FAMILY WHERE OCCUPATION= "MIXED";
iii) SELECT SUM(INCOME) FROM FAMILY WHERE OCCUPATION = "SERVICE";
iv) SELECT AVG(INCOME) FROM FAMILY;

ORACLE NUMBER FUNCTIONS (ALSO KNOWN AS MATH FUNCTIONS)

Number functions accept numeric input and return numeric values. Most of these functions return values that are accurate to 38 decimal digits. The number functions available in Oracle are: ABS ACOS ASIN ATAN ATAN2 BITAND CEIL COS COSH EXP FLOOR LN LOG MOD POWER ROUND (number) SIGN SIN SINH SQRT TAN TANH TRUNC (number).

ABS: ABS returns the absolute value of n.

The following example returns the absolute value of -87:

```
SELECT ABS(-87) "Absolute" FROM DUAL;
```

ACOS: ACOS returns the arc cosine of n. Inputs are in the range of -1 to 1, and outputs are in the range of 0 to pi and are expressed in radians.

The following example returns the arc cosine of .3:

```
SELECT ACOS(.3)"Arc_Cosine" FROM DUAL;
```

Similar to ACOS, you have ASIN (Arc Sine), ATAN (Arc Tangent) functions.

CEIL: Returns the lowest integer above the given number.
Example: The following function return the lowest integer above 3.456;
select ceil(3.456) "Ceil" from dual;

FLOOR: Returns the highest integer below the given number.
Example: The following function return the highest integer below 3.456;
select floor(3.456) "Floor" from dual;

COS: Returns the cosine of an angle (in radians). Example: The following example returns the COSINE angle of 60 radians. Select cos(60) "Cosine" from dual;

SIN: Returns the Sine of an angle (in radians). Example: The following example returns the SINE angle of 60 radians. Select SIN(60) "Sine" from dual;

TAN: Returns the Tangent of an angle (in radians). Example: The following example returns the tangent angle of 60 radians. Select Tan(60) "Tangent" from dual;
Similar to SIN, COS, TAN functions hyperbolic functions SINH, COSH, TANH are also available in oracle.

MOD: Returns the remainder after dividing m with n.
Example: The following example returns the remainder after dividing 30 by 4.
Select mod(30,4) "MOD" from dual;

POWER: Returns the power of m, raised to n.
Example: The following example returns the 2 raised to the power of 3.
select power(2,3) "Power" from dual;

EXP: Returns the e raised to the power of n. Example**:** The following example returns the e raised to power of 2. select exp(2) "e raised to 2" from dual;

LN: Returns natural logarithm of n. Example: The following example returns the natural logarithm of 2. select ln(2) from dual;

LOG: Returns the logarithm, base m, of n. Example**:** The following example returns the log of 100.
select log(10,100) from dual;

ROUND: Returns a decimal number rounded of to a given decimal positions.
select round(3.4573,2) "Round" from dual;

TRUNC: Returns a decimal number Truncated to a given decimal positions.
Example: select trunc(3.4573) "trunc" from dual;

SQRT: Returns the square root of a given number. select sqrt(16) from dual;

Oracle Dual Table: The DUAL is special one row, one column table present by default in all Oracle databases. The owner of DUAL is SYS (SYS owns the data dictionary, therefore DUAL is part of the data dictionary.) but DUAL can be accessed by every user. The table has a single VARCHAR2(1) column called DUMMY that has a value of 'X'. MySQL allows DUAL to be specified as a table in queries that do not need data from any tables. In SQL Server DUAL table does not exist, but you could create one.
The DUAL table was created by Charles Weiss of Oracle corporation to provide a table for joining in internal views.

1. The following command displays the structure of DUAL table:

```
DESC DUAL;
Output:
Name              Null?   Type
------------------------- ------
DUMMY                     VARCHAR2(1)
```

2. The following command displays the content of the DUAL table:

```
SELECT * FROM DUAL;
Output:
DUMMY
----------
X
```

3. The following command displays the number of rows of DUAL table:

```
SELECT COUNT(*) FROM DUAL;
```

4. The following command displays the string value from the DUAL table:

```
SELECT 'ABCDEF12345' FROM DUAL;
```

5. The following command displays the numeric value from the DUAL table:

```
SELECT 123792.52 FROM DUAL;
```

6. The following command tries to delete all rows from the DUAL table:

```
DELETE FROM DUAL; [ Output: ERROR at line 1: ORA-01031: insufficient privileges]
```

7. The following command tries to remove all rows from the DUAL table:

```
TRUNCATE TABLE DUAL;
```

Note: The DELETE command is used to remove rows from a table. After performing a DELETE operation, you need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it. TRUNCATE removes all rows from a table. The operation cannot be rolled back.

Output:

```
TRUNCATE TABLE DUAL
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

8. The following command select two rows from dual:

```
SELECT dummy FROM DUAL
UNION ALL
SELECT dummy FROM DUAL;
```

Output:

```
DUMMY
----------
X
X
```

9. Check the system date from the DUAL table using the following statement:

```
SELECT sysdate FROM DUAL;
```

10. Check the arithmetic calculation from the DUAL table using the following statement:

```
SELECT 15+10-5*5/5 FROM DUAL;
```

11. Following code display the numbers 1..10 from DUAL :

```
SELECT level FROM DUAL CONNECT BY level <=10;
```

12. In the following code, DUAL involves the use of decode with NULL.

```
SELECT decode(null,null,1,0) FROM DUAL;
```

DUAL table : Oracle vs MySQL: We have already learned that DUAL is a special one row one column table. For Oracle, it is useful because Oracle doesn't allow statements like:

```
SELECT 15+10-5*5/5;
```

Output: SELECT 15+10-5*5/5;

* ERROR at line 1: ORA-00923: FROM keyword not found where expected

But the following command will execute (see the output of the previous example):
SELECT 15+10-5*5/5 FROM DUAL;
In case of MySQL the following command will execute: SELECT 15+10-5*5/5;

The following table shows the uses of dummy table in standard DBMS.

DBMS - Dummy-table concept
MSSQL - No dummy-table concept.
MySQL - No dummy-table concept.
Oracle - Dummy-table : DUAL.
Informix - Since version 11.10, a dummy table has been included : sysmaster:sysdual
PostgreSQL - No dummy-table concept.
DB2 - Dummy-table : SYSIBM.SYSDUMMY1

The EMP and DEPT tables in Oracle

Both these tables are owned by the SCOTT user, together with two less frequently used tables: BONUS and SALGRADE. Execute the below code snippets to create and seed the EMP and DEPT tables in your own schema.

```
create table dept(
  deptno number(2,0),
  dname  varchar2(14),
  loc    varchar2(13),
  constraint pk_dept primary key (deptno)
);
```

```
create table bonus(
  ename varchar2(10),
  job   varchar2(9),
  sal   number,
  comm  number
);
```

Emp Table Code:

```
create table emp(
  empno    number(4,0),
  ename    varchar2(10),
  job      varchar2(9),
  mgr      number(4,0),
  hiredate date,
  sal      number(7,2),
  comm     number(7,2),
  deptno   number(2,0),
  constraint pk_emp primary key (empno),
  constraint fk_deptno foreign key (deptno) references
dept (deptno)
);
```

Salgrade Table Code:

```
create table salgrade(
  grade number,
  losal number,
  hisal number
);
```

```
insert into dept values(10, 'ACCOUNTING', 'NEW YORK');
insert into dept values(20, 'RESEARCH', 'DALLAS');
insert into dept values(30, 'SALES', 'CHICAGO');
insert into dept values(40, 'OPERATIONS', 'BOSTON');
```

```
insert into emp values(7839, 'KING', 'PRESIDENT', null, to_date('17-11-1981','dd-mm-yyyy'), 5000, null, 10);
insert into emp values(7698, 'BLAKE', 'MANAGER', 7839, to_date('1-5-1981','dd-mm-yyyy'), 2850, null, 30);
insert into emp values(7782, 'CLARK', 'MANAGER', 7839, to_date('9-6-1981','dd-mm-yyyy'),2450, null, 10);
insert into emp values(7566, 'JONES', 'MANAGER', 7839,to_date('2-4-1981','dd-mm-yyyy'),2975, null, 20);
insert into emp values(7788, 'SCOTT', 'ANALYST', 7566,to_date('13-JUL-87','dd-mm-rr') - 85,
3000, null, 20);
insert into emp values(7902, 'FORD', 'ANALYST', 7566,to_date('3-12-1981','dd-mm-yyyy'),3000, null, 20);
insert into emp values(7369, 'SMITH', 'CLERK', 7902,to_date('17-12-1980','dd-mm-yyyy'),800, null, 20);
insert into emp values(7499, 'ALLEN', 'SALESMAN', 7698, to_date('20-2-1981','dd-mm-yyyy'), 1600, 300, 30);
insert into emp values(7521, 'WARD', 'SALESMAN', 7698, to_date('22-2-1981','dd-mm-yyyy'), 1250, 500, 30);
insert into emp values(7654, 'MARTIN', 'SALESMAN', 7698, to_date('28-9-1981','dd-mm-yyyy'),
 1250, 1400, 30);
insert into emp values(7844, 'TURNER', 'SALESMAN', 7698,to_date('8-9-1981','dd-mm-yyyy'), 1500, 0, 30);
insert into emp values(7876, 'ADAMS', 'CLERK', 7788, to_date('13-JUL-87', 'dd-mm-rr') - 51,
1100, null, 20);
insert into emp values(7900, 'JAMES', 'CLERK', 7698, to_date('3-12-1981','dd-mm-yyyy'),
950, null, 30);
insert into emp values(7934, 'MILLER', 'CLERK', 7782, to_date('23-1-1982','dd-mm-yyyy'),
1300, null, 10);
insert into salgrade values (1, 700, 1200);
insert into salgrade values (2, 1201, 1400);
insert into salgrade values (3, 1401, 2000);
insert into salgrade values (4, 2001, 3000);
insert into salgrade values (5, 3001, 9999);
```

commit;

[Note: The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.]

MYSQL TRANSACTION CONTROL LANGUAGE

Commit, Rollback and Savepoint MySQL commands: Transaction Control Language(TCL) commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

COMMIT command: COMMIT command is used to permanently save any transaction into the database.When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

ROLLBACK command: This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

SAVEPOINT command: SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Auto commit on off : To disable auto-commit mode: SET autocommit=0; / SET autocommit=off; To enable auto-commit mode: SET autocommit = 1; / SET autocommit = ON;

Step by Step Rollback Example:

```
mysql> create table pba(roll int)ENGINE=InnoDB;
mysql> insert into pba values(23);
mysql> insert into pba values(21);
mysql> insert into pba values(45);
mysql> select *from pba;
mysql> commit;
mysql> start transaction;
mysql> delete from pba;
mysql> rollback;
mysql> select *from pba;
```

Step by Step Rollback to Savepoint Example:

```
mysql> start transaction;
mysql> select *from pba;
mysql> insert into pba values(1);
mysql> insert into pba values(2);
mysql> savepoint s1;
mysql> insert into pba values(10);
mysql> insert into pba values(11);
mysql> select *from pba;
mysql> rollback to savepoint s1;
mysql> select *from pba;
mysql> commit;
```

Step by Step RELEASE SAVEPOINT Example:

mysql> start transaction; mysql> select *from pba; mysql> insert into pba values(11); mysql> insert into pba values(12); mysql> savepoint s2; mysql> insert into pba values(13); mysql> insert into pba values(14);	mysql> select *from pba; mysql> release savepoint s2; mysql> select *from pba; mysql> rollback to savepoint s2; ERROR 1305 (42000): SAVEPOINT s2 does not exist

SQL
ORACLE MISCELLANEOUS SINGLE ROW FUNCTIONS

COALESCE: Coalesce function returns the first not null value in the expression list.
Example. The following query returns salary+commision, if commission is null then returns salary, if salary is also null then returns 1000.
select empno,ename,sal,comm,coalesce(sal+comm,sal,1000) “Net Sal” from emp;

ENAME	SALARY	COMM	NET SAL
SMITH	1000	100	1100
SAMI	3000		3000
SCOTT			1000
RAVI		200	1000

DECODE: DECODE(expr, searchvalue1, result1,searchvalue2,result2,..., defaultvalue)
Decode functions compares an expr with search value one by one. If the expr does not match any of the search value then returns the default value. If the default value is omitted then returns null.
Example: The following query returns the department names according the deptno. If the deptno does not match any of the search value then returns “Unknown Department”
select decode(deptno,10,'Sales',20,'Accounts’,30,'Production’, 40,'R&D','Unknown Dept') As DeptName from emp;

GREATEST: GREATEST(expr1, expr2, expr3,expr4...)
Returns the greatest expr from a expr list.
Example 1: select greatest(10,20,50,20,30) from dual;
Example 2: select greatest('SAMI','SCOTT','RAVI','SMITH','TANYA') from dual;

LEAST: LEAST(expr1, expr2, expr3,expr4...)**.** It is simillar to greatest. It returns the least expr from the expression list.
Example 1:
select least(10,20,50,20,30) from dual;
Example 2:
select least('SAMI','SCOTT','RAVI','SMITH','TANYA') from dual;

NVL: NVL2(expr1,expr2) **.**This function is oftenly used to check null values. It returns expr2 if the expr1 is null, otherwise returns expr1.
Example 1: The following example returns 100 because the first argument is not null.
SELECT NVL(100,200) FROM dual;
The following example returns N/A because the first argument is null:
SELECT NVL(NULL, 'N/A') FROM dual;

NVL2: NVL2(expr1,expr2,expr3)**.** NVL2 returns expr2 if expr1 is not null, otherwise return expr3.

A) Oracle NVL2() function with numeric data type example

The following statement returns two because the first argument is null.

```
SELECT NVL2(NULL, 1, 2) FROM dual;
```

B) Oracle NVL2() function with character data type example

The following example returns the second argument which is the ABC string because the first argument is not null.

```
SELECT NVL2(1, 'ABC', 'XYZ') FROM dual;
```

NULLIF: NULLIF(expr1, expr2). Nullif compares expr1 with expr2. If they are equal then returns null, otherwise return expr1.

For example, the following statement returns a null value because the first argument equals the second one.

```
SELECT NULLIF(100,100) FROM dual;
```

However, the following example returns the first value (100) because the two arguments are different:

```
SELECT NULLIF(100,200) FROM dual;
```

The following example causes an error because the first argument is literal NULL:

```
SELECT NULLIF(NULL,100) FROM dual;
```

The following statement also causes an error because the data types of arguments are different.

```
SELECT NULLIF(10,'20') FROM dual;
```

UID: Returns the current session ID of user logged on.

Example: select uid from dual;

USER: Returns the username of the current user logged on.

select user from dual;

SYS_CONTEXT: SYS_CONTEXT returns the value of parameter associated with the context namespace. You can use this function in both SQL and PL/SQL statements.

EXAMPLE: The following query returns the username of the current user.

```
Select sys_context('USERENV','SESSION_USER') “Username” from dual;
```

Similar to SESSION_USER parameter for namespace USERENV the other important parameters are:
ISDBA: To check whether the current user is having DBA privileges or not.
HOST: Returns the name of host machine from which the client is connected.
INSTANCE: The instance identification number of the current instance
IP_ADDRESS: IP address of the machine from which the client is connected.
DB_NAME :Name of the database as specified in the DB_NAME initialization parameter

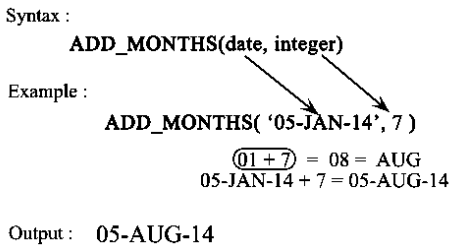
VSIZE: VSIZE(expr)**.** Returns the internal representation of expr in bytes.
Example: The following query return the representation of ename in bytes.
select ename,vsize(ename) as Bytes from emp;

ORACLE DATE FUNCTIONS

ADD_MONTHS()- function returns a date with a given number of months added (date plus integer months). A month is defined by the session parameter NLS_CALENDAR.

Applies to: Oracle 12c, Oracle 11g, Oracle 10g, Oracle 9i

Syntax: ADD_MONTHS(date, integer)



A) Add a number of months to a date

The following example adds 1 month to 29-FEB-2016:
SELECT ADD_MONTHS(DATE '2016-02-29', 1) FROM dual;

B) Add a negative number of months to a date: The following statement illustrates the effect of using a negative month for the ADD_MONTH() function:
SELECT ADD_MONTHS(DATE '2016-03-31', -1) FROM dual;

C) Get the last day of the last month: The following statement returns the last day of the last month.
SELECT LAST_DAY(ADD_MONTHS(SYSDATE , - 1)) FROM dual;

Oracle CURRENT_DATE: CURRENT_DATE returns the current date in the session time zone, in a value in the Gregorian calendar of datatype DATE.
Examples: The following statement changes the default date format to a new one that includes the time data:
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
To find out the session time zone, you use the SESSIONTIMEZONE function as follows:
SELECT SESSIONTIMEZONE FROM DUAL;

Currently, the session time zone is set to -07:00.To get the current date in the session time zone, you use the following statement:

```
SELECT CURRENT_DATE FROM DUAL;
```

If you change the session time zone, the value of the current date is adjusted accordingly as shown in the following example: First, set the session time zone to -09:00:

```
ALTER SESSION SET TIME_ZONE = '-09:00';
```

The new current date was adjusted as expected.

```
SELECT CURRENT_DATE, SESSIONTIMEZONE FROM DUAL;
```

CURRENT_TIMESTAMP: The Oracle CURRENT_TIMESTAMP function returns the current date and time in session time zone.

Noted that the CURRENT_TIMESTAMP function returns a value of TIMESTAMP WITH TIME ZONE while the CURRENT_DATE function returns a value of DATE without time zone data.

Examples: The following statement changes the format of timestamp values to include the time components:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
```

The following example shows the current timestamp in the session time zone:

```
SELECT CURRENT_TIMESTAMP FROM dual;
```

Let’s try to change the session time zone to the new one:

```
ALTER SESSION SET TIME_ZONE = '-08:00';
```

And get the current timestamp again:

```
SELECT CURRENT_TIMESTAMP FROM dual;
```

DBTIMEZONE: The Oracle DBTIMEZONE function returns the database time zone value.

Examples: To get the database time zone, you use the following statement:

```
SELECT DBTIMEZONE FROM dual;
```

If you want to change the database time zone, you use the ALTER DATABASE statement as follows:

```
ALTER DATABASE SET TIME_ZONE = 'Europe/London';
```

To make the new database time zone take effect, you need to bounce the database.

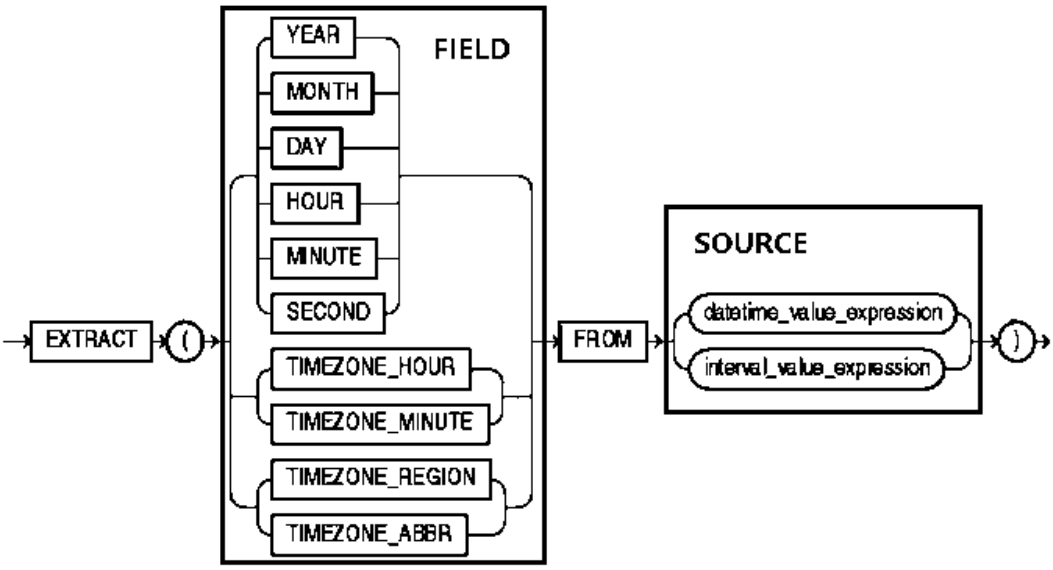
After bouncing the database, you can issue the following SQL statement to validate if the change has taken place:

```
SELECT DBTIMEZONE FROM dual;
```

EXTRACT: The Oracle EXTRACT() function extracts a specific component (year, month, day, hour, minute, second, etc.,) from a datetime or an interval value.

Syntax: The following illustrates the syntax of the Oracle EXTRACT() function:

EXTRACT(field FROM source)



A) Extracting fields from DATE values

You can extract YEAR, MONTH, DAY from a DATE value by using the EXTRACT() function.

The following example extracts the value of the YEAR field from a DATE value.

SELECT EXTRACT(YEAR FROM TO_DATE('31-Dec-1999 15:30:20 ', 'DD-Mon-YYYY HH24:MI:SS')) YEAR FROM DUAL;

In this example, we used the TO_DATE() function to convert a date literal to a DATE value.

Extracting month from a date:

SELECT EXTRACT(MONTH FROM TO_DATE('31-Dec-1999 15:30:20 ', 'DD-Mon-YYYY HH24:MI:SS')) MONTH FROM DUAL;

Extracting day from a date:

SELECT EXTRACT(DAY FROM TO_DATE('31-Dec-1999 15:30:20 ', 'DD-Mon-YYYY HH24:MI:SS')) DAY FROM DUAL;

To extract values of HOUR, MINUTE, and SECOND fields, you use TO_CHAR() function. For example, to extract hour, minute, and second of a current system date, you use the following statement:

SELECT TO_CHAR(SYSDATE, 'HH24') hour, TO_CHAR(SYSDATE, 'MI') minute, TO_CHAR(SYSDATE, 'SS') second FROM DUAL;

B) Extracting fields from INTERVAL YEAR TO MONTH values

For the INTERVAL YEAR TO MONTH, you can extract only YEAR and MONTH fields.
Suppose you have the following interval: INTERVAL '5-2' YEAR TO MONTH
It is 5 years 2 months.

To extract the value of the year field, you use the following statement:

```
SELECT EXTRACT( YEAR FROM INTERVAL '5-2' YEAR TO MONTH ) FROM DUAL;
```

The following example extracts the value of the month field from the interval:

```
SELECT EXTRACT( MONTH FROM INTERVAL '5-2' YEAR TO MONTH ) FROM DUAL;
```

C) Extracting fields from INTERVAL DAY TO SECOND values

For an INTERVAL DAY TO SECOND, you can extract DAY, HOUR, MINUTE, and SECOND as shown in the following example: Extract day from an interval

```
SELECT EXTRACT( DAY FROM INTERVAL '5 04:30:20.11' DAY TO SECOND ) FROM
dual;
```

Extract hour from an interval

```
SELECT EXTRACT( HOUR FROM INTERVAL '5 04:30:20.11' DAY TO SECOND ) FROM
dual;
```

Extract minute from an interval

```
SELECT EXTRACT( MINUTE FROM INTERVAL '5 04:30:20.11' DAY TO SECOND ) FROM
dual;
```

Extract second from an interval

```
SELECT EXTRACT( SECOND FROM INTERVAL '5 04:30:20.11' DAY TO SECOND ) FROM
dual;
```

D) Extracting fields from TIMESTAMP values: You can extract YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND from a TIMESTAMP value as shown in the following examples:

Extracting year from a timestamp:

```
SELECT EXTRACT( YEAR FROM TIMESTAMP '1999-12-31 23:59:59.10' ) FROM dual;
```

Extracting month from a timestamp:

```
SELECT EXTRACT( MONTH FROM TIMESTAMP '1999-12-31 23:59:59.10' ) FROM dual;
```

Extracting day from a timestamp:

```
SELECT EXTRACT( DAY FROM TIMESTAMP '1999-12-31 23:59:59.10' ) FROM dual;
```

Extracting hour from a timestamp:

```
SELECT EXTRACT( HOUR FROM TIMESTAMP '1999-12-31 23:59:59.10' ) FROM dual;
```

Extracting minute from a timestamp:
SELECT EXTRACT(MINUTE FROM TIMESTAMP '1999-12-31 23:59:59.10') FROM dual;
Extracting second from a timestamp:
SELECT EXTRACT(SECOND FROM TIMESTAMP '1999-12-31 23:59:59.10') FROM dual;

FROM_TZ: The Oracle FROM_TZ() function convert a timestamp and a time zone to value with the type of TIME STAMP WITH TIME ZONE.
Syntax: The following illustrates the syntax of the FROM_TZ() function:
FROM_TZ (timestamp, timezone)
Arguments: The Oracle FROM_TZ() function requires two arguments:
1) timestamp: The timestamp is a TIMESTAMP value which should be converted to a TIME STAMP WITH TIME ZONE value.
2) timezone: The timezone is a character string in the format TZH:TZM e.g., -08:00 or a character expression that evaluates to a string in TZR with optional TZD format.
Examples
The following example shows how to convert a timestamp and a time zone to a TIMESTAMP WITH TIME ZONE value:
SELECT FROM_TZ(TIMESTAMP '2017-08-08 08:09:10', '-07:00') FROM DUAL;

LAST_DAY: The Oracle LAST_DAY() takes a DATE argument and returns the last day of the month of that date.
Syntax: The following illustrates the syntax of the Oracle LAST_DAY() function:
LAST_DAY(date)
A) Get the last day of the current month: SELECT LAST_DAY(SYSDATE) FROM dual;
B) Calculate the number of days left of the current month:
SELECT LAST_DAY(SYSDATE) – SYSDATE FROM dual;
C) Return the last day of the last/next month:
The following example uses the ADD_MONTHS() function to add and subtract one month and returns the last day of the last and the next month:
SELECT LAST_DAY(ADD_MONTHS(SYSDATE,-1)) LAST_DAY_LAST_MONTH, LAST_DAY(ADD_MONTHS(SYSDATE,1)) LAST_DAY_NEXT_MONTH FROM dual;
D) Get the last day of February of the leap years:
The following example shows how to use the LAST_DAY() function to get the last day of the month February in both leap years and non-leap years:
SELECT LAST_DAY(DATE '2000-02-01') LAST_DAY_OF_FEB_2000, LAST_DAY(DATE '2016-02-01') LAST_DAY_OF_FEB_2016, LAST_DAY(DATE '2017-02-01') LAST_DAY_OF_FEB_2017 FROM dual;

LOCALTIMESTAMP: The Oracle LOCALTIMESTAMP function returns a TIMESTAMP value that represents the current date and time in the session time.
Notice that the LOCALTIMESTAMP function returns a TIMESTAMP value while the CURRENT_TIMESTAMP function returns a TIMESTAMP WITH TIME ZONE value.
Examples: The following statement shows the returned values of the LOCALTIMESTAMP and CURRENT_TIMESTAMP:

```
SELECT LOCALTIMESTAMP, CURRENT_TIMESTAMP FROM dual;
```

Let's change the session time zone to the new one:

```
ALTER SESSION SET TIME_ZONE = '-9:00';
SELECT LOCALTIMESTAMP, CURRENT_TIMESTAMP FROM dual;
```

MONTHS_BETWEEN: The Oracle MONTHS_BETWEEN() function returns the number of months between two dates.MONTHS_BETWEEN(minuend_date, subtrahend_date);
The Oracle MONTHS_BETWEEN() function requires two arguments, each of which can be a DATE or expression evaluates to a DATE:
1) minuend_date: The minuend_date is a date which is subtracted from.
2) subtrahend_date: The subtrahend is also a date which is to be subtracted.

The MONTHS_BETWEEN() function returns the number of months between two dates which is:

1. a positive integer if minuend_date is later than subtrahend_date.
2. a negative integer if minuend_date is earlier than subtrahend_date.
3. an integer if minuend_date and subtrahend_date are either the same days or they are both the last days of the months.

In other cases, the MONTHS_BETWEEN() function will calculate the fractional portion of the result based on the 31-day month and also consider the difference in time parts of minuend_date and subtrahend_date.
A) Get the difference in months of dates with the same day: The following example returns a difference in months between two dates: July 01 2017 and January 01 2017:

```
SELECT MONTHS_BETWEEN( DATE '2017-07-01', DATE '2017-01-01' ) MONTH_DIFF
FROM DUAL;
```

B) Both date arguments are the last days of the months: The following statement returns 1 month because both arguments are the last day of the months:

```
SELECT MONTHS_BETWEEN( DATE '2017-03-31', DATE '2017-02-28' ) MONTH_DIFF
FROM DUAL;
```

C) Both date arguments are not the same day: The following example returns a decimal because the days of the date arguments are not the same.

```
SELECT MONTHS_BETWEEN( DATE '2017-07-31', DATE '2017-08-15' ) MONTH_DIFF
FROM DUAL;
```

NEW_TIME: The Oracle NEW_TIME() function converts a date from a time zone to another. Note that before using this function, you must set the NLS_DATE_FORMAT parameter to display 24-hour time. Syntax: NEW_TIME(date, from_timezone, to_timezone)
The NEW_TIME() function accepts three arguments
1) date: A date whose time zone should be converted
2) from_timezone: A time zone of the date
3) to_timezone: A time zone to which the date should be converted

The following table illustrates the permitted values for from_timezone and to_timezone:

Time zone	Name
ADT	Atlantic Daylight Time
AST	Atlantic Standard Time
BDT	Bering Daylight Time
BST	Bering Standard Time
CDT	Central Daylight Time
CST	Central Standard Time
EDT	Eastern Daylight Time
EST	Eastern Standard Time
GMT	Greenwich Mean Time
HDT	Alaska-Hawaii Daylight Time
HST	Alaska-Hawaii Standard Time Time
MDT	Mountain Daylight Time
MST	Mountain Standard Time
NST	Newfoundland Standard Time
PDT	Pacific Daylight Time
PST	Pacific Standard Time
YDT	Yukon Daylight Time
YST	Yukon Standard Time

Examples: To demonstrate the NEW_TIME() function, first, we need to change the date format to 24-hour time format as follows:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';

SELECT NEW_TIME( TO_DATE( '08-07-2017 01:30:45', 'MM-DD-YYYY HH24:MI:SS' ),
'AST', 'PST' ) TIME_IN_PST FROM DUAL;
```

ROUND: The Oracle ROUND() function returns a date rounded to a specific unit.
Syntax: ROUND(date, format);
The Oracle ROUND() function takes two arguments:
1) date: The date argument is a DATE value (or an expression which resolves to a DATE value) that should be rounded.
2) format: The format argument is a string format the specifies which unit the date should be rounded to. The format argument is optional. If you omit it, the ROUND() function will round the date to the nearest day.

Format	**Description**
CC, SCC	Century, with or without minus sign (BC)
[S]YYYY, [S]YEAR, YYY, YY, Y	Year (in various appearances)
IYYY, IYY, IY, I	ISO year
Q	Quarter
MONTH, MON, MM, RM	Month (full name, abbreviated name, numeric, Roman numerals)
IW, WW (ISO)	week number
W	Day of the week
DDD, DD, J	Day (of the year/of the month/Julian day)
DAY, DY, D	Closest Sunday
HH, HH12, HH24	Hours
MI	Minutes

Rounding using default format: The following example rounds the date (20-Jul-2017 16:30:15) to the nearest day:

```
SELECT TO_CHAR( ROUND( TO_DATE( '20-Jul-2017 16:30:15',  'DD-Mon-YYYY HH24:MI:SS' ) ), 'DD-Mon-YYYY HH24:MI:SS' )  rounded_result FROM dual;
```

SESSIONTIMEZONE: The Oracle SESSIONTIMEZONE function returns the time zone of the current session. The Oracle SESSIONTIMEZONE function returns either a time zone offset, which is a character type in the format [+|-]TZH:TZM
e.g., -07:00 or a time zone region name e.g., America/Denver.

```
SELECT SESSIONTIMEZONE FROM dual;
```

TO_CHAR: The Oracle TO_CHAR() function converts a DATE or INTERVAL value to a string in a specified date format.
The Oracle TO_CHAR() function is very useful for formatting the internal date data returned by a query in a specific date format. Syntax: TO_CHAR(expr [, date_format] [, nslparam]);

The Oracle TO_CHAR() accepts three arguments:
1) expr: The expr is a DATE or an INTERVAL value that should be converted.
The data type of expr can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, or TIMESTAMP WITH LOCAL TIME ZONE.

2) date_format: The date_format is a string that determines the format that the result string should be in. The date_format argument is optional. If you omit it, the TO_CHAR() function will use the default date format for DATE values, default timestamp format for TIMESTAMP and TIMESTAMP WITH TIME ZONE value, and default timestamp with time zone format for TIMESTAMP WITH TIME ZONE values.
To compose value for the date_format argument, you use the Oracle date format model.

3) nlsparam: The nlsparam argument specifies the languages for names and abbreviations of day and month e.g., Monday, Mon, January, Jan, etc., in the result string.
The nlsparam argument has the following form:
'NLS_DATE_LANGUAGE = language'
This nlsparam argument is also optional. If you omit it, the TO_CHAR() function uses the default date language.

A) Convert current system date: The following statement converts the current system date to a string with the format YYYY-MM-DD:
SELECT TO_CHAR(sysdate, 'YYYY-MM-DD') FROM dual;
SELECT TO_CHAR(sysdate, 'DL') FROM dual;
SELECT TO_CHAR(sysdate, 'DL' , 'NLS_DATE_LANGUAGE = FRENCH') FROM dual;
B) Format an interval example: This example uses the TO_CHAR() function to format an interval:
SELECT TO_CHAR(INTERVAL '600' SECOND, 'HH24:MM') result FROM DUAL;

TO_DATE: The Oracle TO_DATE() function converts a date literal to a DATE value.
Syntax: TO_DATE (string, format, nls_language)
1) string: is a string value which is converted to a DATE value. It can be a value of any data type CHAR, VARCHAR2, NCHAR, or NVARCHAR2.

2) format: is the date and time format for the string. The format argument is optional. If you omit the format, the string must be in the standard date format which is DD-MON-YY e.g., 31-DEC-2000. Noted that if format is J, which is for Julian, then the string must be an integer.
For the detailed information on how to construct the format, check it out the Oracle date format.

3) nls_language: is an expression that specifies the language for day and month names in the string. This nls_language argument has the following form: NLS_DATE_LANGUAGE = language
This ls_language argument is optional. If you omit it, the TO_DATE() function will use the default language for your session.

SELECT TO_DATE('5 Jan 2017', 'DD MON YYYY') FROM dual;

B) Use TO_DATE() function to insert data into a table:

```
CREATE TABLE members (
   member_id NUMBER,
   first_name VARCHAR2 ( 50 ) NOT NULL,
   last_name VARCHAR2 ( 50 ) NOT NULL,
   joined_date DATE NOT NULL,
   PRIMARY KEY ( member_id )
);

INSERT INTO members(member_id,first_name, last_name, joined_date)
VALUES(1,'Laureen','Davidson', TO_DATE('Feb 01 2017','Mon DD YYYY'));

SELECT * FROM members;
```

……..

TRUNC: The Oracle TRUNC() function returns a DATE value truncated to a specified unit.
Syntax: TRUNC(date, format)
The TRUNC() function accepts two arguments:
1) date: The date argument is a DATE value or an expression that evaluates to a DATE value that will be truncated.
2) format: The format argument determines the unit to which the date will be truncated.
The format argument is optional. Its default value is DD that instructs the TRUNC() function to truncate the date to midnight.

A) Truncate a date value using default format: The following statement truncates the date value to midnight:

```
SELECT TO_CHAR( TRUNC(TO_DATE( '04-Aug-2017 15:35:32 ', 'DD-Mon-YYYY
HH24:MI:SS' )), 'DD-Mon-YYYY HH24:MI:SS' ) result FROM dual;
```

B) Get the first day of the month of a date: The following statement returns the first day of the current month.

```
SELECT TRUNC( SYSDATE, 'MM' ) result FROM dual;
```

C) Get the first day of the quarter of a date:

```
SELECT TRUNC( SYSDATE, 'Q' ) result FROM  dual;
```

TZ_OFFSET: The Oracle TZ_OFFSET() function returns the time zone offset from UTC of a valid time zone name or the SESSIONTIMEZONE or DBTIMEZONE function name.
The TZ_OFFSET() function accepts one argument which can be a valid time zone name e.g., 'Europe/London', a function name of SESSIONTIMEZONE or DBTIMEZONE, or a time zone offset from UTC (which simply returns itself).

A) Get time zone offset of a specific time zone: The following statement returns the time zone offset of the Europe/London time zone from UTC:

```
SELECT TZ_OFFSET( 'Europe/London' ) FROM DUAL;
```

B) Get time zone offset of the database time zone: The following example shows the time zone offset of the database time zone from UTC:

```
SELECT TZ_OFFSET( DBTIMEZONE ) FROM DUAL;
```

C) Get time zone offset of the session time zone: The following statement returns the time zone offset of the session time zone from UTC:

```
SELECT TZ_OFFSET( SESSIONTIMEZONE ) FROM DUAL;
```

ASCII: The Oracle ASCII function returns an ASCII code value of a character or character expression. Syntax: ASCII(character_expression)
The ASCII() function accepts one argument: character_expression is a character or character expression.
Return values: 1.The ASCII() function returns an integer that represents the ASCII code value of the character_expression.
2. If the character_expression consists of more than one character, the ASCII() function will return the ASCII code of the first character only.
3. The ASCII() function returns NULL if the character_expresion is NULL.

```
SELECT ASCII( 'A' ), ASCII( 'B' ), ASCII( 'C' ) FROM dual;

SELECT ASCII( 'ABC' ) FROM dual;

SELECT ASCII( NULL ) FROM dual;
```

CHR FUNCTION: The OracleCHR() function converts an ASCII code, which is a numeric value between 0 and 225, to a character. The Oracle CHR() function is the opposite of the ASCII() function.

```
SELECT CHR( 83 ), CHR( 115 ) FROM dual;
SELECT CHR( NULL ) FROM DUAL;
```

DUMP: The Oracle DUMP() function allows you to find the data type, length, and internal representation of a value.

The following illustrates the syntax of the DUMP() function:

```
DUMP ( expression [, return_format] [, start_position] [, length] )
```

The DUMP() function takes four arguments

Expression: Specifies an expression to be evaluated. It can be a column or an expression.

return_format: Determines the format of the returned value. The return_format accepts one of the following values: If you don't specify the return_format, the DUMP() function will return the internal representation of the expression in decimal format (or 10).

start_position: Specifies the starting position in the expression for which to return the internal representation.

Length: Specifies the length, from the start_position, in the expression for which to return the internal representation.

1) Basic Oracle DUMP() function examples: The following example uses the DUMP() function to display the type, length, and internal representation of the string 'Oracle DUMP':

```
SELECT DUMP('Oracle DUMP') AS result FROM DUAL;
```

```
RESULT
-----------------------------------------------
Typ=96 Len=11: 79,114,97,99,108,101,32,68,85,77,80
```

In this result:

The Typ=96: 96 denotes the CHAR data type. (See the table for the mapping of the internal numbers and their corresponding data types)

Len=11: the length of the string is 11.

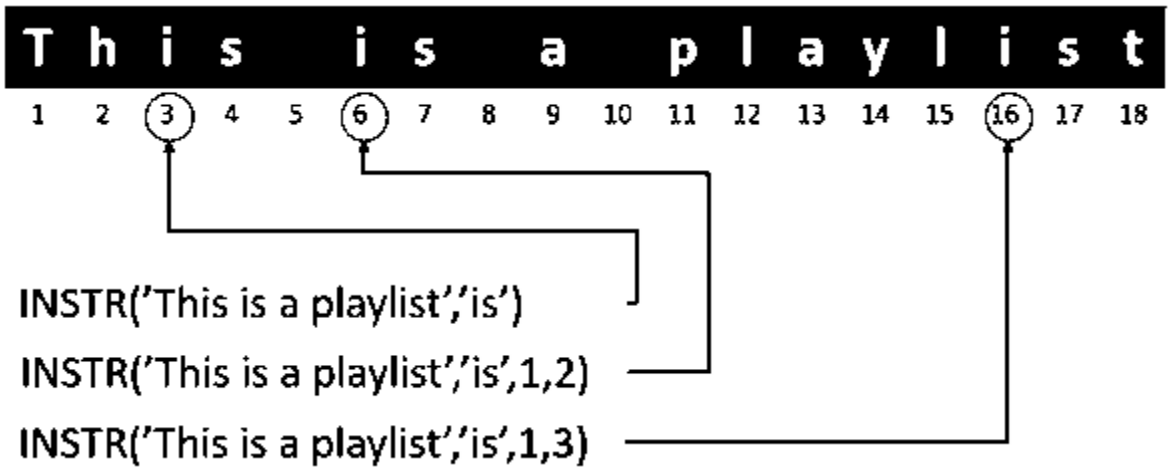
79,114,97,99,108,101,32,68,85,77,80 is the decimal (or ASCII) representation of the string 'Oracle DUMP' e.g., ASCII of O is 79, r is 114, and so on.

2) To display actual characters of the string, you use return format as 17:

```
SELECT DUMP('Oracle DUMP',17) AS result FROM DUAL;
```

INITCAP: The Oracle INITCAP() function converts the first letter of each word to uppercase and other letters to lowercase.
By definition, words are sequences of alphanumeric characters delimited by a space or any other non-alphanumeric letter.
SELECT INITCAP('hi john') FROM DUAL;

INSTR: The Oracle INSTR() function searches for a substring in a string and returns the position of the substring in a string. Syntax: INSTR(string , substring [, start_position [, occurrence]])
The Oracle INSTR() function accepts four arguments:
String: is the string or character expression that contains the substring to be found.
Substring: is the substring to be searched
start_position: is a nonzero integer that specifies where in the string the INSTR() function begins to search. The start_position is calculated using characters as defined by input character set.
If the start_position is positive, then INSTR() function searches and counts forward from the beginning of the string. In case the start_position is negative, the INSTR() function will search and count backward from the end of the string.
The start_position is an optional parameter. The default value of the start_position is 1. It means that, by default, the INSTR() function searches from the begining of the string.
Occurrence: is an positive integer that specifies which occurrence of the substring for which the INSTR() function should search. The occurence is optional and its default value is 1, meaning that the INSTR() funtion searches for the first occurrence of the substring by default.



1) Search from the start of the string: The following statement returns the location of the first occurrence of the is substring in This is a playlist, starting from position 1 (the first character) in the string.
SELECT INSTR('This is a playlist', 'is') substring_location FROM dual;

2) Search for the 2nd and 3nd occurrence of a substring: The following statement returns the location of the 2nd and 3rd occurrences of the substring is in This is a playlist.
SELECT INSTR('This is a playlist', 'is', 1, 2) second_occurrence, INSTR('This is a playlist', 'is', 1, 3) third_occurrence FROM dual;

3) Search for a substring that does not exist in a string: The following example illustrates the result when the substring are is not found in the searched string:
SELECT INSTR('This is a playlist', 'are') substring_location FROM dual;

4) Search backward: The following example searches the first occurrence of the substring is backward from the end of the searched string.
SELECT INSTR('This is a playlist', 'is',-1) substring_location FROM dual;

LENGTH: The Oracle LENGTH() function returns the number of characters of a specified string. It measures the length of the string in characters as defined by the input character set.
SELECT 'Oracle LENGTH' string, LENGTH('Oracle LENGTH') Len FROM dual;

CONCAT: The Oracle CONCAT() function concatenates two strings and returns the combined string.
The CONCAT() function accepts two arguments whose data types can by any of the data types CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.
The CONCAT() function returns a string whose character set depends on the character set of the first string argument.

SELECT CONCAT('Happy',' coding') FROM dual;

SELECT CONCAT(CONCAT('Happy', ' coding'), ' together') FROM dual;

SELECT 'Happy' || ' coding' || ' together' FROM dual;

LOWER: The Oracle LOWER() function converts all letters in a string to lowercase.
SELECT LOWER('PBA INSTITUTE') FROM dual;

LPAD: The Oracle LPAD() function returns a string left-padded with specified characters to a certain length.Syntax: LPAD(source_string, target_length [,pad_string]);
The Oracle LPAD() function takes three arguments:
1) source_string: is the string that will be padded from the left end.

2) target_length: is the length of the result string after padding.
Note that if the target_length is less than the length of the source_string, then LPAD() function will shorten down the source_string to the target_length without doing any padding.
3) pad_string: is the string to be padded.The pad_string argument is optional. If you don't specify it explicitly, the LPAD() function will use a single space for padding.

Example 1: The following example pads a string with the character (*) to a length of 5:

```
SELECT LPAD( 'ABC', 5, '*' ) FROM dual;
```

Example 2: In this example, the source string 'ABC' has length 3, therefore, only two more characters need to be padded to make the length of the result string 5.

```
SELECT LPAD( 'ABCDEF', 5, '*' ) FROM dual;
```

Example 3: LPAD() function to add leading zeros to format numeric strings. See the following statement:

```
SELECT LPAD( '123', 8, '0' ) RESULT FROM dual
UNION
SELECT LPAD( '7553', 8, '0' ) FROM dual
UNION
SELECT LPAD( '98753', 8, '0' ) FROM dual
UNION
SELECT LPAD( '754226', 8, '0' ) FROM dual;
```

LTRIM : Oracle LTRIM() function removes from the left-end of a string all characters contained in a set. Syntax: LTRIM(trim_source,[set])
The LTRIM() function accepts two arguments:
1) trim_source: is the string that unwanted characters should be removed.
2) set: is a set that contains one or more characters which should be removed from the trim_source string.
The set argument is optional. If you don't specify it, the set will default to a single space.
The data types of both trim_source and set can be one of the following types CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.
Example 1: SELECT LTRIM(' XYZ') FROM dual;

Example 2: The following example illustrates how to remove characters in a set of (1,2,3) from the left end of the string 'XYZ123456'

```
SELECT LTRIM( '123456XYZ', '123' ) FROM dual;
```

REGEXP_COUNT: The REGEXP_COUNT() function complements the functionality of the REGEXP_INSTR() function by returning the number of times a pattern occurs in a string.
Syntax: REGEXP_COUNT(string, pattern, position, match_parameter)
Here is the detail of each argument: string (mandatory): Is the input string to search for the pattern.
pattern (mandatory): Is a regular expression to be matched.
The maximum size of the pattern is 512 bytes. The function will convert the type of the pattern to the type of the string if the types of pattern and string are different.
position (optional): Is a positive integer that determines the beginning position in the string which the function starts the search. The default is 1, meaning that the function starts searching at the beginning of the string.
match_parameter (optional): Specify the matching behavior of the function. The match_parameter accepts the values listed in the following table:

Value	Description
'c'	Performs case-sensitive matching.
'i'	Performs case-insensitive matching.
'n'	Allows the period (.), which is the match-any-character character, to match the newline character. If you skip this parameter, then the period (.) does not match the newline character.
'm'	The function treats the string as multiple lines. The function interprets the caret (^) and the dollar sign ($) as the start and end, respectively, of any line anywhere in the string, rather than only at the start or end of the entire string. If you skip this parameter, then function treats the source string as a single line.
'x'	Ignores whitespace characters. By default, whitespace characters match themselves.

Example 1: This example uses the REGEXP_COUNT() function to return the number of numbers in the string.

```
SELECT REGEXP_COUNT('An apple costs 50 cents, a banana costs 10 cents.','\d+') result FROM dual;
```

Example 2: Match on Single Character: Let's count the number of times the character 't' appears in a string.

```
SELECT REGEXP_COUNT ('Timeofindia is a great resource', 't') FROM dual;

SELECT REGEXP_COUNT ('Timeofindia is a great resource', 't', 1, 'i') FROM dual;
```

Example 3: Match on Multiple Characters

```
SELECT REGEXP_COUNT ('The example shows how to use the REGEXP_COUNT function', 'the', 1, 'i') FROM dual;
```

SQL

Example 4 - Match on more than one alternative

SELECT REGEXP_COUNT ('Anderson', 'a|e|i|o|u') FROM dual;

SELECT REGEXP_COUNT ('Anderson', 'a|e|i|o|u', 1, 'i') FROM dual;

REGEXP_REPLACE: The Oracle REGEXP_REPLACE() function replaces a sequence of characters that matches a regular expression pattern with another string.

Syntax: REGEXP_REPLACE (source_string, search_pattern [, replacement_string [, star_position [, nth_occurrence [, match_parameter]]]])

The REGEXP_REPLACE() function takes 6 arguments:

1) source_string: is the string to be searched for.

2) search_pattern: is the regular expression pattern for which is used to search in the source string.

3) replacement_string: is the string that replaces the matched pattern in the source string. This argument is optional and its default value is null.

4) start_position: is an integer that determines the position in the source string where the search starts. The start_position is also optional. If not mentioned, the start_position is 1, which is the beginning position of the source string.

5) nth_occurrence: is a non-positive integer that indicates which position the replacement should take place. If nth_position is 0, the REGEXP_REPLACE() function will replace all occurrences of the match. Otherwise, the REGEXP_REPLACE() function will replace the nth occurrence.

6) match_parameter: is a literal string that changes the default matching behavior of the function. The behavior of the match_parameter in this function is the same for in the REGEXP_SUBSTR() function. Refer to REGEXP_SUBSTR() function for detailed information.

A) Removing special characters from a string: Sometimes, your database may contain special characters. The following statement uses the REGEXP_REPLACE() function to remove special characters from a string:

SELECT REGEXP_REPLACE('Th♥is∞ is a dem☻o of REGEXP_♫REPLACE function','[^a-z_A-Z]') FROM dual;

[^ **Negated set.** Match any character that is not in the set.

a-z **Range.** Matches a character in the range "a" to "z" (char code 97 to 122). Case sensitive.

_ **Character.** Matches a "_" character (char code 95).

A-Z **Range.** Matches a character in the range "A" to "Z" (char code 65 to 90). Case sensitive.

Character. Matches a SPACE character (char code 32).

]

B) Masking sensitive information:

The following statement hides the middle part of a credit card for security purposes. You can apply this technique in E-commerce, Banking, and other Financial applications that require strict security.

SELECT regexp_replace('4024007187788590', '(^\d{3})(.*)(\d{4}$)', '\1**********\3') credit_card FROM dual;

(**Capturing group #1.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

^ **Beginning.** Matches the beginning of the string, or the beginning of a line if the multiline flag (**m**) is enabled.

\d **Digit.** Matches any digit character (0-9).

{3} **Quantifier.** Match 3 of the preceding token.

)

(**Capturing group #2.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

. **Dot.** Matches any character except line breaks.

* **Quantifier.** Match 0 or more of the preceding token.

)

(**Capturing group #3.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

\d **Digit.** Matches any digit character (0-9).

{4} **Quantifier.** Match 4 of the preceding token.

$ **End.** Matches the end of the string, or the end of a line if the multiline flag (**m**) is enabled.

)

C) Removing redundant spaces: The following statement removes redundant spaces, the space character that appears more than one, in a string:

SELECT regexp_replace('This line contains more than one spacing between words', '( ){2,}', ' ') regexp_replace FROM dual;

(**Capturing group #1.** Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.

Character. Matches a SPACE character (char code 32).

)

{2,} **Quantifier.** Match 2 or more of the preceding token.

REGEXP_SUBSTR: The Oracle REGEXP_SUBSTR() function is an advanced version of the SUBSTR()function that allows you to search for substrings based on a regular expression. Instead of returning the position of the substring, it returns a portion of the source string that matches the regular expression.

Syntax: REGEXP_SUBSTR(source_string, pattern[, start_position [, occurrence
[, match_parameter [, subexpr]]]])

The Oracle REGEXP_SUBSTR() function accepts 6 arguments:

1) source_string: is a string to be searched for.

2) pattern: is the regular expression pattern that is used to search for in the source string.

3) start_position: is positive integer that indicates the starting position in the source string where the search begins.

The start_position argument is optional. Its default value is 1. Therefore, if you don't specify it explicitly, the REGEXP_SUBSTR() function will start searching at the beginning of the source string.

4) occurrence: is a positive integer that specifies which occurrence of the search pattern that the REGEXP_SUBSTR() function should search for.The occurrence argument is also optional and it defaults to 1, meaning that the REGEXP_SUBSTR() function should search for the first occurrence of the pattern in the source string.

5) match_parameter: is a literal string that determines the default matching behavior for the REGEXP_SUBSTR() function.

You can use one or more following values for the match_parameter argument:

'i' indicates case-insensitive matching

'c' indicates case-sensitive matching.

'n' allows the period (.) character to match the newline character. If you don't explicitly specify this parameter, the REGEXP_SUBSTR() function will not use the period to match the newline character.

'm' treats the source string as a multiline string.

Because the match_parameter argument is optional, therefore, if you omit it, the REGEXP_SUBSTR() function will behave as follows:

Case sensitivity matching is determined by NLS_SORT parameter.

The period (.) does not match the newline character.

The source string is treated as a single line.

6) subexpr: is a positive integer whose value from 0 to 9 that indicates which sub-expression in the regular expression is the target.

Example: If you want to get the fourth word of the above string, use the REGEXP_SUBSTR() function as follows:

```
SELECT regexp_substr( 'This is a regexp_substr demo', '[[:alpha:]]+', 1, 4 ) the_4th_word
FROM dual;
```

REPLACE: The Oracle REPLACE() function replaces all occurrences of a specified substring in a string with another.

```
SELECT REPLACE( 'This is a test', 'is', 'IS' ) FROM dual;
```

RPAD: The Oracle RPAD() function returns a string right-padded with specified characters to a certain length. Syntax : RPAD(source_string, target_length [,pad_string]);

The Oracle RPAD() function accepts three arguments:

1) source_string: is the string that will be padded from the right end.

2) target_length: is the length of the result string after padding.

Note that if the target_length is less than the length of the source_string, then RPAD() function will shorten down the source_string to the target_length without doing any padding.

3) pad_string: is a string to be padded.The pad_stringis optional and it defaults to a single space if you don't specify it explicitly.

```
SELECT RPAD( 'XYZ', 6, '+' ) FROM dual;

SELECT RPAD( 'Testing', 4, '-' ) FROM dual;
```

In this statement, the length of the source string 'Testing' is 7 while the target length is 4. So the RPAD() function truncates 3 characters right end of the source string which results in the following string: Test

RTRIM: Oracle RTRIM() function removes all characters that appear in a specified set from the right end of a string. Syntax: RTRIM(trim_source,[set])

The RTRIM() function accepts two arguments:

1) trim_source: is the string which the characters that appear in the set will be removed.

2) set: is one or more characters that should be removed from the right end of the trim_source string

The set argument is optional. If you omit it when calling the RTRIM() function, it will default to a single space. In other words, the RTRIM() function will remove the spaces from the right end of the trim_source by default.The data types of both trim_source and set can be one of the following data types: CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.

```
SELECT RTRIM( 'ABC  ' ) FROM dual;
```

SELECT RTRIM('ABC12345543', '345') FROM dual; [For every character in the set ('3','4','5'), the RTRIM() function removed the right-most occurrences of each from the string.]

SOUNDEX: The SOUNDEX() function returns a string that contains the phonetic representation of a string. Syntax: SOUNDEX(expression)
The SOUNDEX() function will return a string, which consists of four characters, that represents the phonetic representation of the expression.
The SOUNDEX() function is useful for comparing words that sound alike but spelled differently in English.
Basic Oracle SOUNDEX() example: This example uses the SOUNDEX() function to return the Soundex of the word 'sea' and 'see'.

```
SELECT SOUNDEX('see') see, SOUNDEX('sea') sea FROM dual;
```

Both words sound the same, they should receive the same Soundex value.

```
SELECT SOUNDEX('see') see, SOUNDEX('power') power FROM dual;
```

TRUNCATE TABLE CASCADE IN ORACLE DATABASE 12C

```
CREATE TABLE t1 (id NUMBER, description VARCHAR2(50), CONSTRAINT t1_pk
PRIMARY KEY (id) );

CREATE TABLE t2 (id NUMBER, t1_id   NUMBER, description VARCHAR2(50),
CONSTRAINT t2_pk PRIMARY KEY (id), CONSTRAINT t2_t1_fk FOREIGN KEY (t1_id)
REFERENCES t1 (id) ON DELETE CASCADE);

CREATE TABLE t3 (id NUMBER, t2_id NUMBER, description VARCHAR2(50),
CONSTRAINT t3_pk PRIMARY KEY (id), CONSTRAINT t3_t2_fk FOREIGN KEY (t2_id)
REFERENCES t2 (id) ON DELETE CASCADE );

-- Insert a data into each table.
INSERT INTO t1 VALUES (1, 't1 ONE');
INSERT INTO t2 VALUES (1, 1, 't2 ONE');
INSERT INTO t2 VALUES (2, NULL, 't2 TWO');
INSERT INTO t3 VALUES (1, 1, 't3 ONE');
INSERT INTO t3 VALUES (2, NULL, 't3 TWO');
COMMIT;

-- Check the contents of the tables.
SELECT (SELECT COUNT(*) FROM t1) AS t1_count,
```

```
(SELECT COUNT(*) FROM t2) AS t2_count,
(SELECT COUNT(*) FROM t3) AS t3_count FROM   dual;
```

DELETE ... [CASCADE]: The presence of the ON DELETE CASCADE relationships allows us to delete from any of the tables, with any dependent child records deleted automatically. The CASCADE keyword in the following delete example is not really necessary, but it's good to use it to remind any other developers that you are expecting a recursive delete.

```
DELETE FROM t1 CASCADE;

-- Check the contents of the tables.
SELECT (SELECT COUNT(*) FROM t1) AS t1_count,
(SELECT COUNT(*) FROM t2) AS t2_count,
(SELECT COUNT(*) FROM t3) AS t3_count FROM   dual;
```

Notice the rows with null values in the foreign key columns are not deleted, as strictly speaking they were not orphaned by the initial deletion.

TRUNCATE ... CASCADE: Rollback the previous deletion to return the data to its original state.

```
ROLLBACK;

-- Check the contents of the tables.
SELECT (SELECT COUNT(*) FROM t1) AS t1_count,
(SELECT COUNT(*) FROM t2) AS t2_count,
(SELECT COUNT(*) FROM t3) AS t3_count FROM   dual;

TRUNCATE TABLE t1 CASCADE;
-- Check the contents of the tables.
SELECT (SELECT COUNT(*) FROM t1) AS t1_count,
(SELECT COUNT(*) FROM t2) AS t2_count,
(SELECT COUNT(*) FROM t3) AS t3_count FROM   dual;
```

TRUNCATE TABLE vs DELETE TABLE: Both the statements will remove the data from the "customers" table but the main difference is that you can roll back the DELETE statement whereas you can't roll back the TRUNCATE TABLE statement.

Oracle Sequence: What is a sequence: A sequence is a list of integers in which their orders are important. For example, the (1,2,3,4,5) and (5,4,3,2,1) are totally different sequences even though they have the same members.
Creating a sequence: The CREATE SEQUENCE statement allows you to create a new sequence object in your own schema.
For example, this statement uses the CREATE SEQUENCE statement to create a new sequence object named item_seq:

```
CREATE SEQUENCE item_seq;
```

You use the sequence object to generate a sequence of unique integers, mostly for surrogate key columns.
Note that Oracle 12c automatically generates a sequence object associated with the identity column of the table.
1. Using a sequence: To access the next available value for a sequence, you use the NEXTVAL pseudo-column:

```
SELECT item_seq.NEXTVAL FROM dual;

SELECT item_seq.CURRVAL FROM dual;

SELECT item_seq.NEXTVAL FROM   dual CONNECT BY level <= 5;
```

2. Sequence Table example:

```
CREATE TABLE items (item_id NUMBER );
INSERT INTO items(item_id) VALUES(item_seq.NEXTVAL);
INSERT INTO items(item_id) VALUES(item_seq.NEXTVAL);
COMMIT;

SELECT item_id FROM items;
```

NEXT_DAY Function: NEXT_DAY returns the date of the first weekday named by char that is later than the date date. The return type is always DATE, regardless of the datatype of date. The argument char must be a day of the week in the date language of your session, either the full name or the abbreviation. The minimum number of letters required is the number of letters in the abbreviated version. Any characters immediately following the valid abbreviation are ignored. The return value has the same hours, minutes, and seconds component as the argument date.

```
SELECT NEXT_DAY('02-FEB-2001','TUESDAY') "NEXT DAY" FROM DUAL;
```

BIN_TO_NUM Function: BIN_TO_NUM converts a bit vector to its equivalent number. Each argument to this function represents a bit in the bit vector. This function takes as arguments any numeric datatype, or any nonnumeric datatype that can be implicitly converted to NUMBER. Each expr must evaluate to 0 or 1. This function returns Oracle NUMBER.
SELECT BIN_TO_NUM(1,0,1,0) FROM DUAL;

ASCIISTR Function: ASCIISTR takes as its argument a string, or an expression that resolves to a string, in any character set and returns an ASCII version of the string in the database character set. Non-ASCII characters are converted to the form \xxxx, where xxxx represents a UTF-16 code unit.
SELECT ASCIISTR('ABÄCDE') FROM DUAL;

COMPOSE Function: COMPOSE takes as its argument a string, or an expression that resolves to a string, in any datatype, and returns a Unicode string in its fully normalized form in the same character set as the input. char can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. For example, an o code point qualified by an umlaut code point will be returned as the o-umlaut code point.
SELECT COMPOSE ('o' || UNISTR('\0308')) FROM DUAL;

CONVERT Function: CONVERT converts a character string from one character set to another. The datatype of the returned value is VARCHAR2.
The char argument is the value to be converted. It can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.
The dest_char_set argument is the name of the character set to which char is converted.
The source_char_set argument is the name of the character set in which char is stored in the database. The default value is the database character set.
Both the destination and source character set arguments can be either literals or columns containing the name of the character set.
For complete correspondence in character conversion, it is essential that the destination character set contains a representation of all the characters defined in the source character set. Where a character does not exist in the destination character set, a replacement character appears. Replacement characters can be defined as part of a character set definition.

Common character sets include:
US7ASCII: US 7-bit ASCII character set
WE8DEC: West European 8-bit character set
F7DEC: DEC French 7-bit character set
WE8EBCDIC500: IBM West European EBCDIC Code Page 500
WE8ISO8859P1: ISO 8859-1 West European 8-bit character set

UTF8: Unicode 4.0 UTF-8 Universal character set, CESU-8 compliant
AL32UTF8: Unicode 4.0 UTF-8 Universal character set

SELECT CONVERT('Ä Ê Í Ó Ø A B C D E ', 'US7ASCII', 'WE8ISO8859P1') FROM DUAL;

DECOMPOSE Function: DECOMPOSE is valid only for Unicode characters. DECOMPOSE takes as its argument a string in any datatype and returns a Unicode string after decomposition in the same character set as the input. For example, an o-umlaut code point will be returned as the "o" code point followed by an umlaut code point.
string can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.
CANONICAL causes canonical decomposition, which allows recomposition (for example, with the COMPOSE function) to the original string. This is the default.
COMPATIBILITY causes decomposition in compatibility mode. In this mode, recomposition is not possible. This mode is useful, for example, when decomposing half-width and full-width katakana characters, where recomposition might not be desirable without external formatting or style information.
CLOB and NCLOB values are supported through implicit conversion. If char is a character LOB value, it is converted to a VARCHAR value before the COMPOSE operation. The operation will fail if the size of the LOB value exceeds the supported length of the VARCHAR in the particular development environment.
SELECT DECOMPOSE ('Châteaux') FROM DUAL;

NLSSORT Function: NLSSORT returns the string of bytes used to sort char.
Both char and 'nlsparam' can be any of the datatypes CHAR, VARCHAR2, NCHAR, or NVARCHAR2. The string returned is of RAW datatype.
The value of 'nlsparam' can have the form 'NLS_SORT = sort'
where sort is a linguistic sort sequence or BINARY. If you omit 'nlsparam', then this function uses the default sort sequence for your session. If you specify BINARY, then this function returns char.
If you specify 'nlsparam', then you can append to the linguistic sort name the suffix _ai to request an accent-insensitive sort or _ci to request a case-insensitive sort. Please refer to Oracle Database Globalization Support Guide for more information on accent- and case-insensitive sorting.
This function does not support CLOB data directly. However, CLOBs can be passed in as arguments through implicit data conversion.

CREATE TABLE test (name VARCHAR2(15));	INSERT INTO test VALUES ('Gaardiner'); INSERT INTO test VALUES ('Gaberd'); INSERT INTO test VALUES ('Gaasten');

```
SELECT * FROM test ORDER BY name;

SELECT * FROM test ORDER BY NLSSORT(name, 'NLS_SORT = XDanish');

SELECT * FROM test WHERE NLSSORT(name, 'NLS_SORT = XDanish') >
NLSSORT('Gaberd', 'NLS_SORT = XDanish');
```

USERENV Function: USERENV returns information about the current session. This information can be useful for writing an application-specific audit trail table or for determining the language-specific characters currently used by your session. You cannot use USERENV in the condition of a CHECK constraint. Table 5-13 describes the values for the parameter argument.
All calls to USERENV return VARCHAR2 data except for calls with the SESSIONID and ENTRYID parameters, which return NUMBER.

```
SELECT USERENV('LANGUAGE') "Language" FROM DUAL;
```

MySQL 8 Temporary Table: MySQL has a feature to create a special table called a Temporary Table that allows us to keep temporary data. We can reuse this table several times in a particular session. It is available in MySQL for the user from version 3.23, and above so if we use an older version, this table cannot be used. This table is visible and accessible only for the current session. MySQL deletes this table automatically as long as the current session is closed or the user terminates the connection. We can also use the DROP TABLE command for removing this table explicitly when the user is not going to use it.
If we use a PHP script to run the code, this table removes automatically as long as the script has finished its execution. If the user is connected with the server through the MySQL client, then this table will exist until the user closes the MySQL client program or terminates the connection or removed the table manually.
A temporary table provides a very useful and flexible feature that allows us to achieve complex tasks quickly, such as when we query data that requires a single SELECT statement with JOIN clauses. Here, the user can use this table to keep the output and performs another query to process it.
A temporary table in MySQL has many features, which are given below:
1. MySQL uses the CREATE TEMPORARY TABLE statement to create a temporary table.
2. This statement can only be used when the MySQL server has the CREATE TEMPORARY TABLES privilege.
3. It can be visible and accessible to the client who creates it, which means two different clients can use the temporary tables with the same name without conflicting with each other. It is because this table can only be seen by that client who creates it. Thus, the user cannot create two temporary tables with the same name in the same session.

4. A temporary table in MySQL will be dropped automatically when the user closes the session or terminates the connection manually.
5. A temporary table can be created by the user with the same name as a normal table in a database. For example, if the user creates a temporary table with the name student, then the existing student table cannot be accessible. So, the user performs any query against the student table, is now going to refer to the temporary student table. When the user removes a temporary table, the permanent student table becomes accessible again.

mysql> CREATE TEMPORARY TABLE Students(student_name VARCHAR(40) NOT NULL, total_marks DECIMAL(12,2) NOT NULL DEFAULT 0.00, total_subjects INT UNSIGNED NOT NULL DEFAULT 0);

```
mysql> CREATE TEMPORARY TABLE Students( student_name VARCHAR(40) NOT NULL, total_marks DECIMAL(12,2) NOT NULL DEFAULT 0.00,
 total_subjects INT UNSIGNED NOT NULL DEFAULT 0);
Query OK, 0 rows affected (0.00 sec)

mysql>
```

mysql>INSERT INTO Students(student_name, total_marks, total_subjects) VALUES ('Joseph', 150.75, 2), ('Peter', 180.75, 2);

mysql> SELECT * FROM Students;

[It is to be noted that when we run a **SHOW TABLES** command, then our temporary table will not be shown on the list. Also, if we close the current session and then will execute the SELECT statement, we will get a message saying that no data available in the database, and even the temporary table will not exist.]

Normal Table to Temporary Table:

1. 1st Create customer & orders tables and insert the values.

2. Create Temp Table:

CREATE TEMPORARY TABLE temp_customers SELECT c.cust_name, c.city, o.prod_name, o.price FROM orders o INNER JOIN customer c ON c.cust_id = o.order_id ORDER BY o.price DESC;

mysql> SELECT * FROM temp_customers;

mysql> SELECT cust_name, prod_name, price FROM temp_customers;

How to Drop Temporary Table in MySQL:

MySQL allows us to remove the temporary table using the DROP TABLE statement. But, it''s a good practice to use the TEMPORARY keyword with the DROP TABLE statement. This keyword helps us to avoid the mistake of deleting a permanent table when the temporary table and permanent table have the same name in the current session. So, it is recommended to use the following query for removing the temporary table:

mysql> DROP TEMPORARY TABLE table_name;

MySQL View Theory: A view is a database object that has no values. Its contents are based on the base table. It contains rows and columns similar to the real table. In MySQL, the View is a virtual table created by a query by joining one or more tables. It is operated similarly to the base table but does not contain any data of its own. The View and table have one main difference that the views are definitions built on top of other tables (or views). If any changes occur in the underlying table, the same changes reflected in the View also.

Syntax: Following is the syntax to create a view in MySQL:

CREATE [OR REPLACE] VIEW view_name AS SELECT columns FROM tables [WHERE conditions];

MySQL Update VIEW:

Syntax: Following is the syntax used to update the existing view in MySQL:

ALTER VIEW view_name AS SELECT columns FROM table WHERE conditions;

Why we use View?

MySQL view provides the following advantages to the user:

1. Simplify complex query

It allows the user to simplify complex queries. If we are using the complex query, we can create a view based on it to use a simple SELECT statement instead of typing the complex query again.

2. Increases the Re-usability

We know that View simplifies the complex queries and converts them into a single line of code to use VIEWS. Such type of code makes it easier to integrate with our application. This will eliminate

the chances of repeatedly writing the same formula in every query, making the code reusable and more readable.
3. Help in Data Security
It also allows us to show only authorized information to the users and hide essential data like personal and banking information. We can limit which information users can access by authoring only the necessary data to them.
4.Enable Backward Compatibility
A view can also enable the backward compatibility in legacy systems. Suppose we want to split a large table into many smaller ones without affecting the current applications that reference the table. In this case, we will create a view with the same name as the real table so that the current applications can reference the view as if it were a table.

MySQL Number Format Function: FORMAT function in MySQL is used to formats the number as a format of "#, ###. ##", rounded it in certain decimal places. After formatting the number, it will return the value as a string.
This function is beneficial when we calculate values in the databases like inventory turnover, or the average net price of products. The calculated result is a decimal value with many decimal places. In such a case, it is required to format those numbers for user understandable.
Therefore, we use the following syntax of FORMAT function in MySQL FORMAT(N, D, locale);
N - It is a number that we want to format.
D - It is a number of decimal places that we want to round the number. If D is 0, the result returns a string without any places.
locale - It is an optional argument. It specifies the locale to use that determines the decimal point result, thousands of separators, and grouping between separators. By default, MySQL will use an en_us locale.
Examples:

1. mysql> SELECT FORMAT(11342.123456, 3);

2. mysql> SELECT FORMAT(18670.2021, 0);

3. mysql> SELECT FORMAT(19600.2021, 2, 'de_DE');

4. mysql> SELECT FORMAT(11342.123456, 1);

5. mysql> SELECT FORMAT(11342.123456, -2);

6. mysql> SELECT FORMAT(11342.123456, -4);

[de_DE locale uses (,) comma operator for decimal mark.]

SQL

MYSQL DATE AND TIME FUNCTIONS

ADDDATE() function: MySQL ADDDATE() adds a time value with a date. The DATE_ADD() is the synonym of ADDDATE().
Syntax: ADDDATE(date, INTERVAL expr unit), ADDDATE(expr,days)
Arguments:

date	A date value.
INTERVAL	Keyword.
expr	A date or datetime expression or a number.
unit	An unit, described in the following table.

Examples:

SELECT ADDDATE('2020-05-15', INTERVAL 10 DAY) as required_date;

SELECT ADDDATE('2020-05-15', INTERVAL 2 YEAR) as required_date;

SELECT ADDDATE('2020-05-15',16);

Unit Value	Expected expr Format
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

ADDTIME(): In MySQL the ADDTIME() returns a time or datetime after adding a time value with a time or datetime. Syntax : ADDTIME(expr1,expr2)

Example: SELECT ADDTIME('2008-05-15 13:20:32.50','2 1:39:27.50') as required_datetime;

CONVERT_TZ(): In MySQL the CONVERT_TZ() returns a resulting value after converting a datetime value from a time zone specified as the second argument to the time zone specified as the third argument. This function returns NULL when the arguments are invalid.
Syntax : CONVERT_TZ (dt, from_tz,to_tz)
Example: SELECT CONVERT_TZ('2008-05-15 12:00:00','+00:00','+10:00');

CURDATE(),CURRENT_DATE(): In MySQL the CURDATE() returns the current date in 'YYYY-MM-DD' format or 'YYYYMMDD' format depending on whether numeric or string is used in the function. Syntax : CURDATE(), CURRENT_DATE()
Example: SELECT CURDATE();

CURRENT_TIME: In MySQL the CURRENT_TIME() returns the current time in 'HH:MM:SS' format or HHMMSS.uuuuuu format depending on whether numeric or string is used in the function
Syntax: CURRENT_TIME(), CURRENT_TIME
Example: SELECT CURRENT_TIME;

CURRENT_TIMESTAMP(): In MySQL the CURRENT_TIMESTAMP returns the current date and time in 'YYYY-MM-DD HH:MM:SS' format or YYYYMMDDHHMMSS.uuuuuu format depending on whether numeric or string is used in the function. Syntax : CURRENT_TIMESTAMP
Example: SELECT CURRENT_TIMESTAMP;

CURTIME(): In MySQL the CURRENT_TIME() returns the current time in 'HH:MM:SS' format or HHMMSS.uuuuuu format depending on whether numeric or string is used in the function
Example: SELECT CURTIME();

DATE_ADD(): MySQL DATE_ADD() adds time values (as intervals) to a date value. The ADDDATE() is the synonym of DATE_ADD(). Syntax : DATE_ADD(date,INTERVAL expr unit)
Example: SELECT DATE_ADD('2020-05-15', INTERVAL 10 DAY) as required_date;

TABLE OF FORMAT SPECIFIERS

Name	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%ac	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, …)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)

%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week
%u	Week (00..53), where Monday is the first day of the week
%V	Week (01..53), where Sunday is the first day of the week; used with %X
%v	Week (01..53), where Monday is the first day of the week; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal "%" character
%x	x, for any "x" not listed above

DATE_FORMAT(): MySQL DATE_FORMAT() formats a date as specified in the argument. The '%' is required before the format specifier characters. Syntax: DATE_FORMAT(date,format)

Example: To get week day name, day of the month with english suffix, month name and year in numeric %W %D %M %Y specifier can be used.

```
SELECT DATE_FORMAT('2008-05-15 22:23:00', '%W %D %M %Y');
```

Example: To get the time in 12-hour format followed by AM or PM %r specifier can be used

```
SELECT DATE_FORMAT('2008-05-15 22:23:00', '%r');
```

DATE_SUB(): MySQL DATE_SUB() function subtract a time value (as interval) from a date.
Syntax : DATE_SUB(date, INTERVAL expr unit)
Example: This example subtracting day from a data.
SELECT DATE_SUB('2008-05-15', INTERVAL 10 DAY);
Example: This example subtracting a date time from a data.
SELECT DATE_SUB('2008-05-15 4:50:20',INTERVAL '1 1:10:10' DAY_SECOND);

DATE(): MySQL DATE() takes the DATE part out from a DATEtime expression. Syntax : DATE_SUB(expr)
Example: This example extract the DATE portion from the specified DATEtime.
SELECT DATE('2008-05-17 11:31:31') as required_DATE;

DATEDIFF(): MySQL DATEDIFF() returns the number of days between two dates or datetimes. This function only calculates the date portion from each expression.
Syntax : DATEDIFF(expr1,expr2)
Example: This example returns the difference between two date.
SELECT DATEDIFF('2008-05-17 11:31:31','2008-04-28');

DAY(): MySQL DAY() returns the day of the month for a specified date. The day returned will be within the range of 1 to 31. If the given date is '0000-00-00', the function will return 0.
Example: SELECT DAY('2008-05-15');

DAYNAME(): MySQL DAYNAME() returns the name of the week day of a date,.
Example: SELECT DAYNAME('2008-05-15');

DAYOFMONTH(): MySQL DAYOFMONTH() returns the day of the month for a specified date.
Example: SELECT DAYOFMONTH('2008-05-15');

DAYOFWEEK(): MySQL DAYOFWEEK() returns the week day number (1 for Sunday,2 for Monday …… 7 for Saturday) for a specific date.
Example: SELECT DAYOFWEEK('2008-05-15');

DAYOFYEAR(): MySQL DAYOFYEAR() returns day of the year for a date. The return value is within the range of 1 to 366.
Example: SELECT DAYOFYEAR('2008-05-15');

EXTRACT(): MySQL EXTRACT() EXTRACTs a part of a given date. This function does not perform date arithmetic. Example: SELECT EXTRACT(YEAR FROM '2008-05-15');

Example: The following statement will EXTRACT the HOUR_SECOND part from 2008-05-15 15:53:20. SELECT EXTRACT(HOUR_SECOND FROM '2008-05-15 15:53:20');

Example: The following statement will EXTRACT the year and month part from 2008-05-15 15:53:20. SELECT EXTRACT(YEAR_MONTH FROM '2008-05-15 15:53:20')

FROM_DAYS(): MySQL FROM_DAYS() returns a date against a datevalue.
Example: The following statement will return a date against the datevalue 733910.
SELECT FROM_DAYS(733910);

FROM_UNIXTIME(): MySQL FROM_UNIXTIME() returns a date /datetime from a version of unix_timestamp. The return value format depending upon the context of the function (whether numeric or string). If specified, the result is formatted according to a given format string.
Syntax: FROM_ UNIXTIME (unix_timestamp, [format])
Example: The following statement will return a date time value from 1255033470.
SELECT FROM_UNIXTIME(1255033470);
Example: The following statement will return a date time value from 1255033470.
SELECT FROM_UNIXTIME(1255033470)+0;
Example: The following statement will return a date time value according to the format string %Y %D %M %h:%i:%s %x.
SELECT FROM_UNIXTIME(1255033470, '%Y %D %M %h:%i:%s %x');

GET_FORMAT(): MySQL GET_FORMAT() converts a date or time or datetime in a formatted manner. Syntax : GET_ FORMAT([date | time | datetime],[‘EUR’ |‘USA’ |’JIS’|’ISO’| ‘INTERNAL’])
Example: The following statement will arrange the date format in EUR.
SELECT GET_FORMAT(DATE,'EUR');
Example: The following statement will format and return the specified date 2009-05-18 in the format obtained from GET_FORMAT(DATE,'EUR').
SELECT DATE_FORMAT('2009-05-18',GET_FORMAT(DATE,'EUR'));
Example: The following statement will format and return the specified time 11:15:46 PM in a specific format as obtained from STR_TO_DATE(TIME,'USA').
SELECT STR_TO_DATE('11:15:46 PM',GET_FORMAT(TIME,'USA'));

HOUR(): MySQL HOUR() returns the HOUR of a time. The return value is within the range of 0 to 23 for time-of-day values. The range of time values may be larger than 23.
Example: The following statement return the HOUR from the given time 15:13:46.
SELECT HOUR('15:13:46');

LAST_DAY(): MySQL LAST_DAY() returns the last day of the corresponding month for a date or datetime value. If the date or datetime value is invalid, the function returns NULL.
Example: SELECT LAST_DAY('2009-05-18');

LOCALTIME(): MySQL LOCALTIME returns the value of current date and time in various format depending on the context of the function.
Example: SELECT LOCALTIME;

LOCALTIMESTAMP(): MySQL LOCALTIMESTAMP returns the value of current date and time in various format depending on the context of the function.
Example: SELECT LOCALTIMESTAMP;

MAKEDATE(): MySQL MAKEDATE() returns a date by taking a value of a year and a number of days. The number of days must be greater than 0 other wise a NULL will be returned.
Example: The statement below will make a date from year 2009 and number of days 138.
SELECT MAKEDATE(2019,138);

MAKETIME(): MySQL MAKETIME() makes and returns a time value from a given hour, minute and seconds. The value of hour may be greater than 24 but value of minutes and second greater than 59 return NULL.
Example: The statement below will make a time value from 15 hours, 25 minutes and 36 seconds.
SELECT MAKETIME(15,25,36);

MICROSECOND(): MySQL MICROSECOND() returns MICROSECONDs from the time or datetime expression. The return value is within the range of 0 to 999999.
Example: SELECT MICROSECOND('2019-05-18 10:15:21.000423');

MINUTE(): MySQL MINUTE() returns a MINUTE from a time or datetime value.
Example: SELECT MINUTE('2009-05-18 10:15:21.000423');

MONTH(): MySQL MONTH() returns the MONTH for the date within a range of 1 to 12 (January to December). It Returns 0 when MONTH part for the date is 0..
Example: SELECT MONTH('2009-05-18');

MONTHNAME(): MySQL MONTHNAME() returns the full name of the month for a given date. The return value is within the range of 1 to 12 (January to December). It Returns NULL when month part for the date is 0 or more than 12
Example: SELECT MONTHNAME('2019-05-18');

NOW(): MySQL NOW() returns the value of current date and time in 'YYYY-MM-DD HH: MM:SS' format or YYYYMMDDHHMMSS.uuuuuu format depending on the context (numeric or string) of the function.

Example: SELECT NOW();

Example: The following statement will return the date and current time in 'YYYY-MM-DD HH:SS:MM' format for the previous day. The keyword 'INTERVAL' have been introduced to get the result.

SELECT NOW(),NOW()-INTERVAL 1 DAY;

Example: The following statement will return the date and time in 'YYYY-MM-DD HH:SS:MM' format before 1 hour of current datetime.

SELECT NOW(),NOW()-INTERVAL 1 HOUR;

PERIOD_ADD(): MySQL PERIOD_ADD() adds a number of months with a period and returns the value in the format YYYYMM OR YYMM. Remember that the format YYYYMM and YYMM are not date values.

Example: The following statement will return a value in YYYYMM or YYMM format after adding 13 months with the period 200905.

SELECT PERIOD_ADD(200905,13);

PERIOD_DIFF(): MySQL PERIOD_DIFF() returns the difference between two periods. Periods should be in the same format i.e. YYYYMM or YYMM. It is to be noted that periods are not date values.

Example: The following statement will return a value in YYYYMM or YYMM format after calculating the difference between two periods 200905 and 200811.

SELECT PERIOD_DIFF(200905,200811);

QUARTER(): MySQL QUARTER() returns the quarter of the year for a date. The return value is in the range of 1 to 4. Syntax : QUARTER(date)

Example: The following statement will return a value between 1 to 4 as a QUARTER of a year for a given date 2009-05-18.

SELECT QUARTER('2009-05-18');

SEC_TO_TIME(): MySQL SEC_TO_TIME() returns a time value by converting the seconds specified in the argument. The return value is in hours, minutes and seconds. The range of the result is in the time data type.

Example: The following statement will return a time value after converting the seconds value 3610.

SELECT SEC_TO_TIME(3610);

SECOND(): MySQL SECOND() returns the second for a time. The return value is in the range of 0 to 59. Example: SELECT SECOND('21:29:46');

STR_TO_DATE(): MySQL STR_TO_DATE() returns a datetime value by taking a string and a specific format string as arguments. If the date or time or datetime value specified as string is illegal, the function returns NULL. Syntax : STR_TO_DATE(str,format)
Example: The following statement will return a valid date from the given string 18,05,2009 according to the format %d,%m,%Y.
SELECT STR_TO_DATE('18,05,2009','%d,%m,%Y');
Example: The following statement will return a valid date from the given string May 18, 2009 according to the format %M %d,%Y.
SELECT STR_TO_DATE('May 18, 2009','%M %d,%Y');
Example: The following statement will return a valid time from the given string 11:59:59 according to the format %h:%i:%s.
SELECT STR_TO_DATE('11:59:59','%h:%i:%s');

SUBDATE(): MySQL SUBDATE() subtracts a time value (as interval) from a given date. Syntax: SUBDATE(date, INTERVAL expr unit)
Example: The following statement will return a date after subtracting 10 days (notice that INTERVAL keyword is used) from the specified date 2008-05-15.
SELECT SUBDATE('2008-05-15', INTERVAL 10 DAY);
Example: The following statement will return a date after subtracting 10 days (notice that INTERVAL keyword is not used) from the specified date 2008-05-15.
SELECT SUBDATE('2008-05-15', 10);

SUBTIME(): MySQL SUBTIME() subtracts one datetime value from another. Syntax: SUBTIME(expr1,expr2);
Example: The following statement will return a datetime value between two datetimes 2009-05-18 10:29:43.999999 and 19 3:31:18.000002 specified in the arguments.
SELECT SUBTIME('2009-05-18 10:29:43.999999','19 3:31:18.000002');

SYSDATE(): MySQL SYSDATE() returns the current date and time in YYYY-MM-DD HH:MM:SS or YYYYMMDDHHMMSS.uuuuuu format depending on the context of the function.
Example: SELECT SYSDATE();

TIME_FORMAT(): MySQL TIME_FORMAT() converts a time in a formatted string using the format specifiers. Syntax : TIME_FORMAT(time,format).

Example: The following statement will convert a given time string 97:15:40 in to %H %k %h %I %l format.

```
SELECT TIME_FORMAT('97:15:40','%H %k %h %I %l');
```

TIME_TO_SEC(): MySQL TIME_TO_SEC() converts a time value in to seconds. Syntax : TIME_TO_SEC(tm)

Example: The following statement will convert the specified time 05:15:40 in seconds.

```
SELECT TIME_TO_SEC('05:15:40');
```

TIME(): MySQL TIME() extracts the time part of a time or datetime expression as string format.

Example: The following statement will return the time portion from the given date-time value 2009-05-18 15:45:57.005678.

```
SELECT TIME('2009-05-18 15:45:57.005678');
```

TIMEDIFF(): MySQL TIMEDIFF() returns the differences between two time or datetime expressions. It is to be noted that two expressions must be of same type. Syntax : TIMEDIFF(expr1,expr2)

Example: The following statement will return the difference between two datetime values 2009-05-18 15:45:57.005678 and 2009-05-18 13:40:50.005670.

```
SELECT TIMEDIFF('2009-05-18 15:45:57.005678', '2009-05-18 13:40: 50.005670');
```

TIMESTAMP(): MySQL TIMESTAMP() returns a datetime value against a date or datetime expression. If two arguments are used with this function, first it adds the second expression with the first and then returns a datetime. Syntax : TIMESTAMP(expr); TIMESTAMP(expr1,expr2)

Example: The following statement will return a datetime value for the given date expression 2009-05-18.

```
SELECT TIMESTAMP('2009-05-18');
```

TIMESTAMPADD(): MySQL TIMESTAMPADD() adds time value with a date or datetime value. The unit for interval as mentioned should be one of the following : FRAC_SECOND (microseconds), SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER or YEAR. Syntax: TIMESTAMPADD(unit,interval,datetime_expr)

Example: The following statement will return a date value after adding 2 months with the specified date 2009-05-18.

```
SELECT TIMESTAMPADD(MONTH,2,'2009-05-18');
```

TIMESTAMPDIFF(): MySQL the TIMESTAMPDIFF() returns a value after subtracting a datetime expression from another. Syntax :
TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)
Example: The following statement will return a value in months by subtracting 2009-05-18 from 2009-07-29. SELECT TIMESTAMPDIFF(MONTH,'2009-05-18','2009-07-29');

TO_DAYS(): MySQL TO_DAYS() returns number of days between a given date and year 0.
Example: The following statement will return the number of days from year 0 to 2009-05-18.
SELECT TO_DAYS('2009-05-18');

UNIX_TIMESTAMP(): MySQL UNIX_TIMESTAMP() returns a Unix timestamp in seconds since '1970- 01-01 00:00:00' UTC as an unsigned integer if no arguments are passed with UNIT_TIMESTAMP(). Example: The following statement will return the unix timestamp in seconds as an unsigned integer since '1970-01-01 00:00:00' UTC.
SELECT UNIX_TIMESTAMP();

UTC_DATE(): MySQL UTC_DATE returns the current UTC (Coordinated Universal Time) date as a value in 'YYYY-MM-DD' or YYYYMMDD format depending on the context of the function i.e. in a string or numeric context. Example: SELECT UTC_DATE,UTC_DATE();

UTC_TIME(): MySQL UTC_TIME returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS format depending on the context of the function i.e. in a string or numeric context.
Example: SELECT UTC_TIME,UTC_TIME();

UTC_TIMESTAMP(): In MySQL the UTC_TIMESTAMP returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format depending on the usage of the function i.e. in a string or numeric context.
Example: SELECT UTC_TIMESTAMP,UTC_TIMESTAMP();

WEEK(): MySQL WEEK() returns the week number for a given date. The argument allows the user to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If no argument is included with the function, it returns the default week format.
Example: The following statement will return the week of the specified date 2009-05-18.
SELECT WEEK('2009-05-18');
Example: The following statement will return the week of the specified date 2009-05-18. 1 as the second argument defines that the first day of the week is assumed as Monday.
SELECT WEEK('2009-05-18',1);

WEEKDAY(): MySQL WEEKDAY() returns the index of the day in a week for a given date (0 for Monday, 1 for Tuesday and .....6 for Sunday).
Example: The following statement will returns the index of the week for the date 2009-05-19.
SELECT WEEKDAY('2009-05-19');

WEEKOFYEAR(): MySQL WEEKOFYEAR() returns the calender week (as a number) of a given date. The return value is in the range of 1 to 53. Syntax: WEEKOFYEAR(date)
Example: The following statement will return the calender week of the specified date 2009-05-19.
SELECT WEEKOFYEAR('2009-05-19');

YEAR(): MySQL YEAR() returns the year for a given date. The return value is in the range of 1000 to 9999 or 0 for 'zero' date.
Example: SELECT YEAR('2009-05-19');

YEARWEEK(): MySQL YEARWEEK() returns year and week number for a given date.
Example: SELECT YEARWEEK('2009-05-18');

SPACE() function: MySQL SPACE() returns the string containing a number of spaces specified as an argument. Example: SELECT 'start', SPACE(10), 'end';

PASSWORD() function: MySQL password() returns a binary string from a plain text password. The function returns NULL if the string supplied as the argument was NULL. MySQL server uses this function to encrypt MySQL passwords for storage in the Password column of the user grant table. Example: SELECT PASSWORD('PBAINST');

MD5() function: MySQL MD5() Calculates an MD5 128-bit checksum for a string. The value is returned as a binary string of 32 hex digits, or NULL if the argument was NULL. The return value can, for example, be used as a hash key. Example: SELECT MD5('PBA INSTITUTE');

MYSQL STRING FUNCTIONS LIST

ASCII(): This function returns the numeric value of the leftmost character of the string str. Returns 0 if str is the empty string. Returns NULL if str is NULL. Syntax: ASCII(str)
Example 1: SELECT ASCII('2');
Example 2: SELECT ASCII(2);
Example 3: SELECT ASCII('An');

BIN(): Returns a string representation of the binary value of N, where N is a longlong (BIGINT) number. Returns NULL if N is NULL. Syntax : BIN(N). Example: SELECT BIN(12);

BIT_LENGTH(): Returns the length of the string str in bits. Syntax : BIT_LENGTH(str)
Example: SELECT BIT_LENGTH('text');

CHAR(): CHAR() interprets each argument N as an integer and returns a string consisting of the characters given by the code values of those integers. NULL values are skipped.
Syntax: CHAR(N,... [USING charset_name])
Example 1: SELECT CHAR(77,121,83,81,'76');
Example 2: SELECT CHAR(77,77.3,'77.3');

CHAR_LENGTH(): Returns the length of the string str, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five 2-byte characters, LENGTH() returns 10, whereas CHAR_LENGTH() returns 5. Syntax : CHAR_LENGTH(str)
Example: SELECT CHAR_LENGTH('shib shankar');

CONCAT(): Returns the string that results from concatenating one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent nonbinary string form. Syntax : CONCAT(str1,str2,...)
Example: SELECT CONCAT('pbainst','.','com');

CONCAT_WS(): CONCAT_WS() stands for Concatenate With Separator and is a special form of CONCAT(). The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is NULL, the result is NULL.
Syntax: CONCAT_WS(separator,str1,str2,...)
Example: SELECT CONCAT_WS(',','1st string','2nd string');

ELT(): ELT() returns the Nth element of the list of strings: str1 if N = 1, str2 if N = 2, and so on. Returns NULL if N is less than 1 or greater than the number of arguments. ELT() is the complement of FIELD(). Syntax : ELT(N,str1,str2,str3,...)
Example: SELECT ELT(4,'this','is','the','elt');

EXPORT_SET(): Returns a string such that for every bit set in the value bits, you get an on string and for every bit not set in the value, you get an off string.
Syntax: EXPORT_SET(bits,on,off[,separator[,number_of_bits]])
Example: SELECT EXPORT_SET(5,'Y','N',',',3);
5 BINARY = 101 => YNY

Binary Base = 2

	Column 8	Column 7	Column 6	Column 5	Column 4	Column 3	Column 2	Column 1
Base^{exp}	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Weight	128	64	32	16	8	4	2	1

FIELD(): Returns the index (position) of str in the str1, str2, str3, ... list. Returns 0 if str is not found. Syntax : FIELD(str,str1,str2,str3,...)
Example: SELECT FIELD('ank', 'b', 'ank', 'of', 'monk');

FIND_IN_SET(): Returns a value in the range of 1 to N if the string str is in the string list strlist consisting of N substrings. A string list is a string composed of substrings separated by ",", characters. Returns NULL if either argument is NULL. This function does not work properly if the first argument contains a comma (",") character. Syntax: FIND_IN_SET(str,strlist)
Example: SELECT FIND_IN_SET('ank','b,ank,of,monk');

HEX(): MySQL HEX() returns a string representation of hexadecimal value of a decimal or string value specified as argument. If the argument is a string, each character in the argument is converted to two hexadecimal digits. If the argument is decimal, the function returns a hexadecimal string representation of the argument , and treated as a longlong(BIGINT) number.
Syntax: HEX(str), HEX(N)
Example: SELECT HEX(157);

INSERT(): Returns the string str, with the substring beginning at position pos and len characters long replaced by the string newstr. Returns the original string if pos is not within the length of the string. Replaces the rest of the string from position pos if len is not within the length of the rest of the string. Syntax : INSERT(str,pos,len,newstr)
Example 1: SELECT INSERT('Originalstring', 4, 5, ' insert ');
Example 2: SELECT INSERT('Originalstring', -3, 5, ' insert ');

INSTR(): MySQL INSTR() takes a string and a substring of it as arguments, and returns an integer which indicates the position of the first occurrence of the substring within the string
Syntax : INSTR(str,substr)
Example: SELECT INSTR('myteststring','st');

LCASE(): MySQL LCASE() converts the characters of a string to lower case characters.
Syntax : LCASE(str)
Example: SELECT LCASE('PBAINST');

LEFT(): MySQL LEFT() returns a specified number of characters from the left of a given string. Both the number and the string are supplied in the arguments as str and len of the function.
Syntax: LEFT(str,len)
Example: SELECT LEFT('pba institute', 3);

LENGTH(): MySQL LENGTH() returns the length of a given string.
Syntax : LENGTH(str)
Example: SELECT LENGTH('text');

LOCATE(): MySQL LOCATE() returns the position of the first occurrence of a string within a string. Both of these strings are passed as arguments. An optional argument may be used to specify from which position of the string (i.e. string to be searched) searching will start. If this position is not mentioned, searching starts from the beginning.
Syntax : LOCATE(substr,str,pos)
Example: SELECT LOCATE('st','pbainst');

LOWER(): MySQL LOWER() converts all the characters in a string to lowercase characters.
Syntax : LOWER(str)
Example: SELECT LOWER('MYTESTSTRING');

LPAD(): MySQL LPAD() left pads a string with another string. The actual string, a number indicating the length of the padding in characters (optional) and the string to be used for left padding - all are passed as arguments. Syntax: LPAD(str,len,padstr)
Example 1: SELECT LPAD('Hello',10,'**');
Example 2: SELECT LPAD('hi',1,'**');

LTRIM(str): MySQL LTRIM() removes the leading space characters of a string passed as argument. Syntax : LTRIM(str)
Example: SELECT LTRIM(' Hello')

MAKE_SET(): MySQL MAKE_SET() returns a set value (a string containing substrings separated by “,” characters) consisting of the strings that have the corresponding bit in the first argument.
Syntax : MAKE_SET(bits,str1,str2,...)
MAKE_SET() function uses the binary representation of the first argument to return the applicable strings in the subsequent arguments.

```
SELECT BIN(1) AS '1', BIN(2) AS '2', BIN(3) AS '3', BIN(4) AS '4', BIN(5) AS '5', BIN(6) AS '6', BIN(7) AS '7', BIN(8) AS '8', BIN(9) AS '9', BIN(10) AS '10';

SELECT MAKE_SET(1, 'a','b','c','d') AS '1', MAKE_SET(2, 'a','b','c','d') AS '2', MAKE_SET(3, 'a','b','c','d') AS '3', MAKE_SET(4, 'a','b','c','d') AS '4', MAKE_SET(5, 'a','b','c','d') AS '5', MAKE_SET(6, 'a','b','c','d') AS '6', MAKE_SET(7, 'a','b','c','d') AS '7', MAKE_SET(8, 'a','b','c','d') AS '8',MAKE_SET(9, 'a','b','c','d') AS '9', MAKE_SET(10, 'a','b','c','d') AS '10';
```

BIN() function to return each number's binary value. We can see that the binary representation of 4 is 100. We need to visualize this backwards in order to apply it to our MAKE_SET() example above. In our case, this is a three-digit binary value, with the right-most digit corresponding to the first string, the next digit corresponds to the second string, and the leftmost digit corresponds with the third string. In binary terms, 1 is "on" and 0 is "off". The MAKE_SET() function only returns strings that have a corresponding 1 in their binary value. Therefore, our example above returns the third string.

Example 1: SELECT MAKE_SET(1,'a','b','c');
Example 2: SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world'); [Multiple Binary Values]
Example 3: SELECT MAKE_SET(5, 'Cat','Dog','Horse','Duck') Result;

MID(): MySQL MID() extracts a substring from a string. The actual string, position to start extraction and length of the extracted string - all are specified as arguments.
Syntax: MID(str,pos,len)
Example: SELECT MID('pbainst',4,3);

OCT(): Returns a string representation of the octal value of N, where N is a longlong (BIGINT) number. Returns NULL if N is NULL. Syntax : OCT(N)
Example: SELECT OCT(12);

ORD(): MySQL ORD() returns the code for the leftmost character if that character is a multi-byte (sequence of one or more bytes) one. If the leftmost character is not a multibyte character, ORD() returns the same value as the ASCII() function. Syntax : ORD(str)
Example: SELECT ORD("pbainst");

POSITION(): MySQL POSITION() returns the position of a substring within a string..
Syntax: POSITION(substr IN str)
Example: SELECT POSITION("i" IN "pbainst");

QUOTE(): Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotation marks and with each instance of backslash ("\"), single quote (""), ASCII NUL, and Control+Z preceded by a backslash. If the argument is NULL, the return value is the word "NULL" without enclosing single quotation marks.
Syntax : QUOTE(str)
Example 1: SELECT QUOTE('pba"inst');
Example 2: SELECT QUOTE('shib shankar ghosh');

REPEAT(): MySQL REPEAT() repeats a string for a specified number of times.
The function returns NULL either any either of the arguments are NULL.
Syntax: REPEAT(str,count) Example: SELECT REPEAT('**-',5);

REPLACE(): MySQL REPLACE() replaces all the occurrences of a substring within a string.
Syntax: REPLACE(str,from_str,to_str) Example: SELECT REPLACE('pbainst','inst','institute');

REVERSE(): Returns a given string with the order of the characters reversed.
Syntax: REVERSE(str) Example: SELECT REVERSE('shib shankar');

RIGHT(): MySQL RIGHT() extracts a specified number of characters from the right side of a given string. Syntax : RIGHT(str,len) Example: SELECT RIGHT('pbainstitute',8);

RPAD(): MySQL RPAD() function pads strings from right. The actual string which is to be padded as str, length of the string returned after padding as len and string which is used for padding as padstr is used as a parameters within the argument. Syntax: RPAD(str,len,padstr)
Example: SELECT RPAD('pbainst',15,'*');

RTRIM(): MySQL RTRIM() removes the trailing spaces from a given string.
Syntax: RTRIM(str) Example: SELECT RTRIM(' pbainst ');

SOUNDEX(): MySQL SOUNDEX() function returns soundex string of a string.
Soundex is a phonetic algorithm for indexing names after English pronunciation of sound. You can use SUBSTRING() on the result to get a standard soundex string. All non-alphabetic characters in str are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.
Syntax: SOUNDEX(str)
Example 1: SELECT SOUNDEX('Sure') AS Sure, SOUNDEX('Shore') AS Shore;
Example 2: SELECT SOUNDEX('Dam') AS Dam, SOUNDEX('Damn') AS Damn, SOUNDEX('Too') AS Too, SOUNDEX('Two') AS Two;

Example 3: create table student_info and insert the values.

```
Select * from student_info;
+------+---------+-----------+-----------+
| id   | Name    | Address   | Subject   |
+------+---------+-----------+-----------+
| 101 | YashPal  | Amritsar  | History   |
| 105 | Gaurav   | Chandigarh | Literature |
| 125 | Raman    | Shimla    | Computers |
+------+---------+-----------+-----------+
```

mysql> Select * from student_info where SOUNDEX(Name) LIKE '%G%';

SPACE(): MySQL SPACE() returns the string containing a number of spaces as specified in the argument. Syntax : SPACE(N) Example: SELECT 'start', SPACE(10), 'end';

SUBSTR(): MySQL SUBSTR() returns the specified number of characters from a particular position of a given string. SUBSTR() is a synonym for SUBSTRING().
Syntax : SUBSTR(str,pos,len) Example: SELECT SUBSTR('shib shankar',4,3);

SUBSTRING(): MySQL SUBSTRING() returns a specified number of characters from a particular position of a given string.Syntax : SUBSTRING(str,pos,len)
Example 1: SELECT SUBSTRING('pbainst',4,3);
Example 2: SELECT SUBSTRING('pbainstitute',5);
Example 3: SELECT SUBSTRING(' pbainstitute ',-5);

SUBSTRING_INDEX(): MySQL SUBSTRING_INDEX() returns the substring from the given string before a specified number of occurrences of a delimiter.
Returns from the left of the final delimiter if the number is positive and right of the final delimiter when the number is negative.
If the number is greater than the number of occurrence of delimiter, the returned substring will be the total string. If the specified number is 0, nothing will be fetched from the given string.
Syntax : SUBSTRING_INDEX(str,delim,count)
Example: SELECT SUBSTRING_INDEX('www.pbainst.info','.',2);

TRIM(): MySQL TRIM() function returns a string after removing all prefixes or suffixes from the given string.Syntax : TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str)
Example 1: SELECT TRIM(' trim ');
Example 2: SELECT TRIM(LEADING 'leading' FROM 'leadingtext');
Example 3: SELECT TRIM(BOTH 'leadtrail' FROM 'leadtrailtextleadtrail');

UNHEX(): MySQL UNHEX() function performs the opposite operation of HEX(). This function interprets each pair of hexadecimal digits (in the argument) as a number and converts it to a character. Syntax: UNHEX(str)
Example 1: SELECT UNHEX('4D7953514C');
Example 2: SELECT UNHEX(HEX('MySQL'));

UPPER(): MySQL UPPER() converts all the characters in a string to uppercase characters.
Syntax: UPPER(str) Example : SELECT UPPER('shibu');

SQL
TRANSCRIPT FUNCTIONS

MySQL BIT_AND() function: MySQL BIT_AND() function returns the bitwise AND of all bits in a given expression.The calculation is performed on 64 bit precession.
Syntax: BIT_AND(expr) Example: SELECT 2 & 15;

MySQL BIT_OR() function: MySQL BIT_OR() function returns the bitwise OR of all bits in a given expression. The calculation is performed on 64 bit precession. If this function does not find a matching row, it returns 0. Syntax: BIT_OR(expr) Example: SELECT 2 | 5;

MySQL BIT_XOR() function: MySQL BIT_XOR() function returns the bitwise XOR of all bits in a given expression. The calculation is performed on 64 bit precession.
Syntax: BIT_XOR(expr) Example: SELECT 2 ^ 15;

MySQL GROUP_CONCAT() function: MySQL GROUP_CONCAT() function returns a string with concatenated non- NULL value from a group. Returns NULL when there are no non-NULL values. Syntax: GROUP_CONCAT(expr);

Table: invoice

invoice_no	book_id	pub_lang	qty	rate	cost
INV0001	BK001	English	15	75	1125
INV0002	BK004	English	8	55	440
INV0003	BK005	NULL	20	20	400
INV0004	BK004	English	15	35	525
INV0005	BK001	English	8	25	200
INV0006	BK003	Hindi	20	45	900

Example: SELECT pub_lang,GROUP_CONCAT(book_id) FROM invoice GROUP BY pub_lang;

MySQL MAX() function: MySQL MAX() function returns the maximum value of an expression.
Example: SELECT pub_lang,MAX(qty) FROM invoice GROUP BY pub_lang HAVING MAX(qty)>=8;

MySQL STD() function: MySQL STD() function returns the population standard deviation of expression. It returns NULL if no matching row is found. Syntax: STD(expr);
Example: SELECT STD(cost) FROM invoice;

MySQL STDDEV_POP() function: MySQL STDDEV_POP() function returns the population standard deviation of an expression (the square root of VAR_POP()). It returns NULL if no matching row is found. Syntax : STDDEV_POP(expr);
Example: SELECT STDDEV_POP(cost) FROM invoice;

MySQL STDDEV_SAMP() function: MySQL STDDEV_SAMP() function returns the sample standard deviation of an expression (the square root of VAR_SAMP()). It returns NULL if no matching rows are found. Syntax : STDDEV_SAMP(expr);
Example: SELECT STDDEV_SAMP(cost) FROM invoice;

MySQL STDDEV() function: MySQL STDDEV() function returns the population standard deviation of expression. The STDDEV() function is used to calculate statistical information for a specified numeric field in a query. It returns NULL if no matching rows found.
Syntax: STDDEV(expr); Example: SELECT STDDEV(cost) FROM invoice;

MySQL VAR_POP() function: MySQL VAR_POP() function returns the population standard variance of an expression. Syntax: VAR_POP(expr)
Example: SELECT VAR_POP(cost) FROM invoice;

MySQL VAR_SAMP() function: MySQL VAR_SAMP() function returns the sample variance of an given expression. Syntax : VAR_SAMP(expr)
Example: SELECT VAR_SAMP(cost) FROM invoice;

MySQL VARIANCE() function: MySQL VARIANCE() function returns the population standard variance of an expression. Syntax: VARIANCE(expr)
Example: SELECT VARIANCE(cost) FROM invoice;

MYSQL 8 ON DELETE CASCADE

```
mysql> CREATE TABLE parent (id INT NOT NULL, PRIMARY KEY (id)) ENGINE=INNODB;

mysql> CREATE TABLE child (
id INT, parent_id INT,
INDEX par_ind (parent_id),
FOREIGN KEY (parent_id) REFERENCES parent(id)
ON DELETE CASCADE) ENGINE=INNODB;

mysql> insert into parent values(1);
mysql> insert into parent values(2);
mysql> insert into parent values(3);
mysql> insert into child values(899,1);
mysql> insert into child values(9666,1);
```

mysql> select *from child;
mysql> select *from parent;
mysql> delete from parent where id=1;
mysql> select *from parent;
mysql> select *from child;

ORACLE JOIN

Oracle Join Conditions: A join queries must have contained at least one join condition, either in the FROM clause or in the WHERE clause. The join condition compares two columns from two different tables. The Oracle Database combines pairs of rows, from each table, participating in joining, which are satisfying the join condition evaluates to TRUE.
A WHERE clause that contains a join condition can also contain other conditions that refer to columns of only one table. These conditions can further restrict the rows returned by the join query.

Oracle Types of JOIN: Equijoins**,** Non-Equi Joins**,** Self Joins**,** Cross Join / Cartesian Products**,** Inner Joins**,** Outer Joins, Left Outer Join, Right Outer Join, Full Outer Join, Natural Join, Antijoins, Semijoins

Equijoins: An equijoin is a join with a join condition containing an equality operator. This is represented by (=) sign. This join retrieves information by using equality condition.

TABLE: EMP_MAST

EMP_NO	EMP_NAME	JOB_NAME	MGR_ID	DEPT_NO
1234	Alex	Clerk	4567	15
2345	Jack	Consultant	3456	25
3456	Paul	Manager	1234	15
4567	Jenefer	Engineer	2345	45

TABLE: DEP_MAST

DEPT_NO	DEP_NAME	LOCATION
15	FINANCE	PARIS
25	MARKETING	LONDON
35	HR	DELHI

Example: SELECT EMP_NO,EMP_NAME,JOB_NAME,DEP_NAME FROM EMP_MAST E,DEP_MAST D WHERE E.DEPT_NO=D.DEPT_NO;

Non-Equi Join: An nonequi join is an inner join statement that uses an unequal operation (i.e.: <>, >, <, !=, BETWEEN, etc.) to match rows from different tables.
Example: SELECT EMP_NO,EMP_NAME,JOB_NAME,DEP_NAME FROM EMP_MAST E,DEP_MAST D WHERE E.DEPT_NO>D.DEPT_NO;

Self Joins: A self join is such a join in which a table is joined with itself. For example, when you require details about an employee and his manager (also an employee).
Example: SELECT a1.emp_no,a2.emp_name,a1.job_name,a2.dept_no FROM emp_mast a1,emp_mast a2 WHERE a1.emp_no=a2.mgr_id;

Cross Joins: A Cross Join or Cartesian join or Cartesian product is a join of every row of one table to every row of another table.
Example: SELECT emp_no,emp_name,job_name,dep_name,location FROM emp_mast CROSS JOIN dep_mast;

Inner Joins: An inner join is a join that returns rows of the tables that satisfy the join condition.
Example: SELECT emp_no,emp_name,job_name,dep_name,location FROM emp_mast INNER JOIN dep_mast USING(dept_no);

Outer Joins: An outer join is such a join which is similar to the equi join, but Oracle will also return non matched rows from the table.
left outer join displays all matching records of both table along with the records in left hand side table of join clause which are not in right hand side table of join clause.
Example1: SELECT emp_no,emp_name,job_name,dep_name,location FROM emp_mast e LEFT OUTER JOIN dep_mast d ON(e.dept_no=d.dept_no);
Example 2: SELECT emp_no,emp_name,job_name,dep_name,location FROM emp_mast e, dep_mast d WHERE e.dept_no=d.dept_no(+);

Right Outer Join: This right outer join displays all matching records of both tables along with the records in left hand side table of join clause which are not in right hand side table of join clause.
Example 1: SELECT emp_no,emp_name,job_name,dep_name,location FROM emp_mast e RIGHT OUTER JOIN dep_mast d ON(e.dept_no=d.dept_no);
Example 2: SELECT emp_no,emp_name,job_name,dep_name,location FROM emp_mast e, dep_mast d WHERE e.dept_no(+)=d.dept_no;

Full Outer Join: A full outer join returns all rows from both the tables left and right side of the join clause, extended with nulls if they do not satisfy the join condition.
Example: SELECT emp_no,emp_name,job_name,dep_name,location FROM emp_mast e FULL OUTER JOIN dep_mast d ON(e.dept_no=d.dept_no);

Natural Join: A natural join is such a join that compares the common columns of both tables with each other. Example: SELECT EMP_NO,EMP_NAME,JOB_NAME,DEP_NAME,LOCATION FROM EMP_MAST NATURAL JOIN DEP_MAST;

Antijoins: An antijoin between two tables returns rows from the first table where no matches are found in the second table. Anti-Joins are only available when performing a NOT IN sub-query.
Example: SELECT * FROM EMP_MAST WHERE DEPT_NO NOT IN (SELECT DEPT_NO FROM DEP_MAST);

Semijoins: A semi-join is such a join where the EXISTS clause is used with a subquery. It can be called a semi-join because even if duplicate rows are returned in the subquery, only one set of matching values in the outer query is returned.
Example: SELECT * FROM DEP_MAST A WHERE EXISTS (SELECT * FROM EMP_MAST B WHERE A.DEPT_NO = B.DEPT_NO);

29. Specify the following queries in ORACLE SQL

create table Client (IdClient CHAR(10) PRIMARY KEY, Name VARCHAR(25) NOT NULL, Address VARCHAR(60) NOT NULL, NumCC CHAR(16) NOT NULL);

create table Orders (IdOrder CHAR(10) PRIMARY KEY, IdClient CHAR(10) NOT NULL REFERENCES Client on delete cascade, DateOrder DATE NOT NULL, DateExped DATE);

create table Author (idAuthor NUMBER PRIMARY KEY, Name VARCHAR(25));

create table Book (ISBN CHAR(15) PRIMARY KEY, Title VARCHAR(60) NOT NULL,
Ano CHAR(4) NOT NULL, PurchasePrice NUMBER(6,2) DEFAULT 0, SalePrice NUMBER(6,2) DEFAULT 0);

create table Author_Book
(ISBN CHAR(15), Author NUMBER, CONSTRAINT al_PK PRIMARY KEY (ISBN, Author), CONSTRAINT BookA_FK FOREIGN KEY (ISBN) REFERENCES Book on delete cascade, CONSTRAINT Author_FK FOREIGN KEY (Author) REFERENCES Author);

create table Books_Order(ISBN CHAR(15), IdOrder CHAR(10),
amount NUMBER(3) CHECK (amount >0), CONSTRAINT lp_PK PRIMARY KEY (ISBN, idOrder), CONSTRAINT Book_FK FOREIGN KEY (ISBN) REFERENCES Book on delete cascade, CONSTRAINT pedido_FK FOREIGN KEY (IdOrder) REFERENCES Orders on delete cascade);

insert into Client values ('0000001','James Smith', 'Picadilly 2','1234567890123456');
insert into Client values ('0000002','Laura Jones', 'Holland Park 13', '1234567756953456');
insert into Client values ('0000003','Peter Doe', 'High Street 42', '1237596390123456');
insert into Client values ('0000004','Rose Johnson', 'Notting Hill 46', '4896357890123456');

```
insert into Client values ('0000005','Joseph Clinton', 'Leicester Square 1', '1224569890123456');
insert into Client values ('0000006','Betty Fraser', 'Whitehall 32', '2444889890123456' );
insert into Orders values ('0000001P','0000001', TO_DATE('01/12/2011'),TO_DATE('03/12/2011'));
insert into Orders values ('0000002P','0000001', TO_DATE('01/12/2011'),null);
insert into Orders values ('0000003P','0000002', TO_DATE('02/12/2011'),TO_DATE('03/12/2011'));
insert into Orders values ('0000004P','0000004', TO_DATE('02/12/2011'),TO_DATE('05/12/2011'));
insert into Orders values ('0000005P','0000005', TO_DATE('03/12/2011'),TO_DATE('03/12/2011'));
insert into Orders values ('0000006P','0000003', TO_DATE('04/12/2011'),null);
insert into Author values (1,'Jane Austin');
insert into Author values (2,'George Orwell');
insert into Author values (3,'J.R.R Tolkien');
insert into Author values (4,'Antoine de Saint-Exupéry');
insert into Author values (5,'Bram Stoker');
insert into Author values (6,'Plato');
insert into Author values (7,'Vladimir Nabokov');
insert into Book values ('8233771378567', 'Pride and Prejudice', '2008', 9.45, 13.45);
insert into Book values ('1235271378662', '1984', '2009', 12.50, 19.25);
insert into Book values ('4554672899910', 'The Hobbit', '2002', 19.00, 33.15);
insert into Book values ('5463467723747', 'The Little Prince', '2000', 49.00, 73.45);
insert into Book values ('0853477468299', 'Dracula', '2011', 9.45, 13.45);
insert into Book values ('1243415243666', 'The Republic', '1997', 10.45, 15.75);
insert into Book values ('0482174555366', 'Lolita', '1998', 4.00, 9.45);
insert into Author_Book values ('8233771378567',1);
insert into Author_Book values ('1235271378662',2);
insert into Author_Book values ('4554672899910',3);
insert into Author_Book values ('5463467723747',4);
insert into Author_Book values ('0853477468299',5);
insert into Author_Book values ('1243415243666',6);
insert into Author_Book values ('0482174555366',7);
insert into Books_Order values ('8233771378567','0000001P', 1);
insert into Books_Order values ('5463467723747','0000001P', 2);
insert into Books_Order values ('0482174555366','0000002P', 1);
insert into Books_Order values ('4554672899910','0000003P', 1);
insert into Books_Order values ('8233771378567','0000003P', 1);
insert into Books_Order values ('1243415243666','0000003P', 1);
insert into Books_Order values ('8233771378567','0000004P', 1);
insert into Books_Order values ('4554672899910','0000005P', 1);
insert into Books_Order values ('1243415243666','0000005P', 1);
insert into Books_Order values ('5463467723747','0000005P', 3);
insert into Books_Order values ('8233771378567','0000006P', 5);
```

1. List of books, the publication year of each of them and their authors, ordered by publication year.
2. List of books that were published before 01-01-2000.

3. List of clients that have bought at least one book.
4. List of clients that bought the book whose ISBN= 4554672899910.
5. List of clients whose name contains ‘Jo’ and the books that they have bought.
6. List of clients that have bought at least a book whose price is greater than 10€.
7. List of clients that have placed more than one order in the same date.
8. List of clients and dates in which they have placed orders that have not been sent yet.
9. List of clients that have not bought books whose price is greater than 10€.
10. List of books whose sale price is greater than 30€ or that were published before 2000.
11. List of books and amount of copies of each of them that have been sold.
12. List of clients and the total amount that they have spent in the bookstore.
13. Profits obtained from sales of books.
14. Total amount of each order (ordered by date) that have been placed after 01/12/2011 and have not been sent yet.
15. Detail of orders (Order number, client name, title, total amount, unit price and total).
16. Orders whose total amount is greater than 100€.
17. Orders and the total amount of each of them that contains more than a book (title).
18. Orders and the total amount of each of them that contains more than 4 copies.
19. List of the most expensive books.
20. List of books that have not been sold or that have been sold but the profit per copy is less than 5€.
21. List of clients that have bought more than one copy of a book sometime.

30. Specify the following queries in Oracle SQL.

```
create table flight( flno number(4,0) primary key, origin varchar2(20), destination varchar2(20), distance number(6,0), departure_date date, arrival_date date, price number(7,2));

create table aircraft( aid number(9,0) primary key, name varchar2(30), distance number(6,0));

create table employee( eid number(9,0) primary key, name varchar2(30), salary number(10,2));

create table certificate( eid number(9,0), aid number(9,0), primary key(eid,aid),
foreign key(eid) references employee, foreign key(aid) references aircraft);
```

```
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (99.0,'Los Angeles','Washington D.C.',2308.0,to_date('04/12/2005 09:30', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 09:40', 'dd/mm/yyyy HH24:MI'),235.98);
```

SQL

```
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (13.0,'Los Angeles','Chicago',1749.0,to_date('04/12/2005 08:45', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 08:45', 'dd/mm/yyyy HH24:MI'),220.98);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (346.0,'Los Angeles','Dallas',1251.0,to_date('04/12/2005 11:50', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 07:05', 'dd/mm/yyyy HH24:MI'),225-43);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (387.0,'Los Angeles','Boston',2606.0,to_date('04/12/2005 07:03', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 05:03', 'dd/mm/yyyy HH24:MI'),261.56);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (7.0,'Los Angeles','Sydney',7487.0,to_date('04/12/2005 05:30', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 11:10', 'dd/mm/yyyy HH24:MI'),278.56);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (2.0,'Los Angeles','Tokyo',5478.0,to_date('04/12/2005 06:30', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 03:55', 'dd/mm/yyyy HH24:MI'),780.99);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (33.0,'Los Angeles','Honolulu',2551.0,to_date('04/12/2005 09:15', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 11:15', 'dd/mm/yyyy HH24:MI'),375.23);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (34.0,'Los Angeles','Honolulu',2551.0,to_date('04/12/2005 12:45', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 03:18', 'dd/mm/yyyy HH24:MI'),425.98);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (76.0,'Chicago','Los Angeles',1749.0,to_date('04/12/2005 08:32', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 10:03', 'dd/mm/yyyy HH24:MI'),220.98);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (68.0,'Chicago','New York',802.0,to_date('04/12/2005 09:00', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 12:02', 'dd/mm/yyyy HH24:MI'),202.45);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (7789.0,'Madison','Detroit',319.0,to_date('04/12/2005 06:15', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 08:19', 'dd/mm/yyyy HH24:MI'),120.33);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (701.0,'Detroit','New York',470.0,to_date('04/12/2005 08:55', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 10:26', 'dd/mm/yyyy HH24:MI'),180.56);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (702.0,'Madison','New York',789.0,to_date('04/12/2005 07:05', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 10:12', 'dd/mm/yyyy HH24:MI'),202.34);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (4884.0,'Madison','Chicago',84.0,to_date('04/12/2005 10:12', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 11:02', 'dd/mm/yyyy HH24:MI'),112.45);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (2223.0,'Madison','Pittsburgh',517.0,to_date('04/12/2005 08:02', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 10:01', 'dd/mm/yyyy HH24:MI'),189.98);
```

SQL

```
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (5694.0,'Madison','Minneapolis',247.0,to_date('04/12/2005 08:32', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 09:33', 'dd/mm/yyyy HH24:MI'),120.11);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (304.0,'Minneapolis','New York',991.0,to_date('04/12/2005 10:00', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 01:39', 'dd/mm/yyyy HH24:MI'),101.56);
INSERT INTO flight (FLNO, origin, destination, distance, departure_date, arrival_date, price) VALUES (149.0,'Pittsburgh','New York',303.0,to_date('04/12/2005 09:42', 'dd/mm/yyyy HH24:MI'),to_date('04/12/2005 12:09', 'dd/mm/yyyy HH24:MI'),1165.00);
Insert into aircraft (AID,name,distance) values ('1','Boeing 747-400','8430');
Insert into aircraft (AID,name,distance) values ('2','Boeing 737-800','3383');
Insert into aircraft (AID,name,distance) values ('3','Airbus A340-300','7120');
Insert into aircraft (AID,name,distance) values ('4','British Aerospace Jetstream 41','1502');
Insert into aircraft (AID,name,distance) values ('5','Embraer ERJ-145','1530');
Insert into aircraft (AID,name,distance) values ('6','SAAB 340','2128');
Insert into aircraft (AID,name,distance) values ('7','Piper Archer III','520');
Insert into aircraft (AID,name,distance) values ('8','Tupolev 154','4103');
Insert into aircraft (AID,name,distance) values ('16','Schwitzer 2-33','30');
Insert into aircraft (AID,name,distance) values ('9','Lockheed L1011','6900');
Insert into aircraft (AID,name,distance) values ('10','Boeing 757-300','4010');
Insert into aircraft (AID,name,distance) values ('11','Boeing 777-300','6441');
Insert into aircraft (AID,name,distance) values ('12','Boeing 767-400ER','6475');
Insert into aircraft (AID,name,distance) values ('13','Airbus A320','2605');
Insert into aircraft (AID,name,distance) values ('14','Airbus A319','1805');
Insert into aircraft (AID,name,distance) values ('15','Boeing 727','1504');
Insert into employee (EID,name,salary) values ('242518965','James Smith','120433');
Insert into employee (EID,name,salary) values ('141582651','Mary Johnson','178345');
Insert into employee (EID,name,salary) values ('11564812','John Williams','153972');
Insert into employee (EID,name,salary) values ('567354612','Lisa Walker','256481');
Insert into employee (EID,name,salary) values ('552455318','Larry West','101745');
Insert into employee (EID,name,salary) values ('550156548','Karen Scott','205187');
Insert into employee (EID,name,salary) values ('390487451','Lawrence Sperry','212156');
Insert into employee (EID,name,salary) values ('274878974','Michael Miller','99890');
Insert into employee (EID,name,salary) values ('254099823','Patricia Jones','24450');
Insert into employee (EID,name,salary) values ('356187925','Robert Brown','44740');
Insert into employee (EID,name,salary) values ('355548984','Angela Martinez','212156');
Insert into employee (EID,name,salary) values ('310454876','Joseph Thompson','212156');
Insert into employee (EID,name,salary) values ('489456522','Linda Davis','27984');
Insert into employee (EID,name,salary) values ('489221823','Richard Jackson','23980');
Insert into employee (EID,name,salary) values ('548977562','William Ward','84476');
Insert into employee (EID,name,salary) values ('310454877','Chad Stewart','33546');
Insert into employee (EID,name,salary) values ('142519864','Betty Adams','227489');
Insert into employee (EID,name,salary) values ('269734834','George Wright','289950');
Insert into employee (EID,name,salary) values ('287321212','Michael Miller','48090');
Insert into employee (EID,name,salary) values ('552455348','Dorthy Lewis','152013');
```

SQL

```
Insert into employee (EID,name,salary) values ('248965255','Barbara Wilson','43723');
Insert into employee (EID,name,salary) values ('159542516','William Moore','48250');
Insert into employee (EID,name,salary) values ('348121549','Haywood Kelly','32899');
Insert into employee (EID,name,salary) values ('90873519','Elizabeth Taylor','32021');
Insert into employee (EID,name,salary) values ('486512566','David Anderson','43001');
Insert into employee (EID,name,salary) values ('619023588','Jennifer Thomas','54921');
Insert into employee (EID,name,salary) values ('15645489','Donald King','18050');
Insert into employee (EID,name,salary) values ('556784565','Mark Young','205187');
Insert into employee (EID,name,salary) values ('573284895','Eric Cooper','114323');
Insert into employee (EID,name,salary) values ('574489456','William Jones','105743');
Insert into employee (EID,name,salary) values ('574489457','Milo Brooks','20');
```

```
Insert into certificate (EID,AID) values ('11564812','2');
Insert into certificate (EID,AID) values ('11564812','10');
Insert into certificate (EID,AID) values ('90873519','6');
Insert into certificate (EID,AID) values ('141582651','2');
Insert into certificate (EID,AID) values ('141582651','10');
Insert into certificate (EID,AID) values ('141582651','12');
Insert into certificate (EID,AID) values ('142519864','1');
Insert into certificate (EID,AID) values ('142519864','2');
Insert into certificate (EID,AID) values ('142519864','3');
Insert into certificate (EID,AID) values ('142519864','7');
Insert into certificate (EID,AID) values ('142519864','10');
Insert into certificate (EID,AID) values ('142519864','11');
Insert into certificate (EID,AID) values ('142519864','12');
Insert into certificate (EID,AID) values ('142519864','13');
Insert into certificate (EID,AID) values ('159542516','5');
Insert into certificate (EID,AID) values ('159542516','7');
Insert into certificate (EID,AID) values ('242518965','2');
Insert into certificate (EID,AID) values ('242518965','10');
Insert into certificate (EID,AID) values ('269734834','1');
Insert into certificate (EID,AID) values ('269734834','2');
Insert into certificate (EID,AID) values ('269734834','3');
Insert into certificate (EID,AID) values ('269734834','4');
Insert into certificate (EID,AID) values ('269734834','5');
Insert into certificate (EID,AID) values ('269734834','6');
Insert into certificate (EID,AID) values ('269734834','7');
Insert into certificate (EID,AID) values ('269734834','8');
Insert into certificate (EID,AID) values ('269734834','9');
Insert into certificate (EID,AID) values ('269734834','10');
Insert into certificate (EID,AID) values ('269734834','11');
Insert into certificate (EID,AID) values ('269734834','12');
Insert into certificate (EID,AID) values ('269734834','13');
Insert into certificate (EID,AID) values ('269734834','14');
Insert into certificate (EID,AID) values ('269734834','15');
Insert into certificate (EID,AID) values ('274878974','10');
Insert into certificate (EID,AID) values ('274878974','12');
Insert into certificate (EID,AID) values ('310454876','8');
Insert into certificate (EID,AID) values ('310454876','9');
Insert into certificate (EID,AID) values ('355548984','8');
Insert into certificate (EID,AID) values ('355548984','9');
Insert into certificate (EID,AID) values ('356187925','6');
Insert into certificate (EID,AID) values ('390487451','3');
Insert into certificate (EID,AID) values ('390487451','13');
Insert into certificate (EID,AID) values ('390487451','14');
Insert into certificate (EID,AID) values ('548977562','7');
Insert into certificate (EID,AID) values ('550156548','1');
Insert into certificate (EID,AID) values ('550156548','12');
Insert into certificate (EID,AID) values ('552455318','2');
Insert into certificate (EID,AID) values ('552455318','7');
Insert into certificate (EID,AID) values ('552455318','14');
Insert into certificate (EID,AID) values ('556784565','2');
Insert into certificate (EID,AID) values ('556784565','3');
Insert into certificate (EID,AID) values ('556784565','5');
Insert into certificate (EID,AID) values ('567354612','1');
Insert into certificate (EID,AID) values ('567354612','2');
Insert into certificate (EID,AID) values ('567354612','3');
Insert into certificate (EID,AID) values ('567354612','4');
Insert into certificate (EID,AID) values ('567354612','5');
Insert into certificate (EID,AID) values ('567354612','7');
Insert into certificate (EID,AID) values ('567354612','9');
Insert into certificate (EID,AID) values ('567354612','10');
Insert into certificate (EID,AID) values ('567354612','11');
Insert into certificate (EID,AID) values ('567354612','12');
Insert into certificate (EID,AID) values ('567354612','15');
Insert into certificate (EID,AID) values ('573284895','3');
Insert into certificate (EID,AID) values ('573284895','4');
Insert into certificate (EID,AID) values ('573284895','5');
Insert into certificate (EID,AID) values ('574489456','6');
Insert into certificate (EID,AID) values ('574489456','8');
Insert into certificate (EID,AID) values ('574489457','7');
```

1. List codes and names of pilots that are certified to pilot Boeing aircrafts.
2. List codes of aircrafts that can fly from Los Angeles to Chicago without refueling.
3. List pilots that have a certificate to pilot aircrafts with a range of 3,000 km. but cannot pilot Boeing aircrafts.
4. List of employees that have the highest salary.

5. List of employees that have the highest number of certificates.
6. List of employees that have at least 3 certificates.
7. List of names of the aircrafts such that all the pilots that can pilot them have a salary greater than 80.000€.
8. For each pilot that can pilot more than 3 aircrafts, show the code of the pilot and the distance that these aircrafts can cover.
9. List the names of the pilots whose salary is less than the cheaper flight from Los Angeles to Honolulu.
10. Show the difference of the average salary of all the employees (pilots included) and the average salary of the pilots.
11. List of the names and salaries of the employees (no pilots) whose salary is greater than the average salary of the pilots.
12. List the names of the pilots that can pilot ONLY aircrafts with a range greater than 1000Km.

31. ORACLE ENTERPRICE EDITION EXERCISE 1

1.1 Create the EMP_2018 table.

SQL> CREATE TABLE EMP_2018 (EMPN NUMBER(4) PRIMARY KEY,ENAME VARCHAR2(20) NOT NULL, JOB VARCHAR2(10), MGR NUMBER(4), HIREDATE DATE,SAL NUMBER(7,2),COMM NUMBER(7,2),DEPTNO NUMBER(2));

SQL> DESC EMP_2018;

1.2 Create the PRODUCT_MASTER_2018 table.

SQL> CREATE TABLE PRODUCT_MASTER_2018 (PRODUCT_NO VARCHAR2(6) PRIMARY KEY,DESCRIPTION VARCHAR2(25), PROFIT_PERCENT NUMBER(4,2),UNIT_MEASURE VARCHAR2(10),QTY_ON_HAND NUMBER(8),RECORD_LVL NUMBER(8), SELL_PRICE NUMBER(8,2),COST_PRICE NUMBER(8,2));

SQL> DESC PRODUCT_MASTER_2018;

1.3 Create the CLIENT_MASTER_2018 table.

SQL> CREATE TABLE CLIENT_MASTER_2018 (CLIENT_NO VARCHAR2(6) PRIMARY KEY, NAME VARCHAR2(20), CITY VARCHAR2(15), STATE VARCHAR2(15), PINCODE NUMBER (6), BAL_DUE NUMBER (10,2));

SQL> DESC CLIENT_MASTER_2018;

1.4 Create the SALESMAN _MASTER_2018 table.

SQL> CREATE TABLE SALESMAN_MASTER_2018 (SALESMAN_NO VARCHAR2(6) PRIMARY KEY,SALESMAN_NAME VARCHAR2(20), ADDRESS1 VARCHAR2(10),ADDRESS2 VARCHAR2(10),CITY VARCHAR2(10), PINCODE NUMBER(7),STATE VARCHAR2(15), SAL_AMT NUMBER(6,2),TGR_TO_GET NUMBER(5),YTD_SALES NUMBER(5) ,REMARKS VARCHAR2(10));

SQL> DESC SALESMAN_MASTER_2018;

1.5 Create the STUDENTS_DETAILS_2019 table.

SQL> CREATE TABLE STUDENTS_DETAILS_2019(ROLL_NO NUMBER(4) PRIMARY KEY,NAME VARCHAR2(20) NOT NULL,DOB DATE, MOBILE_NO NUMBER(10),PARENTS_MOBILE_NO NUMBER(11),EMAIL VARCHAR2(30),PERCENT_10TH NUMBER(4,2), PERCENT_12TH NUMBER(4,2), GRADUATION_DISCIPLINE VARCHAR2(20),PERCENT_GRADUATION NUMBER(4,2),SUB_INTEREST VARCHAR2(20));

Insert students_details_2019 according to your choice.

2.1 Insert records in EMP_2018 table.

SQL> INSERT INTO EMP_2018(EMPN, ENAME,JOB,MGR,HIREDATE, SAL,COMM, DEPTNO) VALUES (&EMPN,'&ENAME','&JOB',&MGR,'&HIREDATE', &SAL,&COMM,&DEPTNO);

SQL> SELECT * FROM EMP_2018;

EMPN	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	12-JAN-83	1100	0	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	0	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	0	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	0	20
7900	JAMES	CLERK	7698	03-DEC-98	950	0	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7839	KING	PRESIDENT	0	17-NOV-81	5000	0	10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	0	10
7566	JONES	MANAGER	7839	02-APR-81	2975	0	20
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	0	30
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7369	SMITH	CLERK	7902	17-DEC-80	800	0	20
7480	JHON	CLERK	7782	12-DEC-82	1300	0	10
7903	OBAMA	MANAGER	0	12-DEC-81	5000	0	20

16 rows selected.

2.2 Insert records in PRODUCT_MASTER_2018, CLIENT_MASTER_2018 and in SALESMAN_MASTER_2018 tables respectively.

a) Data for CLIENT_MASTER_2018 table.

SQL> INSERT INTO CLIENT_MASTER_2018 (CLIENT_NO,NAME,CITY, STATE, PINCODE,BAL_DUE) VALUES ('&CLIENT_NO','&NAME','&CITY','&STATE', &PINCODE,&BAL_DUE);

*The remaining 5 rows will be filled up in the same way as stated above.

```
SQL> SELECT * FROM CLIENT_MASTER_2018;
```

CLIENT	NAME	CITY	STATE	PINCODE	BAL_DUE
C00001	I BYROSS	BOMBAY	MAHARASHTRA	400054	15000
C00002	V SAITWAL	MADRAS	TAMIL NADU	780001	0
C00003	P JAGUSTE	BOMBAY	MAHARASHTRA	400057	5000
C00004	B NAVINDGI	BOMBAY	MAHARASHTRA	400056	0
C00005	R SREEDHARAN	DELHI		100001	2000
C00006	RUKMINI	BOMBAY	MAHARASTHRA	400050	0

SQL> COMMIT;

b) Data for PRODUCT _MASTER_2018 table.

SQL> INSERT INTO PRODUCT_MASTER_2018 (PRODUCT_NO, DESCRIPTION, PROFIT_PERCENT, UNIT_MEASURE,QTY_ON_HAND,RECORD_LVL,SELL_ PRICE,COST_PRICE) VALUES ('&PRODUCT_NO','&DESCRIPTION', &PROFIT_PERCENT, '&UNIT_MEASURE',&QTY_ON_HAND,&RECORD_LVL,&SELL_PRICE, &COST_PRICE);

*The remaining 8 rows will be filled up in the same way as stated above.

```
SQL> SELECT * FROM PRODUCT_MASTER_2018;
```

PRODUC	DESCRIPTION	PROFIT_PERCENT	UNIT_MEASU	QTY_ON_HAND	RECORD_LVL	SELL_PRICE	COST_ P
P00001	1.44 FLOPPIES	5	PIECE	100	20	525	500
P03453	MONITORS	6	PIECE	10	3	12000	11280
P06734	MOUSE	5	PIECE	20	5	1050	1000
P07865	1.22 FLOPPIES	5	PIECE	100	20	525	500
P07868	KEYBOARDS	2	PIECE	10	3	3150	3050
P07885	CD DRIVE	2.5	PIECE	10	3	5250	5100
P07965	540HDD	4	PIECE	10	3	8400	8000
P07975	1.44 DRIVE	5	PIECE	10	3	1050	1000
P08865	1.22 DRIVE	5	PIECE	2	3	1050	1000

9 rows selected.

c) Data for SALESMAN _MASTER_2018 table.

SQL> INSERT INTO SALESMAN_MASTER_2018 (SALESMAN_NO, SALESMAN_NAME,ADDRESS1,ADDRESS2,CITY,PINCODE,STATE,SAL_AMT,TGR_TO_GET,YTD_SALES, REMARKS) VALUES ('&SALESMAN_NO','&SALESMAN_NAME', '&ADDRESS1','&ADDRESS2','&CITY',&PINCODE,'&STATE',&SAL_AMT,&TGR_TO_GET, &YTD_SALES,'&REMARKS');

*The remaining 3 rows will be filled up in the same way as stated above.

SQL> SELECT * FROM SALESMAN_MASTER_2018;

SALESM	SALESMAN	ADDRESS1	ADDRESS2	CITY	PINCO	STATE	SAL_AMT	TGR_TO_GET	YTD_SALES	REMARKS
S00001	KIRAN	A/14	WARLI	BOMBAY	400002	MAHARASHTRA	3000	100	50	GOOD
S00002	MANISH	65	NARIMAN	BOMBAY	400001	MAHARASHTRA	3000	200	100	GOOD
S00003	RAVI	P-7	BANDRA	BOMBAY	400032	MAHARASHTRA	3000	200	100	GOOD
S00004	ASHISH	A-5	JUHU	BOMBAY	400044	MAHARASHTRA	3500	200	150	GOOD

2.3 SQL QUERY:

i) Display the structures of the tables created.
ii) List EMPNO, JOB from table EMP.
iii) Display EMP_2018 Table.
iv) List EMPNO, SALARY, SALARY*1.5 as “NEWSAL” from table.
v) List employee details where JOB is “CLERK”.
vi) List distinct DEPTNO.
vii) List ENAME, SAL of those employees whose salary is > 1000 and <= 1500.
viii) List ENAME, SAL of those employees whose salary is 3000 or 5000.
ix) List ENAME that contains “A” as second letter of their ENAME.
x) List all employee details whose salary is greater than 1500 and DEPT_ID is 10 or 20.
xi) Display only ENAME and JOB in lower case, sentence case respectively and length of ENAME. Then arrange the result according to descending order of salary.

2.4.1. Create a table named as SavingMaster (ID, FirstName, LastName, Address, PinCode)

SQL> create table SavingMaster(ID varchar2(4) primary key, FirstName varchar2(20), LastName varchar2(20), Address varchar2(20),pincode number(7));

2.4.2 Insert 5 records in it.

SQL> insert into SavingMaster (ID,FirstName,LastName,Address,pincode) values ('&ID','&FirstName','&LastName','&Address',&pincode);

```
SQL> select * from SavingMaster;

ID    FIRSTNAME            LASTNAME       ADDRESS          PINCODE
------- -------------------- ------ ---------- ---------- -----
P001 Ram                  Das            Mumbai           450793
P002 Kamal                Meheta         Kolkata          700001
P003 Rita                 Debroy         Maharashtra      678905
P004 Piyush               Das            Hyderabad        879678
P005 Suman                Adhikari       Kolkata          700038
```

2.4.3. Add a column Phone_no in it and update records accordingly.

SQL> Alter table SavingMaster ADD Phone_no number(10);
SQL> desc SavingMaster;
SQL> update SavingMaster set Phone_no = 9008789654 where ID = 'P001';
*The remaining rows will be filled up in the same way as stated above. { Ph no user choice}

2.4.4. Change the name of table as SavingsMaster.

SQL> RENAME SavingMaster to SavingsMaster;

2.4.5 Increase column size of column FirstName by 5.

SQL> Alter table SavingsMaster modify FirstName varchar2(25);
SQL> desc SavingsMaster;

2.4.6. Rename the column Phone_No by Ph_no.

SQL> ALTER table SavingsMaster RENAME column Phone_no to Ph_no;

2.4.7. Delete the record whose address is Mumbai.

SQL> delete from SavingsMaster where address='Mumbai';

SQL> select * from SavingsMaster;

3. Solve queries.

i. Display the date which your computer is showing.
ii. How many columns are there in the table DUAL? Write the SQL query.
iii. What will be the output of the following queries: SELECT 15+10-5*5/5 FROM DUAL;
iv. Display the product details whose profit price is Rs. 25.
v. Project ENAME, JOB of all employees' in the following manner => ADAMS, CLERK.
vi. List all employee details in alphabetical order by JOB.

vii. Display Employees whose ENAME contains string “am” using function according to following format:

EMPNO	SALARY
7876*	$1100

4. Solve queries.

i. Change the Ename of employee in lower case in table EMP_2018.
ii. Change the product quantity to 15 for all items with product quantity less than 10.
iii. Change the name of salesman by ‘Satish’ and salary by 2000, where the records of ‘Manish’ are there.
iv. Confirm your changes.
v. Change the product quantity to 20 for all items with product quantity less than 10.
Create a save point marker. Add two records into table PRODUCT_MASTER_2018. Do not commit. Show the data of the table. Now roll back to save point created previously and show the data of the table and note the effect.
vi. Delete the data from table EMP_2018 who have worked less than 35 years till date.
vii. Increase the salary and commission of all employees of table EMP_2018 by 1000 and 10% accordingly.
viii. Delete all records of those employees having EMPNO less than 7500 and salary in the range of 1500 and 1800 from table EMP_2018.

5. Solve queries.

i. Calculate square root of 15, absolute value of -30, floor value of 35.628, ceiling value of 35.628, round of 35.628 up to 2 decimal point, truncate of 35.628 up to 2 decimal point.
ii. Display EMPNO, no of months worked till date (label the column MONTHS_WORKED), salary review date which is 1st Monday after 6th month of service (label the column SAL_REV).
iii. Display Annual Salary of employees 2018 where annual salary is (sal + commission)*12.
iv. Display total salary, average salary, minimum salary, maximum salary from table EMP_2018.
v. Display total salary department wise from table EMP_2018.
vi. Display DEPTNO, salary, hire date of lowest paid employee for that department excluding department having DEPTNO=30. Exclude any group where the minimum salary is 800.Sort the output in descending order of hire date.
vii. Display minimum average salary among department from table EMP_2018.
viii. Display EMPNO, HIREDATE who joined on December 03, 1981. Display the hire date according to the following format: Thursday, The Third of December, 1981.

6. Solve queries.

i. Retrieve all the information from Salesman_Master_2018 where salesman_name begins with the letter ‘K’.

ii. Retrieve product_no, description, profit_percent and sell_price from the Product_Master_2019 where profit_percent is in between 3 and 6.
iii. Find out the names of all the clients.
iv. List all the clients who are located in Bombay.
v. Find the name of the salesman who has a salary equal to 3000.
vi. Change the city of client_no 'C00005' in Client_Master_2018 to Bombay.
vii. Change the bal_due of client_no 'C00001' in Client_Master_2018 to Rs. 1000.
viii. Change the cost_price of '1.22 Floppies' in Product_Master_2018 to Rs. 950.00.
ix. Change the city of the salesman to Mumbai whose name starts with 'R'.
x. Display the total number of clients' state wise.
xi. Display the employee names whose hire date is on DEC (i.e. DECEMBER).
xii. Display the employee names whose hire date is on current month.
xii. Display the employee names whose hire date is on current month.

7. Create the following table Sales_Order_2019. Use varchar2 data type for all columns except Order_Date.

Order_No	Client_No	Order_Date	Price
O19001	C00006	12-APR-97	20000
O19002	C00002	25-DEC-97	50000
O19003	C00007	03-OCT-97	10000
O19004	C00005	18-JUN-97	15000
O19005	C00002	20-AUG-97	17000
O19006	C00007	02-JAN-97	20000

SQL> CREATE TABLE SALES_ORDER_2019 (ORDER_NO VARCHAR2(6), CLIENT_NO VARCHAR2(6),ORDER_DATE DATE,PRICE VARCHAR2(6));
SQL> DESC SALES_ORDER_2019;
i. Retrieve order information like Order_No, Client_No, Order_Date for all the orders placed by the client in the ascending order of date. The Order_Date should be displayed in 'DD/MM/YY' format.
ii. Display the orders which are taken on and before 18th June 1997.
iii. Find the orders having price 20000 in number format.

POSITION (Position_no->primary key)

8. Create the following tables:
a. EMPLOYEE (Emp_no->primary key)
b. POSITION (Position_no->primary key)
c. DUTY_ ALLOCATION

Positiong_No	Skill
321	Waiter
322	Bartender
323	Busboy
324	Hostess
325	Receptionist
326	Waiter
350	Chef
351	Chef

SQL

EMPLOYEE (Emp_no->primary key)

Emp_no	Name	Skill	Pay_Rate
123456	Ron	Waiter	7.5
123457	Jon	Bartender	8.79
123458	Don	Busboy	4.70
123459	Pam	Hostess	4.90
123460	Pat	Bellboy	4.70
123461	Ian	Receptionist	9.00
123471	Pierre	Chef	14.00
123472	Julie	Chef	14.50

DUTY_ ALLOCATION

Position_no	Emp_no	Day	Shift
321	123456	19-04-1986	1
322	123457	18-04-1986	2
323	123458	18-04-1986	1
321	123461	20-04-1986	2
321	123461	19-04-1986	2
350	123471	18-04-1986	1
323	123458	20-04-1986	3
351	123471	19-04-1986	1

SQL> CREATE TABLE EMPLOYEE (EMP_NO NUMBER(6) PRIMARY KEY,NAME VARCHAR2(15),SKILL VARCHAR2(15),PAY_RATE NUMBER(4,2));

```
SQL>INSERT ALL INTO EMPLOYEE VALUES(123456,'RON','WAITER',7.5)
INTO EMPLOYEE VALUES (123457,'JON','BRATENDER',8.79)
INTO EMPLOYEE VALUES (123458,'DON','BUSYBOY',4.70)
INTO EMPLOYEE VALUES (123459,'PAM','HOSTESS',4.90)
INTO EMPLOYEE VALUES (123460,'PAT','BELLBOY',4.70)
INTO EMPLOYEE VALUES (123461,'IAN','RECEPTIONIST',9.00)
INTO EMPLOYEE VALUES (123471,'PIERRE','CHEF',14.00)
INTO EMPLOYEE VALUES (123472,'JULIE','CHEF',14.50)
SELECT * FROM DUAL;
```

SQL> CREATE TABLE POSITION (POSITIONING_NO NUMBER(3) PRIMARY KEY,SKILL VARCHAR2(15));

```
SQL>INSERT ALL INTO POSITION VALUES (321,'WAITER')
INTO POSITION VALUES (322,'BRATENDER')
INTO POSITION VALUES (323,'BUSYBOY')
INTO POSITION VALUES (324,'HOSTESS')
INTO POSITION VALUES (325,'RECEPTIONIST')
INTO POSITION VALUES (326,'WAITER')
INTO POSITION VALUES (350,'CHEF')
INTO POSITION VALUES (351,'CHEF')
SELECT * FROM DUAL;
```

SQL> CREATE TABLE DUTY_ALLOCATION (POSITIONING_NO NUMBER(3), EMP_NO NUMBER(6), DAY VARCHAR2(20), SHIFT NUMBER(2),FOREIGN KEY(POSITIONING_NO) REFERENCES POSITION, FOREIGN KEY(EMP_NO) REFERENCES EMPLOYEE);

```
SQL>INSERT ALL INTO DUTY_ALLOCATION VALUES (321,123456,'19-04-1986',1)
INTO DUTY_ALLOCATION VALUES (322,123457,'18-04-1986',2)
```

```
INTO DUTY_ALLOCATION VALUES (323,123458,'18-04-1986',1)
INTO DUTY_ALLOCATION VALUES (321,123461,'20-04-1986',2)
INTO DUTY_ALLOCATION VALUES (321,123461,'19-04-1986',2)
INTO DUTY_ALLOCATION VALUES (350,123471,'18-04-1986',1)
INTO DUTY_ALLOCATION VALUES (323,123458,'20-04-1986',3)
INTO DUTY_ALLOCATION VALUES (351,123471,'19-04-1986',1)
SELECT * FROM DUAL;
```

i) Display name, pay_rate of employees with emp_no less than 123460 whose rate of pay is more than the rate of pay is at least one employee with emp_no greater than or equal to 123460 from table EMPLOYEE.
ii) Display name of employee who are assigned to all positions that required Chef's Skill from tables EMPLOYEE, POSITION, DUTY_ALLOCATION.
iii) Find the employee with the lowest pay rate using subquery from table EMPLOYEE.
iv) Get the names of Chef's paid at the minimum pay rate using subquery from table EMPLOYEE.
v) Find the names and pay_rate of all employees who are allocated a duty using subquery from table EMPLOYEE and DUTY_ALLOCATION.
vi)Display emp_no who are waiters or working position no 321 (using set operator) from table EMPLOYEE and DUTY_ALLOCATION.
vii) Display name with the skill of Chef who are assigned a duty using basic set operator from table EMPLOYEE and DUTY_ALLOCATION.
viii) Get emp_no of employee working on at least two date from table DUTY_ALLOCATION.
ix) Display minimum salary department wise only for those departments that have minimum salary greater than department 20.
x) Write a subquery to display the name, department number & salary of any employee whose department no salary match the department no & salary of any employee who earns commission.
xi) Display the employee name & employee no of employee who have no subordinates.
xii)Write a subquery to display the name, department number & salary of any employee whose commission & salary match the commission & salary of any employee located in DAALLAS.
xiii) Create a subquery to display the name, hire date, salary for all employees who have both the same salary & commission as SCOTT.
xiv) Create a subquery to display employees that earn a salary that is higher than the salary of all the clerks. Sort the results on salary from highest to lowest.
xv) Write a subquery to display the employee name, salary, department numbers & maximum salary for all who earn less than the maximum salary in their department.
xvi) Write a subquery to display the 4 most senior employees in the company.
xvii) Write a subquery to display the top 3 earners' names & salaries.

9. Solve Queries.

i) Create table EMPLOYEE_NEW based on structure of EMPLOYEE. Name the colums in your new table EID, ENAME, ESKILL, ERATE respectively.
ii. Modify the table EMPLOYEE_NEW to add a column HIREDATE (Datatype – Date).
iii. Modify the table EMPLOYEE_NEW to allow longer employee name up to 50 characters.
iv. Delete field ESKILL from EMPLOYEE_NEW.
v. Change the data type of hire date to varchar2.
vi. Change the data type of EID to varchar2.
vii. Rename the column ENAME to NAME.
viii. Rename the table EMPLOYEE_NEW to ENEW.
ix. Add comment to ENEW to describe the table. Confirm this comment addition from data dictionary.
x. Delete the primary constraint.
xi. Add primary key constraint name emp_eid_pk on the field EID. Now delete the constraint.
xii. Delete data from EMPLOYEE_NEW. Do not commit. Undo the deletion & show the data.
xiii. Truncate table ENEW. Undo the job & show the data. Now add a row.
xiv. Delete comment of ENEW.
xv. Drop table ENEW.
xvi. Create a table based on structure of EMPLOYEE. Do not copy data, copy only the structure.

10. Solve queries.

i. Create a view names EMP_VIEW based on EMP_ID, FIRST_NAME, DEPT_ID from table EMP_2018.
ii) Change the column heading of the view according to description.
iii) Select view name and text from data dictionary for created view.
iv) Display employee name & dept no from view.
v) Create a view named dept2 that contains employee name, emp id for all employees of dept 2 from the view EMP_VIEW. Do not allow an employee to be assigned in another department through the view.
vi) Create a view named EMP_DEPT_VIEW which contains dept name, min salary, max salary & average salary department wise. Now select data from the view.
vii) Create a sequence named EMPLOYEE_EMPNO_SEQ to be used with primary key of table EMPLOYEE. The sequence should start at 60; have a maximum value 200 & incremented by 10.
viii) Display sequence name, maximum value, increment size & last no for your sequence.
ix) Create an index on the column SKILL of table EMPLOYEE.
x) Display index & uniqueness that exists in the data dictionary for table EMPLOYEE.
xi) Create a synonym for table POSITION. Fetch all the data from the synonym. Drop the table position.
xi) Drop all the sequence, index, synonym created.

32. ORACLE ENTERPRICE EDITION EXERCISE 2

DELETE ALL TABLES, IF EXISTS

```
DROP TABLE companies_departments; DROP TABLE employees_benefits;
DROP TABLE companies; DROP TABLE employees;
DROP TABLE roles; DROP TABLE departments; DROP TABLE benefits;

CREATE TABLE companies (id NUMBER PRIMARY KEY,name VARCHAR2(30)
DEFAULT ' ' NOT NULL);

CREATE TABLE departments (id NUMBER PRIMARY KEY,name VARCHAR2(20)
DEFAULT ' ' NOT NULL);

CREATE TABLE companies_departments (id NUMBER PRIMARY KEY,
company_id NUMBER NOT NULL, department_id NUMBER NOT NULL,
active CHAR DEFAULT '1' CHECK (active in ('0', '1')));

CREATE TABLE roles (id NUMBER PRIMARY KEY,name VARCHAR2(30) NOT NULL,
salary NUMBER NOT NULL, department_id NUMBER NOT NULL);

CREATE TABLE employees (id NUMBER PRIMARY KEY,name VARCHAR2(70)
DEFAULT ' ' NOT NULL, age NUMBER NOT NULL, role_id NUMBER NOT NULL,
company_id NUMBER NOT NULL);

CREATE TABLE benefits ( id NUMBER PRIMARY KEY, name VARCHAR2(70)
DEFAULT ' ' NOT NULL);

CREATE TABLE employees_benefits ( id NUMBER PRIMARY KEY, employee_id NUMBER
NOT NULL, benefit_id NUMBER NOT NULL, benefit_value NUMBER NOT NULL,
received_date DATE NULL );

CREATE SEQUENCE id_company_seq;
CREATE SEQUENCE id_departments_seq;
CREATE SEQUENCE id_companies_departments_seq;
CREATE SEQUENCE id_roles_seq;
CREATE SEQUENCE id_employees_seq;
CREATE SEQUENCE id_benefits_seq;
CREATE SEQUENCE id_employees_benefits_seq;
```

SQL

ALTER TABLE companies_departments ADD FOREIGN KEY (company_id) REFERENCES companies(id);
ALTER TABLE companies_departments ADD FOREIGN KEY (department_id) REFERENCES departments(id);
ALTER TABLE employees ADD FOREIGN KEY (role_id) REFERENCES roles(id);
ALTER TABLE employees ADD FOREIGN KEY (company_id) REFERENCES companies(id);
ALTER TABLE roles ADD FOREIGN KEY (department_id) REFERENCES departments(id);
ALTER TABLE employees_benefits ADD FOREIGN KEY (employee_id) REFERENCES employees(id);
ALTER TABLE employees_benefits ADD FOREIGN KEY (benefit_id) REFERENCES benefits(id);

```
sql> select *from roles;
```

id	name	salary	department_id
1	System Analist	4000	2
2	Software Developer	2300	2
3	Software Engeneer	4500	2
4	Administrative Assistant	1800	1
5	IT Manager	5200	2
6	Project Manager	5000	2
7	Financial Manager	4800	1
8	Accountant	7500	1
9	Human Resources Analyst	3900	3
10	Intern	900	2

```
sql> select *from departments;
```

id	name
1	Financial
2	TI
3	Human Resources

```
sql> select *from companies;
```

id	name
1	Mauris Sapien Cursus Limited
2	Risus Limited
3	Sociis Natoque Penatibus PC
4	Diam At Pretium Ltd
5	Lacus Pede Incorporated
6	Imperdiet Corporation
7	Nec LLC
8	Congue A LLP
9	Posuere Industries
10	Augue Limited

```
sql> select *from benefits;
```

id	name
1	Overtime
2	Food stamps
3	Transportation Vouchers
4	Health Insurence
5	Trainings, courses or congresses

```
sql> select *from companies_departments;
```

id	company_id	department_id	active
1	1	1	1
2	1	2	1
3	1	3	1
4	2	1	1
5	2	2	1
6	2	3	1
7	3	1	1
8	3	2	0
9	3	3	1
10	4	1	1
11	4	2	1
12	4	3	1
13	5	1	1
14	5	2	1
15	5	3	1
16	6	1	1
17	6	2	1
18	6	3	1
19	7	1	1
20	7	2	1
21	7	3	1
22	8	1	1
23	8	2	1
24	8	3	1
25	9	1	1
26	9	2	1
27	9	3	1
28	10	1	1
29	10	2	1
30	10	3	1

SQL

```
sql> select *from employees;
```

id	name	age	role_id	company_id
1	Devin B. Morrow	27	1	1
2	Clark G. Barlow	37	1	2
3	Cameron V. Wade	21	1	3
4	Mannix O. Wheeler	28	1	4
5	Hyacinth Q. Moses	36	1	5
6	Kristen X. Petersen	21	1	6
7	Dana R. Skinner	19	3	7
8	Chancellor T. Camacho	23	3	8
9	Britanney E. Callahan	30	3	9
10	Ryan M. Wilkinson	39	5	10
11	Cody D. Graham	39	5	1
12	Veronica T. Hancock	26	5	2
13	Carson C. Elliott	25	6	3
14	Justin U. Ward	23	6	4
15	Russell V. Banks	29	6	5
16	Mari R. English	36	10	6
17	Giacomo G. Padilla	41	10	7
18	Andrew L. Kennedy	24	4	8
19	Tamara N. Gates	24	4	9
20	Risa W. Jenkins	30	4	10
21	Leigh C. Pitts	30	7	1
22	Jonas O. Mcpherson	28	7	2
23	Charity J. Wall	29	7	3
24	Abigail O. Atkins	40	8	4
25	Lilah D. Taylor	19	8	5
26	Peter N. Sheppard	26	8	6
27	Randall T. Graham	42	4	7
28	Maxwell E. Moon	33	4	8
29	Zephania H. Owens	31	4	9
30	Leilani J. Shields	42	9	10
31	Karina D. Fowler	34	9	1
32	Miriam V. Bryant	38	9	2
33	Lois I. Cruz	19	10	3
34	Paki I. Duke	24	10	4
35	Buckminster E. Jarvis	23	10	5