MICROSOFT SQL SERVER

QUERIES AND NOTES

BHARAT SINGH RAJPUT

B.C.A. III SEMESTER

NATIONAL POST GRADUATE COLLEGE,
LUCKNOW

Create Database

```
CREATE DATABASE <Database_name>;
```

CREATE DATABASE npgcbca3;

Listing Databases/Tables

SHOW DATABASES
SHOW TABLES

Selecting database

```
USE <database_name>
```

Use npgcbca3;

Deleting database

```
DROP <database_name>;
```

Drop npgcbca3;

Creating table

```
CREATE TABLE <Table_name>
      (<Column_name_1> <data_type> <constraints>,
       <Column_name_2> <data_type> <constraints>,
       ..);
```

```
CREATE TABLE Employee
      (emp_id INT PRIMARY KEY,
       dept_id INT,
       emp_name VARCHAR(30),
       emp_address VARCHAR(30),
       emp_sal INT);
```

Constraints :

- ***NOT NULL*** - Ensures that a column cannot have a NULL value
- ***UNIQUE*** - Ensures that all values in a column are different
- ***PRIMARY KEY*** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

- ***FOREIGN KEY*** - Uniquely identifies a row/record in another table. The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables. E.g.

```
CREATE TABLE Orders (
      OrderID int NOT NULL PRIMARY KEY,
      PersonID int FOREIGN KEY REFERENCES Persons(PersonID));
```

- ***CHECK*** - Ensures that all values in a column satisfies a specific condition

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

- ***DEFAULT*** - Sets a default value for a column when no value is specified

```
CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      City varchar(255) DEFAULT 'Sandnes');
```

- ***INDEX*** - Used to create and retrieve data from the database very quickly

```
CREATE INDEX idx_lastname ON Persons (LastName);

-- TO REMOVE INDEX USE
DROP INDEX table_name.index_name;
```

- ***AUTO_INCREMENT*** – Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

```
CREATE TABLE Persons (
      ID int NOT NULL AUTO_INCREMENT = 100,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      PRIMARY KEY (ID));

-- INITIAL VALUE IS 100. SQL SETS IT TO 1 IF NOT SPECIFIED
```

Delete Table

```
DROP <table_name>
```

DROP Employee

Truncate Table (Deletes only data but not table)

```
TRUNCATE <table_name>
```

TRUNCATE Employee

Alter Table

1. Inserting new column

```
ALTER TABLE <table_name> Add <column_name> <data_type>
```

ALTER TABLE Employee Add DOB datetime

2. Deleting a column

```
ALTER TABLE <table_name> DROP COLUMN <column_name>
```

ALTER TABLE Employee DROP COLUMN DOB

3. Changing datatype of column

```
ALTER TABLE <table_name> ALTER COLUMN <column_name> <data_type>
```

ALTER TABLE Employee ALTER COLUMN emp_salary BIGINT

4. Changing column name

```
EXEC sp_RENAME '<table_name.old_column_name>', '<new_name>', 'COLUMN'
```

5. Change table name

```
EXEC sp_RENAME '<table_name.old_table_name>', '<new_name>', 'COLUMN'
```

Create View

In SQL, a view is a virtual table based on the result-set of an SQL statement.
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

```
CREATE VIEW [<view_name>] AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Examples :

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued = No;
```

Then, we can query the view as follows:

```
SELECT * FROM [Current Product List];
```

Another view in the Northwind sample database selects every product in the "Products" table with a unit price higher than the average unit price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, UnitPrice
FROM Products
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products);
```

We can query the view above as follows:

```
SELECT * FROM [Products Above Average Price];
```

Another view in the Northwind database calculates the total sale for each category in 1997. Note that this view selects its data from another view called "Product Sales for 1997":

```
CREATE VIEW [Category Sales For 1997] AS
SELECT DISTINCT Category_Name, Sum(Product_Sales) AS Category_Sales
FROM [Product Sales for 1997]
GROUP BY CategoryName;
```

We can query the view above as follows:

```
SELECT * FROM [Category Sales For 1997];
```

We can also add a condition to the query. Let's see the total sale only for the category "Beverages":

```
SELECT * FROM [Category Sales For 1997]
WHERE CategoryName = 'Beverages';
```

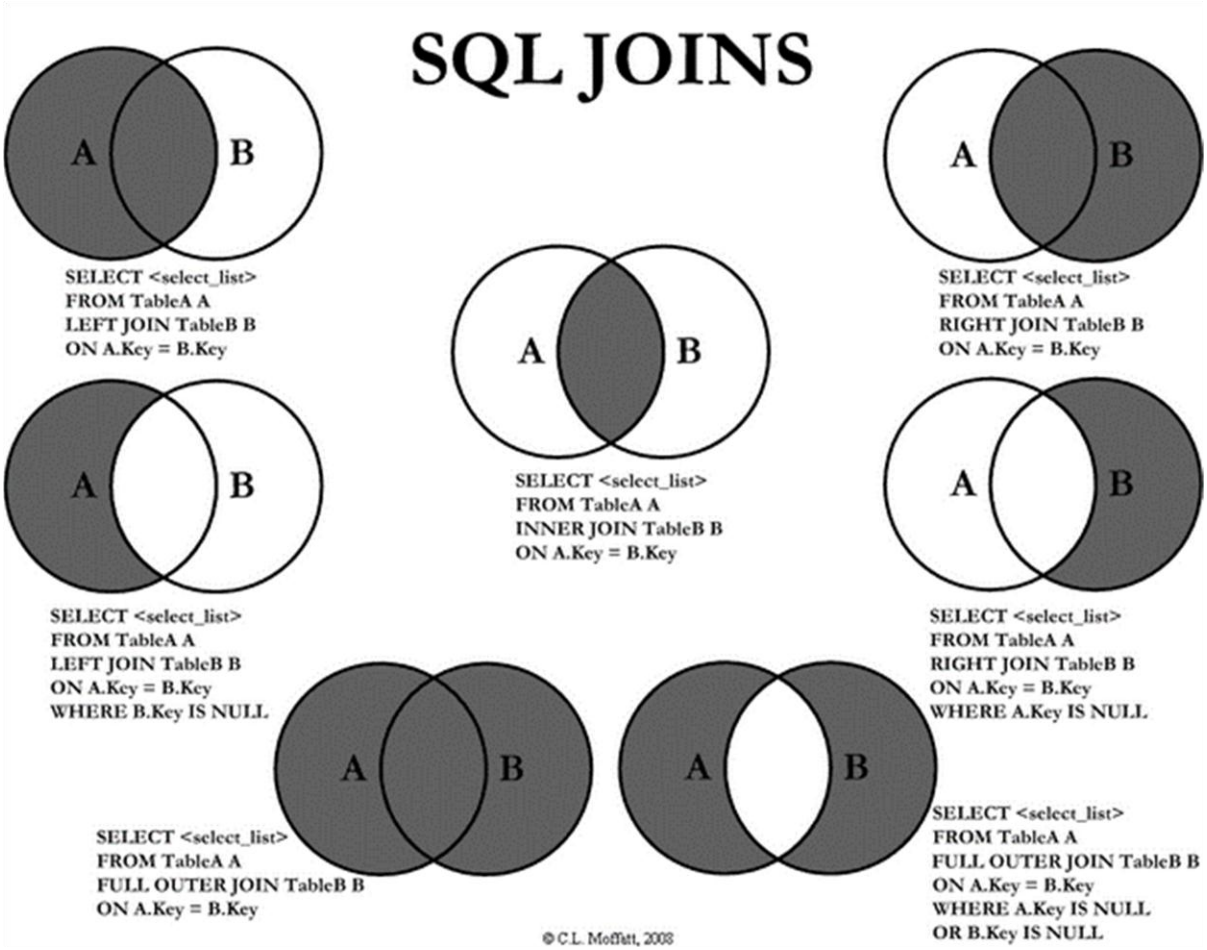
Inserting data into table

```
INSERT INTO <table_name> VALUES(<column1_value>, <column2_value>, ..);

INSERT INTO <table_name> (<Column_name1>, <column_name_2>, …)
Values (<column1_value>, <column2_value>, …);

-- 8/9/2018
INSERT INTO Department VALUES(100, 'SALES');
INSERT INTO Department VALUES(200, 'MARKETING');
INSERT INTO Department VALUES(300, 'ADMINISTRATION');
INSERT INTO Department VALUES(400, 'PRODUCTION');

INSERT INTO Employee VALUES(1000, 100, 'Satish Kumar', 'Lucknow', 10000);
INSERT INTO Employee VALUES(1001, 100, 'Abhay Bhist', 'Delhi', 12000);
INSERT INTO Employee VALUES(1002, 100, 'Kapil Rajput', 'Mumbai', 15000);
INSERT INTO Employee VALUES(1003, 100, 'Ravi Mishra', 'Kanpur', 11400);
INSERT INTO Employee VALUES(1004, 100, 'Neeraj Yadav', 'Banaras', 16500);

INSERT INTO Employee VALUES(1005, 200, 'Chirag Singh', 'Lucknow', 17000);
INSERT INTO Employee VALUES(1006, 200, 'Raj Singh', 'Delhi', 12000);
INSERT INTO Employee VALUES(1007, 200, 'Ramesh Yadav', 'Allahabad', 21000);
INSERT INTO Employee VALUES(1008, 200, 'Rakesh Maurya', 'Mumbai', 22000);
INSERT INTO Employee VALUES(1009, 200, 'Amar Shukla', 'Lucknow', 20000);

INSERT INTO Employee VALUES(1010, 300, 'Avinash Thakur', 'Mumbai', 21000);
INSERT INTO Employee VALUES(1011, 300, 'Vishal Yadav', 'Lucknow', 25000);
INSERT INTO Employee VALUES(1012, 300, 'Surendra Rao', 'Delhi', 26000);
INSERT INTO Employee VALUES(1013, 300, 'Satyendra Pawar', 'Banaras', 26500);
INSERT INTO Employee VALUES(1014, 300, 'Mohit Mishra', 'Kanpur', 26800);

INSERT INTO Employee VALUES(1015, 400, 'Vivek Singh', 'Mumbai', 19000);
INSERT INTO Employee VALUES(1016, 400, 'Vipin Rajput', 'Kanpur', 18000);
INSERT INTO Employee VALUES(1017, 400, 'Vikas Dixit', 'Delhi', 21000);
INSERT INTO Employee VALUES(1018, 400, 'Ajit Kumar', 'Lucknow', 25000);
INSERT INTO Employee VALUES(1019, 400, 'Aryan Srivastava', 'Lucknow',
29000);
```

Printing String/Expression

```
SELECT “String”;
SELECT <Expression>
```

Joins



Exercise

Retrieve data From tables

Write a SQL statement to display all the information of all salesmen.

```
SELECT * FROM Salesman;
```

Write a SQL statement to display a string "This is SQL Exercise, Practice and Solution".

```
SELECT "This is SQL Exercise, Practice and Solution";
```

Write a query to display three numbers in three columns.

```
SELECT 1,2,3;
```

Write a query to display the sum of two numbers 10 and 15 from RDMS server.

```
SELECT 10+15;
```

Write a query to display the result of an arithmetic expression.

```
SELECT (10+1)*5/3+2;
```

Write a SQL statement to display specific columns like name and commission for all the salesmen.

```
SELECT Name, Commission FROM Salesman;
```

Write a query to display the columns in a specific order like order date, salesman id, order number and purchase amount from for all the orders.

```
SELECT * From Orders ORDER BY Ord_Date DESC;
SELECT * From Orders ORDER BY Salesman_ID;
```

Write a query which will retrieve the value of salesman id of all salesmen, getting orders from the customers in orders table without any repeats.

```
SELECT DISTINCT (Salesman_id)  FROM Orders;
```

Write a SQL statement to display names and city of salesman, who belongs to the city of Paris.

```
SELECT Name, City FROM Salesman WHERE city = 'Paris';
```

Write a SQL statement to display all the information for those customers with a grade of 200.

```
SELECT * FROM CUSTOMER WHERE Grade = 200;
```

Write a SQL query to display the order number followed by order date and the purchase amount for each order which will be delivered by the salesman who is holding the ID 5001.

```
SELECT Ord_No, Ord_Date, Purch_Amt FROM Orders WHERE Salesman_ID = 5001;
```

Write a SQL query to display the Nobel prizes for 1970.

```
SELECT * FROM Nobel_Win Where Year = 1970;
```

Write a SQL query to know the winner of the 1971 prize for Literature.

```
SELECT * FROM Nobel_Win Where Year = 1971 AND Subject = 'Literature';
```

Write a SQL query to display the year and subject that won 'Dennis Gabor' his prize.

```
SELECT Year, Subject FROM Nobel_Win Where Winner = 'Dennis Gabor';
```

Write a SQL query to give the name of the 'Physics' winners since the year 1950.

```
SELECT Name FROM Nobel_Win WHERE Year >= 1950 AND Subject = ‘Physics’;
```

Write a SQL query to Show all the details (year, subject, winner, country) of the Chemistry prize winners between the year 1965 to 1975 inclusive.

```
SELECT * FROM Nobel_Win
WHERE Year BETWEEN 1965 AND 1975 AND Subject = 'Chemistry';
```

Write a SQL query to show all details of the Prime Ministerial winners after 1972 of Menachem Begin and Yitzhak Rabin.

```
SELECT * FROM Nobel_Win
```

```
WHERE Year > 1972 AND Winner IN('Menachem Begin', 'Yitzhak Rabin');
```

Write a SQL query to show all the details of the winners with first name Louis.

```
SELECT * FROM Nobel_Win WHERE Winner LIKE 'Louis%';
```

Write a SQL query to show all the winners in Physics for 1970 together with the winner of Economics for 1971.

```
SELECT * FROM Nobel_Win
WHERE (Year = 1970 AND Subject = 'Physics')
      OR (Year = 1971 AND Subject = 'Economics');
```

OR

```
SELECT * FROM Nobel_Win
WHERE (Year = 1970 AND Subject = 'Physics')
UNION
SELECT * FROM Nobel_Win
WHERE (Year = 1971 AND Subject = 'Economics')
```

Write a SQL query to show all the winners of nobel prize in the year 1970 except the subject Physiology and Economics.

```
SELECT * FROM Nobel_Win WHERE Subject NOT IN('Physiology', 'Economics');
```

Write a SQL query to show the winners of a 'Physiology' prize in an early year before 1971 together with winners of a 'Peace' prize in a later year on and after the 1974.

```
SELECT * FROM Nobel_Win
WHERE(Year < 1971 AND Subject = 'Physiology')
UNION
SELECT * FROM Nobel_Win
WHERE (Year > 1974 AND Subject = 'Peace')
```

Write a SQL query to find all details of the prize won by Johannes Georg Bednorz.

```
SELECT * FROM Nobel_Win Where Winner = 'Johannes Georg Bednorz';
```

Write a SQL query to find all the details of the nobel winners for the subject not started with the letter 'P' and arranged the list as the most recent comes first, then by name in order.

```
SELECT * FROM Nobel_Win
WHERE Subject Not Like 'P%' ORDER BY Year DESC, Winner;
```

Write a SQL query to find all the details of 1970 winners by the ordered to subject and winner name; but the list contain the subject Economics and Chemistry at last.

```
SELECT * FROM Nobel_Win
WHERE YEAR = 1970 ORDER BY
CASE WHEN Subject IN ('Chemistry', 'Economics') THEN 1 ELSE 0
END, Subject, Winner;
```

Write a SQL query to find all the products with a price between Rs.200 and Rs.600

```
SELECT * FROM Item_Mast WHERE Pro_Price BETWEEN 200 AND 600;
```

Write a SQL query to calculate the average price of all products of the manufacturer which code is 16.

```
SELECT AVG(Pro_Price) FROM Item_Mast Where Pro_Com = 16;
```

Write a SQL query to find the item name and price in Rs.

```
SELECT Pro_Name AS "Item Name", Pro_Price AS "Price In Rs." From Item_Mast;
```

Write a SQL query to display the name and price of all the items with a price is equal or more than Rs.250, and the list contain the larger price first and then by name in ascending order.

```
SELECT Pro_Name, Pro_Price FROM Item_Mast
WHERE Pro_Price >= 250 ORDER BY Pro_Price DESC, Pro_Name;
```

Write a SQL query to display the average price of the items for each company, showing only the company code.

```
SELECT Pro_Com, AVG(Pro_Price) FROM Item_Mast GROUP BY Pro_Com;
```

Write a SQL query to find the name and price of the cheapest item(s).

```
SELECT Pro_Price, Pro_Name FROM Item_Mast
Where Pro_Price = (SELECT MIN(Pro_Price) FROM Item_Mast);
```

Write a query in SQL to find the last name of all employees, without duplicates.

```
SELECT DISTINCT Emp_Lname FROM EMP_DETAILS
```

Write a query in SQL to find the data of employees whose last name is 'Snares'

```
SELECT * FROM EMP_DETAILS WHERE Emp_Lname = 'Snares'
```

Write a query in SQL to display all the data of employees that work in the department 57.

```
SELECT * FROM EMP_DETAILS WHERE Emp_Dept = 57
```

Write a query in SQL to display name, salary and department ID of employees having maximum salary in their respective department.

```
SELECT Emp_Name, Emp_Sal, Dept_ID FROM Employee
WHERE Emp_Sal IN (Select MAX(Emp_Sal) FROM Employee GROUP BY Dept_ID);
```

Boolean and Relational operators

Write a query to display all customers with a grade above 100.

```
SELECT * FROM Customer WHERE Grade > 100;
```

Write a query statement to display all customers in New York who have a grade value above 100.

```
SELECT * FROM Customer WHERE Grade > 100 AND City = 'New York';
```

Write a SQL statement to display all customers, who are either belongs to the city New York or had a grade above 100.

```
SELECT * FROM Customer WHERE Grade > 100 OR City = 'New York';
```

Write a SQL statement to display all the customers, who are either belongs to the city New York or not had a grade above 100.

```
SELECT * FROM Customer WHERE City = 'New York' OR NOT GRADE > 100;
```

Write a SQL query to display those customers who are neither belongs to the city New York nor grade value is more than 100.

```
SELECT * FROM Customer WHERE Grade <= 100 AND City <> 'New York';
OR
SELECT * FROM Customer WHERE NOT (Grade > 100 OR City = 'New York');
```

Write a SQL statement to display either those orders which are not issued on date 2012-09-10 and issued by the salesman whose ID is 505 and below or those orders which purchase amount is 1000.00 and below.

```
SELECT * FROM Orders
WHERE NOT ((Ord_Date = '2012-09-10' AND Salesman_ID > 505)
OR Purch_Amt > 1000);
```

Write a SQL statement to display salesman_id, name, city and commission who gets the commission within the range more than 0.10% and less than 0.12%.

```
SELECT Salesman_ID, Name, City, Commission FROM Salesman
WHERE Commission > 0.10 AND Commission < 0.12;
```

Write a SQL query to display all orders where purchase amount less than 200 or exclude those orders which order date is on or greater than 10th Feb,2012 and customer id is below 3009.

```
SELECT * FROM Orders
WHERE (Purch_Amt < 200) OR
NOT (Ord_Date >= '2012/02/10' AND Customer_ID < 3009);
```

Write a SQL statement where
i) order dates are anything but 2012-08-17, or customer id is not greater than 3005
ii) and purchase amount is not below 1000.

```
SELECT * FROM Orders
WHERE NOT ((Ord_Date = '2012/08/17' OR Customer_ID > 3005)
AND Purch_Amt < 1000)
```

Write a SQL query to display order number, purchase amount, archived, the unachieved percentage for those order which exceeds the 50% of the target value of 6000.

```
SELECT Ord_No, Purch_Amt,
(100 * Purch_Amt)/6000 AS 'Achieved %',
(100 * (6000 - Purch_Amt))/6000 AS 'Unachieved %'
```

```
FROM Orders
WHERE (100 * Purch_Amt)/6000 > 50
```

NOTE : (Achieved And Unachieved % are calculated using purchase amount)

Write a query in SQL to find the data of employees whose last name is Dosni or Mardy.

```
SELECT * FROM Emp_Details WHERE Emp_Lname IN ('Dosni', 'Mardy')
```

Write a query in SQL to display all the data of employees that work in department 47 or department 63.

```
SELECT * FROM Emp_Details WHERE Emp_Dept IN (47, 63)
```

Wildcard and Special operators

Salesman :

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12
5007	Paul Adam	Rome	0.13

Customer :

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		5005
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	5003
3008	Julian Green	London	300	5002
3003	Jozy Altidor	Moscow	200	5007

Orders :

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.4	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.6	2012-04-25	3002	5001

TestTable :

col1
A001/DJ-402\44_/100/2015
A001_\DJ-402\44_/100/2015
A001_DJ-402-2014-2015
A002_DJ-401-2014-2015
A001/DJ_401
A001/DJ_402\44
A001/DJ_402\44\2015
A001/DJ-402%45\2015/200
A001/DJ_402\45\2015%100
A001/DJ_402%45\2015/300
A001/DJ-402\44

Emp_Details :

EMP_IDNO	EMP_FNAME	EMP_LNAME	EMP_DEPT
127323	Michale	Robbin	57
526689	Carlos	Snares	63
843795	Enric	Dosio	57
328717	Jhon	Snares	63
444527	Joseph	Dosni	47
659831	Zanifer	Emily	47
847674	Kuleswar	Sitaraman	57
748681	Henrey	Gabriel	47
555935	Alex	Manuel	57
539569	George	Mardy	27
733843	Mario	Saule	63
631548	Alan	Snappy	27
839139	Maria	Foster	57

Write a SQL statement to find those salesmen with all information who come from the city either Paris or Rome.

```
SELECT * FROM Salesman WHERE City IN ('Paris', 'Rome');
```

Write a query to filter those salesmen with all information who comes from any of the cities Paris and Rome.

```
SELECT * FROM Salesman WHERE City = 'Paris' OR City = 'Rome';
```

Write a query to produce a list of salesman_id, name, city and commision of each salesman who live in cities other than Paris and Rome.

```
SELECT Salesman_ID, Name, City, Commission FROM Salesman
WHERE NOT City IN ('Paris', 'Rome');
```

Write a query to sort out those customers with all information whose ID value is within any of 3007, 3008 and 3009.

```
SELECT * FROM Customer WHERE Customer_ID IN(3007, 3008, 3009);
```

Write a SQL statement to find those salesmen with all information who gets the commission within a range of 0.12 and 0.14.

```
SELECT * FROM Salesman WHERE Commission BETWEEN 0.12 AND 0.14
```

Write a query to filter all those orders with all information which purchase amount value is within the range 500 and 4000 except those orders of purchase amount value 948.50 and 1983.43.

```
SELECT * FROM Orders WHERE (Purch_Amt BETWEEN 500 AND 4000)
AND NOT Purch_Amt IN (948.50, 1983.43);
```

Write a SQL statement to find those salesmen with all other information and name started with any latter within 'A' and 'K'.

```
SELECT * FROM Salesman WHERE Name BETWEEN 'A%'AND 'K%'
```

Write a SQL statement to find those salesmen with all other information and name started with other than any latter within 'A' and 'L'.

```
SELECT * FROM Salesman WHERE NOT Name BETWEEN 'A%'AND 'L%'
```

Write a SQL statement to find that customer with all information whose name begin with the letter 'B'.

```
SELECT * FROM Customer WHERE Cust_Name LIKE 'B%'
```

Write a SQL statement to find all those customers with all information whose names are ending with the letter 'n'.

```
SELECT * FROM Customer WHERE Cust_Name LIKE '%n'
```

Write a SQL statement to find those salesmen with all information whose name containing the 1st character is 'N' and the 4th character is 'l' and rests may be any character.

```
SELECT * FROM Salesman WHERE Name LIKE 'N__l%'
```

Write a SQL statement to find those rows from the table testtable which contain the escape character underscore (_) in its column 'col1'.

```
SELECT * FROM TestTable WHERE Col1 Like '%/_%' ESCAPE '/'
```

Write a SQL statement to find those rows from the table testtable which does not contain the character underscore (_) in its column 'col1'.

```
SELECT * FROM TestTable WHERE NOT Col1 Like '%/_%' ESCAPE '/'
```

Write a SQL statement to find those rows from the table testtable which contain the escape character (/) in its column 'col1'.

```
SELECT * FROM TestTable WHERE Col1 Like '%//%' ESCAPE '/'
```

Write a SQL statement to find those rows from the table testtable which does not contain the escape character (/) in its column 'col1'.

```
SELECT * FROM TestTable WHERE NOT Col1 Like '%//%' ESCAPE '/'
```

Write a SQL statement to find those rows from the table testtable which contain the string (_/) in its column 'col1'.

```
SELECT * FROM TestTable WHERE Col1 Like '%/_//%' ESCAPE '/'
```

Write a SQL statement to find those rows from the table testtable which does not contain the

string (_/) in its column 'col1'.

```
SELECT * FROM TestTable WHERE NOT Col1 Like '%/_//%' ESCAPE '/'
```

Write a SQL statement to find those rows from the table testtable which contain the character (%) in its column 'col1'.

```
SELECT * FROM TestTable WHERE Col1 Like '%/%%' ESCAPE '/'
```

Write a SQL statement to find those rows from the table testtable which does not contain the character (%) in its column 'col1'.

```
SELECT * FROM TestTable WHERE NOT Col1 Like '%/%%' ESCAPE '/'
```

Write a SQL statement to find that customer with all information who does not get any grade except NULL.

```
SELECT * FROM Customer WHERE Grade IS NULL
```

Write a SQL statement to find that customer with all information who gets a grade except NULL value.

```
SELECT * FROM Customer WHERE NOT Grade IS NULL
```

Write a query in SQL to display all the data of employees whose last name begins with an 'D'.

```
SELECT * FROM Emp_Details WHERE Emp_Lname LIKE 'D%'
```

Aggregate Functions

Orders :

ord_no	purch_amt	ord_date	customer_id	salesman_id
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.50	2012-08-17	3009	5003
70005	2400.60	2012-07-27	3007	5001
70008	5760.00	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.40	2012-10-10	3009	5003
70011	75.29	2012-08-17	3003	5007
70013	3045.60	2012-04-25	3002	5001
70001	150.50	2012-10-05	3005	5002
70007	948.50	2012-09-10	3005	5002
70012	250.45	2012-06-27	3008	5002

Customer :

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006
3009	Geoff Cameron	Berlin	100	5003
3003	Jozy Altidor	Moscow	200	5007
3001	Brad Guzan	London		5005

Salesman :

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11

5006	Mc Lyon	Paris	0.14
5007	Paul Adam	Rome	0.13
5003	Lauson Hen	San Jose	0.12

Item_Mast :

pro_id	pro_name	pro_price	pro_com
101	Mother Board	3200.00	15
102	Key Board	450.00	16
103	ZIP drive	250.00	14
104	Speaker	550.00	16
105	Monitor	5000.00	11
106	DVD drive	900.00	12
107	CD drive	800.00	12
108	Printer	2600.00	13
109	Refill cartridge	350.00	13
110	Mouse	250.00	12

Emp_Department :

dpt_code	dpt_name	dpt_allotment
57	IT	65000
63	Finance	15000
47	HR	240000
27	RD	55000
89	QC	75000

Emp_Details :

emp_idno	emp_fname	emp_lname	emp_dept
631548	Alan	Snappy	27
839139	Maria	Foster	57
127323	Michale	Robbin	57
526689	Carlos	Snares	63
843795	Enric	Dosio	57
328717	Jhon	Snares	63
444527	Joseph	Dosni	47
659831	Zanifer	Emily	47
847674	Kuleswar	Sitaraman	57
748681	Henrey	Gabriel	47
555935	Alex	Manuel	57
539569	George	Mardy	27
733843	Mario	Saule	63

Write a SQL statement to find the total purchase amount of all orders.

```
SELECT SUM(Purch_Amt) FROM Orders
```

Write a SQL statement to find the average purchase amount of all orders.

```
SELECT AVG(Purch_Amt) FROM Orders
```

Write a SQL statement to find the number of salesmen currently listing for all of their customers.

```
SELECT COUNT(DISTINCT Salesman_ID) FROM Orders
```

Write a SQL statement know how many customer have listed their names.

```
SELECT COUNT(*) FROM Customer
```

Write a SQL statement find the number of customers who gets at least a gradation for his/her performance.

```
SELECT COUNT (ALL GRADE) FROM CUSTOMER
```

Write a SQL statement to get the maximum purchase amount of all the orders.

```
SELECT MAX(Purch_Amt) FROM Orders
```

Write a SQL statement to get the minimum purchase amount of all the orders.

```
SELECT MIN(Purch_Amt) FROM Orders
```

Write a SQL statement which selects the highest grade for each of the cities of the customers.

```
SELECT MAX(Grade), City FROM Customer GROUP BY City
```

Write a SQL statement to find the highest purchase amount ordered by the each customer with their ID and highest purchase amount.

```
SELECT MAX(Purch_Amt), Customer_ID FROM Orders GROUP BY Customer_ID
```

Write a SQL statement to find the highest purchase amount ordered by the each customer on a particular date with their ID, order date and highest purchase amount.

```
SELECT MAX(Purch_Amt), Customer_ID, Ord_Date FROM Orders
GROUP BY Customer_ID, Ord_Date
```

Write a SQL statement to find the highest purchase amount on a date '2012-08-17' for each salesman with their ID.

```
SELECT MAX(Purch_Amt), Salesman_ID FROM Orders
WHERE Ord_Date = '2012-08-17' GROUP BY Salesman_ID
```

Write a SQL statement to find the highest purchase amount with their ID and order date, for only those customers who have highest purchase amount in a day is more than 2000.

```
SELECT MAX(Purch_Amt), Customer_ID, Ord_Date FROM Orders
GROUP BY Customer_ID, Ord_Date HAVING MAX(Purch_Amt) > 2000
```

Write a SQL statement to find the highest purchase amount with their ID and order date, for those customers who have a higher purchase amount in a day is within the range 2000 and 6000.

```
SELECT MAX(Purch_Amt), Customer_ID, Ord_Date FROM Orders
GROUP BY Customer_ID, Ord_Date HAVING MAX(Purch_Amt) BETWEEN 2000 AND 6000
```

Write a SQL statement to find the highest purchase amount with their ID and order date, for only those customers who have a higher purchase amount in a day is within the list 2000, 3000, 5760 and 6000.

```
SELECT MAX(Purch_Amt), Customer_ID, Ord_Date FROM Orders
GROUP BY Customer_ID, Ord_Date HAVING MAX(Purch_Amt) IN(2000, 3000, 5760,
6000)
```

Write a SQL statement to find the highest purchase amount with their ID, for only those customers whose ID is within the range 3002 and 3007.

```
SELECT MAX(Purch_Amt), Customer_ID FROM Orders
WHERE Customer_ID BETWEEN 3002 AND 3007 GROUP BY Customer_ID
```

Write a SQL statement to display customer details (ID and purchase amount) whose IDs are within the range 3002 and 3007 and highest purchase amount is more than 1000.

```
SELECT MAX(Purch_Amt), Customer_ID FROM Orders
```

```
WHERE Customer_ID BETWEEN 3002 AND 3007
GROUP BY Customer_ID HAVING Max(Purch_Amt) > 1000
```

Write a SQL statement to find the highest purchase amount with their ID, for only those salesmen whose ID is within the range 5003 and 5008.

```
SELECT MAX(Purch_Amt), Salesman_ID FROM Orders
WHERE Salesman_ID BETWEEN 5003 AND 5008 GROUP BY Salesman_ID
```

Write a SQL statement that counts all orders for a date August 17th, 2012.

```
SELECT COUNT(*) FROM Orders WHERE Ord_Date = '2012/08/17'
```

Write a SQL statement that count the number of salesmen for whom a city is specified. Note that there may be spaces or no spaces in the city column if no city is specified.

```
SELECT COUNT(*) FROM Salesman WHERE City IS NOT NULL
```

Write a query that counts the number of salesmen with their order date and ID registering orders for each day.

```
SELECT Ord_Date, Salesman_ID, COUNT(*) FROM Orders
GROUP BY Ord_Date, Salesman_ID
```

Write a SQL query to calculate the average price of all the products.

```
SELECT AVG(Pro_Price) FROM Item_Mast
```

Write a SQL query to find the number of products with a price more than or equal to Rs.350.

```
SELECT COUNT(*) FROM Item_Mast WHERE Pro_Price >= 350
```

Write a SQL query to display the average price of each company's products, along with their code.

```
SELECT AVG(Pro_Price), Pro_Com From Item_Mast GROUP BY Pro_Com
```

Write a query in SQL to find the sum of the allotment amount of all departments.

```
SELECT SUM(Dpt_Allotment) FROM Emp_Department
```

Write a query in SQL to find the number of employees in each department along with the department code.

```
SELECT COUNT(*), Emp_Dept FROM Emp_Details GROUP BY Emp_Dept
```

Formatting Output

Write a SQL statement to display the commission with the percent sign (%) with salesman ID, name and city columns for all the salesmen.

```
SELECT Salesman_ID, Name, City, CONCAT((100 * Commission),'%') AS "%
Commission" FROM Salesman
```

Write a SQL statement to find out the number of orders booked for each day and display it in such a format like "For 2001-10-10 there are 15 orders".

```
SELECT CONCAT('For ', Ord_Date, ' there are ', COUNT(*), ' Orders')
AS "Number Of Orders" FROM Orders GROUP BY Ord_Date
```

Write a query to display the orders according to the order number arranged by ascending order.

```
SELECT * FROM Orders ORDER BY Ord_No
```

Write a SQL statement to arrange the orders according to the order date in such a manner that the latest date will come first then previous dates.

```
SELECT * FROM Orders ORDER BY Ord_Date DESC
```

Write a SQL statement to display the orders with all information in such a manner that, the older order date will come first and the highest purchase amount of same day will come first.

```
SELECT * FROM Orders ORDER BY Ord_Date ASC, Purch_Amt DESC
```

Write a SQL statement to display the customer name, city, and grade, etc. and the display will be arranged according to the smallest customer ID.

```
SELECT Cust_Name, City, Grade FROM Customer ORDER BY Customer_ID
```

Write a SQL statement to make a report with salesman ID, order date and highest purchase amount in such an arrangement that, the smallest salesman ID will come first along with their smallest order date.

```
SELECT Salesman_ID, Ord_Date, MAX(Purch_Amt) FROM Orders
GROUP BY Salesman_ID, Ord_Date
ORDER BY Salesman_ID, Ord_Date
```

Write a SQL statement to display customer name, city and grade in such a manner that, the customer holding highest grade will come first.

```
SELECT Cust_Name, City, Grade FROM Customer ORDER BY Grade DESC
```

Write a SQL statement to make a report with customer ID in such a manner that, the largest number of orders booked by the customer will come first along with their highest purchase amount.

```
SELECT Customer_ID, COUNT(*), MAX(Purch_Amt) FROM Orders
GROUP BY Customer_ID ORDER BY COUNT DESC
```

Write a SQL statement to make a report with order date in such a manner that, the latest order date will come last along with the total purchase amount and total commission (15% for all salesmen) for that date.

```
SELECT Ord_Date, SUM(Purch_Amt), ((SUM(Purch_Amt)/100) * 15) AS "Commission"
FROM Orders GROUP BY Ord_Date ORDER BY Ord_Date
```

Query on Multiple Tables

Write a query to find those customers with their name and those salesmen with their name and city who lives in the same city.

```
SELECT Customer.Cust_Name, Salesman.Name, Salesman.City FROM Salesman,
Customer
WHERE Customer.City = Salesman.City
```

Write a SQL statement to find the names of all customers along with the salesmen who works for them.

```
SELECT Customer.Cust_Name, Salesman.Name FROM Salesman, Customer
WHERE Customer.Salesman_ID = Salesman.Salesman_ID
```

Write a SQL statement to display all those orders by the customers not located in the same cities where their salesmen live.

```
SELECT Orders.Ord_No, Customer.Cust_Name, Orders.Customer_ID,
Orders.Salesman_ID
FROM Orders, Customer, Salesman
WHERE Customer.City <> Salesman.City
AND Orders.Customer_ID = Customer.Customer_ID
AND Orders.Salesman_ID = Salesman.Salesman_ID
```

Write a SQL statement that finds out each order number followed by the name of the customers who made the order.

```
SELECT Orders.Ord_No, Customer.Cust_Name
FROM Orders, Customer
WHERE Orders.Customer_ID = Customer.Customer_ID
```

Write a SQL statement that sorts out the customer and their grade who made an order. Each of the customers must have a grade and served by at least a salesman, who belongs to a city.

```
SELECT Customer.Customer_ID, Customer.Grade, Salesman.City
FROM Customer, Salesman, Orders
WHERE Orders.Customer_ID = Customer.Customer_ID AND
Orders.Salesman_ID = Salesman.Salesman_ID
AND Customer.Grade IS NOT NULL
AND Salesman.City IS NOT NULL
```

Write a query that produces all customers with their name, city, salesman and commission, who served by a salesman and the salesman works at a rate of the commission within 12% to 14%.

```
SELECT Customer.Cust_Name, Customer.City, Salesman.Name,
Salesman.Salesman_ID, CONCAT(Salesman.Commission * 100, '%') AS "Commission
%" FROM Salesman, Customer
WHERE Customer.Salesman_ID = Salesman.Salesman_ID
AND Salesman.Commission BETWEEN 0.12 AND 0.14
```

Write a SQL statement that produces all orders with the order number, customer name, commission rate and earned commission amount for those customers who carry their grade is 200 or more and served by an existing salesman.

```
SELECT Orders.Ord_No, Customer.Cust_Name,
CONCAT((100 * Salesman.Commission), '%') AS "Commission %",
(Orders.Purch_Amt * Salesman.Commission) AS "Commission_Amt"
FROM Customer, Orders, Salesman
WHERE Orders.Salesman_ID = Salesman.Salesman_ID
AND Orders.Customer_ID = Customer.Customer_ID
AND Customer.Grade >= 200
```

SORTING and FILTERING on HR Database

Write a query in SQL to display the full name (first and last name), and salary for those employees who earn below 6000.

```
SELECT CONCAT(First_Name, ' ', Last_Name) AS "Full Name", Salary FROM
Employees WHERE SALARY < 6000
```

Write a query in SQL to display the first and last_name, department number and salary for those employees who earn more than 8000.

```
SELECT First_Name, Last_Name, Department_ID, Salary FROM Employees
WHERE SALARY > 8000
```

Write a query in SQL to display the first and last name, and department number for all employees whose last name is “McEwen”.

```
SELECT First_Name, Last_Name, Department_ID FROM Employees
WHERE Last_Name = 'McEwen'
```

Write a query in SQL to display all the information for all employees without any department number.

```
SELECT * FROM Employees WHERE Department_ID IS NULL
```

Write a query in SQL to display all the information about the department Marketing.

```
SELECT * FROM Departments WHERE Department_Name = 'Marketing'
```

Write a query in SQL to display the full name (first and last), hire date, salary, and department number for those employees whose first name does not containing the letter M and make the result set in ascending order by department number.

```
SELECT CONCAT(First_Name, ' ', Last_Name) AS "Full Name",
Hire_Date, Salary, Department_ID FROM Employees
WHERE NOT First_Name LIKE '%M%' ORDER BY Department_ID
```

Write a query in SQL to display all the information of employees whose salary is in the range of 8000 and 12000 and commission is not null or department number is except the number 40, 120 and 70 and they have been hired before June 5th, 1987.

```
SELECT * FROM Employees
WHERE (Salary BETWEEN 8000 AND 12000 AND Commission_PCT IS NOT NULL)
OR (Department_ID IN(40, 120, 70) AND Hire_Date < '1987/06/05')
```

Write a query in SQL to display the full name (first and last name), and salary for all employees who does not earn any commission.

```
SELECT CONCAT(First_Name, ' ', Last_Name) AS "Full Name", Salary
FROM Employees WHERE Commission_PCT = 0
```

Write a query in SQL to display the full name (first and last), the phone number and email separated by hyphen, and salary, for those employees whose salary is within the range of 9000 and 17000. The column headings assign with Full_Name, Contact_Details and Remuneration respectively.

```
SELECT CONCAT(First_Name, ' ', Last_Name) AS "Full_Name",
CONCAT(Phone_Number, '-', Email) AS "Contact_Details",
Salary AS "Remuneration"
FROM Employees WHERE Salary BETWEEN 9000 AND 17000
```

Write a query in SQL to display the first and last name, and salary for those employees whose

first name is ending with the letter m.

```
SELECT First_Name, Last_Name, Salary FROM Employees
WHERE First_Name Like '%m'
```

Write a query in SQL to display the full name (first and last) name, and salary, for all employees whose salary is out of the range 7000 and 15000 and make the result set in ascending order by the full name.

```
SELECT CONCAT(First_Name, ' ', Last_Name) AS "Full_Name",
Salary FROM Employees
WHERE NOT Salary BETWEEN 7000 AND 15000
ORDER BY CONCAT(First_Name, ' ', Last_Name)
```

Write a query in SQL to display the full name (first and last), job id and date of hire for those employees who was hired during November 5th, 2007 and July 5th, 2009.

```
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) AS "FULL_NAME",
JOB_ID, HIRE_DATE FROM EMPLOYEES
WHERE HIRE_DATE BETWEEN '2007-11-05' AND '2009-07-5'
```

Write a query in SQL to display the the full name (first and last name), and department number for those employees who works either in department 70 or 90.

```
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) AS "FULL_NAME",
DEPARTMENT_ID FROM EMPLOYEES WHERE DEPARTMENT_ID IN (70, 90)
```

Write a query in SQL to display the full name (first and last name), salary, and manager number for those employees who is working under a manager.

```
SELECT CONCAT(FIRST_NAME, ' ', LAST_NAME) AS "FULL_NAME",
SALARY, MANAGER_ID FROM EMPLOYEES WHERE MANAGER_ID IS NOT NULL
```

Write a query in SQL to display all the information from Employees table for those employees who was hired before June 21st, 2002.

```
SELECT * FROM EMPLOYEES WHERE HIRE_DATE < '2002-06-21'
```

Write a query in SQL to display the first and last name, email, salary and manager ID, for those employees whose managers are hold the ID 120, 103 or 145.

```
SELECT FIRST_NAME, LAST_NAME, EMAIL, SALARY, MANAGER_ID FROM EMPLOYEES
WHERE MANAGER_ID IN (120, 103, 145)
```

Write a query in SQL to display all the information for all employees who have the letters D, S, or N in their first name and also arrange the result in descending order by salary.

```
SELECT * FROM EMPLOYEES
WHERE FIRST_NAME LIKE '%D%'
OR FIRST_NAME LIKE  '%S%'
OR FIRST_NAME LIKE '%N%'
ORDER BY  SALARY DESC
```

Write a query in SQL to display the full name (first name and last name), hire date, commission percentage, email and telephone separated by '-', and salary for those employees who earn the salary above 11000 or the seventh digit in their phone number equals 3 and make the result set in a descending order by the first name.

```
SELECT CONCAT(FIRST_NAME, ` `, LAST_NAME) AS “FULL_NAME”,
HIRE_DATE, COMMISSION_PCT,
CONCAT(EMAIL, `-`, PHONE_NUMBER) AS “CONTACT_DETAILS”
```

```
FROM EMPLOYEES
WHERE SALARY > 11000
OR PHONE_NUMBER LIKE '_____7%'
ORDER BY FIRST_NAME DESC
```

Write a query in SQL to display the first and last name, and department number for those employees who holds a letter s as a 3rd character in their first name.

```
SELECT FIRST_NAME, LAST_NAME, DEPARTMENT_ID FROM EMPLOYEES
WHERE FIRST_NAME LIKE '__s%'
```

Write a query in SQL to display the employee ID, first name, job id, and department number for those employees who is working except the departments 50,30 and 80.

```
SELECT EMPLOYEE_ID, FIRST_NAME, JOB_ID, DEPARTMENT_ID FROM EMPLOYEES
WHERE NOT DEPARTMENT_ID IN (80, 30, 50)
```

Write a query in SQL to display the employee Id, first name, job id, and department number for those employees whose department number equals 30, 40 or 90.

```
SELECT EMPLOYEE_ID, FIRST_NAME, JOB_ID, DEPARTMENT_ID FROM EMPLOYEES
WHERE DEPARTMENT_ID IN (40, 30, 90)
```

Write a query in SQL to display the ID for those employees who did two or more jobs in the past.

```
SELECT COUNT(*), EMPLOYEE_ID FROM JOB_HISTORY
GROUP BY EMPLOYEE_ID
HAVING COUNT(*) > 1
```

Write a query in SQL to display job ID, number of employees, sum of salary, and difference between highest salary and lowest salary for a job.

```
SELECT JOB_ID, COUNT(*) AS "NUM_EMPLOYEES",
SUM(SALARY) AS "SUM_SALARY",
(MAX(SALARY) - MIN(SALARY)) AS "DIFFERENCE"
FROM EMPLOYEES GROUP BY JOB_ID
```

Write a query in SQL to display job ID for those jobs that were done by two or more for more than 300 days.

```
SELECT JOB_ID FROM JOB_HISTORY
WHERE (END_DATE - START_DATE) > 300
GROUP BY JOB_ID HAVING COUNT(*) > 1
```

Write a query in SQL to display the country ID and number of cities in that country we have.

```
SELECT COUNTRY_ID, COUNT(*) AS "# CITIES" FROM LOCATIONS
GROUP BY COUNTRY_ID
```

Write a query in SQL to display the manager ID and number of employees managed by the manager.

```
SELECT MANAGER_ID, COUNT(*) AS "NUM_EMPLOYEES" FROM EMPLOYEES
GROUP BY MANAGER_ID
```

Write a query in SQL to display the details of jobs in descending sequence on job title.

```
SELECT * FROM JOBS ORDER BY JOB_TITLE DESC
```

Write a query in SQL to display the first and last name and date of joining of the employees

who is either Sales Representative or Sales Man.

```
SELECT FIRST_NAME, LAST_NAME, HIRE_DATE FROM EMPLOYEES
WHERE JOB_ID IN ('SA_MAN', 'SA_REP')
```

Write a query in SQL to display the average salary of employees for each department who gets a commission percentage.

```
SELECT AVG(SALARY), DEPARTMENT_ID FROM EMPLOYEES
WHERE COMMISSION_PCT IS NOT NULL GROUP BY DEPARTMENT_ID
```

Write a query in SQL to display those departments where any manager is managing 4 or more employees.

```
SELECT DEPARTMENT_ID FROM EMPLOYEES GROUP BY DEPARTMENT_ID
HAVING COUNT(*) >= 4
```

Write a query in SQL to display those departments where more than ten employees work who got a commission percentage.

```
SELECT DEPARTMENT_ID FROM EMPLOYEES WHERE COMMISSION_PCT IS NOT NULL
GROUP BY DEPARTMENT_ID HAVING COUNT(*) > 10
```

Write a query in SQL to display the employee ID and the date on which he ended his previous job.

```
SELECT EMPLOYEE_ID, MAX(END_DATE) FROM JOB_HISTORY GROUP BY EMPLOYEE_ID
```

Write a query in SQL to display the details of the employees who have no commission percentage and salary within the range 7000 to 12000 and works in that department which number is 50.

```
SELECT * FROM EMPLOYEES WHERE COMMISSION IS NULL
AND SALARY BETWEEN 7000 AND 12000
AND DEPARTMENT_ID = 50
```

Write a query in SQL to display the job ID for those jobs which average salary is above 8000.

```
SELECT JOB_ID FROM EMPLOYEES GROUP BY JOB_ID HAVING AVG(SALARY) > 8000
```

Write a query in SQL to display job Title, the difference between minimum and maximum salaries for those jobs which max salary within the range 12000 to 18000.

```
SELECT JOB_TITLE, (MAX_SALARY - MIN_SALARY) AS "DIFFERENCE"
FROM JOBS WHERE MAX_SALARY BETWEEN 12000 AND 18000
```

Write a query in SQL to display all those employees whose first name or last name starts with the letter D.

```
SELECT * FROM EMPLOYEES WHERE FIRST_NAME LIKE 'D%' OR LAST_NAME LIKE 'D%'
```

Write a query in SQL to display the details of jobs which minimum salary is greater than 9000.

```
SELECT * FROM JOBS WHERE MIN_SALARY > 9000
```

Write a query in SQL to display those employees who joined after 7th September, 1987.

```
SELECT * FROM EMPLOYEES WHERE HIRE_DATE > '1987-09-07'
```

JOINS

Write a SQL statement to prepare a list with salesman name, customer name and their cities for the salesmen and customer who belongs to the same city.

```
SELECT SALESMAN.NAME, CUSTOMER.CUST_NAME, SALESMAN.CITY
FROM SALESMAN INNER JOIN CUSTOMER ON SALESMAN.CITY = CUSTOMER.CITY
```

THIS IS SAME AS

```
SELECT SALESMAN.NAME, CUSTOMER.CUST_NAME, SALESMAN.CITY
FROM SALESMAN, CUSTOMER WHERE SALESMAN.CITY = CUSTOMER.CITY
```

Write a SQL statement to make a list with order no, purchase amount, customer name and their cities for those orders which order amount between 500 and 2000.

```
SELECT ORDERS.ORD_NO, ORDERS.PURCH_AMT, CUSTOMER.CUST_NAME, CUSTOMER.CITY
FROM CUSTOMER INNER JOIN ORDERS
ON CUSTOMER.CUSTOMER_ID = ORDERS.CUSTOMER_ID
WHERE ORDERS.PURCH_AMT BETWEEN 500 AND 2000
```

Write a SQL statement to know which salesman are working for which customer.

```
SELECT SALESMAN.NAME, CUSTOMER.CUST_NAME
FROM CUSTOMER INNER JOIN SALESMAN
ON SALESMAN.SALESMAN_ID = CUSTOMER.SALESMAN_ID
```

Write a SQL statement to find the list of customers who appointed a salesman for their jobs who gets a commission from the company is more than 12%.

```
SELECT CUSTOMER.CUST_NAME,
(100 * SALESMAN.COMMISSION) AS "COMMISSION %", SALESMAN.NAME
FROM CUSTOMER INNER JOIN SALESMAN
ON CUSTOMER.SALESMAN_ID = SALESMAN.SALESMAN_ID
WHERE SALESMAN.COMMISSION > 0.12
```

Write a SQL statement to find the list of customers who appointed a salesman for their jobs who does not live in the same city where their customer lives, and gets a commission is above 12% .

```
SELECT CUSTOMER.CUST_NAME,
(100 * SALESMAN.COMMISSION) AS "COMMISSION %", SALESMAN.NAME
FROM CUSTOMER INNER JOIN SALESMAN
ON CUSTOMER.SALESMAN_ID = SALESMAN.SALESMAN_ID
WHERE SALESMAN.COMMISSION > 0.12
AND SALESMAN.CITY <> CUSTOMER.CITY
```

Write a SQL statement to find the details of a order i.e. order number, order date, amount of order, which customer gives the order and which salesman works for that customer and how much commission he gets for an order.

```
SELECT ORDERS.ORD_NO, ORDERS.PURCH_AMT, ORDERS.ORD_DATE,
CUSTOMER.CUST_NAME, SALESMAN.NAME AS "SALESMAN_NAME",
(SALESMAN.COMMISSION * 100) AS "COMMISSION %"
FROM ORDERS INNER JOIN CUSTOMER
ON ORDERS.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
INNER JOIN SALESMAN
ON ORDERS.SALESMAN_ID = SALESMAN.SALESMAN_ID
```

Write a SQL statement to make a join on the tables salesman, customer and orders in such a form that the same column of each table will appear once and only the relational rows will come.

```
SELECT * FROM ORDERS NATURAL JOIN CUSTOMER NATURAL JOIN SALESMAN
```

Write a SQL statement to make a list in ascending order for the customer who works either through a salesman or by own.

```
SELECT CUSTOMER.CUSTOMER_ID, CUSTOMER.CUST_NAME, CUSTOMER.CITY,
SALESMAN.NAME AS "SALESMAN"
FROM CUSTOMER LEFT JOIN SALESMAN
ON CUSTOMER.SALESMAN_ID = SALESMAN.SALESMAN_ID
ORDER BY CUSTOMER.CUSTOMER_ID
```

Write a SQL statement to make a list in ascending order for the customer who holds a grade less than 300 and works either through a salesman or by own.

```
SELECT CUSTOMER.CUSTOMER_ID, CUSTOMER.CUST_NAME, CUSTOMER.CITY,
CUSTOMER.GRADE, SALESMAN.NAME AS "SALESMAN"
FROM CUSTOMER LEFT JOIN SALESMAN
ON CUSTOMER.SALESMAN_ID = SALESMAN.SALESMAN_ID
WHERE GRADE < 300
```

Write a SQL statement to make a report with customer name, city, order number, order date, and order amount in ascending order according to the order date to find that either any of the existing customers have placed no order or placed one or more orders.

```
SELECT CUSTOMER.CUST_NAME, CUSTOMER.CITY, ORDERS.ORD_NO,
ORDERS.ORD_DATE, ORDERS.PURCH_AMT
FROM CUSTOMER LEFT JOIN ORDERS
ON CUSTOMER.CUSTOMER_ID = ORDERS.CUSTOMER_ID
ORDER BY ORDERS.ORD_DATE
```

Write a SQL statement to make a report with customer name, city, order number, order date, order amount salesman name and commission to find that either any of the existing customers have placed no order or placed one or more orders by their salesman or by own.

```
SELECT CUSTOMER.CITY, CUSTOMER.CUST_NAME, ORDERS.ORD_NO,
ORDERS.PURCH_AMT, SALESMAN.NAME AS "SALESMAN",
SALESMAN.COMMISSION FROM  CUSTOMER LEFT JOIN ORDERS
ON CUSTOMER.CUSTOMER_ID = ORDERS.CUSTOMER_ID
LEFT JOIN SALESMAN ON SALESMAN.SALESMAN_ID = CUSTOMER.SALESMAN_ID
```

Write a SQL statement to make a list in ascending order for the salesmen who works either for one or more customer or not yet join under any of the customers.

```
SELECT SALESMAN.SALESMAN_ID, SALESMAN.NAME,
SALESMAN.CITY, CUSTOMER.CUST_NAME FROM SALESMAN LEFT JOIN CUSTOMER
ON CUSTOMER.SALESMAN_ID = SALESMAN.SALESMAN_ID
ORDER BY SALESMAN.SALESMAN_ID
```

Write a SQL statement to make a list for the salesmen who works either for one or more customer or not yet join under any of the customers who placed either one or more orders or no order to their supplier.

```
SELECT SALESMAN.SALESMAN_ID, SALESMAN.NAME, SALESMAN.CITY,
CUSTOMER.CUST_NAME,CUSTOMER.GRADE, ORDERS.ORD_NO, ORDERS.PURCH_AMT
FROM SALESMAN
LEFT JOIN CUSTOMER ON CUSTOMER.SALESMAN_ID = SALESMAN.SALESMAN_ID
LEFT JOIN ORDERS ON ORDERS.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
```

Write a SQL statement to make a list for the salesmen who either work for one or more customers or yet to join any of the customer. The customer may have placed, either one or more orders on or above order amount 2000 and must have a grade, or he may not have placed any order to the

associated supplier.

```
SELECT CUSTOMER.CUST_NAME, CUSTOMER.CITY, CUSTOMER.GRADE,
SALESMAN.SALESMAN_NAME, ORDERS.ORD_NO, ORDERS.ORD_DATE, ORDERS.PURCH_AMT
FROM SALESMAN LEFT JOIN CUSTOMER
ON CUSTOMER.SALESMAN_ID = SALESMAN.SALESMAN_ID LEFT JOIN ORDERS
ON ORDERS.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
WHERE (PURCH_AMT >= 2000 AND GRADE IS NOT NULL)
OR ORDERS.ORD_NO IS NULL
```

Write a SQL statement to make a report with customer name, city, order no. order date, purchase amount for those customers from the existing list who placed one or more orders or which order(s) have been placed by the customer who is not on the list.

```
SELECT CUSTOMER.CUST_NAME, CUSTOMER.CITY, ORDERS.ORD_NO,
ORDERS.ORD_DATE, ORDERS.PURCH_AMT
FROM ORDERS LEFT JOIN CUSTOMER
ON ORDERS.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
```

Write a SQL statement to make a report with customer name, city, order no. order date, purchase amount for only those customers on the list who must have a grade and placed one or more orders or which order(s) have been placed by the customer who is neither in the list not have a grade.

```
SELECT CUSTOMER.CUST_NAME, CUSTOMER.CITY, ORDERS.ORD_NO, ORDERS.ORD_DATE,
ORDERS.PURCH_AMT
FROM  CUSTOMER FULL OUTER JOIN ORDERS
ON CUSTOMER.CUSTOMER_ID = ORDERS.CUSTOMER_ID
WHERE (CUSTOMER.CUSTOMER_ID IS NOT NULL AND CUSTOMER.GRADE IS NOT NULL)
OR (CUSTOMER.CUSTOMER_ID IS NULL AND CUSTOMER.GRADE IS NULL)
```

Write a SQL statement to make a cartesian product between salesman and customer i.e. each salesman will appear for all customer and vice versa.

```
SELECT SALESMAN.SALESMAN_NAME, CUSTOMER.CUST_NAME FROM SALESMAN, CUSTOMER
OR
SELECT SALESMAN.SALESMAN_NAME, CUSTOMER.CUST_NAME
FROM SALESMAN CROSS JOIN CUSTOMER
```

Write a SQL statement to make a cartesian product between salesman and customer i.e. each salesman will appear for all customer and vice versa for that customer who belongs to a city.

```
SELECT SALESMAN.SALESMAN_NAME, CUSTOMER.CUST_NAME , SALESMAN.CITY
FROM SALESMAN CROSS JOIN CUSTOMER
WHERE SALESMAN.CITY IS NOT NULL
```

Write a SQL statement to make a cartesian product between salesman and customer i.e. each salesman will appear for all customer and vice versa for those salesmen who belongs to a city and the customers who must have a grade.

```
SELECT SALESMAN.SALESMAN_NAME, CUSTOMER.* , SALESMAN.*
FROM SALESMAN CROSS JOIN CUSTOMER
WHERE SALESMAN.CITY IS NOT NULL AND CUSTOMER.GRADE IS NOT NULL
```

Write a SQL statement to make a cartesian product between salesman and customer i.e. each salesman will appear for all customer and vice versa for those salesmen who must belong a city which is not the same as his customer and the customers should have an own

grade.

```
SELECT SALESMAN.SALESMAN_NAME, CUSTOMER.* , SALESMAN.*
FROM SALESMAN CROSS JOIN CUSTOMER
WHERE SALESMAN.CITY IS NOT NULL AND CUSTOMER.GRADE IS NOT NULL
AND CUSTOMER.CITY <> SALESMAN.CITY
```

Write a SQL query to display all the data from the item_mast, including all the data for each item's producer company.

```
SELECT * FROM ITEM_MAST INNER JOIN COMPANY_MAST
ON ITEM_MAST.PRO_COM = COMPANY_MAST.COM_ID
```

Write a SQL query to display the item name, price, and company name of all the products.

```
SELECT ITEM_MAST.PRO_NAME, ITEM_MAST.PRO_PRICE, COMPANY_MAST.COM_NAME
FROM ITEM_MAST INNER JOIN COMPANY_MAST
ON COMPANY_MAST.COM_ID = ITEM_MAST.PRO_COM
```

Write a SQL query to display the average price of items of each company, showing the name of the company.

```
SELECT COMPANY_MAST.COM_NAME, AVG(ITEM_MAST.PRO_PRICE) AS 'AVERAGE_PRICE'
FROM COMPANY_MAST INNER JOIN ITEM_MAST
ON COMPANY_MAST.COM_ID = ITEM_MAST.PRO_COM
GROUP BY COMPANY_MAST.COM_NAME
```

Write a SQL query to display the names of the company whose products have an average price larger than or equal to Rs. 350.

```
SELECT COMPANY_MAST.COM_NAME, AVG(ITEM_MAST.PRO_PRICE) AS 'AVERAGE_PRICE'
FROM COMPANY_MAST INNER JOIN ITEM_MAST
ON COMPANY_MAST.COM_ID = ITEM_MAST.PRO_COM
GROUP BY COMPANY_MAST.COM_NAME
HAVING AVG(ITEM_MAST.PRO_PRICE) > 350
```

Write a SQL query to display the name of each company along with the ID and price for their most expensive product.

```
SELECT COMPANY_MAST.COM_ID, COMPANY_MAST.COM_NAME,
MAX(ITEM_MAST.PRO_PRICE) AS 'MAX_PRICE'
FROM COMPANY_MAST INNER JOIN ITEM_MAST
ON COMPANY_MAST.COM_ID = ITEM_MAST.PRO_COM
GROUP BY COMPANY_MAST.COM_ID, COMPANY_MAST.COM_NAME
```

Write a query in SQL to display all the data of employees including their department.

```
SELECT * FROM EMP_DETAILS INNER JOIN EMP_DEPARTMENT ON
EMP_DEPARTMENT.DPT_CODE = EMP_DETAILS.EMP_DEPT
```

Write a query in SQL to display the first name and last name of each employee, along with the name and sacntion amount for their department.

```
SELECT EMP_DETAILS.EMP_FNAME, EMP_DETAILS.EMP_LNAME,
EMP_DEPARTMENT.DPT_NAME, EMP_DEPARTMENT.DPT_ALLOTMENT
FROM EMP_DETAILS INNER JOIN EMP_DEPARTMENT
ON EMP_DEPARTMENT.DPT_CODE = EMP_DETAILS.EMP_DEPT
```

Write a query in SQL to find the first name and last name of employees working for departments with a budget more than Rs. 50000.

```
SELECT EMP_DETAILS.EMP_FNAME, EMP_DETAILS.EMP_LNAME,
```

```
EMP_DEPARTMENT.DPT_NAME, EMP_DEPARTMENT.DPT_ALLOTMENT
FROM EMP_DETAILS INNER JOIN EMP_DEPARTMENT
ON EMP_DEPARTMENT.DPT_CODE = EMP_DETAILS.EMP_DEPT
WHERE EMP_DEPARTMENT.DPT_ALLOTMENT > 50000
```

Write a query in SQL to find the names of departments where more than two employees are working.

```
SELECT EMP_DEPARTMENT.DPT_NAME, COUNT(EMP_DETAILS.*) AS "NUM_EMPLOYEES"
FROM EMP_DETAILS INNER JOIN EMP_DEPARTMENT
ON EMP_DEPARTMENT.DPT_CODE = EMP_DETAILS.EMP_DEPT
GROUP BY EMP_DEPARTMENT.DPT_NAME
HAVING COUNT(EMP_DETAILS.*) > 1
```