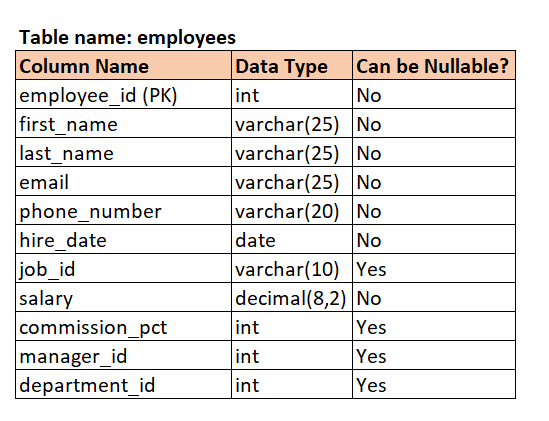
1. **From the Table DDL of employees table given below, Write a query to create a table employees**.

**Note:**

* Make sure to apply **NOT NULL** and Primary Key (PK) condition for the columns where ever applicable.
* While creating a table, table name and column name should be exactly same as given in DDL (including upper/lower case)



**Ans:**

**create table employees(**

**employee\_id int not null primary key,**

**first\_name varchar(25) not null,**

**last\_name varchar(25) not null,**

**email varchar(25)not null,**

**phone\_number varchar(20) not null,**

**hire\_date date not null,**

**job\_id varchar(10),**

**salary decimal(8,2) not null,**

**commission\_pct int ,**

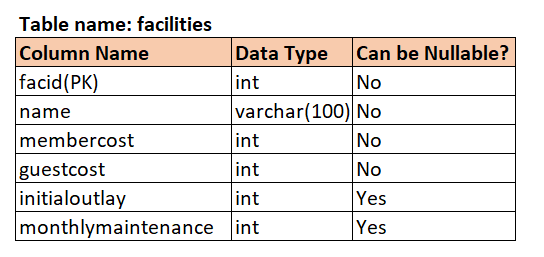
**manager\_id int,**

**department\_id int**

**);**

1. **Write a query to drop the table facilities.**

**Hint: Refer the table for the field names and its data type of the table.**



**Input format**

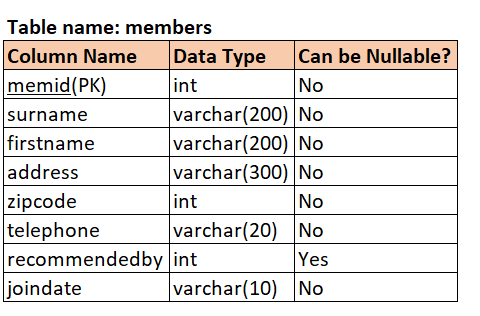
The required tables are populated in the backend.

**Ans:**

**drop table facilities;**

1. **From the DDL given below,**

**Write a query to change the column name zipcode to pincode in members table.**



**Input format**

The members table is already created.

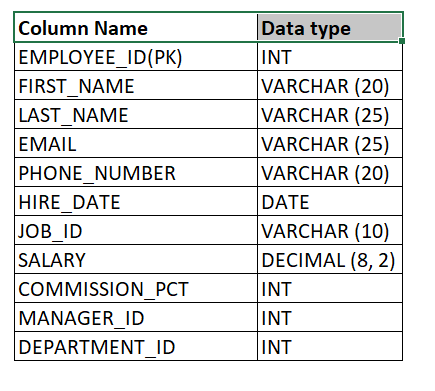
**Ans:**

**alter table members**

**change column zipcode pincode int;**

1. **Write a query to insert any 4 records to the table 'employees'**

**The column names and data types are given below.**

****

**Input format**

The required input table is created in the back end.

**Ans:**

**INSERT into employees(EMPLOYEE\_ID,FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,JOB\_ID,SALARY,COMMISSION\_PCT,MANAGER\_ID,DEPARTMENT\_ID)**

**VALUES(1,'Adhi','B','adhi@gmail.com','129988','2004/04/22','100',700000.00,11,141,111);**

**INSERT into employees(EMPLOYEE\_ID,FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,JOB\_ID,SALARY,COMMISSION\_PCT,MANAGER\_ID,DEPARTMENT\_ID)**

**VALUES(2,'nikita','M','niki@gmail.com','16166','2022/04/04','200',600000.00,12,121,1221);**

**INSERT into employees(EMPLOYEE\_ID,FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,JOB\_ID,SALARY,COMMISSION\_PCT,MANAGER\_ID,DEPARTMENT\_ID)**

**VALUES(3,'monish','K','monish@gmail.com','18881','2004/08/18','300',650000.00,13,131,1331);**

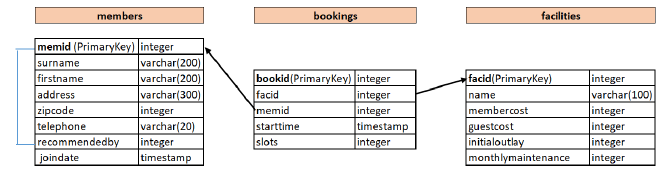
**INSERT into employees(EMPLOYEE\_ID,FIRST\_NAME,LAST\_NAME,EMAIL,PHONE\_NUMBER,HIRE\_DATE,JOB\_ID,SALARY,COMMISSION\_PCT,MANAGER\_ID,DEPARTMENT\_ID)**

**VALUES(4,'aryan','M','aryan@gmail.com','19991','2022/05/13','400',200000.00,14,141,1441);**

1. **Write a query to update facility name as 'Snooker Table-2' instead of facility name 'Pool Table' infacilities table.**

**name column in facilities table refers to facilities name.**

**Refer the table details below:**

****

**Input format**

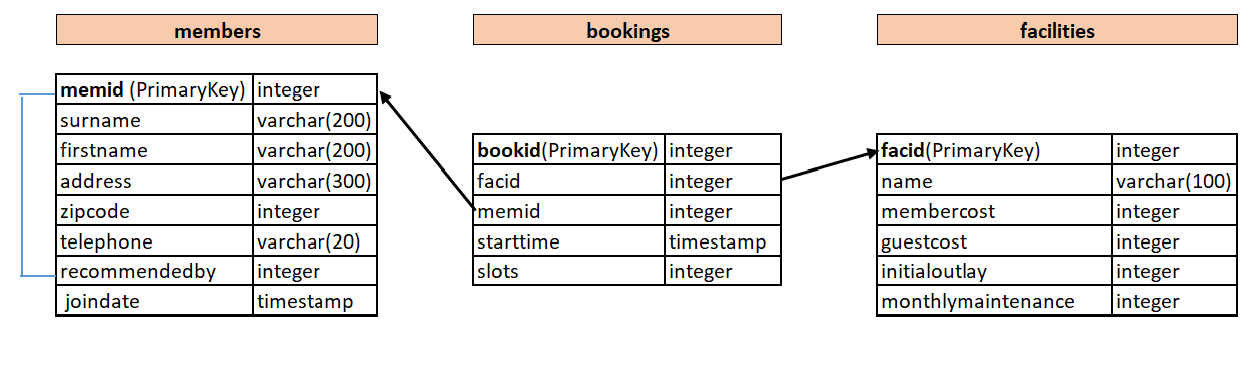
The required input table is created in the back end.

**Ans:**

**update facilities set name='Snooker Table-2' where name='Pool Table';**

1. **Write a query to delete all the records from 'members' table**

Refer the table details below:

****

**Note:**

Table names are case sensitive.

**members**table is already created in the backend.

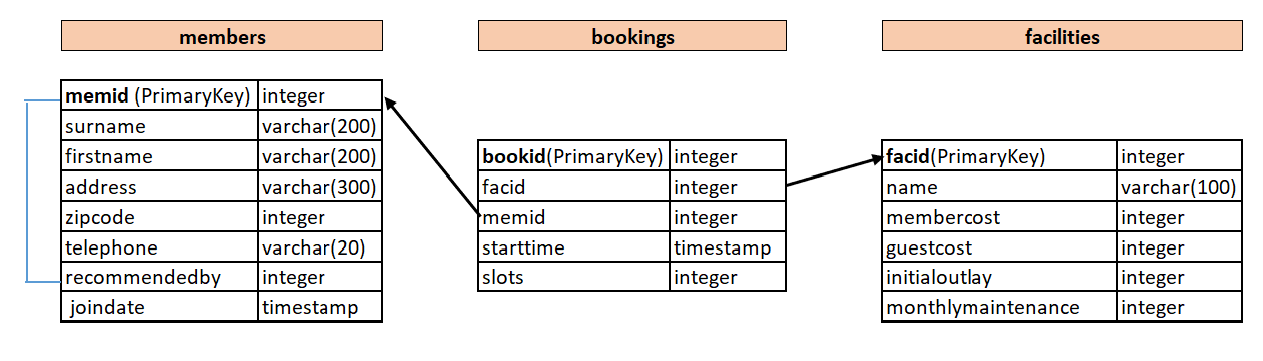
**Input format**

The required input table is created in the back end.

**Ans:**

**delete from members;**

1. **Write a query update the column membercost to 1000 from the table facilities where the monthlymaintenance is greater than 500.**

****

**Note:**

Table names are case sensitive.

**facilities**table is already created in the backend.

**Input format**

The required input table is created in the back end.

**Ans:**

**update facilities**

**set membercost=1000**

**where monthlymaintenance>500;**

1. **From the table schema given below, write a query to create a table orders.**

(Follow the same table names/column names as given along with the case)

The customer table is already created with primary key as **customerNumber**

Note:

In **orders** table, **orderNumber**is the Primary key and **customerNumber** is the Foreign Key referencing customerNumber of customers table.

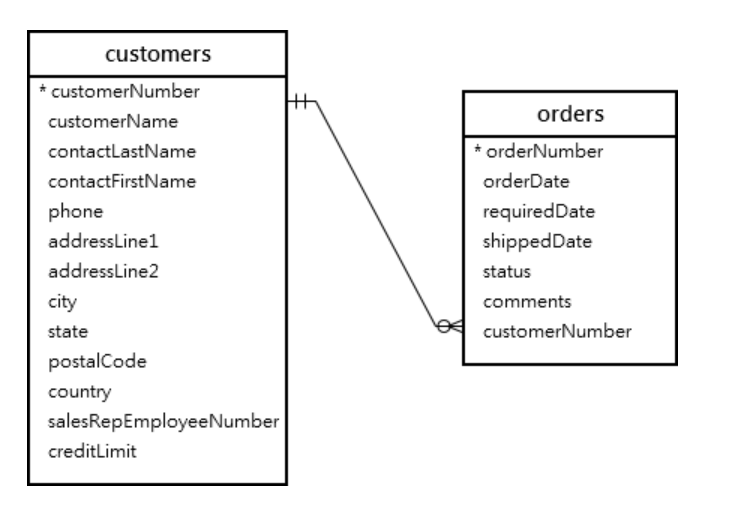
While creating orders table,

* orderNumber is the Primary Key
* define the foreign key relation between 2 tables.
* define all the columns with NOT NULL condition
* Follow the below data type:

orderNumber (int), orderDate(date), requiredDate(date), shippedDate(date), status(varchar(15)), comments(text),customerNumber(int)

**Input format**

The customers table is created in the backend.



**Ans:**

**create table orders(**

**orderNumber int not null primary key,**

**orderDate date not null,**

**requiredDate date not null,**

**shippedDate date not null,**

**status varchar(15) not null,**

**comments text not null,**

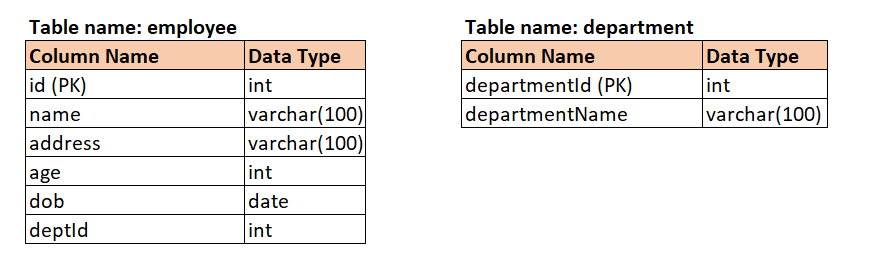
**customerNumber int not null,**

**foreign key (customerNumber) references customers(customerNumber)**

**);**

1. **Adding Foreign key Constraint in the existing table**

We have created an employee and department tables with the below definition without adding foreign key relation

****

We want to link departmentId (department table) and deptId column (employee table) to maintain data integrity by establishing foreign key relation.

Write a alter query to employee table so that deptId will be foreign key referencing departmentId of department table.with reference name as depIdFk.

Input format

The employee and department table are created without referencing foreign key relation.

**Ans:**

**alter table employee**

**add constraint depIdFk**

**foreign key(deptId) references department(departmentId);**

1. From the Schema given below, write a query to produce a count of the number of recommendations each member has made. Sort the output by ascending order of memid (recommendedby). If a particular member did not recommend anyone, ignore that memid. Output recommendedby and the corresponding count.

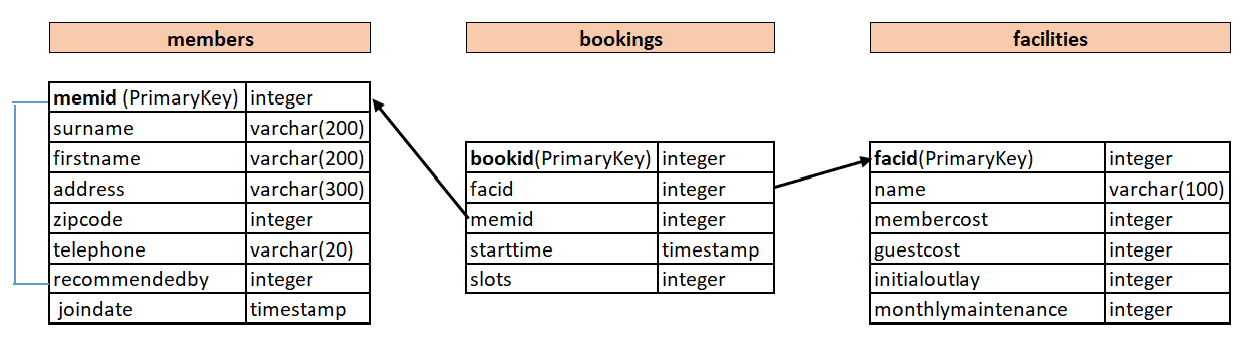
Note:

1. Refer table **members** from the given Schema.

2. 'recommendedby' field denotes 'memid' who recommended that particular member in the members table.

3. Table names are case sensitive.

4. Refer Output Format section for the output header names.



**Input format**

The required input table is populated in the back end.

**Output format**

The output should have the below header for the query to be considered.

**recommendedby, count**

**Ans:**

**select recommendedby,count(recommendedby) as count from members**

**where recommendedby is not null**

**group by recommendedby**

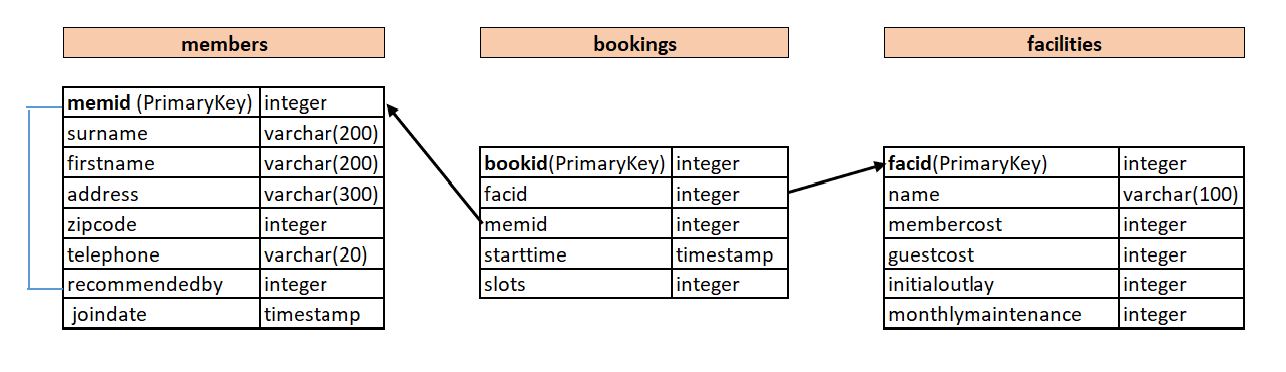
**order by recommendedby;**

1. **From the Schema given below, write a query to produce a list of the total number of slots booked per facility per month for July and August 2012. (month should be fetched as 7 and 8 corresponding to July and August)**

The output should have the details of facility id (facid), month, total slots booked sorted by the facid and month in ascending order.

Note:

1. Refer table **bookings** from the given Schema.
2. Refer the field 'starttime' to filter month
3. Table names are case sensitive.
4. Refer Output Format section for the output header names.

****  
**Input format**

The required input table is populated in the back end.

**Output format**

The output should have the below header for the query to be considered.

**facid, month, Total Slots**

**Sample Output Rows:**

*facid | month | Total Slots*

0 | 7 | 78

0 | 8 | 8

1 | 7 | 4

1 | 8 | 7

**Ans:**

**select facid,extract(month from starttime) as month,sum(slots) as "Total Slots"**

**from bookings**

**where starttime>='2012-07-01' and starttime <='2012-08-31'**

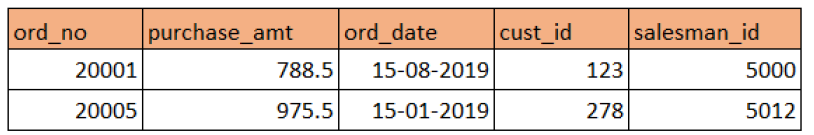
**group by facid,month**

**order by facid,month;**

1. **Given a table with order details. Please find the table details and sample data below.**

TABLE NAME: AGG\_ORDERS

FIELD NAMES: ord\_no, purchase\_amt, ord\_date, cust\_id, salesman\_id



**Note:** This is only a sample data.

Write a SQL statement to find the highest purchase amount with their Customer ID and order date, for only those customers who have the highest purchase amount in a day equal to any of these values 1000, 2200, 3700, 4000. Sort the records in ascending order of Customer ID.

Note: If a customer has done 2 purchases on a same day with purchase amount falls in our list (1000, 2200, 3700, 4000), then output should display both the records.Group by ord\_no,cust\_id,ord\_date in same order.

**Input format**

The required input details will be populated in the backend.

Table Name: AGG\_ORDERS

**Output format**

Output displays Customer ID, Order date along with their highest purchase amount from the given list.

Follow the output header as

**Customer\_ID, Date, Amount**

**Ans:**

**select cust\_id as Customer\_ID ,ord\_date as Date,max(purchase\_amt) as Amount**

**from AGG\_ORDERS**

**where purchase\_amt in (1000,2200,3700,4000)**

**group by ord\_no,cust\_id,ord\_date**

**order by Customer\_ID;**

1. **You are given two tables of XYZ company, EMPLOYEE\_XYZ and INCENTIVES\_XYZ which has details about employees of a company and their incentives. Write a SQL Query that fetches the employee Id as ID, first name of the employees as First\_name and sum of the incentives as Incentive for each IDs from the Incentives table.**

﻿Display in ascending order of employee ID.

Table names are case sensitive.

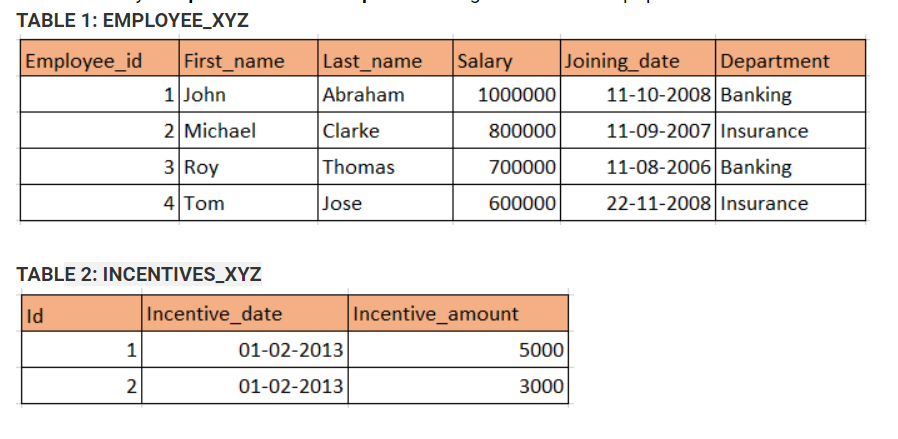
 TABLE 1: EMPLOYEE\_XYZ

Column Names: Employee\_id, First\_name,Last\_name,Salary,Joining\_date,Department

 TABLE 2: INCENTIVES\_XYZ

Column Names: Id, Incentive\_date,Incentive\_amount

 Below are only sample tables with sample data. Original data will be populated in the back end.

****

**Input format**

The required table will be populated in the back end.

**Output format**

The output consists of ID, First\_name and Incentive.

Follow output header as below. (case sensitive)

**ID, First\_name, Incentive**

**Ans:**

**select Employee\_id as ID,First\_name as First\_name,sum(Incentive\_amount) as Incentive from EMPLOYEE\_XYZ e**

**join INCENTIVES\_XYZ i**

**on e.Employee\_id=i.id**

**group by i.id**

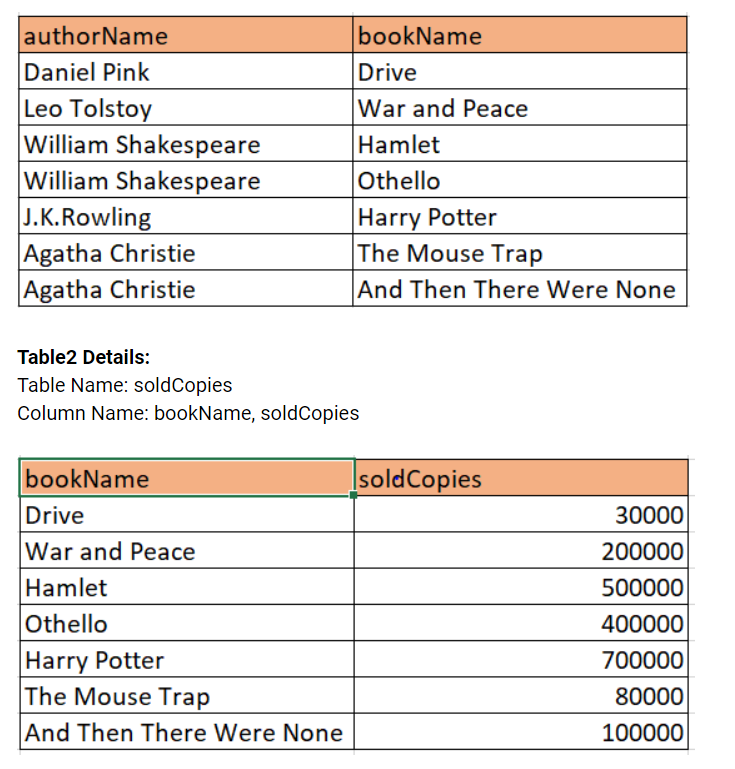
**order by Employee\_Id;**

1. **Dave owns a book shop. He is planning to stock all the best selling books in his shop. He collected all the information on author, book names and the number of copies sold in the below format.**

Table1 Details: Table names are case sensitive

Table Name: authorBooks

Column Name: authorName, bookName

****

Create an SQL query that shows the top 3 authors who sold the most books in total along with total number of copies sold.

 Sort the output in descending order of sold copies.

The headers in the output must be named as follows

Author\_Namesold\_sum

**Input format**

The input tables are populated in the back end.

**Output format**

The output consists of Author Name and sum of the books sold arranged in descending order based on sold copies.

The headers in the output must be named as follows (case sensitive)

**Author\_Name, sold\_sum**

**Ans:**

**select authorName as Author\_Name,sum(soldCopies) as sold\_sum from authorBooks a**

**join soldCopies s**

**on a.bookName=s.bookName**

**group by a.authorName**

**order by sold\_sum desc**

**limit 3;**

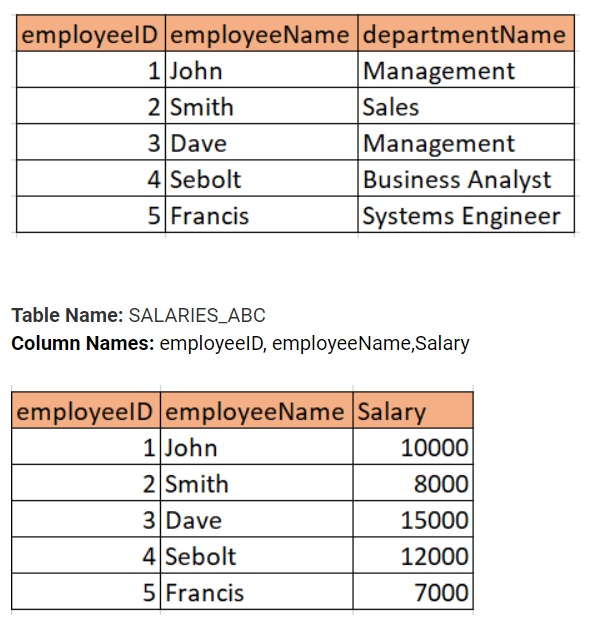
1. **Here is the sample employee and salary information of ABC Infotech Limited company. The company has total of 30 employees.**

Below are only the sample tables. The original table will be populated in the backend

Table names are case sensitive

Table Name: EMPLOYEES\_ABC

Column Names: employeeID, employeeName,departmentName

****

Write a query to display every department name where the average salary per employee is lower than 7000.

The headers in the output must be named as follows

**Department\_Name, Avg\_salaries**

**Input format**

The required input table will be populated in the backend.

**Output format**

Output displays Department Names and average salaries.

The headers in the output must be named as follows (case sensitive)

**Department\_Name, Avg\_salaries**

**Ans:**

**select departmentName as Department\_Name,avg(Salary) as Avg\_salaries**

**from EMPLOYEES\_ABC e**

**join SALARIES\_ABC s**

**on e.employeeID=s.employeeID**

**group by departmentName**

**having Avg\_salaries<7000;**

1. **Suppose that a website contains two tables, the Customers table, and the Orders table. Write a SQL query to find all customers who never order anything.**

Sort the result by ascending order of Customer Names

Table Details

Table1:

Tablename: **CUSTOMERS**

Columnname: id, cust\_name

+----+-------+

| id | cust\_name|

+----+-------+

| 1 | Prasanth |

| 2 | Harish |

| 3 | Sam  |

| 4 | Nadhini |

+----+-------+

Table2:

Table name: **ORDERS**

Column name: id, customer\_id

+----+------------+

| id| customer\_id|

+----+------------+

| 1 | 3     |

| 2 | 1     |

+----+------------+

Sample output

+-----------+

| Customers |

+-----------+

| Harish |

| Nandhini |

+-----------+

**Note:**

Table names are case-sensitive.

Use the same output header as given in the 'Output Format' section.

The above data are only sample data

**Input format**

The required tables will be populated in the back end.

**Output format**

The output will have the below header

**Ans:**

**select cust\_name as Customers from CUSTOMERS**

**where id not in (select customer\_id from ORDERS)**

**order by cust\_name asc;**

1. **Write a query to display the name (first name and last name) for those employees who get more salary than the employee whose ID is 105.**

Table details are as follows:

Table Name: EMPLOYEES\_SUBQ (Table names are case sensitive)

Column Names (along with data types)

employee\_id - INT *Primary key*,

first\_name - VARCHAR(30), last\_name - VARCHAR(30),

email - VARCHAR(30), phone\_number - VARCHAR(20),

hire\_date - DATE,

job\_id - VARCHAR(10),

salary - DECIMAL(18,2),

manager\_id - INT,

department\_id - INT

**Input format**

The required input table is populated in the back end.

**Output format**

The output header should be as follows:

**first\_name, last\_name**

**Ans:**

**select first\_name,last\_name from EMPLOYEES\_SUBQ where salary >**

**(select salary from EMPLOYEES\_SUBQ WHERE employee\_id=105);**

1. **Write a query to display the name (first name and last name), salary, department id, job id for those employees who work in the same designation(job id) as the employee works whose id is 110.**

Table details are as follows:

Table Name: EMPLOYEES\_SUBQ (Table names are case sensitive)

Column Names (along with data types)

employee\_id - INT *Primary key*,

first\_name - VARCHAR(30), last\_name - VARCHAR(30),

email - VARCHAR(30), phone\_number - VARCHAR(20),

hire\_date - DATE,

job\_id - VARCHAR(10),

salary - DECIMAL(18,2),

manager\_id - INT,

department\_id– INT

**Input format**

The required input table is populated in the back end.

**Output format**

The output header should be as follows:

**first\_name, last\_name, salary, department\_id, job\_id**

**Ans:**

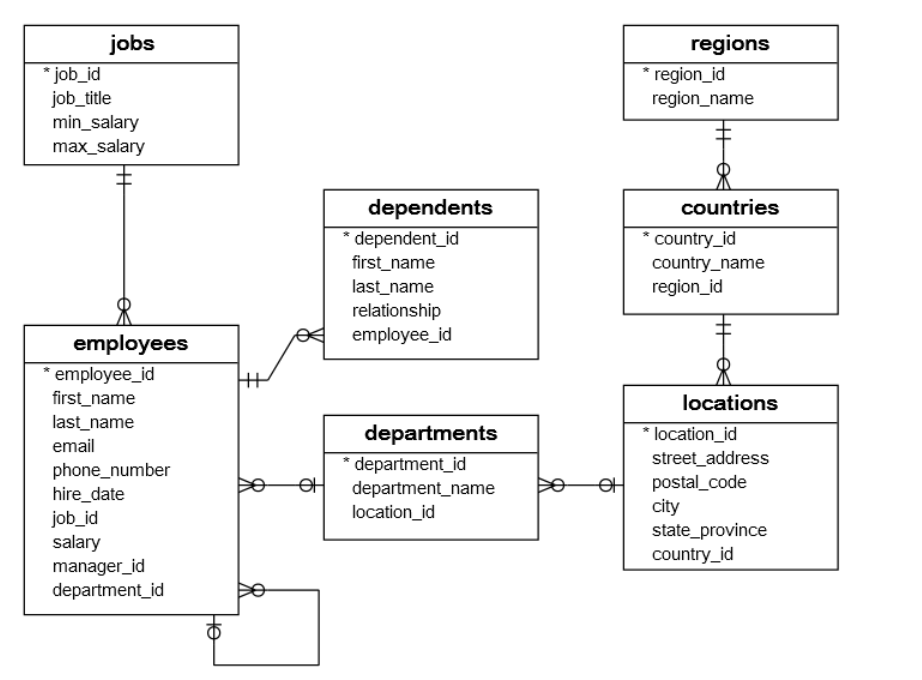
**select first\_name,last\_name,salary,department\_id,job\_id from EMPLOYEES\_SUBQ**

**where job\_id=(select job\_id from EMPLOYEES\_SUBQ where employee\_id=110);**

1. **From the below ER diagram of HR database, Write a query to find all the employees (employee\_id, first\_name, last\_name, salary, department\_id) whose salary is higher than the average salary of the employees in their departments. Use Correlated Subquery concept.**

 Refer the tables employees from the below diagram for the column names (Table names are case sensitive)

Sort the output by ascending order of department\_id, first\_name, last\_name.



**Input format**

The required input tables are populated in the back end.

**Output format**

The output header should be as follows:

**employee\_id, first\_name, last\_name, salary, department\_id**

Sort the output by ascending order of department\_id, first\_name, last\_name.

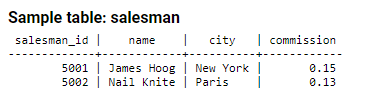
**Ans:**

**select employee\_id,first\_name,last\_name,salary,department\_id from employees e**

**where salary>(select avg(salary)from employees where department\_id=e.department\_id)**

**order by department\_id,first\_name,last\_name;**

1. **From the following table, create a view 'newyorkstaff' for those salespersons belonging to the city 'New York'.**



Note: Table name and column names are case sensitive.

**Output format**

The output will display the View of salespersons belonging to the city 'New York'.

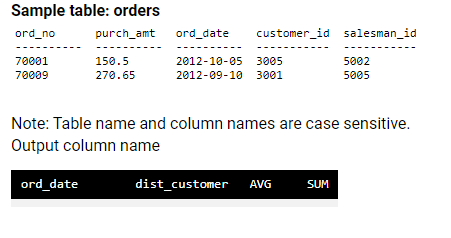
**Ans:**

**create view newyorkstaff as**

**select \* from salesman**

**where city='New York';**

1. **From the following table, create a view 'totalforday' to count the number of unique customers, compute the average and total purchase amount of customer orders by each date. Round off the average and sum values to 2 decimal places using round() and Give an Alice name to Average as AVG and for the sum as SUM.**



**Output format**

The output will display order date, Count of distinct customer id followed by Average, and Sum value.

**Ans:**

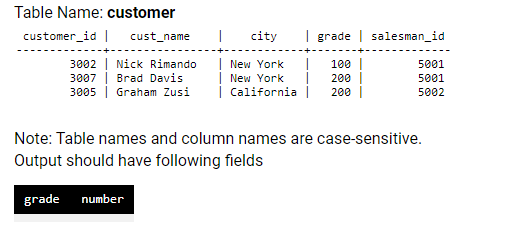
**create view totalforday as**

**select ord\_date,count(distinct customer\_id)as dist\_customer,round(avg(purch\_amt),2) as AVG,**

**round(sum(purch\_amt),2) as SUM from orders**

**group by ord\_date;**

1. **From the following table, create a view 'gradecount' to count the number of customers in each grade.**

****

**Output format**

The output will display the number of customers in each grade using views.

**Ans:**

**create view gradecount as**

**select grade,count(customer\_id) as number**

**from customer**

**group by grade;**

1. **Given the Employees table, Create a Cursor to find the list of all employees in a department with department Id passes as an argument.**

 As Cursor can be implemented inside Stored Procedure, Create a procedure

Procedure Name: EmployeeList(IN dept\_id INT, INOUT employee\_list VARCHAR(2000))

Department ID should be passed as an argument. employee\_list will have all the employee names corresponding to the particular department.

**Table Name: Employees**

EMPLOYEE\_ID INT PRIMARY KEY,

EMP\_NAME VARCHAR(100),

EMAIL VARCHAR(30),

PHONE\_NUMBER VARCHAR(20),

HIRE\_DATE DATE,

JOB\_ID VARCHAR(10),

EMP\_SALARY INTEGER,

MANAGER\_ID INT,

DEPARTMENT\_ID INT

Each Employee name should end with ;

**Sample Output:**

@employeelist

Dave Sebolt;BrianCaulfield;Robert Brown;

Ans:

**DELIMITER $$**

**CREATE PROCEDURE EmployeeList(**

**In dept\_id INT,INOUT employee\_list VARCHAR(2000))**

**BEGIN**

**DECLARE v\_finished INTEGER DEFAULT 0;**

**DECLARE v\_ename VARCHAR(100) DEFAULT "";**

**DECLARE C1 CURSOR FOR**

**SELECT EMP\_NAME FROM Employees WHERE DEPARTMENT\_ID=dept\_id;**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET v\_finished = 1;**

**OPEN C1;**

**get\_emp: LOOP**

**FETCH C1 INTO v\_ename;**

**IF v\_finished = 1 THEN**

**LEAVE get\_emp;**

**END IF;**

**SET employee\_list = CONCAT(employee\_list,v\_ename,";");**

**END LOOP get\_emp;**

**CLOSE C1;**

**END$$**

**DELIMITER ;**

**Footer:**

**SET @employeelist = '';**

**CALL EmployeeList(97, @employeelist);**

**SELECT @employeelist;**

1. **Given Product table, Implement a Cursor inside Stored Procedure to count how many products(product\_name) of each product type exists with product type passed as a input argument to the procedure.**

 As Cursor can be implemented inside Stored Procedure, Create a procedure

Procedure Name:

ProductCount(IN in\_prod\_type VARCHAR(30))

Product type should be passed as an argument.

**Table Name: Product**

product\_id int

product\_typevarchar(30)

product\_namevarchar(30)

Count will be displayed with header name ProductCount

**Sample Output:**

ProductCount

10

**Note:**

Table names are case sensitive

Use proper delimiter to define procedure. At the end of procedure, reset the delimiter to ;

Implement the cursor concept inside the stored procedure. Calling the stored procedure will be taken care in the back end.

**Input format**

The required input table is created and populated in the backend.

**Ans:**

**DELIMITER $$**

**CREATE PROCEDURE ProductCount(**

**IN in\_prod\_type varchar(30))**

**BEGIN**

**DECLARE v\_finished integer default 0;**

**DECLARE count\_ int default 0;**

**DECLARE pro\_name varchar(100) default '';**

**declare C1 CURSOR for select product\_type**

**from Product**

**where product\_type=in\_prod\_type;**

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET v\_finished =1;**

**OPEN C1;**

**Me :LOOP**

**FETCH C1 into pro\_name;**

**if v\_finished= 1 then leave Me;**

**end if;**

**SET count\_=count\_+1;**

**END LOOP;**

**select count\_ as ProductCount;**

**CLOSE C1;**

**END $$**

**DELIMITER ;**

**Footer:**

**CALL ProductCount('Hygiene');**

1. **Create a trigger named product\_availability that inserts mapping records into the products\_to\_stores table.**

**This trigger is used to enforce business logic; in particular, it helps define the product availability for the different stores.**

**When an item is being inserted into the products table, the trigger will check the availability field.**

**If availability column is marked with the LOCAL value, the product will be made available in one store only. (We need to insert a new row in products\_to\_stores table with the new product id and store\_id as 1.)**

**Example:**

INSERT INTO products\_to\_stores (product\_id, store\_id) VALUES (*new product id*, 1);

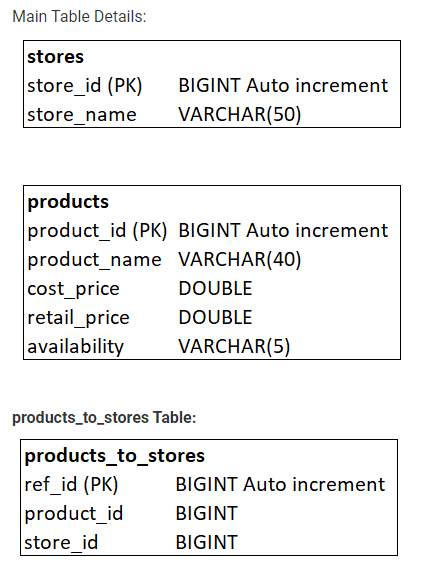
Any other value will instruct the trigger to make the product available to the two stores that we created earlier.

(We need to insert 2 new rows in products\_to\_storestable with the new product id and store\_id as 1 as one row and new product id and store\_id 2 as second row )

**Example:**

INSERT INTO products\_to\_stores (product\_id, store\_id) VALUES (new product id, 1';

INSERT INTO products\_to\_stores (product\_id, store\_id) VALUES (new product id, 2);



**Note:**

1. **Use proper delimiters before defining triggers.**
2. **In Solution, write a trigger alone with the given specifications. The insert query and selecting products\_to\_stores table will be taken care in the backend.**

**Input format**

The required input table is populated in the backend.

**Output format**

The output will have the below header:

**ref\_id, product\_id, store\_id**

**Ans:**

**DELIMITER $$**

**create trigger product\_availability**

**after insert on products**

**for each row**

**begin**

**if new.availability='LOCAL' THEN**

**insert into products\_to\_stores(product\_id,store\_id)**

**values(new.product\_id,1);**

**else**

**insert into products\_to\_stores(product\_id,store\_id)**

**values(new.product\_id,1);**

**insert into products\_to\_stores(product\_id,store\_id)**

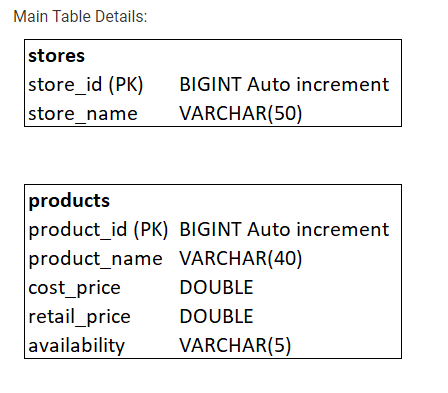
**values(new.product\_id,2);**

**end if;**

**end $$**

**DELIMITER ;**

1. **Create a trigger named product\_archiver that inserts a new row in archived\_products table after we delete a particular product in products table.**



**Note:**

* **Use proper delimiters before defining triggers.**
* **In Solution, write a trigger alone with the given specifications. The delete query which will trigger the insert and selecting rows from archived\_products table will be taken care in the backend.**

**Input format**

The required input tables are created and populated in the backend.

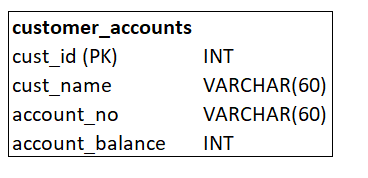
**Output format**

The output will have the below header:

**product\_id, product\_name, cost\_price, retail\_price, availability**

1. **Create a trigger called 'account\_balance\_validator' which should be triggered before update operation taking place on customer\_accounts table.If customer account balance is updated the amount which is less than 0, then the account balance should be updated as 0.**

Table Name: customer\_accounts



**Note:**

* Use proper delimiters before defining triggers.
* In Solution, write a trigger alone with the given specifications. Updating sample row and selecting the rows from customer\_accounts table will be taken care in the backend.
* Table names are case sensitive

**Input format**

The required input table is populated in the backend.

**Output format**

The output will have the below header:

**cust\_id, cust\_name, account\_no, account\_balance**

**Ans:**

**delimiter $$**

**create trigger account\_balance\_validator before**

**update on customer\_accounts**

**for each row begin**

**if new.account\_balance<0**

**then**

**set new.account\_balance=0;**

**end if;**

**end $$**

**delimiter ;**

**Footer:**

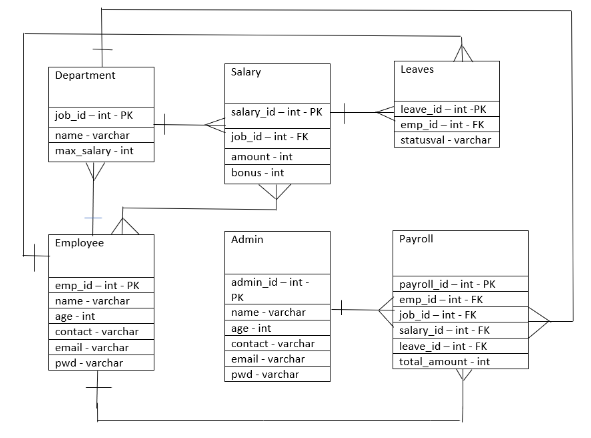
**UPDATE customer\_accounts**

**SET**

**account\_balance = -56**

**WHERE cust\_id = 3;**

1. **Given the schema of a Payroll Processing System,**



**Write a query to count the number of Employee with age<35.**

**Note:**

The required input tables are populated in the back end.

The table names are case-sensitive.

Refer to the schema and the output section for further clarifications.

**Input format**

No console input.

**Output format**

The output prints the count of the Employee with age<35.

Output Header: AgeCount

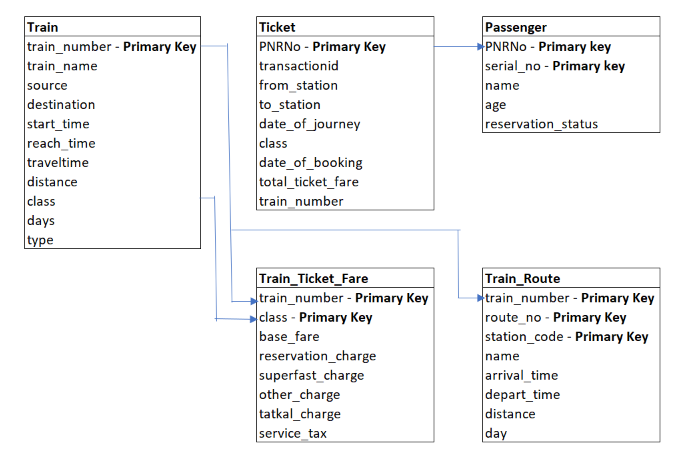
**Ans:**

**select count(emp\_id) as AgeCount from Employee where age<35;**

1. Given the schema of Railway Reservation System, write a query to print the station codes replacing 'M' with 'K' along with the station name.Sort the output by ascending order of station code.

**Note:**

* Refer the table **Train\_Route**from the given schema.
* Table names are case sensitive.
* Refer Output Format section for the output header names.



**Input format**

The required input tables are populated in the back end.

**Output format**

The output should have the below header for the query to be considered.

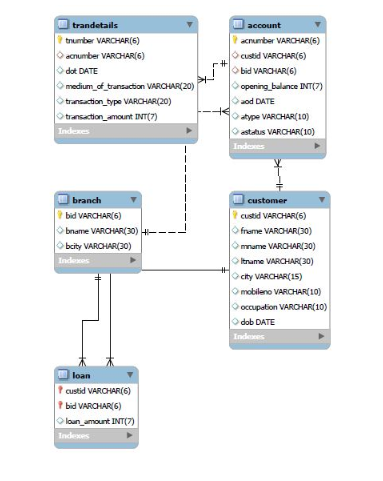
**station\_code, name**

**Ans:**

**select station\_code,name from Train\_Route**

**where station\_code like 'M%'**

**order by station\_code asc;**

1. **Given the schema of a Bank Management system**

**Write a query to display the custid, fname and customer's dob. Display in sorted order of date of birth year and within that sort by firstname**.

**Note:**

The required input tables are populated in the back end.

The table names are case-sensitive.

Refer to the schema and the output section for further clarifications.

**Input format**

No console input.

**Output format**

The output prints the custid, fname and customer's dob in sorted order of date of birth year and within that in the sorted order of firstname.

**Ans:**

**select custid,fname,dob from customer**

**order by dob,fname;**