# Structured Query Language

- DDL (Data Definition Language)
- DML(Data Manipulation Language)
- DCL( Data Control Language)
- TCL(Transaction Control language)

## DDL (Data Definition Language)

- Create

```
create table employees(
Emp_id int(10) not null,
first_name varchar(20) not null,
Last_name varchar(20),
Salary int(10),
primary key(Emp_id)
);
```

Press ctrl + Shift + enter to execute command

To see result

```
select * from employees;
```

or

```
describe employees;
```

## copy table data from another table

Here we are copying from table dept where dept is tech and copy all data , we can choose selected columns too in place of *

⇨ `select * into employees from depts where dept= 'Tech';`

- Alter

⇨ `alter table employees add column contact int(10) after Last_name;`

- Modify

⇨ `alter table employees modify column contact int(15);`

⇨ `alter table employees modify column contact int(15) after Salary;`

- Rename

⇨ `alter table employees rename column contact to job_code;`

- Truncate

⇨ `Truncate table employees;`

- Drop

⇨ `Drop table employees;`

# DML(Data Manipulation Language)

- Insert

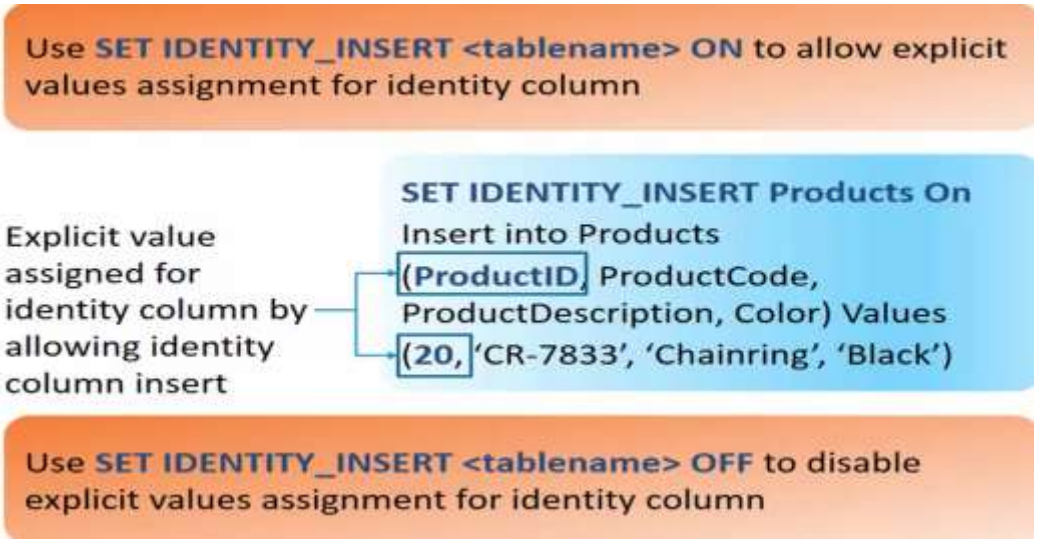
```
⇨ Insert into employees
  (Emp_id,First_name,Last_name,Salary
  values (101,'Steven','King',10000)

⇨ insert into employees
  (Emp_id,First_name,Last_name,Salary)
  values (102,'Edwin','Thomas',15000);
⇨ insert into employees
  (Emp_id,First_name,Last_name,Salary)
  values (103,'Harry','',17000);
```

Here we can use auto increment in emp_id that if I have a pattern `IDENTITY [( seed, increment)]`

```
⇨ create table employees(
  Emp_id int(10) identity(1,1) not
  null, first_name varchar(20) not
  null, Last_name varchar(20),
  Salary int(10), primary key(Emp_id));
```

Here to turn this on off we have to use another command example if we want to insert emp_id = 100 after 5 employees we have to turn on identity_insert `IDENTITY_INSERT <tablename> ON` after that we have to turn it off so that we can start auto increment again `IDENTITY_INSERT <tablename> OFF` else it will ask to insert emp_id or it will show null




- Update

```
update employees set
last_name='Potter' where Emp_id=103;
update employees set
First_name='Paul' where Emp_id=101;
update employees set Last_name=''
where Emp_id=102;
```

- Delete

```
delete from employees where Emp_id in
(101,103);
```

# DCL( Data Control Language)

- Grant

grant <Privilege list> on <Relation Name> to <user>

- Revoke

  grant <Privilege list> on <Relation Name> to <user>

## TCL(Transaction Control language)

- Commit
- Rollback
- Savepoint

## SQL Constraints

- Unique Key

```
Create Table Employee
    (
        [EmpID] [Int]
        , [EmpName] [varchar](100) NOT NULL
        , [Gender] [varchar](8) NULL
        , [StartDate] [date] NULL
        , [EndDate] [date] NULL
        , CONSTRAINT uc_employee UNIQUE (EmpID, EmpName)
    )
```

- Foreign key

```
Create Table Salary
    (
        [EmpID] [Int]
        , [Salary] [Int]
        , CONSTRAINT fk_employee FOREIGN KEY(EmpID) References
        Employee (EmpID)
        )
```

# SQL Operators- Filters

## Where Clause

⇨ `select * from employees where emp_id=101;`

# SQL Operators- Logical

- AND

⇨ `select * from employees where First_name='Steven' and Salary=15000;`

result: display those where both the condition are be true

- OR

⇨ `select * from employees where First_name='Steven' or Salary=15000;`

result: display those where any of the give condition is true

- NOT

⇨
```
select * from employees where
First_name='Steven' and
Salary!=10000;
```
result: display those where any of the give condtition is true but there second is Salary is not equal to 10000

# SQL Operators- Comparision

- = Equal to

⇨
```
select * from employees where
salary=25000;
```

- > Greater than

⇨
```
select * from employees where
salary>25000;
```

- >= Greater than or Equal to

⇨
```
select * from employees where
salary>=25000;
```

- < Lesser than

⇨ `select * from employees where salary<25000;`

- <= Less than or Equal to

⇨ `select * from employees where salary<=25000;`

- <> or != Not Equal to

⇨ `select * from employees where salary!=25000;`

# SQL Operators- Special

- Between Checks attribute value within range

⇨ `select * from employees where salary between 10000 and 25000;`

- Like Checks an attribute values matches a given string pattern

⇨ `select * from employees where First_name like 'Steven';`

⇨ `select * from employees where First_name like 'S%';`

all first name start with S

- Is Null Check an attribute value is null

```
select * from employees where
Last_name='';
```

```
select * from employees where
Last_name is null;
```

not worked

- In Checks an attribute value matches any value within a value list

```
select * from employees where salary
in (10000,12000,20000);
```

- Distinct Limit values to unique values

```
select distinct(First_name) from
employees;
```

# SQL Operators- Aggregations

- Avg() Return the average values from the specified columns

```
select avg(Salary)from employees;
```

- Count() Return the number of table rows

```
select count(*) from employees;
```

- Max() Return the largest value among the records

```
select max(Salary) from employees;
```

- Min() Return the smallest value among the records

```
⇨ select min(Salary) from employees;
```

- Sum() Return the sum of specified column values

```
⇨ select sum(Salary) from employees;
```

# Top N

Draw top N values from the table

ex: For finding top 3 highest paid employees

- Top N with numbers only

```
⇨ Select top 3 salary from employees;
```

- Top N with percent option

```
⇨ select top 3 percent salary from
  employees;
```

- Top N with tie option

```
⇨ select top 3 with tie salary from
  employees order by First_name;
```

# SQL GROUP BY Clause

arrange identical data in group

ex: for that we need to add new column Dept in table

```
⇨ alter table employees add column Dept
  Varchar(20);
```

Now adding data in Dept Column

```
⇨ update employees set Dept='Sales'
  where Emp_id=101;
⇨ update employees set Dept='Tech'
  where Emp_id=102;
⇨ update employees set Dept='Marketing'
  where Emp_id=103;
⇨ update employees set Dept='Content'
  where Emp_id=104;
⇨ update employees set Dept='Sales'
  where Emp_id=105;
⇨ update employees set Dept='Tech'
  where Emp_id=106;

=> update employees set
   Dept='Marketing' where Emp_id=107;
=> update employees set Dept='Content'
   where Emp_id=108;
```

Now Grouping maximum salary Dept wise with enployee name

```
⇨ select First_name, max(Salary),Dept
  from employees group by Dept
```

# SQL HAVING Clause

-Use with aggregate functions due to its non-performance in the WHERE clause.

-Must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

```
⇨ select First_name, max(Salary),Dept
  from employees group by Dept having
  count(Dept>=2;

⇨ select First_name, avg(salary), Dept
  from employees group by Dept having
  count(dept)>=2;
```

# SQL ORDER BY Clause

-Use to sort output in SELECT statement.
-Defaut is to sort in ASC (Asecnding)
-Can sort in Reverse (Decreasing) order with "DESC" after the column name

⇨ `select * from employees order by salary desc;`

# SQL OVER Clause

-Everything will be same but a new column will created that will show avg, sum etc by in group

- Basic

⇨ `select *, sum(salary) over (partition by dept) from employees;`


| ProductID | CityID | MonthID | SalesQuantity | SalesValue |
|---|---|---|---|---|
| 1 | 1 | 201512 | 100 | 8000.00 |
| 1 | 2 | 201605 | 33 | 5640.00 |
| 1 | 6 | 201511 | 600 | 7590.00 |
| 1 | 7 | 201603 | 55 | 3433.00 |
| 1 | 9 | 201601 | 659 | 9432.00 |
| 2 | 5 | 201602 | 110 | 8999.00 |
| 2 | 12 | 201602 | 785 | 8324.00 |

| ProductID | CityID | MonthID | SalesQuantity | SalesValue | TotalQuantity |
|---|---|---|---|---|---|
| 1 | 1 | 201512 | 100 | 8000.00 | 1447 |
| 1 | 2 | 201605 | 33 | 5640.00 | 1447 |
| 1 | 6 | 201511 | 600 | 7590.00 | 1447 |
| 1 | 7 | 201603 | 55 | 3433.00 | 1447 |
| 1 | 9 | 201601 | 659 | 9432.00 | 1447 |
| 2 | 5 | 201602 | 110 | 8999.00 | 895 |
| 2 | 12 | 201602 | 785 | 8324.00 | 895 |

- By using order by to arrange

⇨ `select *, sum(salary) over (partition by dept order by dept desc) from employees;`

- Set custom name to new column

```
select *, sum(salary) over (partition
by dept order by dept desc) as
max_salary from employees;
```

- Creating new column on bases of formula

```
select *, round
((salary*100/sum(salary)) over
(partition by dept order by dept
desc),2) as percent_salary_per_dept
from employees;
```


# SQL Rank

Use to rank the table data ( rank(), dense_rank(), Row_number ()

```
Select * , rank() over (partition by
dept order by salary ) from
employees;
```

# SQL Convert

Use to convert one date format to another

```
select convert(date, getdate(),109)
```

| Without Century | With Century | Input\Output |
|---|---|---|
| - | 0 or 100 | mon dd yyyy hh:miAM (or PM) |
| 1 | 101 | 1 = mm/dd/yy<br>101 = mm/dd/yyyy |
| 2 | 102 | 2 = yy.mm.dd<br>102 = yyyy.mm.dd |
| 3 | 103 | 3 = dd/mm/yy<br>103 = dd/mm/yyyy |
| 4 | 104 | 4 = dd.mm.yy<br>104 = dd.mm.yyyy |
| 5 | 105 | 5 = dd-mm-yy<br>105 = dd-mm-yyyy |
| 6 | 106 | 6 = dd mon yy<br>106 = dd mon yyyy |
| 7 | 107 | 7 = Mon dd, yy<br>107 = Mon dd, yyyy |
| 8 | 108 | hh:mm:ss |
| - | 9 or 109 | mon dd yyyy hh:mi:ss:mmmAM (or PM) |
| 10 | 110 | 10 = mm-dd-yy<br>110 = mm-dd-yyyy |
| 11 | 111 | 11 = yy/mm/dd<br>111 = yyyy/mm/dd |
| 12 | 112 | 12 = yymmdd<br>112 = yyyymmdd |
| - | 13 or 113 | dd mon yyyy hh:mi:ss:mmm (24h) |

| Without Century | With Century | Input\Output |
|---|---|---|
| 14 | 114 | hh:mi:ss:mmm (24h) |
| - | 20 or 120 | yyyy-mm-dd hh:mi:ss (24h) |
| - | 21 or 121 | yyyy-mm-dd hh:mi:ss.mmm (24h) |
| - | 126 | yyyy-mm-ddThh:mi:ss.mmm (no spaces) |
| - | 127 | yyyy-mm-ddThh:mi:ss.mmmZ (no spaces) |
| - | 130 | dd mon yyyy hh:mi:ss:mmmAM |
| - | 131 | dd/mm/yy hh:mi:ss:mmmAM |

# SQL Datepart

Extract day, month , quarter , year , week no. , no. of day of year etc

⇨ `Select datepart(yyyy, '14 Nov 2015')`

| Date Part | Abbreviation |
|---|---|
| year | yy, yyyy |
| quarter | qq, q |
| month | mm, m |
| dayofyear | dy, y |
| day | dd, d |
| week | wk, ww |
| weekday | dw, w |
| hour | hh |
| minute | mi, n |
| second | ss, s |
| millisecond | ms |
| microsecond | mcs |
| nanosecond | ns |

# SQL Dateadd

For add and subtract days , months , year etc

⇨ `Select datedd(dd,2, '14 Nov 2015')`

SELECT DATEADD(dd, 2, '14 Nov 2015')

Output: 16 Nov 2015

SELECT DATEADD(yy, -1, '14 Nov 2015', )

Output: 14 Nov 2014

SELECT DATEADD(mm, -1, '14 Nov 2015')

Output: 14 Oct 2015

# SQL Datediff

For finding difference between two dates

⇨ `Select datediff(mm, '14 Nov 2015', '14 Dec 2015')`

SELECT DATEDIFF(mm, '14 Nov 2015', '14 Dec 2015')
**Output: 1**

SELECT DATEDIFF(dd, '14 Nov 2015', '14 Dec 2015')
**Output: 30**

SELECT DATEDIFF(yy, '14 Nov 2014', '14 Nov 2015')
**Output: 1**

# SQL Choose and IFF

Both are use to give logic or for notify in new column

| Discount Percentage | Category |
|---|---|
| > 0.30 | Bumper Sale |
| = 0.30 | Average Sale |
| < 0.30 | Regular Sale |

```
SELECT Description, DiscountPct
IIF(DiscountPct > 0.30,'Bumper Sale',
IIF(DiscountPct = 0.30, 'Average Sale','Regular Sale') ) Category
From [AdventureWorks2014].[sales].[SpecialOffer]
```

And

| Shift Name | Category |
|---|---|
| Day | General Shift |
| Evening | Reporting Shift |
| Night | Critical Shift |

```
SELECT Name,
CHOOSE(ShiftID,'General Shift', 'Reporting Shift', 'Critical
Shift') Category FROM
[AdventureWorks2014].[HumanResources].[Shift]
```

# SQL UNION

-Used to combine the result-set of two or more SELECT statements removing duplicates.
-Each SELECT satement within the UNION must have the same number of columns.
-The selected columns must be of similar data types and must be in the same order in each SELECT statement.

Ex- Lets create 2 tables first

First Table

```
create table product1(Category_id
int(5),Product_name varchar(20));
```

Insert values in table in product1

```
insert into product1 values
(1,'Nokia');
insert into product1 values (2,'HP');
insert into product1 values
(3,'Samsung');
insert into product1 values
(6,'Nikon');
```

Second Table

```
create table product2(
  Category_id int(5),
  Product_name varchar(20)
  );
```

Insert values in table in product2

```
insert into product2 values
(1,'Samsung');
insert into product2 values (2,'LG');
insert into product2 values (3,'HP');
insert into product2 values
(5,'Dell');
insert into product2 values
(6,'Apple');
insert into product2 values (10,'X-
Box');
```

## SQL UNION

All Element from both table but common elements will occurs only once

```
select Product_name from Product1
union select Product_name from
Product2;
```

## SQL UNION ALL

All Elements from both tables and common elements will repeat too

```
select Product_name from Product1
union all select Product_name from
Product2;
```

# SQL JOINS

Creating another Table we will use employee table as out first table, Dept is Common in both tables

Department Table

```
create table Department(
dept_id int,
dept varchar(20),
dept_loc varchar(20),
);

Inpute into Department values
(1,'Content','Chicago');
Inpute into Department values
(2,'Support','New Jersey');
Inpute into Department values
(3,'Sales','Bostan');
Inpute into Department values
(4,'HR','Chicago');
Inpute into Department values
(5,'Operations','New York');
```

- Inner Join

  It show only Common element from Table 1 and Table 2

```
Select employees.First_name,
employees.Last_name, employees.Dept,
Department.dept_loc from employees
inner join Department On
employees.Dept=Department.dept;
```

OR

⇨
```
Select e.First_name, e.Last_name,
e.Dept, d.dept_loc
from employees e
inner join Department d
on e.Dept=d.dept;
```

- Left Join

  It return all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

⇨
```
Select e.First_name, e.Last_name,
e.Dept, d.dept_loc from employees e
left join Department d on
e.Dept=d.dept;
```

- Right Join

  It return all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

⇨
```
Select e.First_name, e.Last_name,
e.Dept, d.dept_loc from employees e
right join Department d on
e.Dept=d.dept;
```

- Full Outer Join

  It combines the result of both left and right outer joins. The joined tables will contain all records from both the tables and fill in NULLs for missing matches on either side.

⇨
```
Select e.First_name, e.Last_name,
e.Dept, d.dept_loc from employees e
```

```
left join Department d on
e.Dept=d.dept;
Union
Select e.First_name, e.Last_name,
e.Dept, d.dept_loc from employees e
right join Department d on
e.Dept=d.dept;
```

- Self Join
- Cartesian Join (Cross Join)

  It produces the result set with the number of rows in the first table multiplied by the number of rows in the second.

  ⇨
  ```
  Select * from employees cross join
  Department;
  ```

# SQL View

- View

  It are use to show selected data by some keyword , here how to create view

  Create view <viewname> as <sql select quary>

```
Create View
CanadaInternetSales2008
AS
Select INTSALES.* From FactInternetSales IntSales
INNER JOIN
DimSalesTerritory ST
ON
IntSales.SalesTerritoryKey = ST.SalesTerritoryKey
INNER JOIN
DimDate DT
ON
IntSales.OrderDateKey = DT.DateKey
Where ST.SalesTerritoryRegion= "Canada"
AND DT.CalendarYear = 2008
```

- Drop view

  Drop view <viewname>

- Update/alter view ( same as we create view)

  Alter view <viewname> as <sql select quary>

# Advance SQL

## Subquaries

⇨
```
Select productcode , productname,
productline, MSRP from products where
productcode in (select productcode from
orderlists where priceeach > 100);
```

## Stored Procedure

Record a code to reuse that again by just call the keyword

- Simple

⇨
```
delimiter &&
Create procedure agedplayer()
Begin
Select short_name , age, nationality from
football
Where age > 30;
End &&
Delimiter ;

Call agedplayer();
```

- Using IN parameter for arranging

```
delimiter //
Create procedure top_age(in var int)
Begin
Select short_name , age, nationality from
football
Order by age desc limit var;
End //
Delimiter ;

Call top_age(3);
```

- Using IN parameter for updating

```
delimiter //
Create procedure update_age(in temp_name
varchar, new_age int)
Begin
Update football set age=new_age where
short_name=temp_name;
End //
Delimiter ;

Call update_age('Neto',10);
```

- Using OUT parameter for count

```
delimiter //
Create procedure Spain_player(out s_player
int)
Begin
Select count(short_name) into s_player
from football where nationality = 'Spain';
End //
Delimiter ;

Call spain_player(@splr);
Select @splr as Number_of_Spain_Players;
```

## Triggers

It is special type of store procedure that run automatically when the event occurred in database server.

- Before insert Trigger

```
Create table student(Roll_no int, Name varchar(20), Age int, Marks float);

delimiter //
Create trigger marks_verify
Before insert on student
For each row
If new.marks<0 then set new.marks=50;
End if ; //

insert into student
values(501,'Ruth',11,76),
(502,'Max',10,-49.9),
(503,'James',11,-34),
(504,'Millian',10,90.6);

select * from student;

drop trigger marks_verify;
```

## Views in SQL

- In a Table

```
create view foot_ball
as
select short_name, age from football;

select * from foot_ball;
```

- From two tables

```
Use classicmodels

Create view product_description
As
Select productname, quantityinstock, MSRP,
textdescription from products as p
Inner join productlines as pl
On p.productline = pl.productline;

select * from product_description;

rename table product_description to
vehicle_description;
```

## Windows functions

- Create new column

```
select emp_id, first_name, Last_name,
Dept, sum(salary) over (partition by dept)
as total_salary from employees;
```

- Row Number

```
Select row_number() over (Order by salary)
as row_num, emp_id, first_name, salary
from employees;
```

- Count Duplicate Values

```
Create table demo(St_id int, St_Name
varchar(20));

Insert into demo values (101,'Shane'),
(102,'Bradley'), (103,'Herath'),
(103,'Herath'), (104,'Nathan'),
(105,'Kevin'), (105,'Kevin');
```

⇨ `Select st_id, St_name, row_number() over (partition by St_id, St_name order by st_id) as row_num from demo;`

- Rank Function

⇨ `Create table demo1(St_id int);`

⇨ `Insert into demo1 values (101), (102), (103), (103), (104), (105), (106), (107);`

⇨ `Select st_id, Rank() over (order by st_id) as Test_Rank from demo1;`

- First_Value Function

  - Sort over salary

⇨ `Select First_name, Last_name, salary, first_value(First_name) over (order by salary desc) as Highest_Salary from employees;`

  - Sort by Dept and Highest Salary

⇨ `Select First_name, Last_name, salary, dept, first_value(First_name) over (partition by dept order by salary desc) as Highest_Salary from employees;`

# Project 1

# Datasets Used: cricket_1.csv, cricket_2.csv
--cricket_1 is the table for cricket test match 1.
--cricket_2 is the table for cricket test match 2.

# Q1. Find all the players who were present in the test match 1 or test match 2.

```
SELECT * FROM cricket_1 UNION SELECT * FROM
cricket_2;
```

# Q2. Write a MySQl query to find the players from the test match 1 having popularity higher than the average popularity.

```
select player_name , Popularity from cricket_1
WHERE Popularity > (SELECT AVG(Popularity) FROM
cricket_1);
```

# Q3. Find player_id and player name that are common in the test match 1 and test match 2.

```
SELECT player_id, player_name FROM cricket_1
WHERE cricket_1.player_id IN (SELECT player_id
FROM cricket_2);
```

# Q4. Retrieve player_id, runs, and player_name from cricket_1 table and display list of the players where the runs are more than the average runs.

```
SELECT player_id , runs , player_name FROM
cricket_1 WHERE runs>(SELECT AVG(runs) FROM
cricket_1);
```

# Q5. Write a query to extract the player_id, runs and player_name from the table "cricket_1" where the runs are greater than 50.

```
SELECT player_id , runs , player_name FROM
cricket_1 WHERE runs > 50;
```

# Q6. Write a query to extract all the columns from cricket 1 where player_name starts with 'y' and ends with 'v'.

```
 SELECT * FROM cricket_1 WHERE player_name LIKE
'y%v';
```

# Q7. Write a query to extract all the columns from cricket_1 where player_name does not end with 't'.

```
SELECT * FROM cricket_1 WHERE player_name NOT
LIKE '%t';
```

---

# Dataset Used: new cricket.csv

---

# Qll. Extract the Player_Id and Player_name of the players where the charisma value is null.

```
SELECT Player_Id , Player_Name FROM new_cricket
WHERE Charisma IS NULL;
```

# Q12. Separate all Player_Id into single numeric ids (example PL1 = 1).

```
SELECT Player_Id, SUBSTR(Player_Id,3) FROM
new_cricket;
```

# Q13. Write a MySQL query to extract Player_Id , Player_Name and charisma where the charisma is greater than 25.

```
SELECT Player_Id , Player_Name , charisma FROM new_cricket WHERE charisma > 25;
```

# Project 2

**# Question 1**:

# 1) Create a Database bank

```
CREATE DATABASE bank;
use bank;
```

**# Question 2:**

# 2) Create a table with the name “Bank_details" with the following columns

-- Product with string data type
-- Quantity with numerical data type
-- Price with real number data type
-- Purchase cost with decimal data type
-- Estimate sale price with data type float

```
create table bank_details(
Product char(10),
Quantity int,
Price real,
Purchase_cost decimal(6,2),
Estimated_sale_price float
);
```

**# Question 3:**

# 3) Display all columns and their datatype and size in Bank Details

```
Describe Bank_details;
```

**# Question 4:**

# 4) Insert two records into  bank details

-- 1st record with values --
--Product: Paycard
--Quantity: 3
--Price: 330
--Purchase_cost:  8008
--Estimate cost price:  9009

-- 2st record with values --
--Product: PayPoints
--Quantity: 4
--Price: 200
--Purchase_cost:  8000
--Estimate cost price:  6800

```
Insert into bank_detail value ('Paycard',3,330,8008,9009);
Insert into bank_detail value ('PayPoints',4,200,8000,6800);
```

**# Question 5:**

# 5) Add a column: Geo Location to the existing bank details table with data type varchar and size 20

```
Alter table bank_details add column Geo_Location varchar(20);
```

**# Question 6:**

# 6) What is the value of  Geo Location for product:"Paycard".

```
Select Geo_location from bank_details where product = 'Paycard';
```

**# Question 7:**

# 7) How many characters does the product: ‘Paycard’ have in bank details table.

```
Select char_length(product) from bank_details where Product =
'paycard';
```

**# Question 8**:
# 8) Alter the Product field from CHAR to VARCHAR in bank details

```
Alter table bank_details modify product varchar(10);
```

**# Question 9:**
# 9) Reduce the size of the Product field from 1o to 6 and check if it is possible

```
Alter table bank_details modify product varchar(6);
```

-- Output
-- Error: #### - Data too long for column 'product'
-- Justification: Product field consists of Data values with size more than 6 Characters.

**# Question 10:**
# 10) Output of Product field as NEW_PRODUCT in Bank_Details

```
Select PRODUCT as NEW_PRODUCT from Bank_details;
```

**# Question 11:**
# 11) Display only one record from bank_details

```
Select * from bank_details limit 1;
```

**# Question 12:**
# 12) Display the first five characters of the Geo_Location field of bank details

```
Select substr(Geo_location,1,5) from bank_details;
```

# Project 3

**# Question 1:**

# 1) Create a table named as Bank Holidays with below fields

-- a) Holiday field which displays only date

-- b) Start time field which displays hours and minutes

-- c) End time field which also displays hours and minutes and time zone

```
create table bank_holidays(
Holiday date,
Start_Time datetime,
End_Time timestamp
);
```

**# Question 2:**

# 2) Step 1:Insert today's date details in all fields of Bank Holidays

-- Step 2: After step1, perform the below

-- Postpone Holiday to next day by updating the Holiday field

```
-- Step1:
    insert into bank_holidays value
    (current_date(),current_date(),current_date());
```

-- Step 2: To add 1 day interval

```
Update bank_holidays set holiday = date_add(holiday , interval 1 day);
```

To sub 5 day interval

```
Update bank_holidays set holiday = date_add(holiday , interval -5 day);
```

**# Question 3:**

# 3) Update the timestamp with current European time.

```
Update bank_holidays set End_Time= utc_timestamp();
```