Section – A: Theory Questions

1. **What are the different types of joins in SQL?**
   Inner join: returns only the rows that have matching values in both tables.
   Left outer join: returns all the rows from the left table and the matched rows from the right table.
   Right outer join: returns all the rows from the right table and the matched rows from the left table.
   Full outer join: returns all the rows from both tables and includes null values where there are no matches.
   Cross join: returns the cartesian product of the two tables.
   Self-join: a join where a table is joined with itself.

2. **What is the difference between inner, left, and right joins?**
   An "INNER JOIN" returns only the rows that have matching values in both tables.
   A "LEFT JOIN" returns all the rows from the left (first) table and the matching rows from the right (second) table. If there is no match, NULL values will be returned for the right table.
   A "RIGHT JOIN" returns all the rows from the right (second) table and the matching rows from the left (first) table. If there is no match, NULL values will be returned for the left table.

   Syntax:
   ```
   select * from table1 inner join table2 on table1.column = table2.column;
   select * from table1 left join table2 on table1.column = table2.column;
   select * from table1 right join table2 on table1.column = table2.column;
   ```
   Note: The major difference in the syntax is the "inner", "left", "right" keywords used to specify the type of join.The structure of the query remains largely the same, with the exception of the keyword used to specify the type of join.

3. **what are explicit and implicit inner joins**
   Explicit inner join
   ```
   select * from table1, table2 where table1.column = table2.column
   ```
   Implicit inner join
   ```
   select * from table1 inner table2 on table1.column = table2.column
   ```

4. **What is a self join and when would you use it?**
   A **self-join** is a join where a table is joined with itself. It is used to combine data from the same table when the table has a foreign key that references itself. Self-joins are useful for hierarchical data, where each row in the table has a parent row that is also in the same table. They are also useful for querying data that has a recursive relationship, such as a directory structure or a bill of materials.

5. **What is the difference between cross and self-joins?**
   cross join:
   select * from table1 cross join table2;
   select * from table1, table2;
   self join: in order to use self-join, we use the alias of the same table
   SELECT t1.emp_id, t2.emp_id AS manager_id FROM table1 AS t1 INNER JOIN table1 AS t2 ON t1.emp_id = t2.manager_id -- employee under his manager
   SELECT t1.emp_id, t2.emp_id AS manager_id FROM table1 AS t1 INNER JOIN table1 AS t2 ON t2.emp_id = t1.manager_id -- manager under his employee

Section – B: Practice Questions

-- --

Database Used: Banking.sql

-- --

1. **Print credit card transactions with the sum of transaction_amount on all Fridays and the sum of transaction_amount on all other days.**

```
with trans
as (select t2.* from
(select account_number from bank_account_details where account_type in ('Credit Card',
'Add-on Credit Card')) as t1,
bank_account_transaction as t2
where t1.account_number = t2.account_number)

-- Friday and non-Friday
select abs(sum(if(dayname(transaction_date)='Friday',transaction_amount,0))) as 'Friday
transaction'
 ,abs(sum(if(dayname(transaction_date) != 'Friday', transaction_amount, 0))) as
'Other_days transaction'
from trans;
```

2. **Show the details of credit cards along with the aggregate transaction amount during holidays and non-holidays**.

```
with trans
as (select t2.* from
(select account_number from bank_account_details where account_type in ('Credit Card',
'Add-on Credit Card')) as t1,
bank_account_transaction as t2
where t1.account_number = t2.account_number)

select trans.Account_Number,
      sum(IF(trans.transaction_date in (select holiday from Bank_Holidays),
abs(trans.Transaction_amount), null)) as 'Holiday_amt',
      sum(IF(trans.transaction_date not in (select holiday from Bank_Holidays),
abs(trans.Transaction_amount), null)) as 'Holiday_amt'
from trans group by trans.account_number;
```

3. **Generate a report to Send Ad-hoc holiday greetings - “Happy Holiday” for all transactions that occurred during Holidays in 3rd month.**

```
with trans
as (select t2.Account_Number, t1.account_type, t2.Transaction_Date from
(select account_number, Account_type from bank_account_details where account_type in
('Credit Card', 'Add-on Credit Card')) as t1,
bank_account_transaction as t2
where t1.account_number = t2.account_number)

select trans.*, "Happy Holiday" as message from trans, Bank_Holidays where
trans.Transaction_Date = Bank_Holidays.Holiday and month(Bank_Holidays.Holiday) = 3;
```

4. **Calculate the Bank accrued interest with respect to their RECURRING DEPOSITS for any deposits older than 30 days**.

```
Note: Accrued interest calculation =  transaction_amount * interest_rate # Note: use
CURRENT_DATE()
with trans
as (
   select

BANK_ACCOUNT_TRANSACTION.Account_Number,BANK_ACCOUNT_TRANSACTION.
Transaction_amount,
BANK_ACCOUNT_TRANSACTION.Transaction_Date,t1.interest_rate,
      datediff(BANK_ACCOUNT_TRANSACTION.transaction_date, concat(t1.year,'-
',t1.month,'-05')) as interest_dt,
      BANK_ACCOUNT_TRANSACTION.Transaction_amount * t1.interest_rate as
interest_amt
   from (select * from BANK_INTEREST_RATE where account_type = 'RECURRING
DEPOSITS') as t1,
   BANK_ACCOUNT_TRANSACTION inner join Bank_Account_Details
   on BANK_ACCOUNT_TRANSACTION.Account_Number =
Bank_Account_Details.Account_Number
   and Bank_Account_Details.Account_type = 'RECURRING DEPOSITS'
   )

select *
from trans;
```

5. **Display the Savings Account number whose corresponding Credit cards and AddonCredit card transactions have occurred more than one time**

```
select tb1.* from bank_account_transaction as tb1
where tb1.account_number in  (select distinct t1.Linking_Account_Number from
bank_account_relationship_details as t1, bank_account_relationship_details as t2
where t1.account_number = t2.linking_account_number and t2.account_type in ('Credit
Card', 'Add-on Credit Card') and t1.Linking_Account_Number is not null);

-- more than one time
select account_number from bank_account_transaction group by account_number having
count(*) > 1 and account_number = '4000-1956-5698';
```

6. **Display the Savings Account number whose corresponding AddonCredit card transactions have occurred at least once. (not applicable using code eval)**

```
Prerequisite ( The below record is needed for the next two questions.
If you had already added the above, ignore this )
-- INSERT INTO BANK_CUSTOMER_EXPORT VALUES ('123008',"Robin", "3005-1,
Heathrow",          "NY", "1897614000");
select tb1.* from bank_account_transaction as tb1
where tb1.account_number in  (select distinct t1.Linking_Account_Number from
bank_account_relationship_details as t1, bank_account_relationship_details as t2
where t1.account_number = t2.linking_account_number and t2.account_type in ('Credit
Card', 'Add-on Credit Card') and t1.Linking_Account_Number is not null);

-- occurred at least once
select account_number from bank_account_transaction group by account_number having
count(*) >= 1 and account_number = '4000-1956-5698';
```

7. **Print the customer_id and length of customer_id using Natural join on**

```
Tables :bank_customer  and  bank_customer_export
Note: Do not use table alias to refer to column names.
select *, length(Customer_id) as 'length' from bank_customer natural join
bank_customer_export;
```

8. **Print customer_id, customer_name, and other common columns from both the Tables: bank_customer & bank_customer_export without missing any matching customer_id key column records.**

```
Note: refer to data type conversion if found any missing records
select t1.customer_id, t1.customer_name, t1.Address, t1.state_code, t1.Telephone
from bank_customer as t1 join bank_customer_export on t1.customer_id =
bank_customer_export.customer_id;
```