1.compare correlated subquery and nested subqueries.
Correlated Subquery:

A correlated subquery is used when you want to reference values from the outer query within the subquery. In other words, the subquery's execution depends on the values of the outer query.

The subquery is executed once for each row processed by the outer query. It means that the subquery is aware of the current row being processed by the outer query.

Eg: An example of a correlated subquery might be finding all employees whose salary is greater than the average salary in their department.

Nested Subquery:

A nested subquery is used to retrieve values from a subquery independently of the outer query. The subquery can be evaluated once and its result can be used as a single value or a list of values in the outer query.

The subquery is executed once, and its result is treated as a single value or a list, which can be used in the outer query's comparison.

An example of a nested subquery might be finding the employees with the highest salary. The subquery finds the maximum salary, and the outer query then retrieves employees with that salary.

2.How could a subquery be called when it is found in “from” clause?
subqueries that occur as nested SELECT statements in the FROM clause of an outer SELECT statement. Such subqueries are sometimes called derived tables or table expressions because the outer query uses the results of the subquery as a data source.

3.What is the satisfactory normal form of a good relation?
The most commonly satisfactory normal form for a good relation is the Third Normal Form (3NF), which eliminates data redundancy and ensures data integrity by removing transitive dependencies. However, in some cases, higher normal forms like Boyce-Codd Normal Form (BCNF) or Fourth Normal Form (4NF) may be necessary, depending on specific data requirements. The choice of normal form should be based on your data and design considerations.

4.List the qualities needed for a good design of a relation.
Simplicity: Keep the design straightforward and easy to understand.
Data Integrity: Ensure accuracy and consistency of data with constraints.
Efficiency: Optimize query performance and minimize redundancy.
Normalization: Organize data efficiently and prevent anomalies.
Consistency: Maintain naming and data type consistency.
Scalability: Accommodate potential data growth without performance issues.

5.Find the closure set of a given relation R(V,W,X,Y,Z,P,S,T) with functional dependencies as FD:{{VW},Y->V,{WX}->{YZ},V->{ST},X->P}
i)Start with the attributes of the given relation: {V, W, X, Y, Z, P, S, T}
ii)Apply the functional dependencies one by one:
Add VW to the set.
Add YV (due to Y->V).
Add Y and Z (due to {WX}->{YZ}).
Add ST (due to V->{ST}).

Add P (due to X->P).
iii)No more attributes can be added.
The closure set is: {V, W, X, Y, Z, P, S, T, VW, YV, WX, Y, Z, ST, P}.

Part-B
1.Given Relation: Student(sid, sname, sage, marks,mailid)
sid -> sname, sage
sname -> mailid
(sid, sname) -> marks
a)Candidate Keys:
Candidate keys are minimal sets of attributes that can uniquely identify each tuple in the relation.
From the given functional dependencies, we can see that both 'sid' and 'sname' are candidates for keys because 'sid' determines both 'sname' and 'sage,' and 'sname' determines 'mailid.' A candidate key can consist of one or more attributes.
Prime Attributes:
Prime attributes are attributes that are part of the candidate key(s).
In this case, 'sid' and 'sname' are prime attributes.
Non-prime Attributes:
Non-prime attributes are attributes that are not part of any candidate key and can be derived from the candidate key(s) through functional dependencies.
In this case, 'sage,' 'marks,' and 'mailid' are non-prime attributes.
there are **two candidate keys**: **{sid} and {sname},** two prime attributes: sid and sname, and three non-prime attributes: sage, marks, and mailid.

b. If (sname, sage) is to be a key of the relation, it means that it should uniquely identify each tuple. Therefore, for it to be a key, there should be no two tuples with the same combination of (sname, sage).
it appears that there is some value X missing between "Sindhu" and "80." To make (sname, sage) a key, this missing value should be unique and not conflict with any other (sname, sage) combination. So, X should be a value that is not already present in the 'sname' and 'sage' columns, ensuring uniqueness for each combination of (sname, sage).

2.
1. **name, deptno -> grade**
This is a partial dependency as it suggests that an employee's grade depends on their name and department. To check the closure, we can use the given functional dependencies.
By using closure properties:
name, deptno -> grade (given)
name, deptno -> eid (by transitive dependency, using 3)
name, deptno -> name (by transitive dependency, using 4)
The closure of name, deptno is {name, deptno, eid, grade}, so this dependency is fully functional.
2. eid, deptno -> grade
This is another partial dependency.
By using closure properties:
eid, deptno -> grade (given)
eid, deptno -> name (by transitive dependency, using 4)

The closure of eid, deptno is {eid, deptno, name, grade}, so this dependency is fully functional.
3. name -> eid
This is a full functional dependency as it directly determines eid.
4. eid -> name
This is also a full functional dependency as it directly determines name.

Candidate Keys:
From the given functional dependencies and closures, we can see that {name, deptno} and {eid, deptno} are candidate keys because they uniquely determine all attributes in the relation.
Prime Attributes:
The prime attributes are {name, deptno, eid, grade}.
Non-Prime Attributes:
The non-prime attributes are {name, deptno, eid, grade} (same as prime attributes).
Determine Normal Form:
1NF (First Normal Form):
All attributes must be atomic.
In this case, all attributes are atomic, so the relation is in 1NF.
2NF (Second Normal Form):
All non-prime attributes are fully functionally dependent on the candidate keys.
all non-prime attributes are fully functionally dependent on the candidate keys, so the relation is in 2NF.
3NF (Third Normal Form):
There are no transitive dependencies of non-prime attributes on the candidate keys.
In this case, the relation is also in 3NF because there are no transitive dependencies.
The highest normal form reached is 3NF, and the relation schema is already in 3NF.
Therefore, there is no need for further decomposition.

3.a)Write a SQL query to find those employees who receive a higher salary than the a employee with ID 163. Return first name, last name.

```
SELECT FIRST_NAME, LAST_NAME
FROM EMPLOYEE
WHERE SALARY > (SELECT SALARY FROM EMPLOYEE WHERE EMPLOYEE_ID =
163);
```

b) Write a SQL query to find out which employees have the same designation as the employee whose ID is 169. Return first name , last name, department ID and job ID.

```
SELECT E.FIRST_NAME, E.LAST_NAME, E.DEPARTMENT_ID, E.JOB_ID
FROM EMPLOYEE E
WHERE E.JOB_ID = (SELECT JOB_ID FROM EMPLOYEE WHERE EMPLOYEE_ID =
169);
```

c) Write a SQL query to find those employees whose salary matches the lowest salary of any of the departments, Return first name, last name and department II).

```
SELECT E.FIRST_NAME, E.LAST_NAME, E.DEPARTMENT_ID
FROM EMPLOYEE E
```

```
WHERE SALARY = (SELECT MIN(SALARY) FROM EMPLOYEE WHERE
DEPARTMENT_ID = E.DEPARTMENT_ID);
```

d) Write a SQL query to find those employees who earn more than the average salary Return employee ID, first name, last name.

```
SELECT EMPLOYEE_ID, FIRST_NAME, LAST_NAME
FROM EMPLOYEE
WHERE SALARY > (SELECT AVG(SALARY) FROM EMPLOYEE);
```

e) Write a SQL query to find the employee whose salary is 3000 and reporting person's ID is 121. Return all fields.

```
SELECT *
FROM EMPLOYEE
WHERE SALARY = 3000 AND MANAGER_ID = 121;
```

f) Write a SQL query to find all those employees who work in the Finance department. Return department ID, name (first), job ID and department name,

```
SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, E.JOB_ID, E.FIRST_NAME
FROM DEPARTMENT D
JOIN EMPLOYEE E ON D.DEPARTMENT_ID = E.DEPARTMENT_ID
WHERE D.DEPARTMENT_NAME = 'Finance';
```

g) Write a SQL query to find those employees whose department is located at Toronto Return first name, last name, employee ID. job ID.

```
SELECT E.FIRST_NAME, E.LAST_NAME, E.EMPLOYEE_ID, E.JOB_ID
FROM EMPLOYEE E
WHERE E.DEPARTMENT_ID IN (SELECT DEPARTMENT_ID FROM DEPARTMENT
WHERE LOCATION_ID IN (SELECT LOCATION_ID FROM LOCATION WHERE CITY =
'Toronto'));
```