1a) Define the following terms:
i)Join ii)Division iii)Cartesian product iv)Union v)Set difference.

i)Join
A SQL join clause combines records from two or more tables in a relational database. It creates a set that can be saved as a table or used as it is. A JOIN is a means for combining fields from two tables (or more) by using values common to each.

ii)Division
The formal definition of division is as follows:
A/B = □x(A) - □x((□x(A) ✕ B) – A)
The division operator denoted by ÷ is suited to queries that include the phrase "for all".
suppose that we wish to find all customers who hav3e an account at all the branches located at Brooklyn. We obtain all branches in Booklyn by the expression
r1= Π br_name amount (σ br_city = "Brooklyn" (branch)
we can find all customer-name, br_name pairs for which the customer has an account at a branch by writing r2=Π customer_name,br_name (account |x| depositor)
now we have to find customers who appear in r2 with every branch name in r1.the operation that provide this is r2 ÷ r1.

iii) **Cartesian product**
A Cartesian product of n sets, also known as a n-foldCartesian product, can be represented by an array of n dimensions, where each element is an n-tuple. An ordered pair is a 2-tuple or couple

iv)Union
It performs binary union between two given relations and is defined as –
r ∪ s = { t | t ∈ r or t ∈ s}
Notion – r U s
Where r and s are either database relations or relation result set (temporary relation).
For a union operation to be valid, the following conditions must hold –
r, and s must have the same number of attributes.
Attribute domains must be compatible.
Duplicate tuples are automatically eliminated.
∏ author (Books) ∪ ∏ author (Articles)

v)Set difference
The result of set difference operation is tuples, which are present in one relation but are not in the second relation.
Notation – r – s

2b) Explain about SELECT and PROJECT operations with example.

a) Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − σp(r)

Where σ stands for selection predicate and r stands for relation. p is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

σsubject = "database"(Books)

Output − Selects tuples from books where subject is 'database'.

σsubject = "database" and price = "450"(Books)

Output − Selects tuples from books where subject is 'database' and 'price' is 450.

σsubject = "database" and price = "450" or year > "2010"(Books)

Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

b) Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − ∏A1, A2, An (r)

Where A1, A2 , An are attribute names of relation r.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

∏ subject, author (Books)

3. a. **Define functional dependency. Write all its inference rules** **[7]**

In relational database theory, a functional dependency is a constraint between two sets of attributes in a
relation from a database. In other words, functional dependency is a constraint that describes the relationship between attributes in a relation.

Let A, B and C and D be arbitrary subsets of the set of attributes of the giver relation R, and let AB be
the union of A and B. Then,⇒→

- **Reflexivity**

If B is subset of A, then A → B

- **Augmentation**

If A → B, then AC → BC

- **Transitivity:**

If A → B and B → C, then A → C.

- **Projectivity or Decomposition Rule**

If A → BC, Then A → B and A → C

Proof :
Step 1 : A → BC (GIVEN)
Step 2 : BC → B (Using Rule 1, since B ⊆ BC)
Step 3 : A → B (Using Rule 3, on step 1 and step 2)

- **Union or Additive Rule :**

If A→B, and A→C Then A→BC.
Proof :
Step 1 : A → B (GIVEN)
Step 2 : A → C (given)
Step 3 : A → AB (using Rule 2 on step 1, since AA=A)
Step 4 : AB → BC (using rule 2 on step 2)
[5]
Step 5 : A → BC (using rule 3 on step 3 and step 4)

- **Pseudo Transitive Rule :**

If A → B, DB → C, then DA → C
Proof :
Step 1 : A → B (Given)
Step 2 : DB → C (Given)
Step 3 : DA → DB (Rule 2 on step 1)
Step 4 : DA → C (Rule 3 on step 3 and step 2)

3b) Differences between 3NF and BCNF with example [3]

3NF : A relation is in 3NF if it is in 2NF and no non-prime attribute transitively depends on the primary key. In other words, a relation R is in 3NF if for each functional dependency X ⟶ A in R at least one of the following conditions are met:

1. X is a key or super key in R
2. A is a prime attribute in R

Example
Given the following relation:
EMP_DEPT(firstName, employeeNumber, dateOfBirth, address, departmentNumber, departmentName)
An employee can only work in one department and each department has many employees.
The candidate key is **employee Number**.
Consider the following functional dependencies:

1. employeeNumber ⟶ firstName, dateOfBirth, address, departmentNumber
2. departmentNumber ⟶ departmentName

Given the definition above it is possible to conclude that the relation EMP_DEPT is not in 3NF because the second functional dependency does not meet any of the 2 conditions of the 3NF:

1. departmentNumber is not a key or super key in EMP_DEPT
2. departmentName is not a prime attribute in EMP_DEPT

BCNF : A relation R is in BCNF if it is in 3NF and for each functional dependency X ⟶ A in R, X is a key or super key in R. In other words, the only difference between 3NF and BCNF is that in BCNF it is not present the second condition of the 3NF. This makes BCNF stricter than 3NF as any relation that is in BCNF will be in 3NF but not necessarily every relation that is in 3NF will be in BCNF.

Example

Given the following relation:
STUDENT_COURSE(studentNumber, socialSecurityNumber, courseNumber)
A student can assist to many courses and in a course there can be many students.
The candidate keys are:

1. **socialSecurityNumber**, **courseNumber**
2. **studentNumber**, **courseNumber**

Consider the following functional dependencies:

1. studentNumber ⟶ socialSecurityNumber
2. socialSecurityNumber ⟶ studentNumber

Given the definition above it is possible to conclude that STUDENT_COURSE is not in BCNF as at least studentNumber is not a key or super key in STUDENT_COURSE.

4a) Why do we need normalization? Explain 1NF, 2NF and 3NF with example. [10]

1NF:
As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row.
Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.
The Primary key is usually a single column, but sometimes more than one column can be combined to create a single primary key. For example consider a table which is not in First normal form.

Student	Age	Subject
Adam	15	Biology,
Alex	14	Maths
Stuart	17	Maths

The following table is in 1NF.

Student	Age	Subject
Adam	15	Biology
Adam	15	Maths
Alex	14	Maths
Stuart	17	Maths

Second Normal Form (2NF)
As per the Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails Second normal form.

Third Normal Form (3NF)
Third Normal form applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another nonprime attribute. So this transitive functional dependency should be removed from the table and also the table must be in Second normal form.

5a) Explain i) ACID properties ii) Strict two phase locking. [5+5]

i)ACID Properties
Consistency: The consistency requirement here is that sum of A and B be unchanged by the execution of transaction. Without the consistency requirement, money could be created or destroyed by a transaction. It can be verified easily that if the database is consistent before an execution of the transaction, the database remains consistent after the execution of the transaction. Ensuring consistency for an individual transaction if the responsibility of the application programmer, who codes the transaction.
Durability: Once the execution of the transaction completes successfully, and the user who initiated the transaction has been notified that the transfer of funds has taken place, it must be the case that no system failure will result in a loss of data corresponding to this transfer of funds.
Atomicity: That is the reason for the atomicity requirement: if the atomicity property is present, all actions of the transaction are reflected in the database or none are. The database system keeps track of the old values on the disk, on which transaction performs a write and if the transaction does not complete, the database system restores the old values to make it appear that the transaction never occurred. Ensuring atomicity is the responsibility of the database system itself; it is handled by transaction-management component.
• **Isolation:** Even if the consistency and atomicity properties are ensured for each transaction, if several transactions are executed concurrently, their operations may interleave in some undesirable way resulting in an inconsistent state.
The isolation property of a transaction ensures that the concurrent execution of transactions results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in some order.

ii) Strict two phase locking [5]

Consider the partial schedule of the figure. Each transaction observes the two-phase locking protocol, but the failure of T5 after the read(A) step of T7 leads to cascading rollback of T6 and T7.

T5: T6: T7:
lock-X(A);
Read (A);
lock-S(B);
Read (B);
Write(A);
Unlock(A)
lock-X(A);
Read (A);
Write(A);
Unlock(A)
Lock-S(B);
Lock-S(A);
Read (A);

Cascading rollbacks can be avoided by a modification of two-phase locking called the strict two-phase locking protocol. This protocol requires not only that locking be two phase, but also that all exclusive-mode locks taken by a transaction be held until that transaction commits. This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction form reading the data.

6a) Explain the mechanism to control concurrent access to data item [10]

Lock based protocols:

1. Locks
2. Granting of locks
3. The 2-phase locking protocol
4. Implementation of locking

A lock is a mechanism to control concurrent access to data item.

Data items can be locked in 2 modes.

1. Shared
2. Exclusive

Exclusive(x) – Data item can be both read as well as written. X-lock is requested using lock-X instructions.

Shared(x) – Data item can only be read s-lock is requested using lock-s instructions.

Lock requests are made to concurrency control manager. The compatibility relation between the two modes of locking discussed in this section appears in the matrix comp of Figure. An element comp (A, B) of the matrix has the value true if and only if mode A is compatible with mode B.

	S	X
S	True	false
X	false	false

A transaction to unlock a data item immediately after its final access of that data item is not always desirable, since Serializability may not be ensured.

```
T1: lock-X(B);
Read (B);
B := B – 50;
Write(B);
[10]
Unlock(B);
Lock-X(A);
Read(A);
A := A + 50;
Write(A);
Unlock(A);
T2: lock-S(A);
Read (A);
Unlock(A);
Lock-S(B);
Read(B);
Unlock(B);
Display(A + B);
```

Granting of Lock:

When a transaction requests a lock on a data item in a particular mode, and no other transaction has a lock on the same data item in a conflicting mode, the lock can be granted. However, care must be taken to avoid the following scenario. Suppose a transaction T2 has a shared-mode lock on the data item.

Clearly, T1 has to wait for T2 to release the shared-mode lock. Meanwhile, a transaction T3 may request a shared-mode lock on the same data item. The lock request is compatible with the lock granted to T2, so T3 may be granted the shared-mode lock. At this point T2 may release the lock, but still T1 has to wait for T3 to finish. But again, there may be a new transaction T4 that requests a shared-mode lock on the same data item, and is granted the lock before T3 releases it.

In fact, it is possible that there is a sequence of transaction releases the lock a short while after it is granted, but T1 never gets the exclusive-mode lock on the data item. The transaction T1 may never make progress, and is said to be **starved.**

THE TWO – PHASE LOCKING PROTOCOL

One protocol that ensures Serializability is the **two-phase locking protocol**. This protocol requires that each transaction issue lock and unlock requests in two phases:

1. **Growing phase.** A transaction may obtain locks, but may not release any lock.

2. **Shrinking phase.** A transaction may release locks, but may not obtain any new locks.
Initially, a transaction is in the growing phase. The transaction acquires locks as need. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock request.

Implementation of locking:

A lock-manager can be implemented as a process that receives messages from transactions and sends messages in reply.
The lock manager process requests in the following way:
1. When a lock request message arrives, it adds a record to the end of the linked list for the data item, if the linked list is present. Otherwise it creates a new linked list, containing only the record for the request.
2. When the lock manager receives an unlock message from a transaction, it deletes the record for that data item in the linked list corresponding to that transactions.
3. If a transaction aborts, the lock manager deletes any waiting request made by the transactions.

7a) What is a trigger? Explain DML triggers, with an example. [10]

Triggers are stored programs, which are automatically executed or fired when some events occur.
Triggers are, in fact, written to be executed in response to any of the following events −
A database manipulation (DML) statement (DELETE, INSERT, or UPDATE)
A database definition (DDL) statement (CREATE, ALTER, or DROP).
A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
Triggers can be defined on the table, view, schema, or database with which the event is associated. To demonstrate triggers we will be using the CUSTOMERS table we had created and used in the previous chapters −

```
Select * from customers;
+----+----------+-----+-----------+----------+
| ID | NAME | AGE | ADDRESS | SALARY |
+----+----------+-----+-----------+----------+
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | MP | 4500.00 |
+----+----------+-----+-----------+----------+
```

The following program creates a row-level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values –

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW  WHEN (NEW.ID > 0)
DECLARE
sal_diff number;
```

```
BEGIN
        sal_diff := :NEW.salary - :OLD.salary;
        dbms_output.put_line('Old salary: ' || :OLD.salary);
        dbms_output.put_line('New salary: ' || :NEW.salary);
        dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result –

```
Trigger created.
```

8a) Explain the classification of failures.   [5]

Failure of classification:

1. Transaction Failure – Transaction has to abort when it fails to execute or reaches a point where it cannot go further.

Transaction failures can be either Logical or System errors.

2. System Crash: Interruption to power-supply.

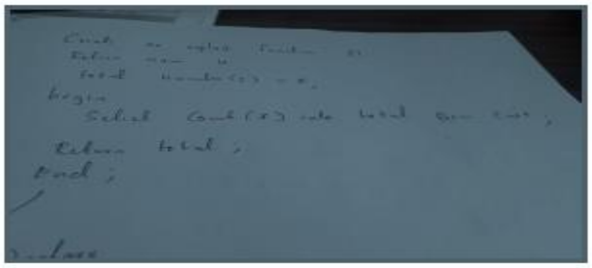
3. Disk-Failure- hard-disk/ Storage drivers used to fail.

8b) Define stored procedure? Explain with example.

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[ (parameter [,parameter]) ]
IS
  [declaration_section]
BEGIN
  executable_section
  [EXCEPTION
   exception_section]
END [procedure_name];
```

When we create a procedure or function, you may define parameters.
There are three types of parameters that can be declared:

1. IN - The parameter can be referenced by the procedure or function. The value of the parameter can not be overwritten by the procedure or function.
2. OUT - The parameter can not be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. IN OUT - The parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.



```
Create or replace function [illegible]
Return number is
  total number(2) := 0;
begin
  Select Count(*) into total from [illegible];
Return total;
End;
/
[illegible]
```

9a) Write in detail about the recovery algorithm along with different phases [10]

ARIES recovers from a system crash in 3 phase:

1. Analysis Pass
2. Redo Pass
3. Undo Pass

Analysis Pass: This pass determines which transactions to undo, which pages were dirty at the time of crash and the LSN from which the redo pass should start.
Redo Pass: This pass starts from a position determined during analysis pass and performs a redo, repeating history, to bring the database to a state it was in before crash.
Undo Pass: This pass toll-back all transactions that were in-complete at the time of crash.

10a) Write short notes on the following: [5+5]

I. Transaction Isolation levels

The isolation levels specified by the SQL standard are as follows:

• Serializable usually ensures serializable execution. However, as we shall explain shortly, some database systems implement this isolation level in a manner that may, in certain cases, allow non serializable executions.

• Repeatable read allows only committed data to be read and further require that, between two reads of a data item by a transaction, no other transaction is allowed to update it. However, the transaction may not be serializable with respect to other transactions. For instance, when it is searching for data satisfying some conditions, a transaction may find some of the data inserted by a committed transaction, but may not find other data inserted by the same transaction.

• Read committed allows only committed data to be read, but does not require repeatable reads. For instance, between two reads of a data item by the transaction, another transaction may have updated the data item and committed.

• Read uncommitted allows uncommitted data to be read. It is the lowest isolation level allowed by SQL.

All the isolation levels above additionally disallow dirty writes, that is, they Disallow writes to a data item that has already been written by another transaction that has not yet committed or aborted.

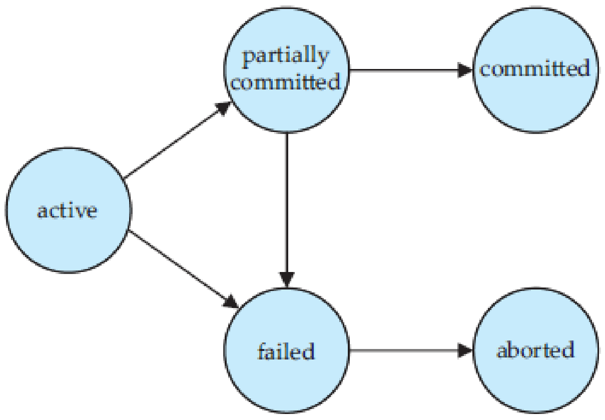
II. Various states of the transaction

A transaction must be in one of the following states:

• **Active,** the initial state; the transaction stays in this state while it is executing.
• **Partially committed**, after the final statement has been executed.
• **Failed**, after the discovery that normal execution can no longer proceed.

• **Aborted**, after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
• **Committed,** after successful completion.

We say that a transaction has committed only if it has entered the committed state.



Similarly, we say that a transaction has aborted only if it has entered the aborted state. A transaction is said to have terminated if it has either committed or aborted. A transaction starts in the active state. When it finishes its final statement, it enters the partially committed state. At this point, the transaction has completed its execution, but it is still possible that it may have to be aborted, since the actual output may still be temporarily residing in main memory, and thus a hardware failure may preclude its successful completion.