DBMS

ACID [ACID Properties]

1. • A transaction is a collection of operations involving data items in a database.
• A transaction usually means that the data in the database has changed.
• DBMS must ensure four important transaction properties to maintain data in the face of concurrent access and system failures, that is ACID properties.
• ACID means Atomicity Consistency Isolation Durability.
• Atomicity.
Atomicity requires that each transaction is all or nothing. If one part of the transaction fails, the entire transaction fails, and the database state is left unchanged.
• Consistency
If each transaction is consistent and the database starts as consistent, it ends up as consistent.
• Isolation
The execution of one transaction is isolated from that of another transactio
It ensures concurrent transaction execution which results in a system state that would be obtained if the transaction were executed serially i.e one after the other.
• Durability
Durability means that once a transaction has been committed, it will remain even in the event of power loss, crashes or errors.
For instance, once a group of SQL statements executes in a relational database, the results need to be stored permanently.

Q.2. [Log-Based Recovery]

• The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
• If any operation is performed on the database, then it will be

recorded in the log.
- But the process of storing the logs should be done before the actual transaction is applied in the database.
- For example, there is transaction to modify the city of student. The following logs are written for this transaction.

(i) When the transaction is initiated, then it writes 'start' log.
< Tn , Start >

(ii) When the transaction modifies the city from 'Noida' to 'Bangalore' then another log is written to file
< Tn , City , 'Noida' , 'Bangalore' >

(iii) When the transaction is finished, then it writes another log to indicate the end of transaction.
< Tn , Commit >

- There are two approaches to modify the database

1. Deferred database modification :

- The deferred modification technique occurs if the transaction does not modify the database until it has committed.
- In this method, all the logs are created and stored in the stable storage and the database is updated when a transaction commits.

2. Immediate database modification :

- The immediate modification technique occurs if database modification occurs while the transaction is still active.
- In this technique, the database is modified immediately after every operation. It follows an actual database modifications.

- Recovery using log records

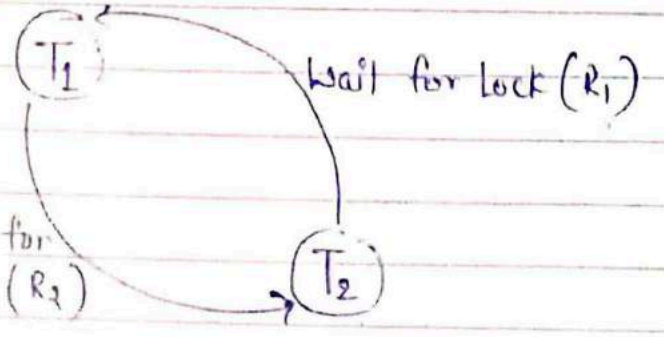
- When the system is crashed, then the system consults the log to find which transactions need to be undone & which need to be redone.

1. If the log contains the records $\langle T_i , start \rangle$ & $\langle T_i , Commit \rangle$ or $\langle T_i , Commit \rangle$ &, then the Transaction T_i needs to be redone.
2. If log contains record $\langle T_n , start \rangle$ but does not contain the

record either < T_i, commit > or < T_i, abort >, then the transaction T_i needs to be undone.

3. **Deadlock**

- A deadlock is a condition in which two or more transactions are waiting for ~~each other deadlock~~ (T_1 & T_2) one another to give up locks.
- For example, Transaction A might hold a lock on some rows in the Accounts table and needs to update some rows in the Orders table to finish. Transaction B holds locks on those very rows in the Orders table but needs to update the rows in the Accounts table held by Transaction A. Transaction A cannot complete its transaction because of the locks on Orders & vice-versa.
- In a database, when a transaction waits indefinitely to obtain a lock then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for graph to detect the deadlock cycle in the database.
- Wait for Graph
  - This is the suitable method for deadlock detection. In this method a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.
  - The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.



4. Timestamp based protocols

- Timestamp based protocols in DBMS is an algorithm which uses th System Time or Logical counter as a timestamp to serialize the execution of concurrent transactions.
- The timestamp based protocol ensures that every conflicting read & write operations are executed in a timestamp order.
- The older transactions is always given priority in this method. It uses a system time to determine the time stamp of the transacti
- This is the most commonly used concurrency protocol.
- Lock-based protocols help you to manage the order between the conflicting transactions when they will execute.
- Time-stamp based protocols manage conflicts as soon as an opesatio is created.

Example:

Suppose there are three transactions T1, T2 & T3.

T1 entered at time 0010

T2 entered at time 0020

T3 entered at time 0030

Priority will be given to transaction T1 then transactions T2 & last Transaction T3.

- The main idea for this protocol is to order the transactions based on their timestamps.
- The schedule is equivalent to the particular Serial order corresponding to the order of Transaction timestamps
- Basic Timestamp Ordering.
- Every transaction is issued a timestamp based on when it enters the system.
- Suppose if an old transaction T_i has timestamp $TS(T_i)$, a new transaction T_j is assigned timestamp $TS(T_j)$ such that $TS(T_i) < TS(T_j)$. The protocol manages concurrent execution such that the timestamp determine the serializability order.

- The timestamp ordering protocol ensures that any conflicting read & write operations are executed in timestamp order.
- Whenever some Transaction T tries to issue a R_item(X) or a W_item(X), the Basic TO algorithm compares the timestamp of T with R_TS(X) & W_TS(X) to ensure that the Timestamp ~~compares the time~~ order is not violated. This describes the Basic TO protocol in the following two cases:

1. Whenever a Transaction T issues a W_item(X) operation, check following conditions:
   - If R_TS(X) > TS(T) or if W_TS(X) > TS(T), then abort and rollback T and reject the operation else,
   - Execute W_item(X) operation of T and set W_TS(X) to TS(T)
2. Whenever a Transaction T issues a R_item(X) operation, check the following conditions:
   - If W_TS(X) > TS(T), then abort and reject T and reject the operation, else
   - If W_TS(X) <= TS(T), then execute the R_item(X) operatio of T and set R_TS(X) to larger of TS(T) and current R_TS

Advantages:
- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks!

Disadvantages:
Starvation is possible if the same transaction is restarted and continual aborted.

5. **Lossless join decomposition.**

- There are two possibilities when a relation R is decomposed in R_1 & R_2 They are
  (i) lossy decomposition i.e., $R_1 \bowtie R_2 \supset R$.
  (ii) Lossless decomposition i.e., $R_1 \bowtie R_2 = R$.
- For a decomposition to be lossless, it should hold the following conditions
  (1) Union of attribute R_1 & R_2 must be equal to attribute R each attribute of R must be either in R_1 or in R_2 i.e. $Att(R_1) \cup Att(R_2) = Att(R$
  (2) Intersection of attributes of R_1 & R_2 must not be null i.e., $Att(R_1) \cap Att(R_2) \neq \phi$
  (3) Common attributes must be a key for atleast one relation (R_1 or R_2) i.e. $Att(R_1) \cap Att(R_2) \rightarrow Att(R_1)$ or $Att(R_1) \cap Att(R_2) \rightarrow Att(R_2)$

Example
A relation R(A,B,C,D) with FD set {A → BC} is decomposed into R_1(A,B,C & R_2(A, D). This is a lossless join decomposition because
- First rule holds true $Att(R_1) \cup Att(R_2) = (ABC) \cup (AD) = (ABCD) = Att(R$
- Second rule holds true as $Att(R_1) \cap Att(R_2) = (ABC) \cap (AD) = A \neq \phi$
- Third rule holds true as $Att(R_1) \cap Att(R_2) = A$ is a key of R_1(ABC because A → BC is given.

Dependency Preserving Decomposition

- If we decompose a relation R into relations R_1 & R_2, all dependencies of R must be part of either R_1 & or R_2 or must be derivable from combination of functional dependencies (FD) of R_1 & R_2.
- Suppose a relation R(A, B, C, D) with FD set {A → BC} is decompos into R_1(ABC) & R_2(AD) which is dependency preserving because FD A → BC is a part of R_1(ABC)
- Example
Consider a schema R(A, B, C, D) & functional dependencies A → B & C → D which is decomposed into R_1(AB) & R_2(CD)

This decomposition is dependency preserving decomposition because
- $A \rightarrow B$ can be ensured in $R_1(AB)$
- $C \rightarrow D$ can be ensured in $R_2(CD)$

9. **Deadlock Prevention.**

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The DBMS analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then DBMS never allows that transaction to be executed.

1. Wait - Die Scheme

- In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.
- Let's assume there are two transactions T_i & T_j and let TS(T) is a timestamp of any transaction T. If T_2 holds a lock by some other transaction and T_1 is requesting for resources held by T_2 then the following actions are performed by DBMS:

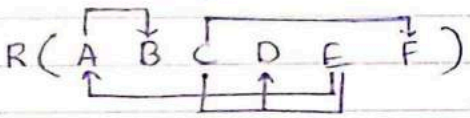
1. Check if $TS(T_i) < TS(T_j)$ - If T_i is the older transaction & T_j has held some resource, then T_i is allowed to wait until the data item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
2. Check if $TS(T_i) < TS(T_j)$ - If T_i is older transaction & has held some resource & if T_j is waiting for it, then T_j is killed & restarted later with the random delay but with the same timestamp

2. Wound wait scheme (Preemptive) :
Older transactions bounds (forces rollback) of the younger transactio instead of waiting for it. Younger transactions may wait for older ones. May be fewer rollbacks then the wait-die scheme.

3. Starvation :
Starvation occurs when a particular transaction consistently waits or restarted and never gets a chance to proceed further. In a deadl resolution scheme, it is possible that the same transaction may consistently be selected as a victim and rolled back.

10. R(A B C D E F)
FD = { C → F , E → A , EC → D , A → B }

→ Construct an arrow diagram on R using FD to calculate the candidate key.



- From the above diagram we can see that an attribute EC is not determined by any of the given FD, hence EC will be the integral part of Candidate key.
- Closure of EC
  EC + = ADFBEC
  Since the closure of EC contains all the attributes of R hence EC is a candidate key.
- Definition of 2NF : No non-prime attributes should be partially dependent on Candidate key.

Since R has 6 attributes & Candidate key is EC. Therefore prime attributes (part of candidate key) are E & C while non-prime attributes are A, B, D, F

a) FD : C → F does not satisfy the definition of 2NF, as a non prime attribute (F) is partially dependent on candidate key EC.

b) FD : E → A does not satisfy the definition of 2NF as a non prime attribute (A) is partially dependent on candidate key EC.

c) FD: EC → D satisfies the definition of 2NF that non-prime attribute C is fully dependent on candidate key EC.

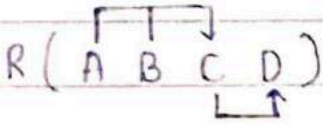
d) FD : A → B does not satisfies the definition of 2NF as no two non-prime attribute can depend on other non-prime attribute.

∴ The R(A B C D E F) is not in 2NF.

Q10. R(A B C D)

FD = {AB → C, C → D}

→ Let us construct an arrow diagram on R using FD to calculate the candidate key.



· From the above diagram of R, we can see that an attribute AB is not determined by any of the given FD, hence AB will be the integral part of Candidate key.

· Closure of AB

AB + = ABCD

since the closure of AB contains all attributes of R, hence AB is a Candidate key.

· Definition of 3NF :

A relational schema R is said to be in 3NF, first it should be in 2NF and no non-prime attribute should be transitively dependent on key of the table.

If AB → C & C → D exist then AB → D also exists which is a

transitive dependency and it should not hold

- Since R has 4 attributes : A, B, C, D and Candidate key is A, B Therefore prime attribute is A & B while non-prime attribute are C & D.
- Given FD are $AB \rightarrow C$ and $C \rightarrow D$ so we can write $AB \rightarrow D$ (which is a transitive dependency)
- In above FD : $AB \rightarrow D$, a non prime attribute D is transitively depending on Key of table hence as per the definition of 3NF it is not in 3NF, because no non-prime attribute should be transitively dependent on key of table.

Q 12. Normalization

- Normalization is the process of minimizing redundancy from a relation or set of relations.
- Redundancy in relation may cause insertion, deletion and update anomalies. So it helps to minimize the redundancy in relations.
- Normal forms are used to eliminate or reduce redundancy in database tables.
- A large database defined as a single relation may result in data duplication.
- This repetition of data may result in :

Q 13. Normalization

- Normalization is the process of organizing the data in the database
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update and Deletion anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

• Types of Normal forms

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. Following are the types of Normal forms

(i) 1 NF

In this, relation is in 1 NF if there exists no multivalued attributes. For example: A Relation student has three attributes (Rno, Name, Course) and course is a multivalued attribute

Student

Rno	Name	Course
1	A	C, C++
2	B	Java
3	C	C, Python

=>

Rno	Name	Course
1	A	C
1	A	C++
2	B	Java
3	C	C
3	C	Python

The final relation becomes

Rno	Name	Course 1	Course 2
1	A	C	C++
2	B	Java	Null
3	C	C	Python

Base table

Rno	Name
1	A
2	B
3	C

Referential tab

Rno	Course
1	C
1	C++
2	Java
3	C
3	Python

(ii) 2 NF

A relation will be in 2NF if it is in 1 NF and all non-key attributes are fully functional dependent on primary key.

Example:

C_id	S_id	location
1	1	Agra
1	3	Delhi
2	1	Agra
3	2	Mumbai
4	3	Delhi

Candidate key = { C_id, S_id }
Prime attribute = { C_id, S_id }
Non-prime attribute = { location }

In the above relation the location depends on S_id, so partial dependency exists
Hence we have to normalize to 2NF

R_1

C_id	S_id
1	1
1	3
2	1
3	2
4	4

R_2

S_id	location
1	Agra
3	Delhi
2	Mumbai

(iii) 3 NF

A relation is in third Normal form if there is no transitive dependency for non-prime attributes as well as it should be 2 NF.

Example :

Student

Rno	State	City
1	MH	Pune
2	GJ	Surat
3	MH	Pune
4	GJ	Surat
5	MP	Indore

FD : { Rno → State, State → city }
Ck : { Rno }
Prime attribute = { Rno } Non-Prime = { state, city }

Rno → state & state → city

So Rno → city

Not in 3NF since there exist transitive dependency

So Normalize to 3 NF

Rno	state
1	MH
2	GJ
3	MH
4	GJ
5	MP

State	City
MH	Pune
GJ	Surat
MP	Indore

(iv) BCNF

A relation is in BCNF iff in every non-trivial functional dependency $X \rightarrow Y$, X is a super key.

Example:

Id	Subject	Professor
101	Java	Mayank
101	C++	Kartik
102	Java	Sarthak
103	C#	Laksh
104	Java	Mayank

- One student can enroll in more than 1 subject
  e.g. student with id 101 has enrolled in Java & C++.
- As each professor teaches only one subject but one subject may be taught by multiple professors. This shows that there is a dependency between the subject & the professor, and the subject is always dependent on the professor (professor → subject). As professor column is non-prime attribute while subject is a prime attribute. This is not allowed in BCNF. For BCNF, the deriving attribute (professor here) must be a prime attribute

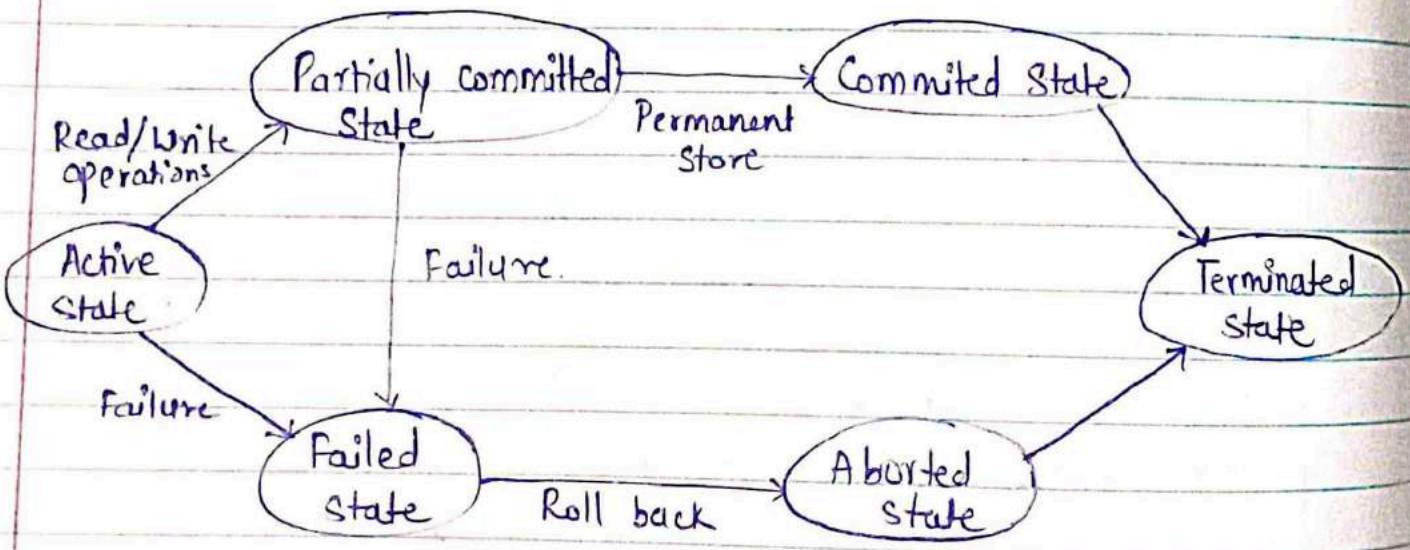
• To normalize it to BCNF we will decompose the table into two tables i.e Student & Professor table

P_id	S_id	Professor
1	101	Mayank
2	101	Kartik
3	102	Sarthak
4	103	Laksh
5	104	Mayank

Professor	Subject
Mayank	Java
Kartik	C++
Sarthak	Java
Laksh	C#
Mayank	Java

Q13. **Transaction states**

• The states through which a transaction passes during its lifetime.
• These are the states that describe the current state of the transaction and how we will proceed with the processing. These states govern ~~the~~ the rules determining whether the transaction will commit or abort.



Active State

A transaction will be called in an active state if its instructions are executed. This is because all the changes made by this transaction are now stored in the buffer in main memory.

Partially committed state

After the last instruction of a transaction has been executed it enters this partially committed state. After entering the state, the transaction is said to be partially committed because it has not performed the commit operation. However, the transaction is not considered fully committed because all the transaction changes are still stored in the buffer in main memory.

Committed state

After all the changes executed by the transaction have been committed and stored successfully in the database, it enters into state called the committed state. Now, the transaction is considered fully committed.

Abort state (Roll back state)

When a transaction is being executed in active state or partially committed state, and some failure occurs, due to which it becomes impossible to continue the execution, it enters into a failed state.

Failed state

After the transaction is failed and enters into a failed state all the changes made by this transaction have to be undone. To undo the changes made by this transaction, it becomes necessa to roll back all the transaction operations. After the transaction fully rolls back, it enters into an aborted state.

Terminated state

When a transaction successfully perform the redo operation in case of a committed transaction or performs the undo operation in case of a failed transaction it is said to be in terminated

state. Therefore to maintain the correctness of the databases, it will execute all the transactions successfully or none of the transactions will be executed in case of any failure occurs.

Q15. **Trigger**

- A trigger is a stored procedure in database which automatically invokes ~~the~~ whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
- In SQL, triggers are called only either before or after the below events:
  (i) INSERT Event : This event is called when the new row is entered in the table.
  (ii) UPDATE Event : This event is called when the existing record is changed or modified in the table.
  (iii) DELETE Event : This event is called when the existing record is removed from the table.

-Types of triggers in ~~SQL~~.
Following are the six types of trigger

- Syntax :

```
create trigger [trigger_name]
[before / after]
{ insert / update / delete }
on [tablename]
[for each row]
[trigger_body]
```

- Types of triggers in SQL

Following are the six types of trigger in SQL

1. AFTER INSERT Trigger

This trigger is invoked after the insertion of data in the table

2. AFTER UPDATE Trigger

This trigger is invoked in SQL after the modification of the data in the table.

3. AFTER DELETE Trigger

This trigger is invoked in SQL after deleting the data from the table.

4. BEFORE INSERT Trigger

This trigger is invoked before inserting the record in the table.

5. BEFORE UPDATE Trigger

This trigger is invoked before the updation of record in the table.

6. BEFORE DELETE Trigger

This trigger is invoked before deleting the record from the table.

- Example :

1. After Delete

```
create trigger after_salaries_delete1
  after delete
  on employee1 for each row
  update salaryBudget
  set total = total - old.salary ;
```

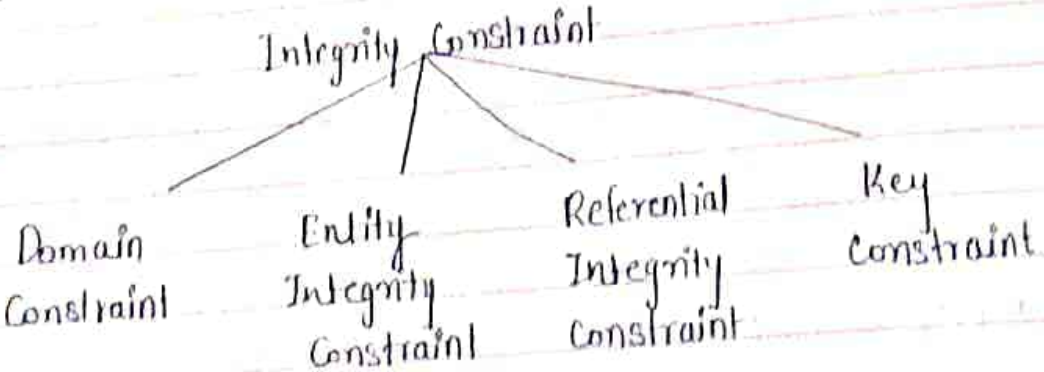
- Advantages

1. Triggers help in executing the scheduled tasks because they are called automatically.
2. Triggers catch the errors in the database layer of various businesses.
3. They allow the database users to validate values before inserting & updating.

Disadvantages :
1. They are not compiled
2. It is not possible to find & debug the errors in triggers.
3. Trigger increases high load on database system.

Q16. **Integrity Constraint**

- Integrity Constraints are set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to database.
- Types of integrity constraint



1. Domain constraint
- Domain constraint can be defined as the definition of valid set of values for an attribute.
- The data type of domain includes strings, character, integer, time, date, currency, etc. The value of attribute must be available in the corresponding domain.
- E.g

ID	Name	Semester	Age
1000	Tom	1st	17
1001	~~Jakson~~ John	2nd	24
1002	Kate	5th	A

↑ Not allowed. Because Age is an integer attribu

2. Entity Integrity constraints

- The entity integrity constraints states that primary key can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field

Example:

Employee

Emp_id	Emp_name	Salary
123	Jack	30000
142	Harry	60000
164	John	20000
	~~Jackson~~	27000

↑ Not allowed as primary key can't contain a null value.

3. Referential Integrity constrainls

- A referential integrity constraint is specified between two tables.
- In referential integrity constraint, if a foreign key in Table 1 refers to Primary key of Table 2, then every value of foreign key in Table 1 must be null or be available in Table 2.

Example:

Table 1

Emp_id	Name	Age	D_no
1	Jack	20	11
2	Harry	40	24
3	John	27	18
4	Devil	38	13

← foreign key

Not allowed as D_no 18 is not defined as primary key of table 2 and in table 1 D_no is a foreign key defined

Table 2

D_no	D_location
11	Mumbai
24	Delhi
13	Noida

Primary key

PAGE No.
DATE / /

6. Book(bid, title, author, cost)
Store (store_no, city, stock, inventory_val)
stock (store_no, bid, quantity).

1. modify the cost of DBMS books by 10%.
~~Update~~
UPDATE Book SET cost = cost * + (cost * 10.0/100.0)
WHERE title ~~LIKE '%DBMS%'~~;
OR
title = 'DBMS';

2. Find the author of the books which are available in Mumbai Store.
SELECT DISTINCT Book.author
FROM Book
INNER JOIN Stock ON Book.bid = Stock.bid
INNER JOIN Store ON Stock.Store_no = Store.Store_no.
WHERE Store.city = 'mumbai';

3. Find the title of the most expensive book
~~SELECT DISTINCT Book.~~
SELECT title
FROM Book
WHERE cost = (SELECT MAX(cost) FROM Book);

4. Find the total quantity of books in each Store.

4. SELECT Store_no SUM(quantity) AS total quantity
FROM STOCK;

5. Add a new record in Book table.
INSERT INTO Book (bid, title, author, cost) VALUES
(104, 'Algorithms and Data Structures', 'Goodrich and Tamassia',
300);

7. Employee (eid, e_name, street, city)
Works (eid, cid, salary)
Company (cid, c_name, city)
Manager (eid, m_name)

1. Find the names of all employees having 's'.
SELECT e_name
FROM Employee
WHERE e_name LIKE '%s%';

2. Display annual salary of all employees.
SELECT e_name, salary * 12 AS annual_salary
FROM Works
INNER JOIN Employee ON Employee.eid = Works.eid;

3. Find the name, street and city of all employees who work for 'Accenture' and earn more than 30000.
SELECT e_name, street, city
FROM Employee
INNER JOIN Works ON Employee.eid = Works.eid
INNER JOIN Company ON Works.cid = Company.cid
WHERE Company.c_name = 'Accenture' AND
Works.salary > 30000;

PAGE No.
DATE / /

4. Give total number of employees.
```
SELECT COUNT(eid) AS total_employees
FROM Employee;
```

5. Delete a record of manager whose name is Jone.
```
DELETE FROM Manager WHERE m_name = 'Jones';
```

8. Department (did, dname, mgrid, loc_id)
Location (loc_id, street, pincode, city, state, cid)
Employee (eid, fname, lname, email, phno, jobid, mgrid, data, salary, commission, did)

1- Display department no, lastname, dept name for all employees.
```
SELECT Employee.did, Employee.lname, Department.dname
FROM Employee
JOIN Department ON Employee.did = Department.did;
```

2. Create unique listing of all jobs that are in Department also include location of department.
```
SELECT DISTINCT Employee.jobid, Department.loc_id,
Location.city
FROM Employee
JOIN Department ON Employee.did = Department.did
JOIN Location ON Department.loc_id = Location.loc_id;
```

3. Display lname, dname, loc-id and city of all employees who earn commission.
```
SELECT Employee.lname, Department.dname, Department.loc_id,
Location.city FROM Employee
JOIN Department ON Employee.did = Department.did
JOIN Location ON Department.loc_id = Location.loc_id
WHERE Employee.commission IS NOT NULL;
```

PAGE No.
DATE / /

4. Display the difference between highest and lowest salary and label output as 'Difference'.

```
SELECT MAX(salary) - MIN(salary) AS Difference
FROM Employee;
```

5. List department id from department that do not contain job id 20 using set operation.

```
SELECT did FROM department EXCEPT
SELECT did FROM Employee WHERE jobid = 20;
```