DBMS CAT2

Q1.What do you mean by functional dependency?

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table. The left side of FD is known as a determinant, the right side of the production is known as a dependent.

> **For example:**Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

> Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types are: Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Q2) What do you mean by transitive functional dependency?

Whenever some indirect relationship happens to cause functional dependency (FC), it is known as Transitive Dependency. Thus, if A -> B and B -> C are true, then A -> C happens to be a transitive dependency.

Thus, to achieve 3NF, one must eliminate the Transitive Dependency.

Example

<Show_Telecast>

Show_ID	Telecast_ID	Telecast_Type	CD_Cost ($)

F08	S09	Thriller	50
F03	S05	Romantic	30
F05	S09	Comedy	20

The table above is not in its 3NF because it includes a transitive functional dependency.

Show_ID -> Telecast_ID

Telecast_ID -> Telecast_Type

Thus, the following has a transitive type of functional dependency.

Show_ID -> Telecast_Type

The statement given above states the relation <Show_Telecast> violates the 3NF (3rd Normal Form). If we want to remove this violation, then we have to split the tables for the removal of the transitive functional dependency.

<Show>

Show_ID	**Telecast_ID**	**CD_Cost ($)**
F08	S09	50
F03	S05	30
F05	S09	20

<Telecast>

Telecast_ID	**Telecast_Type**
S09	Thriller
S05	Romantic
S09	Comedy

Now the above relation is in the Third Normal Form (3NF) of Normalization.

Q3) What is partial functional dependency?

Partial Dependency occurs when a non-prime attribute is functionally dependent on part of a candidate key.

The 2nd Normal Form (2NF) eliminates the Partial Dependency.

Let us see an example –

Example

<StudentProject>

StudentID	ProjectNo	StudentName	ProjectName
S01	199	Katie	Geo Location
S02	120	Ollie	Cluster Exploration

In the above table, we have partial dependency; let us see how −

The prime key attributes are **StudentID** and **ProjectNo**, and

StudentID = Unique ID of the student**StudentName** = Name of the student**ProjectNo** = Unique ID of the project**ProjectName** = Name of the project

As stated, the non-prime attributes i.e. **StudentName** and **ProjectName** should be functionally dependent on part of a candidate key, to be Partial Dependent.

The **StudentName** can be determined by **StudentID**, which makes the relation Partial Dependent.

The **ProjectName** can be determined by **ProjectNo**, which makes the relation Partial Dependent.

Therefore, the <StudentProject> relation violates the 2NF in Normalization and is considered a bad database design.

To remove Partial Dependency and violation on 2NF, decompose the tables −

<StudentInfo>

StudentID	ProjectNo	StudentName
S01	199	Katie
S02	120	Ollie

<ProjectInfo>

ProjectNo	ProjectName
199	Geo Location

120	Cluster Exploration

Now the relation is in 2nd Normal form of Database Normalization.

Q4) What is full functional dependency?

An attribute is fully functional dependent on another attribute, if it is Functionally Dependent on that attribute and not on any of its proper subset.

For example, an attribute Q is fully functional dependent on another attribute P, if it is Functionally Dependent on P and not on any of the proper subset of P.

Let us see an example −

<ProjectCost>

ProjectID	ProjectCost
001	1000
001	5000

<EmployeeProject>

EmpID	ProjectID	Days
E099	001	320
E056	002	190

The above relations states that −

Days are the number of days spent on the project.

EmpID, ProjectID, ProjectCost -> Days

However, it is not fully functional dependent.

Whereas the subset **{EmpID, ProjectID}** can easily determine the {Days} spent on the project by the employee.

This summarizes and gives our fully functional dependency −

{EmpID, ProjectID} -> (Days)

Q5) Explain 2NF with example.

In the 2NF, relational must be in 1NF.

In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Q6) Explain 1NF with example.

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE

14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Q7) Explain 3NF with example.

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

   Example:

   EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so o n

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Q8) Explain BCNF with example.

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India

264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Q9) Explain 4NF with example.

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.

- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Q10) Differentiate between 3NF and BCNF.

Difference between 3NF and BCNF :

S.NO.	3NF	BCNF
1.	In 3NF there should be no transitive dependency that is no non prime attribute should be transitively dependent on the candidate key.	In BCNF for any relation A->B, A should be a super key of relation.
2.	It is less stronger than BCNF.	It is comparatively more stronger than 3NF.
3.	In 3NF the functional dependencies are already in 1NF and 2NF.	In BCNF the functional dependencies are already in 1NF, 2NF and 3NF.
4.	The redundancy is high in 3NF.	The redundancy is comparatively low in BCNF.
5.	In 3NF there is preservation of all functional dependencies.	In BCNF there may or may not be preservation of all functional dependencies.
6.	It is comparatively easier to achieve.	It is difficult to achieve.

S.NO.	3NF	BCNF
7.	Lossless decomposition can be achieved by 3NF.	Lossless decomposition is hard to achieve in BCNF.

Q11) What is Multivalued dependency?

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	MANUF_YEAR	COLOR
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

1. BIKE_MODEL → → MANUF_YEAR
2. BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

q12) Find the Normal Form

R(ABCDE)

AB->C

C->E

D->A

Q18) Explain ACID Property.

A transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.
In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.

Atomicity:
By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.
—**Abort**: If a transaction aborts, changes made to the database are not visible.
—**Commit**: If a transaction commits, changes made are visible.
Atomicity is also known as the 'All or nothing rule'.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be

executed in its entirety in order to ensure the correctness of the database state.

Consistency:

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,
The total amount before and after the transaction must be maintained.
Total **before T** occurs = **500 + 200 = 700**.
Total **after T occurs** = **400 + 300 = 700**.
Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.
Let **X**= 500, **Y** = 500.
Consider two transactions **T** and **T''.**

T	T''
Read (X)	Read (X)
X: = X*100	Read (Y)
Write (X)	Z: = X + Y
Read (Y)	Write (Z)
Y: = Y – 50	
Write (Y)	

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result, interleaving of operations takes place due to which **T''** reads the correct value of **X** but the incorrect value of **Y** and sum computed by
T'': (X+Y = 50, 000+500=50, 500)
is thus not consistent with the sum at end of the transaction:
T: (X+Y = 50, 000 + 450 = 50, 450).
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

Some important points:

Property	Responsibility for maintaining properties
Atomicity	Transaction Manager
Consistency	Application programmer
Isolation	Concurrency Control Manager
Durability	Recovery Manager

The **ACID** properties, in totality, provide a mechanism to ensure the correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

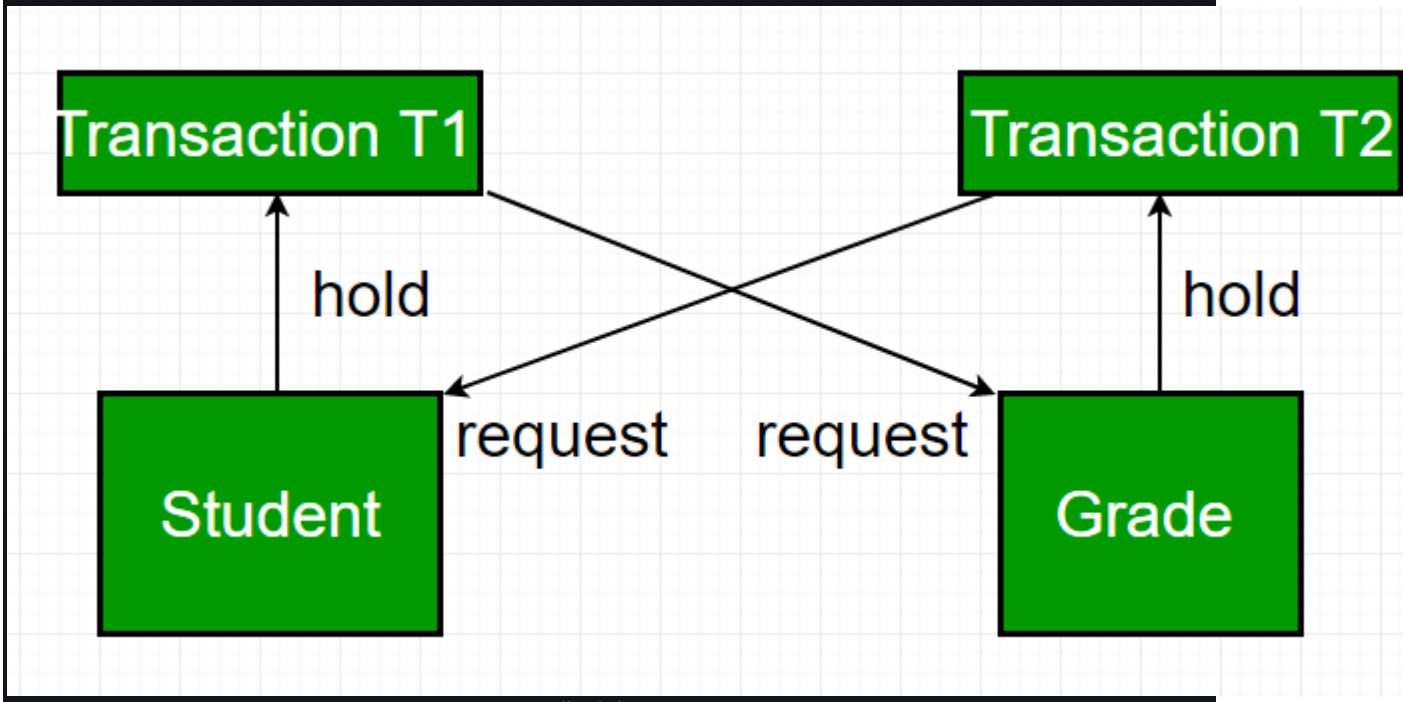
Q19) What is serializability?

Q20) What do you mean by deadlock?

a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as it brings the whole system to a Halt.

Example – let us understand the concept of Deadlock with an example :
Suppose, Transaction T1 holds a lock on some rows in the Students table and **needs to update** some rows in the Grades table.
Simultaneously, Transaction **T2 holds** locks on those very rows (Which T1 needs to update) in the Grades table **but needs** to update the rows in the Student table **held by Transaction T1**.
Now, the main problem arises. Transaction T1 will wait for transaction T2 to give up the lock, and similarly, transaction T2 will wait for

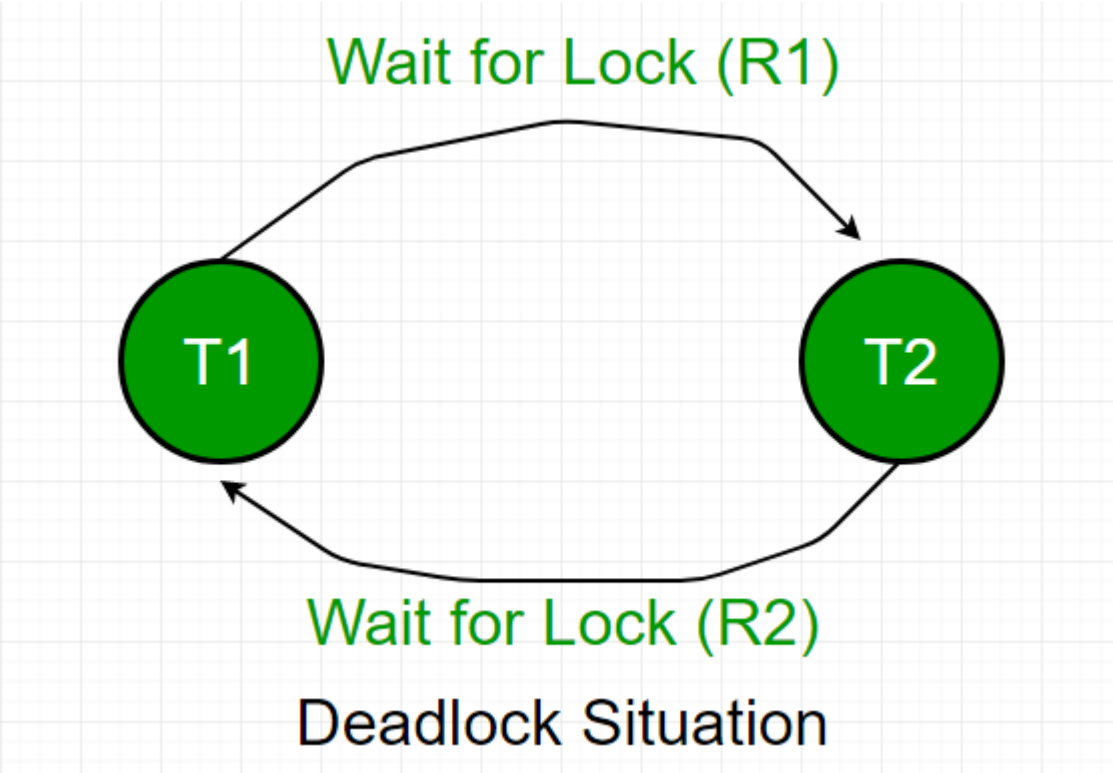
transaction T1 to give up the lock. As a consequence, All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.



Deadlock in DBMS

Deadlock Avoidance –
When a database is stuck in a deadlock, It is always better to avoid the deadlock rather than restarting or aborting the database. The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases.
One method of avoiding deadlock is using application-consistent logic. In the above-given example, Transactions that access Students and Grades should always access the tables in the same order. In this way, in the scenario described above, Transaction T1 simply waits for transaction T2 to release the lock on Grades before it begins. When transaction T2 releases the lock, Transaction T1 can proceed freely.
Another method for avoiding deadlock is to apply both row-level locking mechanism and READ COMMITTED isolation level. However, It does not guarantee to remove deadlocks completely.
Deadlock Detection –
When a transaction waits indefinitely to obtain a lock, The database management system should detect whether the transaction is involved in a deadlock or not.
Wait-for-graph is one of the methods for detecting the deadlock situation. This method is suitable for smaller databases. In this method,

a graph is drawn based on the transaction and their lock on the resource. If the graph created has a closed-loop or a cycle, then there is a deadlock.
For the above-mentioned scenario, the Wait-For graph is drawn below



Deadlock prevention –
For a large database, the deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that deadlock never occurs. The DBMS analyzes the operations whether they can create a deadlock situation or not, If they do, that transaction is never allowed to be executed.
Deadlock prevention mechanism proposes two schemes :

- **Wait-Die Scheme –**
  In this scheme, If a transaction requests a resource that is locked by another transaction, then the DBMS simply checks the timestamp of both transactions and allows the older transaction to wait until the resource is available for execution. Suppose, there are two transactions T1 and T2, and Let the timestamp of any transaction T be TS (T). Now, If there is a lock on T2 by some other transaction and T1 is requesting for resources held by T2, then DBMS performs the following actions:

Checks if TS (T1) < TS (T2) – if T1 is the older transaction and T2 has held some resource, then it allows T1 to wait until resource is available for execution. That means if a younger transaction has locked some resource and an older transaction is waiting for it, then an older transaction is allowed to wait for it till it is available. If T1 is an older transaction and has held some resource with it and if T2 is waiting for it, then T2 is killed and restarted later with random delay but with the same timestamp. i.e. if the older transaction has held some resource and the younger transaction waits for the resource, then the younger transaction is killed and restarted with a very minute delay with the same timestamp.
This scheme allows the older transaction to wait but kills the younger one.

- **Wound Wait Scheme –**
  In this scheme, if an older transaction requests for a resource held by a younger transaction, then an older transaction forces a younger transaction to kill the transaction and release the resource. The younger transaction is restarted with a minute delay but with the same timestamp. If the younger transaction is requesting a resource that is held by an older one, then the younger transaction is asked to wait till the older one releases it.

Following table list the differences between Wait – Die and Wound -Wait scheme prevention schemes :

Wait – Die	Wound -Wait
It is based on a non-preemptive technique.	It is based on a preemptive technique.
In this, older transactions must wait for the younger one to release its data items.	In this, older transactions never wait for younger transactions.
The number of aborts and rollback is higher in these techniques.	In this, the number of aborts and rollback is lesser.

Q21) Define transaction with example.

Q23. R(A, B, C)

A→B

B→ C

Find the closure of each attribute.

A+ = A U B

AB U C

ABC

B+ = B U C

BC

C+ = C

Q24)What do you mean by deadlock?

A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in waiting state forever.

For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

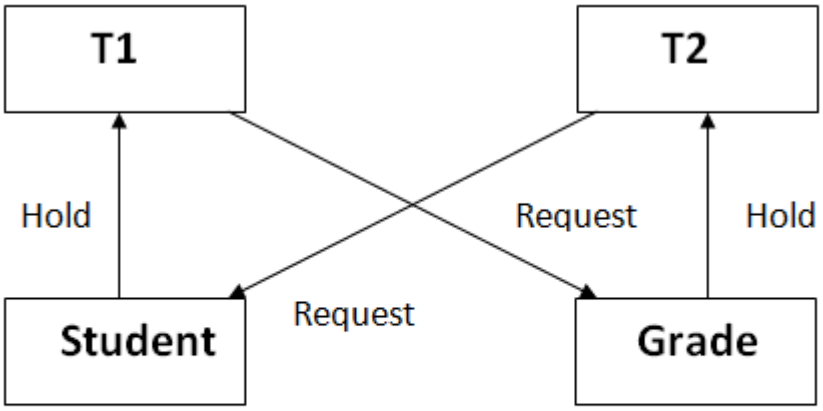


Figure: Deadlock in DBMS

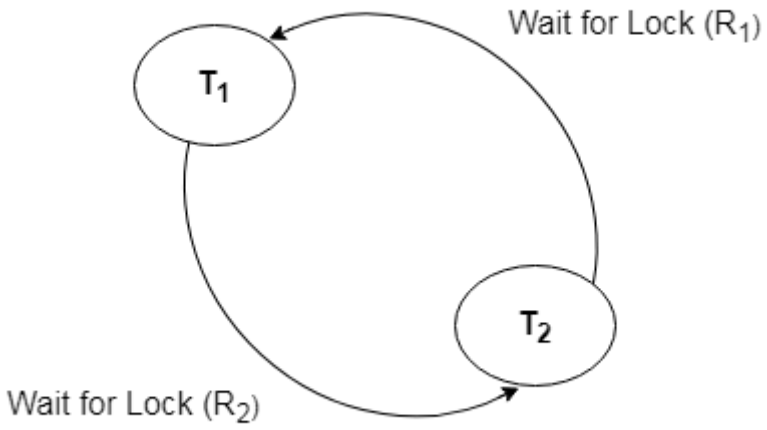
25. Explain Deadlock detection.

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 the 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.

The wait for a graph for the above scenario is shown below:



Q26) What is serial execution in transaction?

In serial execution, the second transaction can begin its execution only after the first transaction has completed. This is possible on a uniprocessor system.

Example

Given below is an example of serial execution −

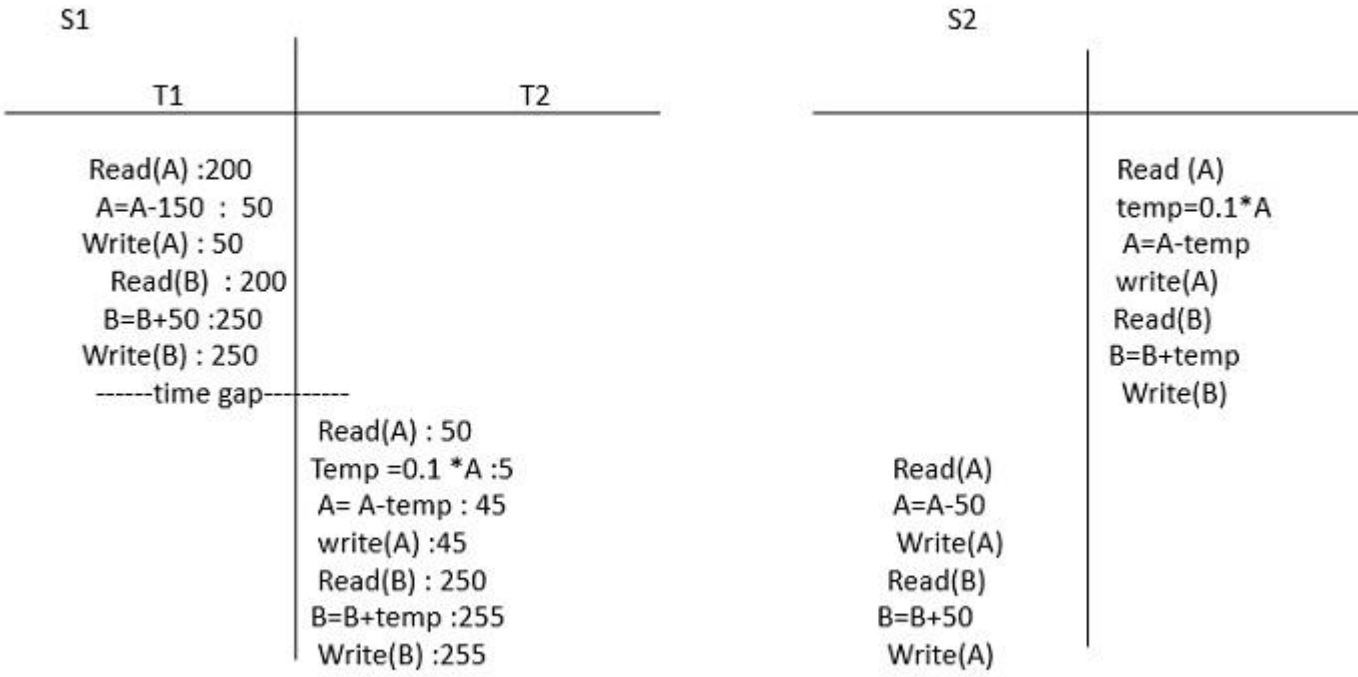
Let us consider two transactions T1 and T2 where T1 performs transfer Rs.150 from account A to account B and similarly T2 transfers 10% of balance from A to B.

```
T1          T2
Read(A)     Read(A)
A=A-150     temp =0.1 *A
Write(A)    A=A-temp
Read(B)     Write(A)
B=B+150     Read(B)
Write(B)    B=B+temp
            Write(B)
```

The order in which the instructions of transaction T1 and T2 are executed is called a Schedule.

The possible serial schedules are as follows −

Let A= 200, B=200



Schedule S1 and S2 keeps the database in consistent state.

In general if the system consists of n number of transactions, then we can generate n! number of valid serial schedules.

Q27. Explain conflict serializable schedule.

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

28. What is concurrent execution in transaction?

Transaction-processing systems usually allow multiple transactions to run concurrently. Allowing multiple transactions to update data concurrently causes several complications with consistency of the data.

Ensuring consistency in spite of concurrent execution of transactions requires extra work; it is far easier to insist that transactions run serially—that is, one at a time, each starting only after the previous one has completed.