DBMS 3: Transactions + Indexing

Transactions:

- Transaction is a set of logically related operations. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of a bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

1. Open_Account(X)
2. Old_Balance = X.balance
3. New_Balance = Old_Balance - 800
4. X.balance = New_Balance
5. Close_Account(X)

Y's Account

1. Open_Account(Y)
2. Old_Balance = Y.balance
3. New_Balance = Old_Balance + 800
4. Y.balance = New_Balance
5. Close_Account(Y)

Operations of Transaction:

Following are the main operations of the transaction:

Read(X): Read operation is used to read the value of X from the database and store it in a buffer in the main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit a transaction from an account that consists of the following operations:

1. R(X);
2. X = X - 500;
3. W(X);

Let's assume the value of X before starting the transaction is 4000.

- The first operation reads X's value from the database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 (updating value in buffer, then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

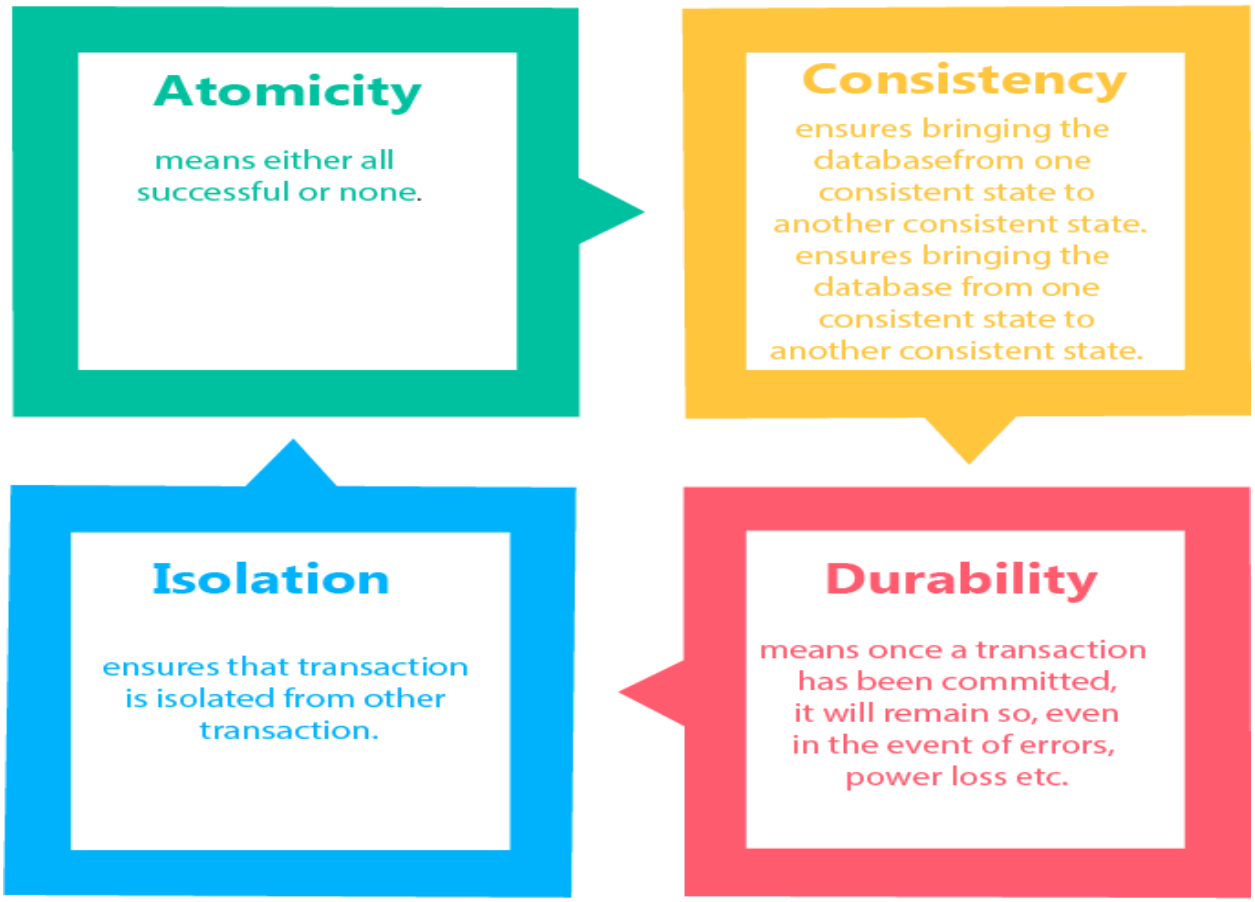
Rollback: It is used to undo the work done.

Transaction Property:

The transaction has four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction (aka ACID Properties)

1. A → Atomicity 2. C → Consistency 3. I → Isolation 4. D → Durability



Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either runs to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that the following transaction **T** consists of **T1** and **T2**. **A** consists of **Rs 600** and **B** consists of **Rs 300**. Transfer **Rs 100** from account **A** to account **B**.

T1	T2
Read(A) A = A - 100 Write(A)	Read(B) B = B + 100 Write(B)

After completion of the transaction, **A** consists of **Rs 500** and **B** consists of **Rs 400**.

If transaction **T** fails after the completion of transaction **T1** but before completion of transaction **T2**, then the amount will be deducted from **A** but not added to **B**. This shows the inconsistent database state. In order to ensure the correctness of the database state, the transaction must be executed in its entirety.

Consistency

- The execution of a transaction should leave a database in either its prior stable state or a new stable state.
- The consistent property of the database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example, The total amount must be maintained before or after the transaction.

1. Total before T occurs = 600 + 300 = 900
2. Total after T occurs = 500 + 400 = 900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

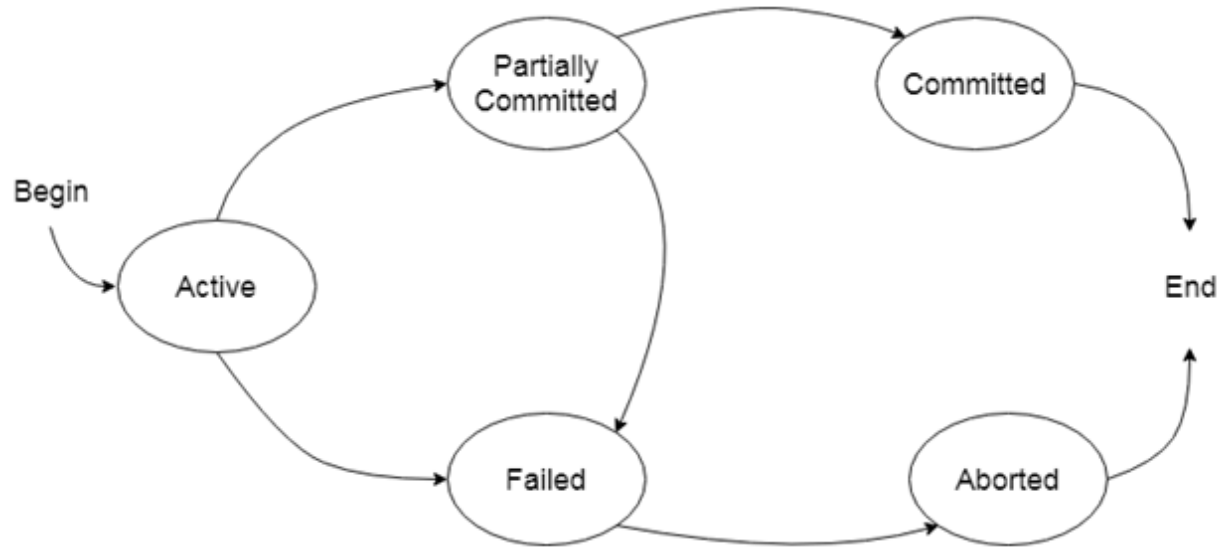
- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction **T1** is being executed and using the data item **X**, then that data item **X** can't be accessed by any other transaction **T2** until the transaction **T1** ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction:

In a database, the transaction can be in one of the following states -



Active State

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: insertion, deletion, or updating a record begins here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total amount calculation example, a final display of the total amount step is executed in this state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total amount calculation, if the database is not able to fire a query to fetch the amount, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to their consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction.
    2. Kill the transaction.

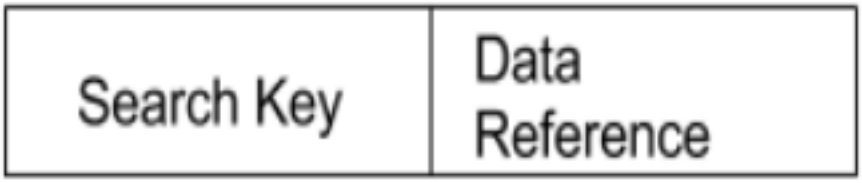
Indexing

Indexing is a way to optimize performance of a database by minimizing the number of disk accesses required when a query is processed.

An index or database index is a data structure which is used to quickly locate and access the data in a database table.

Indexes are created using some database columns.

- The first column is the Search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.
- The second column is the Data Reference which contains a set of pointers holding the address of the disk block where that particular key value can be found.



Structure of an index

B-Tree

1. B-Tree is a self-balancing search tree. In most of the other self-balancing search trees, it is assumed that everything is in the main memory. To understand the use of B-Trees, we must think of the huge amount of data that cannot fit in the main memory.
2. When the number of keys is high, the data is read from the disk in the form of blocks. Disk access time is very high compared to main memory access time.
3. The main idea of using B-Trees is to reduce the number of disk accesses. Most of the tree operations (search, insert, delete, max, min, ..etc) require O(h) disk accesses where h is the height of the tree.
4. The height of B-Trees is kept low by putting the maximum possible keys in a B-Tree node. Generally, a B-Tree node size is kept equal to the disk block size. Since h is low for B-Tree, total disk accesses for most of the operations are reduced significantly compared to balanced Binary Search Trees.

Properties of B-Tree

1) All leaves are at the same level.

2) A B-Tree is defined by the term *minimum degree* ‘t’. The value of t depends upon disk block size.

3) Every node except root must contain at least t-1 keys. The root may contain a minimum of 1 key.

4) All nodes (including root) may contain at most 2t – 1 keys.

5) Number of children of a node is equal to the number of keys in it plus 1.

6) All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in the range from k1 and k2.

7) Like other balanced Binary Search Trees, time complexity to search, insert and delete is O(Logn).