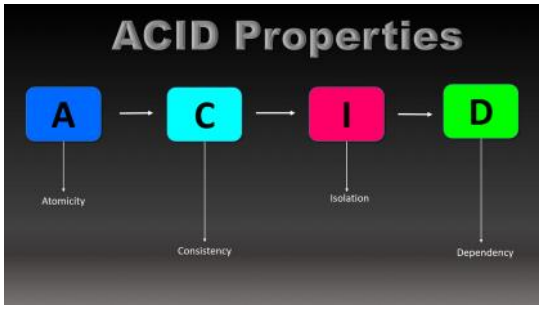
Transaction

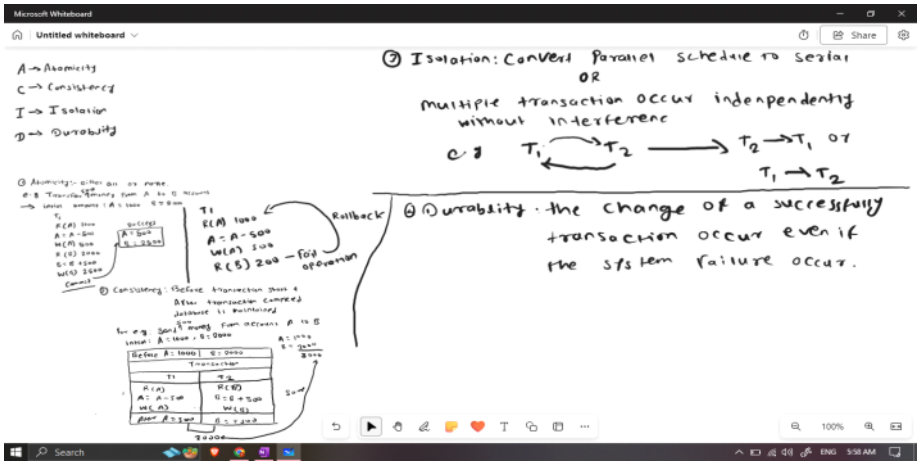
Monday, January 6, 2025 4:51 AM

What is a transaction?

- It is a set of operations used to perform a logical unit of work.
- A transaction generally represents a change in the database.
- Operation : Read,Write
- Example of a transaction to transfer $150 from account A to account B:
  1. read(A)
  2. A:= A – 150
  3. write(A)
  4. read(B)
  5. B:= B + 150
  6. write(B)

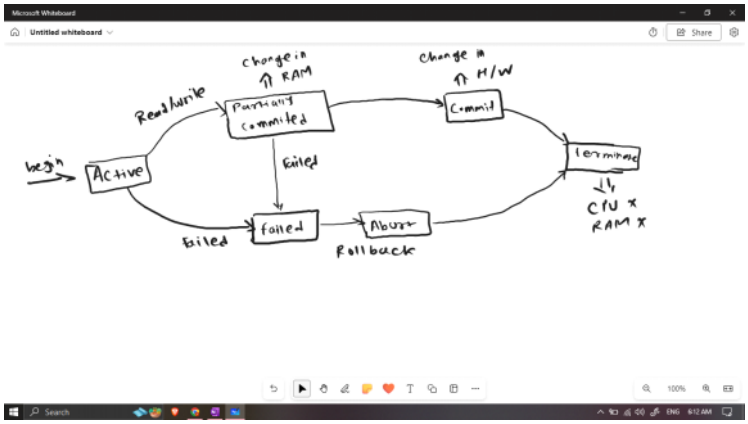
What is ACID property?





Dry Run :
https://wbd.ms/share/v2/aHR0cHM6Ly93aGl0ZWJvYXJkLm1pY3Jvc29mdC5jb20vYXBpL3YxLjAvd2hpdGVib2FyZHMvcmVkZWVtL2Y0ZThhNmZjNDQ2MDRlMzk5NGQ5MTEzZWZlOTU2MGJmX0JCQTcxNzYyLTEyRTAtNDJFMS1CMzI0LTVCMTMxRjQyNEUzRF9hZGFmYzExNC0yYTQxLTQ4ZDMtOTFkNy1mMDhkZGQ2NTczOTk=

Transaction State :-



What is a schedule?

It is a chronological execution sequence of multiple transactions.

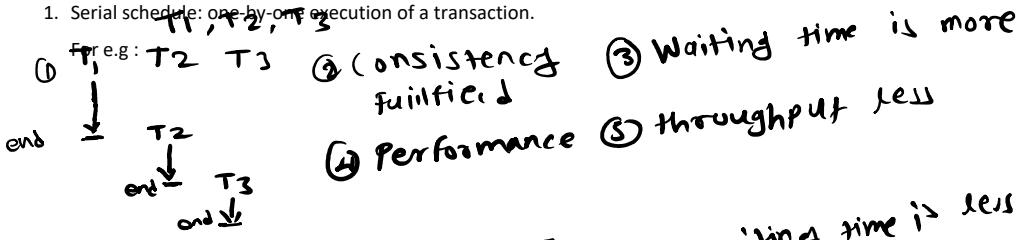
OR

A set of operations from one transaction to another transaction.

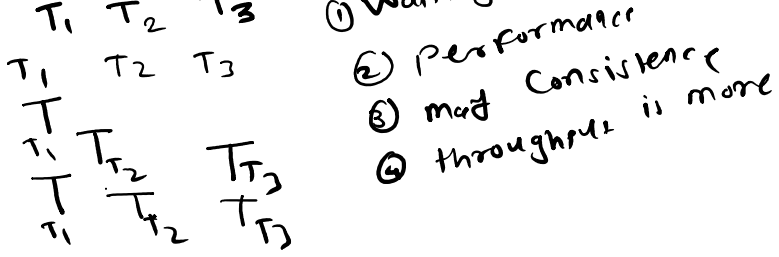
Type of schedule :

1. Serial schedule: one-by-one execution of a transaction.

For e.g : ~~T1,T2,T3~~ T1 T2 T3



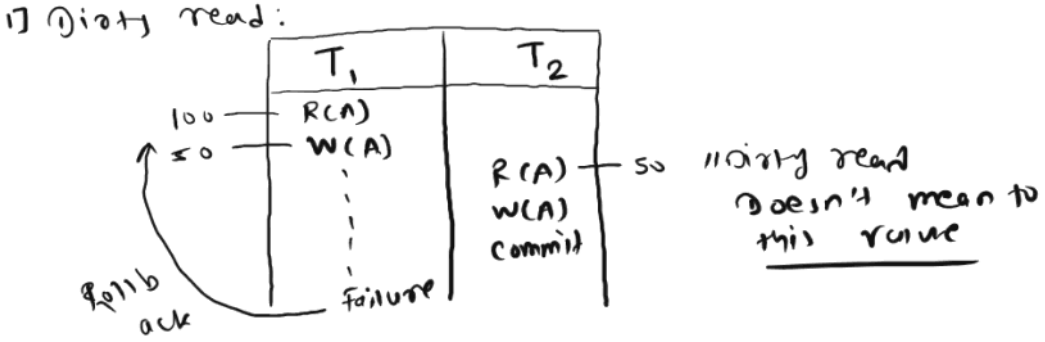
1. Parallel schedule : ~~for~~ e.g T1 T2 T3



Types of Problems in Concurrency:

- Dirty read
- Incorrect summary
- Lost update
- Unrepeatable read
- Phantom read

1. Dirty Read :

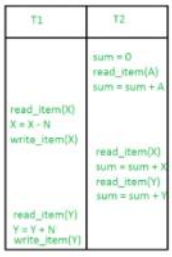


2. Incorrect summary :

Incorrect Summary Problem:

Consider a situation, where one transaction is applying the aggregate function on some records while another transaction is updating these records. The aggregate function may calculate some values before the values have been updated and others after they are updated.

Example:



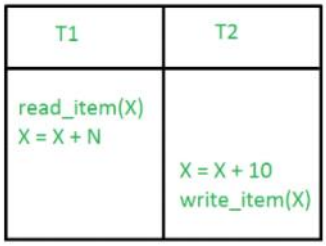
In the above example, transaction 2 is calculating the sum of some records while transaction 1 is updating them. Therefore the aggregate function may calculate some values before they have been updated and others after they have been updated.

3. Lost Update Problem :

Lost Update Problem:

In the lost update problem, an update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

Example:



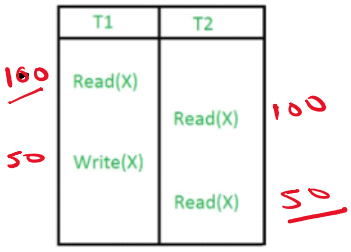
In the above example, transaction 2 changes the value of X but it will get overwritten by the write commit by transaction 1 on X *(not shown in the image above)*. Therefore, the update done by transaction 2 will be lost. Basically, the write commit done by the **last transaction** will overwrite all previous write commits.

4. Unrepeatable Read Problem :

Unrepeatable Read Problem:

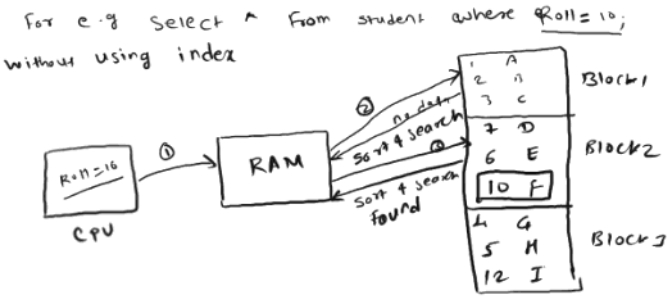
The unrepeatable problem occurs when two or more read operations of the same transaction read different values of the same variable.

Example:

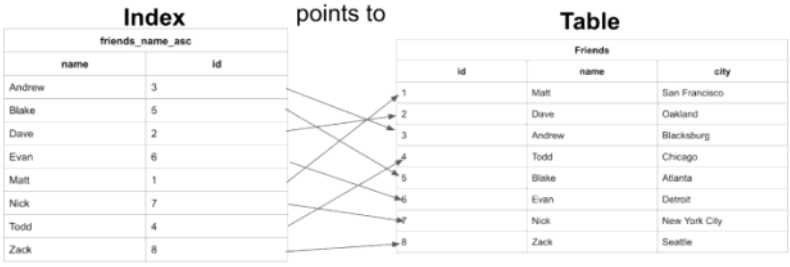


In the above example, once transaction 2 reads the variable X, a write operation in transaction 1 changes the value of the variable X. Thus, when another read operation is performed by transaction 2, it reads the new value of X which was updated by transaction 1.

What is indexing?
Indexing is the way to get an unordered table into an order that will maximize the query's efficiency while searching.



Using Indexing :



Benefits: Reduce I/O cost, Increase performance, Faster join, Searching,Sorting

Types of Indexing?

- Clustered
- Non-clustered

Clustered Indexes

Clustered indexes are the unique index per table that uses the primary key to organize the data that is within the table. The clustered index ensures that the primary key is stored in increasing order, which is also the order the table holds in memory.

- Clustered indexes do not have to be explicitly declared.
- Created when the table is created.
- Use the primary key sorted in ascending order.

Creating clustered Indexes

The clustered index will be automatically created when the primary key is defined:

```
CREATE TABLE friends (id INT PRIMARY KEY, name VARCHAR, city VARCHAR);
```

The created table, "friends", will have a clustered index automatically created, organized around the Primary Key "id" called "friends_pkey":

friends_pkey	Friends		
id	id	name	city
1	1	Matt	San Francisco
2	2	Dave	Oakland
3	3	Andrew	Blacksburg
4	4	Todd	Chicago
5	5	Blake	Atlanta
6	6	Evan	Detroit
7	7	Nick	New York City
8	8	Zack	Seattle

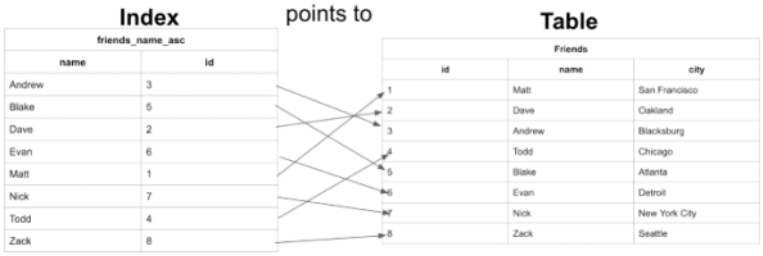
When searching the table by "id", the ascending order of the column allows for optimal searches to be performed. Since the numbers are ordered, the search can navigate the B-tree allowing searches to happen in logarithmic time.

However, in order to search for the "name" or "city" in the table, we would have to look at every entry because these columns do not have an index. This is where non-clustered indexes become very useful.

Non-clustered :

Non-clustered Indexes

Non-clustered indexes are sorted references for a specific field, from the main table, that hold pointers back to the original entries of the table. The first example we showed is an example of a non-clustered table:



They are used to increase the speed of queries on the table by creating columns that are more easily searchable. Non-clustered indexes can be created by data analysts/ developers after a table has been created and filled.

Note: Non-clustered indexes are **not** new tables. Non-clustered indexes hold the field that they are responsible for sorting and a pointer from each of those entries back to the full entry in the table.

Creating non-clustered databases(PostgreSQL)

To create an index to sort our friends' names alphabetically:

```
CREATE INDEX friends_name_asc ON friends(name ASC);
```

- Indexing can vastly reduce the time of queries
- Every table with a primary key has one clustered index
- Every table can have many non-clustered indexes to aid in querying
- Non-clustered indexes hold pointers back to the main table
- Not every database will benefit from indexing
- Not every index will increase the query speed for the database

B-Tree : https://www.scaler.com/topics/sql/b-tree-index/
Index : https://www.geeksforgeeks.org/difference-between-dense-index-and-sparse-index-in-dbms/