Purpose of Normalization

**Normalization** is the process of structuring and handling the relationship between data to minimize redundancy in the relational table and avoid the unnecessary anomalies properties from the database like insertion, update and delete. It helps to divide large database tables into smaller tables and make a relationship between them. It can remove the redundant data and ease to add, manipulate or delete table fields.

A normalization defines rules for the relational table as to whether it satisfies the normal form. A **normal form** is a process that evaluates each relation against defined criteria and removes the multivalued, joins, functional and trivial dependency from a relation. If any data is updated, deleted or inserted, it does not cause any problem for database tables and help to improve the relational table' integrity and efficiency.

Objective of Normalization

1. It is used to remove the duplicate data and database anomalies from the relational table.
2. Normalization helps to reduce redundancy and complexity by examining new data types used in the table.
3. It is helpful to divide the large database table into smaller tables and link them using relationship.
4. It reduces the chances for anomalies to occur in a database.

**Types of Anomalies** :

Following are the types of anomalies that make the table inconsistency, loss of integrity, and redundant data.

**1. Data redundancy** occurs in a relational database when two or more rows or columns have the same value or repetitive value leading to unnecessary utilization of the memory.

**Student Table:**

| **StudRegistration** | **CourseID** | **StudName** | **Address** | **Course** |
| --- | --- | --- | --- | --- |
| **205** | **6204** | **James** | **Los Angeles** | **Economics** |
| **205** | **6247** | **James** | **Los Angeles** | **Economics** |
| **224** | **6247** | **Trent Bolt** | **New York** | **Mathematics** |
| **230** | **6204** | **Ritchie Rich** | **Egypt** | **Computer** |
| **230** | **6208** | **Ritchie Rich** | **Egypt** | **Accounts** |

There are two students in the above table, 'James' and 'Ritchie Rich', whose records are repetitive when we enter a new CourseID. Hence it repeats the studRegistration, StudName and address attributes.

**2. Insert Anomaly:** An insert anomaly occurs in the relational database when some attributes or data items are to be inserted into the database without existence of other attributes. For example, In the Student table, if we want to insert a new courseID, we need to wait until the student enrolled in a course. In this way, it is difficult to insert new record in the table. Hence, it is called insertion anomalies.

**3. Update Anomalies:** The anomaly occurs when duplicate data is updated only in one place and not in all instances. Hence, it makes our data or table inconsistent state. For example, suppose there is a student 'James' who belongs to Student table. If we want to update the course in the Student, we need to update the same in the course table; otherwise, the data can be **inconsistent**. And it reflects the changes in a table with updated values where some of them will not.

**4. Delete Anomalies:** An anomaly occurs in a database table when some records are lost or deleted from the database table due to the deletion of other records. For example, if we want to remove Trent Bolt from the Student table, it also removes his address, course and other details from the Student table. Therefore, we can say that deleting some attributes can remove other attributes of the database table.

So, we need to avoid these types of anomalies from the tables and maintain the integrity, accuracy of the database table. Therefore, we use the normalization concept in the database management system.

**Normalization**

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

* Making relations very large.
* It isn't easy to maintain and update data as it would involve searching many records in relation.
* Wastage and poor utilization of disk space and resources.
* The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is 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.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

**Data modification anomalies can be categorized into three types:**

* **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
* **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
* **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

**Following are the various types of Normal forms:**

| **Normal Form** | **Description** |
| --- | --- |
| 1NF A relation is in 1NF if it contains an atomic value. |
| [2NF](https://www.javatpoint.com/dbms-second-normal-form) | A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key. |
| [3NF](https://www.javatpoint.com/dbms-third-normal-form) | A relation will be in 3NF if it is in 2NF and no transition dependency exists. |
| BCNF | A stronger definition of 3NF is known as Boyce Codd's normal form. |
| [4NF](https://www.javatpoint.com/dbms-forth-normal-form) | A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency. |
| [5NF](https://www.javatpoint.com/dbms-fifth-normal-form) | A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless. |

Advantages of Normalization

* Normalization helps to minimize data redundancy.
* Greater overall database organization.
* Data consistency within the database.
* Much more flexible database design.
* Enforces the concept of relational integrity.

Disadvantages of Normalization

* You cannot start building the database before knowing what the user needs.
* The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
* It is very time-consuming and difficult to normalize relations of a higher degree.
* Careless decomposition may lead to a bad database design, leading to serious problems.

First Normal Form (1NF)

* 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** |

Second Normal Form (2NF)

* 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 |

n 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 |

Third Normal Form (3NF)

* 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 on

**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 |

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.

**Fourth normal form (4NF)**

* 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 |

Fifth normal form (5NF)

* A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
* 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
* 5NF is also known as Project-join normal form (PJ/NF).

Example

| **SUBJECT** | **LECTURER** | **SEMESTER** |
| --- | --- | --- |
| Computer | Anshika | Semester 1 |
| Computer | John | Semester 1 |
| Math | John | Semester 1 |
| Math | Akash | Semester 2 |
| Chemistry | Praveen | Semester 1 |

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

**P1**

| **SEMESTER** | **SUBJECT** |
| --- | --- |
| Semester 1 | Computer |
| Semester 1 | Math |
| Semester 1 | Chemistry |
| Semester 2 | Math |

**P2**

| **SUBJECT** | **LECTURER** |
| --- | --- |
| Computer | Anshika |
| Computer | John |
| Math | John |
| Math | Akash |
| Chemistry | Praveen |

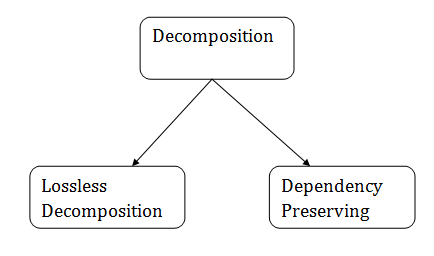
**P3**

| **SEMSTER** | **LECTURER** |
| --- | --- |
| Semester 1 | Anshika |
| Semester 1 | John |
| Semester 1 | John |
| Semester 2 | Akash |
| Semester 1 | Praveen |

Relational Decomposition

* When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
* In a database, it breaks the table into multiple tables.
* If the relation has no proper decomposition, then it may lead to problems like loss of information.
* Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Types of Decomposition



Lossless Decomposition

* If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
* The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
* The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.

**Example:**

**EMPLOYEE\_DEPARTMENT table:**

| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** | **DEPT\_ID** | **DEPT\_NAME** |
| --- | --- | --- | --- | --- | --- |
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |
| 60 | Jack | 40 | Noida | 678 | Testing |

The above relation is decomposed into two relations EMPLOYEE and DEPARTMENT

**EMPLOYEE table:**

| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** |
| --- | --- | --- | --- |
| 22 | Denim | 28 | Mumbai |
| 33 | Alina | 25 | Delhi |
| 46 | Stephan | 30 | Bangalore |
| 52 | Katherine | 36 | Mumbai |
| 60 | Jack | 40 | Noida |

**DEPARTMENT table**

| **DEPT\_ID** | **EMP\_ID** | **DEPT\_NAME** |
| --- | --- | --- |
| 827 | 22 | Sales |
| 438 | 33 | Marketing |
| 869 | 46 | Finance |
| 575 | 52 | Production |
| 678 | 60 | Testing |

Now, when these two relations are joined on the common column "EMP\_ID", then the resultant relation will look like:

**Employee ⋈ Department**

| **EMP\_ID** | **EMP\_NAME** | **EMP\_AGE** | **EMP\_CITY** | **DEPT\_ID** | **DEPT\_NAME** |
| --- | --- | --- | --- | --- | --- |
| 22 | Denim | 28 | Mumbai | 827 | Sales |
| 33 | Alina | 25 | Delhi | 438 | Marketing |
| 46 | Stephan | 30 | Bangalore | 869 | Finance |
| 52 | Katherine | 36 | Mumbai | 575 | Production |
| 60 | Jack | 40 | Noida | 678 | Testing |

Hence, the decomposition is Lossless join decomposition.

Dependency Preserving

* It is an important constraint of the database.
* In the dependency preservation, at least one decomposed table must satisfy every dependency.
* If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.
* For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

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".

UNIT V

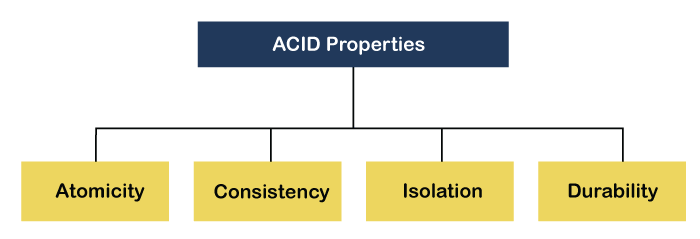
ACID Properties in DBMS

DBMS is the management of data that should remain integrated when any changes are done in it. It is because if the integrity of the data is affected, whole data will get disturbed and corrupted. Therefore, to maintain the integrity of the data, there are four properties described in the database management system, which are known as the **ACID** properties. The ACID properties are meant for the transaction that goes through a different group of tasks, and there we come to see the role of the ACID properties.

In this section, we will learn and understand about the ACID properties. We will learn what these properties stand for and what does each property is used for. We will also understand the ACID properties with the help of some examples.

ACID Properties

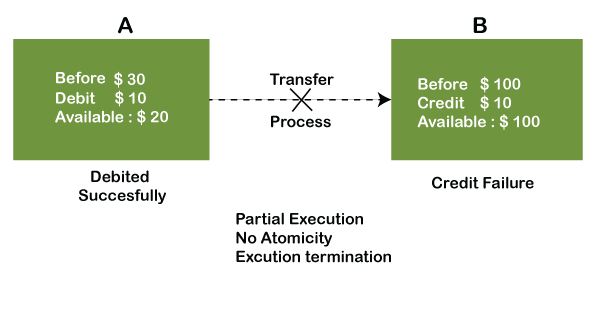
The expansion of the term ACID defines for:



1) Atomicity

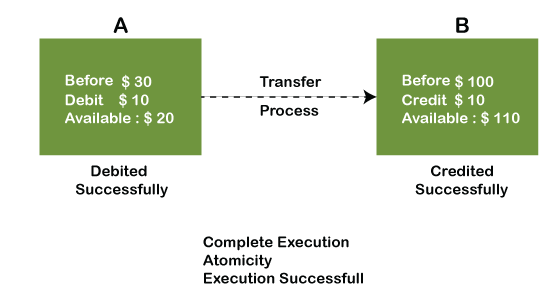
The term atomicity defines that the data remains atomic. It means if any operation is performed on the data, either it should be performed or executed completely or should not be executed at all. It further means that the operation should not break in between or execute partially. In the case of executing operations on the transaction, the operation should be completely executed and not partially.

**Example:** If Remo has account A having $30 in his account from which he wishes to send $10 to Sheero's account, which is B. In account B, a sum of $ 100 is already present. When $10 will be transferred to account B, the sum will become $110. Now, there will be two operations that will take place. One is the amount of $10 that Remo wants to transfer will be debited from his account A, and the same amount will get credited to account B, i.e., into Sheero's account. Now, what happens - the first operation of debit executes successfully, but the credit operation, however, fails. Thus, in Remo's account A, the value becomes $20, and to that of Sheero's account, it remains $100 as it was previously present.



In the above diagram, it can be seen that after crediting $10, the amount is still $100 in account B. So, it is not an atomic transaction.

The below image shows that both debit and credit operations are done successfully. Thus the transaction is atomic.

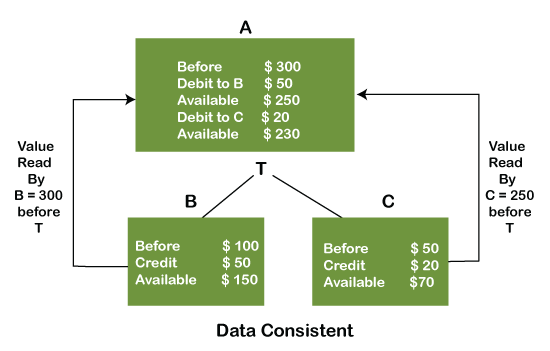


Thus, when the amount loses atomicity, then in the bank systems, this becomes a huge issue, and so the atomicity is the main focus in the bank systems.

2) Consistency

The word **consistency** means that the value should remain preserved always. In [DBMS](https://www.javatpoint.com/dbms-tutorial), the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

**Example:**

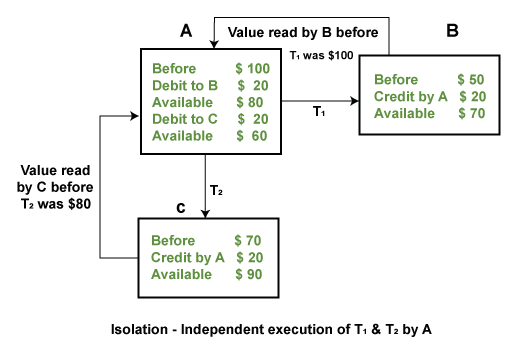


In the above figure, there are three accounts, A, B, and C, where A is making a transaction T one by one to both B & C. There are two operations that take place, i.e., Debit and Credit. Account A firstly debits $50 to account B, and the amount in account A is read $300 by B before the transaction. After the successful transaction T, the available amount in B becomes $150. Now, A debits $20 to account C, and that time, the value read by C is $250 (that is correct as a debit of $50 has been successfully done to B). The debit and credit operation from account A to C has been done successfully. We can see that the transaction is done successfully, and the value is also read correctly. Thus, the data is consistent. In case the value read by B and C is $300, which means that data is inconsistent because when the debit operation executes, it will not be consistent.

3) Isolation

The term 'isolation' means separation. In DBMS, Isolation is the property of a database where no data should affect the other one and may occur concurrently. In short, the operation on one database should begin when the operation on the first database gets complete. It means if two operations are being performed on two different databases, they may not affect the value of one another. In the case of transactions, when two or more transactions occur simultaneously, the consistency should remain maintained. Any changes that occur in any particular transaction will not be seen by other transactions until the change is not committed in the memory.

**Example:** If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



4) Durability

Durability ensures the permanency of something. In DBMS, the term durability ensures that the data after the successful execution of the operation becomes permanent in the database. The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives. However, if gets lost, it becomes the responsibility of the recovery manager for ensuring the durability of the database. For committing the values, the COMMIT command must be used every time we make changes.

Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Thus, it was a precise introduction of ACID properties in DBMS. We have discussed these properties in the transaction section also.

**Concurrent Executions in DBMS**

Concurrent execution refers to the **simultaneous execution** of more than one transaction. This is a common scenario in multi-user database environments where many users or applications might be accessing or modifying the database at the same time. Concurrent execution is crucial for achieving high throughput and efficient resource utilization. However, it introduces the potential for conflicts and data inconsistencies.

**Advantages of Concurrent Execution**

1. **Increased System Throughput:** Multiple transactions can be in progress at the same time, but at different stages
2. **Maximized Processor Utilization:** If one transaction is waiting for I/O operations, another transaction can utilize the processor.
3. **Decreased Wait Time:** Transactions no longer have to wait for other long transactions to complete.
4. **Improved Transaction Response Time:** Transactions get processed faster because they can be executed in parallel.
5. **Potential Problems with Concurrent Execution**

# 1. Lost Update Problem (Write-Write conflict):

One transaction's updates could be overwritten by another.

**Examples:**

T1 | T2

----------|-----------

Read(A) |

A = A+50 |

| Read(A)

| A = A+100

Write(A) |

| Write(A)

**Result:** T1's updates are lost.

**2.Temporary Inconsistency or Dirty Read Problem (Write-Read conflict):**

One transaction might read an inconsistent state of data that's being updated by another.

**Examples:**

T1 | T2

------------------|-----------

Read(A) |

A = A+50 |

Write(A) |

| Read(A)

| A = A+100

| Write(A)

Read(A)(rollbacks)|

| commit

**Result:** T2 has a "dirty" value, that was never committed in T1 and doesn't actually exist in the database.

**3. Unrepeatable Read Problem (Read-Write conflict):**

when a single transaction reads the same row multiple times and observes different values each time. This occurs because another concurrent transaction has modified the row between the two reads.

**Examples:**

T1 | T2

----------|----------

Read(A) |

| Read(A)

| A = A+100

| Write(A)

Read(A) |

**Result:** Within the same transaction, T1 has read two different values for the same data item. This inconsistency is the unrepeatable read.

Serializability in DBMS

The backbone of the most modern application is the form of DBMS. When we design the form properly, then it provides high-performance and relative storage solutions to our application. In this topic, we are going to explain the serializability concept and how this concept affects the DBMS deeply. We also understand the concept of serializability with some examples. Finally, we will conclude this topic with an example of the importance of serializability.

What is Serializability in DBMS?

In the field of computer science, serializability is a term that is a property of the system that describes how the different process operates the shared data. If the result given by the system is similar to the operation performed by the system, then in this situation, we call that system serializable. Here the cooperation of the system means there is no overlapping in the execution of the data. In DBMS, when the data is being written or read then, the DBMS can stop all the other processes from accessing the data.

What is a Serializable Schedule?

* In DBMS, the Serializable schedule is a property in which the read and write operation sequence does not disturb the serializability property. This property ensures that the transaction is executed automatically with the other transaction. In DBMS, the order of the serializability must be the same as some serial schedules of the same transaction.
* In DBMS, there are several algorithms available to check the serializability of the database. One of the most important algorithms is the conflict serializability algorithm. The conflict serializability algorithm is the ability to check the potential of the conflict in the database. When the two transactions access the same data, this conflict occurs in the database. If there is no conflict, then there is guaranteed serializability in the database. However, if there is a conflict occurs, then there is a chance of serializability.

**What is Concurrency Control?**

**Concurrency Control** in Database Management System is a procedure of managing simultaneous operations without conflicting with each other. It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.

Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical Database, it would have a mix of READ and WRITE operations and hence the concurrency is a challenge.

DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system. Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

**Potential problems of Concurrency**

Here, are some issues which you will likely to face while using the DBMS Concurrency Control method:

* **Lost Updates** occur when multiple transactions select the same row and update the row based on the value selected
* Uncommitted dependency issues occur when the second transaction selects a row which is updated by another transaction (**dirty read**)
* **Non-Repeatable Read** occurs when a second transaction is trying to access the same row several times and reads different data each time.
* **Incorrect Summary issue**occurs when one transaction takes summary over the value of all the instances of a repeated data-item, and second transaction update few instances of that specific data-item. In that situation, the resulting summary does not reflect a correct result.

**Why use Concurrency method?**

Reasons for using Concurrency control method is DBMS:

* To apply Isolation through mutual exclusion between conflicting transactions
* To resolve read-write and write-write conflict issues
* To preserve database consistency through constantly preserving execution obstructions
* The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
* Concurrency control helps to ensure serializability

**Example**

Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time.

However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control in DBMS, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

**Concurrency Control Protocols**

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose. Following are the Concurrency Control techniques in DBMS:

* Lock-Based Protocols
* Two Phase Locking Protocol
* Timestamp-Based Protocols
* Validation-Based Protocols

**What is Concurrency Control?**

**Concurrency Control** in Database Management System is a procedure of managing simultaneous operations without conflicting with each other. It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.

Concurrent access is quite easy if all users are just reading data. There is no way they can interfere with one another. Though for any practical Database, it would have a mix of READ and WRITE operations and hence the concurrency is a challenge.

DBMS Concurrency Control is used to address such conflicts, which mostly occur with a multi-user system. Therefore, Concurrency Control is the most important element for proper functioning of a Database Management System where two or more database transactions are executed simultaneously, which require access to the same data.

**Potential problems of Concurrency**

Here, are some issues which you will likely to face while using the DBMS Concurrency Control method:

* **Lost Updates** occur when multiple transactions select the same row and update the row based on the value selected
* Uncommitted dependency issues occur when the second transaction selects a row which is updated by another transaction (**dirty read**)
* **Non-Repeatable Read** occurs when a second transaction is trying to access the same row several times and reads different data each time.
* **Incorrect Summary issue**occurs when one transaction takes summary over the value of all the instances of a repeated data-item, and second transaction update few instances of that specific data-item. In that situation, the resulting summary does not reflect a correct result.

**Why use Concurrency method?**

Reasons for using Concurrency control method is DBMS:

* To apply Isolation through mutual exclusion between conflicting transactions
* To resolve read-write and write-write conflict issues
* To preserve database consistency through constantly preserving execution obstructions
* The system needs to control the interaction among the concurrent transactions. This control is achieved using concurrent-control schemes.
* Concurrency control helps to ensure serializability

**Example**

Assume that two people who go to electronic kiosks at the same time to buy a movie ticket for the same movie and the same show time.

However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control in DBMS, it is possible that both moviegoers will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Both moviegoers can still access information written in the movie seating database. But concurrency control only provides a ticket to the buyer who has completed the transaction process first.

**Concurrency Control Protocols**

Different concurrency control protocols offer different benefits between the amount of concurrency they allow and the amount of overhead that they impose. Following are the Concurrency Control techniques in DBMS:

* Lock-Based Protocols
* Two Phase Locking Protocol
* Timestamp-Based Protocols
* Validation-Based Protocols

**Lock-based Protocols**

**Lock Based Protocols** in DBMS is a mechanism in which a transaction cannot Read or Write the data until it acquires an appropriate lock. Lock based protocols help to eliminate the concurrency problem in DBMS for simultaneous transactions by locking or isolating a particular transaction to a single user.

A lock is a data variable which is associated with a data item. This lock signifies that operations that can be performed on the data item. Locks in DBMS help synchronize access to the database items by concurrent transactions.

All lock requests are made to the concurrency-control manager. Transactions proceed only once the lock request is granted.

**Binary Locks:**A Binary lock on a data item can either locked or unlocked states.

**Shared/exclusive:** This type of locking mechanism separates the locks in DBMS based on their uses. If a lock is acquired on a data item to perform a write operation, it is called an exclusive lock.

**1. Shared Lock (S):**

A shared lock is also called a Read-only lock. With the shared lock, the data item can be shared between transactions. This is because you will never have permission to update data on the data item.

For example, consider a case where two transactions are reading the account balance of a person. The [database](https://www.guru99.com/introduction-to-database-sql.html) will let them read by placing a shared lock. However, if another transaction wants to update that account’s balance, shared lock prevent it until the reading process is over.

**2. Exclusive Lock (X):**

With the Exclusive Lock, a data item can be read as well as written. This is exclusive and can’t be held concurrently on the same data item. X-lock is requested using lock-x instruction. Transactions may unlock the data item after finishing the ‘write’ operation.

For example, when a transaction needs to update the account balance of a person. You can allows this transaction by placing X lock on it. Therefore, when the second transaction wants to read or write, exclusive lock prevent this operation.

**3. Simplistic Lock Protocol**

This type of lock-based protocols allows transactions to obtain a lock on every object before beginning operation. Transactions may unlock the data item after finishing the ‘write’ operation.

**4. Pre-claiming Locking**

Pre-claiming lock protocol helps to evaluate operations and create a list of required data items which are needed to initiate an execution process. In the situation when all locks are granted, the transaction executes. After that, all locks release when all of its operations are over.

**Starvation**

Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.

Following are the reasons for Starvation:

* When waiting scheme for locked items is not properly managed
* In the case of resource leak
* The same transaction is selected as a victim repeatedly

**Deadlock**

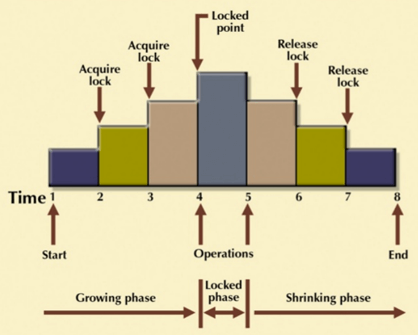
Deadlock refers to a specific situation where two or more processes are waiting for each other to release a resource or more than two processes are waiting for the resource in a circular chain.

**Two Phase Locking Protocol**

**Two Phase Locking Protocol** also known as 2PL protocol is a method of concurrency control in DBMS that ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously. Two Phase Locking protocol helps to eliminate the concurrency problem in DBMS.

This locking protocol divides the execution phase of a transaction into three different parts.

* In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
* The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
* In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.



The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:

* **Growing Phase**: In this phase transaction may obtain locks but may not release any locks.
* **Shrinking Phase**: In this phase, a transaction may release locks but not obtain any new lock

It is true that the 2PL protocol offers serializability. However, it does not ensure that deadlocks do not happen.

In the above-given diagram, you can see that local and global deadlock detectors are searching for deadlocks and solve them with resuming transactions to their initial states.

**Strict Two-Phase Locking Method**

Strict-Two phase locking system is almost similar to 2PL. The only difference is that Strict-2PL never releases a lock after using it. It holds all the locks until the commit point and releases all the locks at one go when the process is over.

**Centralized 2PL**

In Centralized 2 PL, a single site is responsible for lock management process. It has only one lock manager for the entire DBMS.

**Primary copy 2PL**

Primary copy 2PL mechanism, many lock managers are distributed to different sites. After that, a particular lock manager is responsible for managing the lock for a set of data items. When the primary copy has been updated, the change is propagated to the slaves.

**Distributed 2PL**

In this kind of two-phase locking mechanism, Lock managers are distributed to all sites. They are responsible for managing locks for data at that site. If no data is replicated, it is equivalent to primary copy 2PL. Communication costs of Distributed 2PL are quite higher than primary copy 2PL

**Timestamp-based Protocols**

**Timestamp based Protocol** in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions. The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order.

The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. 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. Timestamp-based protocols manage conflicts as soon as an operation is created.

EXAMPLE:

Suppose there are there transactions T1, T2, and T3.

T1 has entered the system at time 0010

T2 has entered the system at 0020

T3 has entered the system at 0030

Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.