## Anomalies in DBMS

Anomaly means inconsistency in the pattern from the normal form. In Database Management System (DBMS), anomaly means the inconsistency occurred in the relational table during the operations performed on the relational table.

There can be various reasons for anomalies to occur in the database. For example, if there is a lot of redundant data present in our database then DBMS anomalies can occur. If a table is constructed in a very poor manner then there is a chance of database anomaly. Due to database anomalies, the integrity of the database suffers.

The other reason for the database anomalies is that all the data is stored in a single table. So, to remove the anomalies of the database, normalization is the process which is done where the splitting of the table and joining of the table (different types of join) occurs.

We will see the anomalies present in a table by the different examples:

Example 1:

| Worker_id | Worker_name | Worker_dept | Worker_address |
|---|---|---|---|
| 65 | Ramesh | ECT001 | Jaipur |
| 65 | Ramesh | ECT002 | Jaipur |
| 73 | Amit | ECT002 | Delhi |
| 76 | Vikas | ECT501 | Pune |
| 76 | Vikas | ECT502 | Pune |
| 79 | Rajesh | ECT669 | Mumbai |

In the above table, we have four columns which describe the details about the workers like their name, address, department and their id. The above table is not normalized, and there is definitely a chance of anomalies present in the table.

**There can be three types of an anomaly in the database:**

**Updation / Update Anomaly**

When we update some rows in the table, and if it leads to the inconsistency of the table then this anomaly occurs. This type of anomaly is known as an updation anomaly. In the above table, if we want to update the address of Ramesh then we will have to update all the rows where Ramesh is present. If during the update we miss any single row, then there will be two addresses of Ramesh, which will lead to inconsistent and wrong databases.

**Insertion Anomaly**

If there is a new row inserted in the table and it creates the inconsistency in the table then it is called the insertion anomaly. For example, if in the above table, we create a new row of a worker, and if it is not allocated to any department then we cannot insert it in the table so, it will create an insertion anomaly.

**Deletion Anomaly**

If we delete some rows from the table and if any other information or data which is required is

also deleted from the database, this is called the deletion anomaly in the database. For example,

in the above table, if we want to delete the department number ECT669 then the details of Rajesh will also be deleted since Rajesh's details are dependent on the row of ECT669. So, there will be deletion anomalies in the table.

To remove this type of anomalies, we will normalize the table or split the table or join the tables. There can be various normalized forms of a table like 1NF, 2NF, 3NF, BCNF etc. we will apply the different normalization schemes according to the current form of the table.

Example 2:

| Stu_id | Stu_name | Stu_branch | Stu_club |
|---|---|---|---|
| 2018nk01 | Shivani | Computer science | literature |
| 2018nk01 | Shivani | Computer science | dancing |
| 2018nk02 | Ayush | Electronics | Videography |
| 2018nk03 | Mansi | Electrical | dancing |
| 2018nk03 | Mansi | Electrical | singing |
| 2018nk04 | Gopal | Mechanical | Photography |

In the above table, we have listed students with their name, id, branch and their respective clubs.

**Updation / Update Anomaly**

In the above table, if Shivani changes her branch from Computer Science to Electronics, then we will have to update all the rows. If we miss any row, then Shivani will have more than one branch, which will create the update anomaly in the table.

**Insertion Anomaly**

If we add a new row for student Ankit who is not a part of any club, we cannot insert the row into the table as we cannot insert null in the column of stu_club. This is called insertion anomaly.

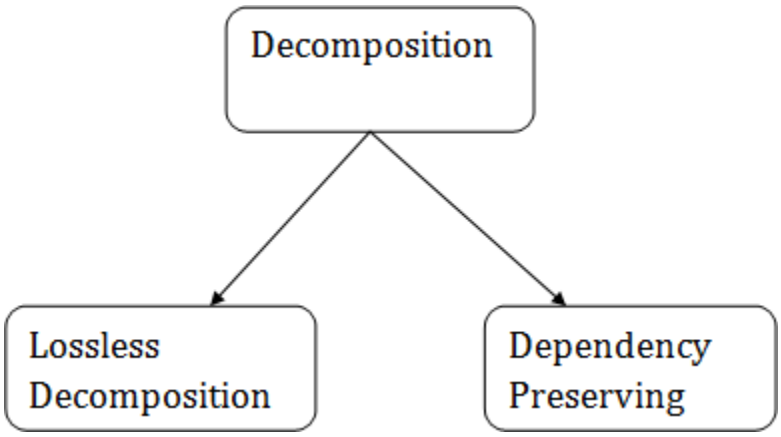
**Deletion Anomaly**

If we remove the photography club from the college, then we will have to delete its row from the table. But it will also delete the table of Gopal and his details. So, this is called deletion anomaly and it will make the database inconsistent.

## 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.

## 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.

1. X → Y

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.

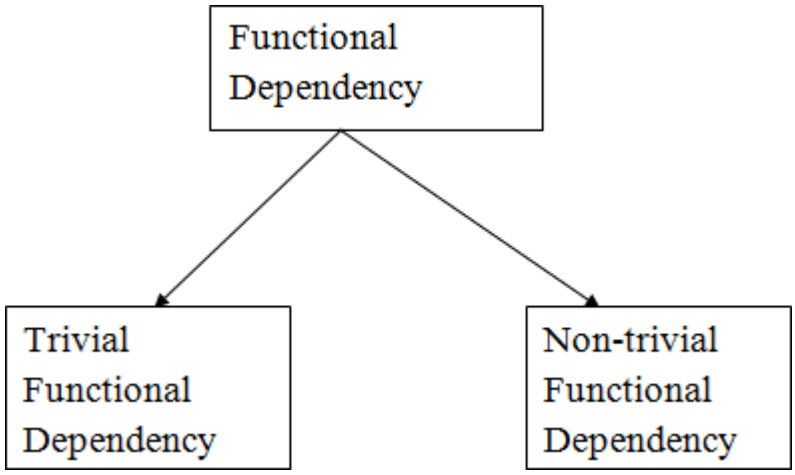
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

## Types of Functional dependency




1. 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

**Example:**

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
3. Employee_Id is a subset of {Employee_Id, Employee_Name}.
4. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial de pendencies too.

2. 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.

**Example:**

1. ID → Name,
2. Name → DOB

**Inference Rule (IR):**

- The Armstrong's axioms are the basic inference rule.
- Armstrong's axioms are used to conclude functional dependencies on a relational database.
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.
- The Functional dependency has 6 types of inference rule:
  1. Reflexive Rule ($IR_1$)
- In the reflexive rule, if Y is a subset of X, then X determines Y.

1. If X ⊇ Y then X → Y

**Example:**

1. X = {a, b, c, d, e}
2. Y = {a, b, c}

2. Augmentation Rule ($IR_2$)

The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.

1. If X → Y then XZ → YZ

**Example:**

1. For R(ABCD), **if** A → B then AC → BC

3. Transitive Rule ($IR_3$)

In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.

1. If X → Y and Y → Z then X → Z

4. Union Rule ($IR_4$)

Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.

1. If X → Y and X → Z then X → YZ

1. X → Y (given)
2. X → Z (given)
3. X → XY (using $IR_2$ on 1 by augmentation with X. Where XX = X)
4. XY → YZ (using $IR_2$ on 2 by augmentation with Y)
5. X → YZ (using $IR_3$ on 3 and 4)

5. Decomposition Rule ($IR_5$)

Decomposition rule is also known as project rule. It is the reverse of union rule.
This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

1. If X → YZ then X → Y and X → Z
   1. X → YZ (given)
   2. YZ → Y (using $IR_1$ Rule)
   3. X → Y (using $IR_3$ on 1 and 2)

6. Pseudo transitive Rule ($IR_6$)

In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.

1. If X → Y and YZ → W then XZ → W

   1. X → Y (given)
   2. WY → Z (given)
   3. WX → WY (using $IR_2$ on 1 by augmenting with W)
   4. WX → Z (using $IR_3$ on 3 and 2)

**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.

| Normal Form | Description |
|---|---|
| 1NF | A relation is in 1NF if it contains an atomic value. |
| 2NF | A relation will be in 2NF if it is in 1NF and all non-key attributes arefully functional dependent on the primary key. |
| 3NF | 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 | A relation will be in 4NF if it is in Boyce Codd's normal form andhas no multi-valued dependency. |
| 5NF | 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 |

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 | Compute |

**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.
3. **Example:**
4. **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.