**What is Redundancy in DBMS?**

Redundancy in DBMS is the problem that arises when the database is not normalized. It is the concept of storing multiple copies of the same data in different parts of the database.

**Example for Redundancy in DBMS?**

Let's understand the concept of redundancy in DBMS with a simple student table.

| **student\_id** | **student\_name** | **student\_age** | **dept\_id** | **dept\_name** | **dept\_head** |
| --- | --- | --- | --- | --- | --- |
| 1 | Tony Stark | 18 | 100 | Computer Science | Steve Rogers |
| 2 | Thor Odinson | 18 | 100 | Computer Science | Steve Rogers |

In this student table, we have repeated the same department details, dept\_id, dept\_name, and dept\_head in every student record. This causes redundancy in the student table.

**How does Data Redundancy Occur?**

Data redundancy in DBMS may occur due to any of the below reasons.

1. The database is not normalized through DBMS [normalization](https://www.scaler.com/topics/dbms/normalization-in-dbms/).
2. The same data is stored in multiple places by multiple systems causing redundancy in data.
3. Mistake during database design causes the same data to be stored multiple times.

**Problems caused by redundancy in Database**

Redundancy in DMBS can cause several problems while performing operations on data such as insert, delete, and update. Let's use the below student table to understand **insertion**, **updation**, and **deletion** anomalies.

| **student\_id** | **student\_name** | **student\_age** | **dept\_id** | **dept\_name** | **dept\_head** |
| --- | --- | --- | --- | --- | --- |
| 1 | Tony Stark | 18 | 100 | Computer Science | Steve Rogers |
| 2 | Thor Odinson | 18 | 100 | Computer Science | Steve Rogers |
| 3 | Bruce Banner | 18 | 101 | Mechanical | Natasha Romanoff |

**Insertion Anomaly**

An insertion anomaly occurs when specific details cannot be inserted into the database without the other details.

**Example:** Without knowing the department details, we cannot insert the student details in the above table. The student details (student\_id, student\_name, and student\_age) depends on the department details (dept\_id, dept\_name, and dept\_head).

**Deletion Anomaly**

Deletion anomaly occurs when deleting specific details loses some unrelated information from the database.

**Example:** If we delete the student with student\_id **3** from the above student table, we also lose the department details with dept\_id **101**. Deleting student details result in losing unrelated department details.

**Updation Anomaly**

Updation anomaly occurs when there is data inconsistency resulting from a partial data update.

**Example:** We wanted to update the dept\_head to **Peter Parker** for dept\_id **101**; we need to update it in all places. If the update didn't occur in all the places (partial update), it may result in data inconsistency.

**How To Avoid Redundancy in DBMS?**

Redundancy in DBMS can be avoided by following the below approaches.

* Redundancy in DBMS can be avoided by normalizing the data through database [normalization](https://www.scaler.com/topics/dbms/normalization-in-dbms/).
* Redundancy can be avoided using Master Data. Master data is a single source of data accessed by several applications and systems.
* Proper database architecture design can avoid data redundancy.

**Advantages of Data Redundancy**

* Data redundancy can help disaster recovery by backing up the data in a different place.
* Data redundancy can help during malicious attacks. Data integrity can be verified if we have multiple copies of the same data.

**Disadvantages of Data Redundancy**

* Data redundancy can cause an increase in storage space due to duplicate data which may increase the cost of the data storage.
* Data redundancy increase the size of the database, which increases the complexity of performing operations on the data.
* Data redundancy can cause inconsistency in data due to partial updates to the database.

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.

Play Videox[[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack" \t "_blank)

**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](https://www.javatpoint.com/dbms-first-normal-form) | 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 is said to be in **1NF (first normal form)**, if it doesn’t contain any multi-valued attribute. In other words you can say that a relation is in 1NF if each attribute contains only atomic(single) value only.
* As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.
* **Example**: Let’s say a company wants to store the names and contact details of its employees. It creates a table in the database that looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| Emp\_Id | Emp\_Name | Emp\_Address | Emp\_Mobile |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 , 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123, 8123450987 |

* Two employees (Jon & Lester) have two mobile numbers that caused the Emp\_Mobile field to have multiple values for these two employees.
* This table is **not in 1NF**as the rule says “each attribute of a table must have atomic (single) values”, the Emp\_Mobile values for employees Jon & Lester violates that rule.
* To make the table complies with 1NF we need to create separate rows for the each mobile number in such a way so that none of the attributes contains multiple values.

|  |  |  |  |
| --- | --- | --- | --- |
| Emp\_Id | Emp\_Name | Emp\_Address | Emp\_Mobile |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 |
| 102 | Jon | Kanpur | 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123 |
| 104 | Lester | Bangalore | 8123450987 |