

Views and normalisation

Normalisation

Normalisation

Normalisation is a **database design technique** used to **organise data in a relational database**. It reduces data redundancy and ensures data integrity.

Normalisation involves breaking down a large, complex table into smaller, related tables and establishing relationships between them.

The process is guided by a set of rules and principles, often described using **normal forms**.

The most common normal forms are:

1. **First Normal Form (1NF)**
2. **Second Normal Form (2NF)**
3. **Third Normal Form (3NF)**

and so on, up to higher levels of normalisation.



Normalisation

Advantages



- 01. It helps maintain **data integrity** by reducing data redundancy and preventing anomalies from operations like *insertion*.
- 02. Normalisation reduces data redundancy, which leads to more **efficient storage utilisation**.
- 03. It's **easier to maintain and update** because we don't need to update multiple places to change a single piece of data.
- 04. Normalised databases offer **scalability** with efficient indexing and data integrity, supporting larger datasets.
- 05. Normalised databases excel in **flexible** querying, supporting complex queries through table relationships.



Disadvantages

- 01. Normalised databases may be **complex to design and query** due to multiple table joins, potentially affecting performance.
- 02. Query **performance may be slow**, especially with complex joins and large datasets, resulting in long execution times.
- 03. While normalised databases are great for reads, they **may struggle with writes** due to relationship overhead.
- 04. Normalisation reduces redundancy but **adds storage overhead** due to primary and foreign key maintenance.
- 05. Higher normal forms (e.g. 4NF, 5NF) can be **challenging and unnecessary**, potentially resulting in overly complex designs.

Denormalisation

Denormalisation, on the other hand, is a **database design technique** that intentionally **introduces redundancy into a database** by combining tables or adding redundant data to one or more tables.

Advantages



01. Redundant data offers **faster query performance** for complex queries and reporting due to reduced join complexity.
02. It **simplifies schema structures**, aiding query development and maintenance, ideal for reporting and analytics.
03. Denormalisation **suits read-heavy scenarios**, prioritising faster queries over write complexities.
04. Fewer joins in denormalised databases means simpler SQL queries and **reduced risk of performance issues**.



Disadvantages

01. **Write operations**, such as inserts, updates, and deletes, **can be slower and more complex** due to redundancy.
02. **Data integrity is challenging** because ensuring consistent updates to redundant data is complex and error-prone.
03. Redundant data **consumes more storage space**, which can be a significant concern in systems with large datasets.
04. As the database complexity increases, **managing schema changes becomes increasingly challenging**.

Data anomalies

Data anomalies are **unexpected occurrences** in a dataset that can erode *data accuracy*, *reliability*, and *usability*, eventually undermining data quality for decision-making, analysis, and reporting.

Causes

Can arise from a variety of causes, including:


1. **Data entry errors:** Human-made mistakes like typos or missing data.
2. **Data storage inconsistencies:** Differences in how data are stored.
3. **Integration issues:** Merging data from diverse sources with format conflicts.
4. **System glitches:** Technical problems in data processing.


Prevention and mitigation


Strategies include:


1. **Prevention:**
 - a. Normalise data.
 - b. Apply validation rules.
 - c. Implement data entry controls.
2. **Mitigation:**
 - a. Perform data cleansing.
 - b. Conduct data audits.
 - c. Establish error-handling protocols.


Managing data anomalies


 **Data integrity:** Anomalies can compromise data integrity by violating database rules and constraints.

 **Data cleaning:** Detecting and rectifying anomalies involves data cleaning to correct errors and inconsistencies.

 **Data integration:** Integrating data from multiple sources can increase data anomalies.

 **Anomalies vs outliers:** Outliers fall outside the expected range but, unlike anomalies, might not imply errors.

 **Data governance:** Vital for preventing, detecting, and addressing data anomalies to uphold data quality.

 **Continuous monitoring:** Continuous monitoring and auditing are essential for maintaining anomaly-free data.

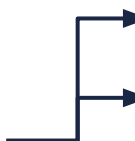
Data anomalies – Update anomaly

An update anomaly occurs in a database when **updating a piece of information requires modifying multiple rows or records in a table**, and failing to do so can lead to inconsistencies or inaccuracies in the data.

Let's say Carmel is promoted to *Head Chef* and the company wants to **update this information**.

In a denormalised table like this, you would need to update multiple rows with the same `Employee_id`.

If we forget to update one of these rows, it would lead to an **inconsistency** in the data.



Employee_id	Name	Job_code	Job_title	State_code	Home_state
E001	Carmel	J01	Chef	26	Cape Town
E001	Carmel	J02	Waiter	26	Cape Town
E002	Stefanie	J02	Waiter	56	Joburg
E002	Stefanie	J03	Bartender	56	Joburg
E003	Lisa	J01	Chef	5	Nairobi

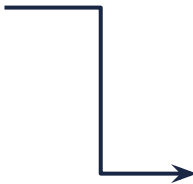
Data anomalies – Insertion anomaly

An insertion anomaly occurs when **a new record can't be added without introducing incomplete data** because certain required information is not yet available for the entity being represented.

A new employee, *John*, **doesn't yet have** a Job_code or State_code.

In a denormalised table like this, you might be forced to insert incomplete or inaccurate data.

This is common where adding a new record may entail duplicating information.



Employee_id	Name	Job_code	Job_title	State_code	Home_state
E001	Carmel	J01	Chef	26	Cape Town
E002	Stefanie	J02	Waiter	56	Joburg
E003	Lisa	J01	Chef	5	Nairobi
E004	John	NULL	NULL	NULL	NULL

Data anomalies – Deletion anomaly

A deletion anomaly is a type of data anomaly in a database where **deleting a single piece of data results in the unintentional loss of related data** that are still valid and necessary.

Let's say Lisa decides to leave the company, and her **record is removed** from the table.

All information related to **Nairobi** (State_code 5) was removed from the table.

Now, there is no record of any employee in Nairobi, which is **not what we intended**.

Employee_id	Name	Job_code	Job_title	State_code	Home_state
E001	Carmel	J01	Chef	26	Cape Town
E002	Stefanie	J02	Waiter	56	Joburg
E003	Lisa	J01	Chef	5	Nairobi

Normalisation



Key normalisation principles

Atomic values: Each column in a table should contain only atomic (indivisible) values. This eliminates repeating groups and ensures data are stored in a tabular format (1NF).

No partial dependencies: In a table with a composite primary key, non-key attributes should be fully functionally dependent on the entire primary key (2NF).

No transitive dependencies: Non-key attributes should not depend on other non-key attributes within the same table (3NF).

- To normalise a database, one must know what the **requirements** are for each of the three normal forms that we'll go over.
- One of the key requirements to remember is that normal forms are progressive. That is, in order to have 3NF we must have 2NF, and in order to have 2NF we must have 1NF.

First Normal Form (1NF)

1NF is the initial stage of database normalisation. It sets the foundation for organising data in a relational database in a structured and consistent manner.

A table is said to be in 1NF if it meets the following criteria:

1. Each cell in the table must **not hold more than one value**, which is referred to as atomicity.
2. The table must have a **primary key** for identification.
3. The table should have **no duplicated rows or columns**.

How 1NF works

1. **Atomicity:** "Atomic" means that a value cannot be divided into smaller parts that have meaning on their own. Atomic values ensure that data are stored at its smallest meaningful level. This prevents the inclusion of arrays, lists, or multiple values within a single cell.
2. **Tabular structure:** Data must be organised in a tabular structure with rows and columns, where each column represents a distinct attribute, and each cell holds a single atomic value.



1NF addresses data anomalies by enforcing the rule of atomic values and organising data in a structured, non-redundant manner. This helps **prevent update anomalies** and **improve data consistency**, ultimately **enhancing data quality** and **reducing the risk of anomalies** in the database.

First Normal Form (1NF)

Example:

Suppose we have the following unnormalised table:

Employee_id	Name	Job_codes	Job_titles	State_codes	Home_states
E001	Carmel	J01, J02	Chef, Waiter	26, 26	Cape Town
E002	Stefanie	J02, J03	Waiter, Bartender	56, 56	Joburg
E003	Lisa	J01	Chef	5	Nairobi



Several columns contain non-atomic, comma-separated values. We need to break down these columns into separate rows, for a more normalised structure.

In our 1NF table, each column contains atomic values and each row is uniquely identified by the Employee_id.

Employee_id	Name	Job_code	Job_title	State_code	Home_state
E001	Carmel	J01	Chef	26	Cape Town
E001	Carmel	J02	Waiter	26	Cape Town
E002	Stefanie	J02	Waiter	56	Joburg
E002	Stefanie	J03	Bartender	56	Joburg
E003	Lisa	J01	Chef	5	Nairobi

Second Normal Form (2NF)

2NF is the next level of database normalisation.

A table is considered to be in 2NF if it meets the following criteria:

1. It is already in 1NF (First Normal Form).
2. It does not contain **partial dependencies**, which means that non-key attributes are fully functionally dependent on the entire primary key.



2NF addresses data anomalies by eliminating partial dependencies which **enhances data integrity**, **reduces update anomalies**, **improves insertion and deletion operations**, and **promotes structured data organisation**. It is crucial for maintaining a reliable and efficient relational database.

How 2NF works

1. **Identify primary key:** Find the unique identifier for each row, which can be a single attribute or a combination.
2. **Ensure full dependencies:** Non-key attributes must depend entirely on the primary key, with no partial dependencies.
3. **Table splitting** (if needed): If partial dependencies exist, consider dividing the table into related tables, each with its own primary key.
4. **Create relationships:** When tables are split, establish links using foreign keys in child tables referencing parent table primary keys for data consistency.

Second Normal Form (2NF)

Example:

Our 1NF table has a **partial dependency** issue because the combination of {Employee_id, Job_code} forms the **primary key**, and **non-key attributes** (e.g. Job_title) **depend only on part** of the primary key (Job_code).

To bring this table into 2NF, we can **split it** into two separate tables that are **linked** by the Employee_id column:

1. Employee information.
2. Job-related information.

Employee_id	Name	Job_code	Job_title	State_code	Home_state
E001	Carmel	J01	Chef	26	Cape Town
E001	Carmel	J02	Waiter	26	Cape Town
E002	Stefanie	J02	Waiter	56	Joburg
E002	Stefanie	J03	Bartender	56	Joburg
E003	Lisa	J01	Chef	5	Nairobi



Employee_id	Name	State_code	Home_state
E001	Carmel	26	Cape Town
E002	Stefanie	56	Joburg
E003	Lisa	5	Nairobi

Employee_id	Job_code	Job_title
E001	J01	Chef
E001	J02	Waiter
E002	J02	Waiter
E002	J03	Bartender
E003	J01	Chef

Third Normal Form (3NF)

A table is considered to be in **3NF** if it satisfies the following criteria:

1. It is already in **2NF**.
2. It does not contain **transitive dependencies**: In 3NF, a table should ensure that **non-key attributes** (attributes not part of the primary key) are not transitively dependent on the primary key through other non-key attributes.



3NF addresses data anomalies, particularly those associated with transitive dependencies, by enforcing direct dependencies on the primary key, enhancing **data integrity**, **reducing update anomalies**, and **promoting structured data organisation**.

How 3NF works

1. **Eliminates transitive dependencies**: Transitive dependencies occur when a non-key attribute is dependent on another non-key attribute. 3NF requires that all non-key attributes be directly dependent on the primary key.
2. **Enhances data integrity**: Maintains data accuracy by preventing non-key attributes from relying on other non-key attributes.
3. **Reduces update anomalies**: Minimises the risk of unintended data inconsistencies during updates.
4. **Organised data**: Encourages structured data by linking non-key attributes directly to the primary key.

Third Normal Form (3NF)

Example:

In the employee information table, the primary key is {Employee_id}, and there are **no partial dependencies**. However, there is a **transitive dependency** between State_code and Home_state because State_code indirectly determines Home_state through Employee_id.

To achieve **3NF**, we would **split** this table into two separate tables linked by State_code:

1. Employee information.
2. Location information.

Employee_id	Name	State_code	Home_state
E001	Carmel	26	Cape Town
E002	Stefanie	56	Joburg
E003	Lisa	5	Nairobi



Employee_id	Name	State_code
E001	Carmel	26
E002	Stefanie	56
E003	Lisa	5

State_code	Home_state
26	Cape Town
56	Joburg
5	Nairobi