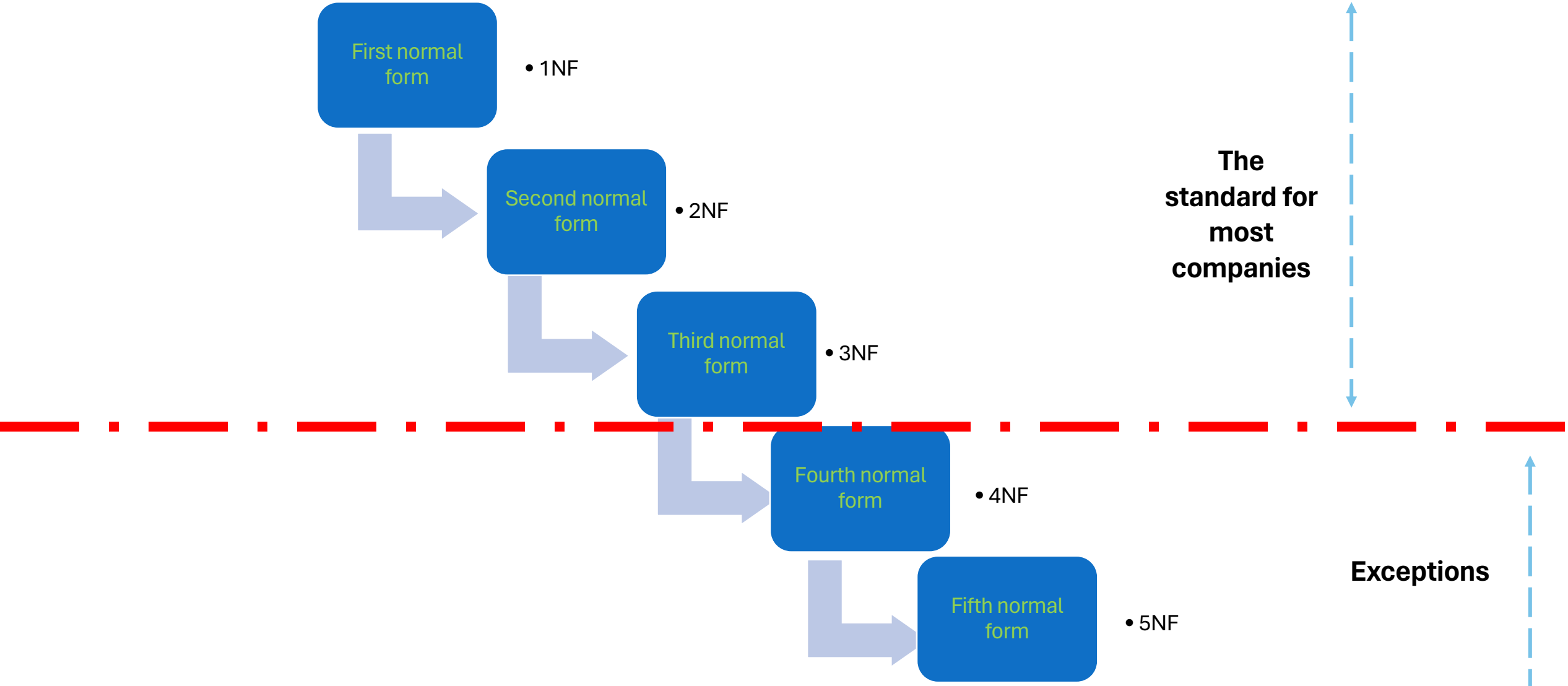
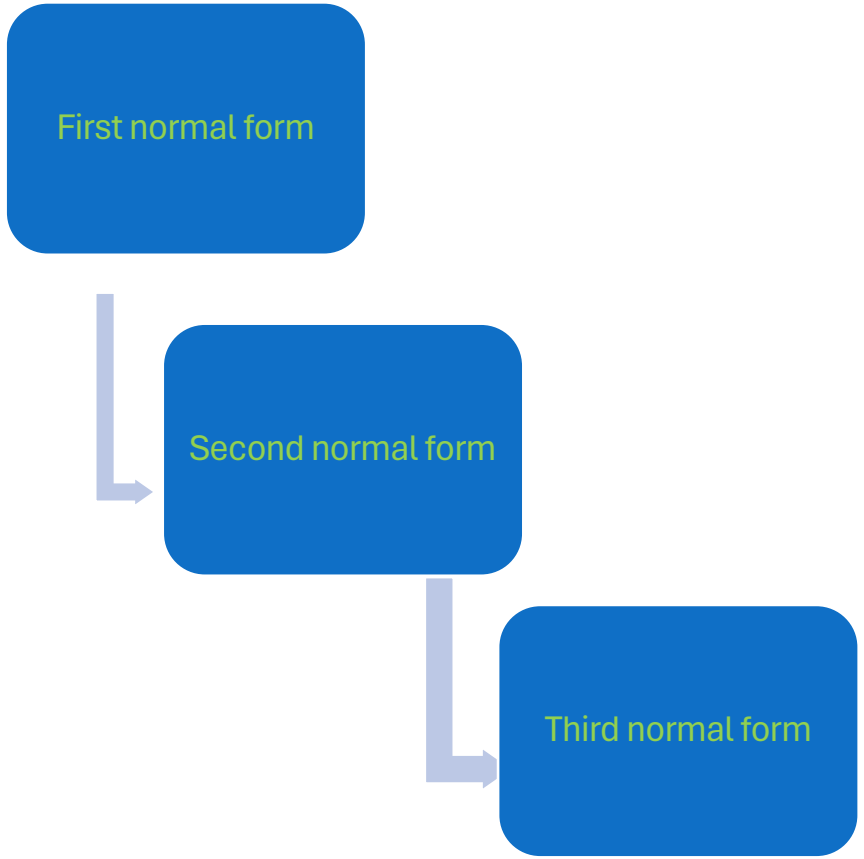
Lecture 2: Database normalization for SQL

Dr Anesu Nyabadza

Levels of data normalisation



Requirements for each of the Levels of data normalisation

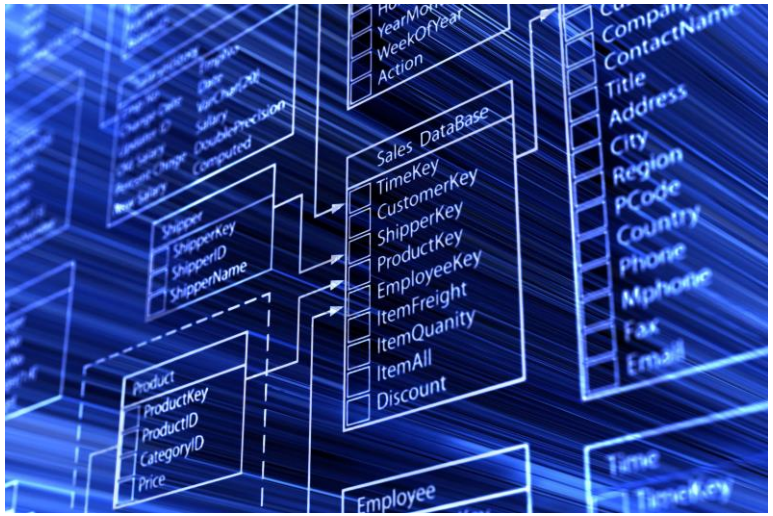


1. Eliminate duplicate columns from the same table.
2. Eliminate duplicate rows from the same table.
3. Each cell must only have 1 value
4. Create separate tables for related data and identify each row with a unique column or set of columns (primary key).
5. There must be no repeating groups (e.g., product_id_1, product_id_2), these must be put in another table.

1. Meet all the requirements of 1NF.
2. Every non-primary key attribute/column must be fully dependent on the entire primary key.

1. Meet all the requirements of 2NF.
2. No non-key attribute must be dependent on another non-primary key attribute.

Why do companies normalise databases?



Integrity

- Normalisation maintains data integrity by minimizing redundancy and inconsistencies in the database.
- By organizing data into separate tables and eliminating duplicate information, companies can ensure that each piece of data is stored only once, reducing the risk of errors and inconsistencies.

Storage

Normalised databases require less storage space compared to **denormalised databases** with redundant data.

- Leading to cost savings, especially for companies dealing with big data. (e.g., duplicates removed)

Performance

- Normalisation can improve query performance by reducing the number of **joins** needed to retrieve data.
- leading to faster response times and improved overall system performance.

Why do companies normalise databases?

Maintenance

- Normalised databases are easier to maintain/update/backup because data is organized logically and stored in separate tables, making changes to the database schema or adding new data easily.
- reduces the risk of errors during updates.

Consistency

Normalisation increases consistency by ensuring that each piece of data is stored in only one place.
reduces the chances of data anomalies such as update/insert/delete anomalies, insert anomalies

Data Analysis and Reporting

- provides a solid foundation for data analysis and reporting.
- By structuring data logically and consistently, companies can easily perform complex queries, generate accurate reports, and derive valuable insights from their data.

Reduced Attack Surface

- minimises redundancy and ensures that each piece of data is stored in only one place.
- This reduces the overall size of the database and limits the potential attack surface that could be exploited by malicious actors.

Access Control

- Views on certain tables

Data masking

- data masking techniques by segregating sensitive data into separate tables.
- Companies can apply data masking to sensitive columns or tables

Examples of the levels-NF1

1. Eliminate duplicate columns from the same table.
2. Eliminate duplicate rows from the same table.
3. Each cell must only have 1 value
4. Create separate tables for related data and identify each row with a unique column or set of columns (primary key).
5. There must be no repeating groups (e.g., product_id_1, product_id_2), these must be put in another table.

Student_ID	Student_Name	Math_Grade	CS_Grade	History_Grade
1	"Maria García"	"A"	"B+"	"C"
2	"Satoshi Nakamoto"	"B"	"A"	"B"
3	"Mai Nguyen"	"C+"	"A-"	"B+"

- The table contains repeating groups (Math_Grade, CS_Grade, History_Grade) for each student.
- The structure is not flexible, making it difficult to add new subjects or scale the table.
- Let us create a new table where each row represents a single student-subject-grade combination.

Student_ID	Student_Name	Subject	Grade
1	"Maria García"	"Mathematics"	"A"
1	"Maria García"	"Computer Scie	"B+"
1	"Maria García"	"History"	"C"
2	"Satoshi Nakamoto"	"Mathematics"	"B"
2	"Satoshi Nakamoto"	"Computer Scie	"A"
2	"Satoshi Nakamoto"	"History"	"B"
3	"Mai Nguyen"	"Mathematics"	"C+"
3	"Mai Nguyen"	"Computer Scie	"A-"
3	"Mai Nguyen"	"History"	"B+"

Examples of the levels-NF1

1. Eliminate duplicate columns from the same table.
2. Eliminate duplicate rows from the same table.
3. Each cell must only have 1 value
4. Create separate tables for related data and identify each row with a unique column or set of columns (primary key).
5. There must be no repeating groups (e.g., product_id_1, product_id_2), these must be put in another table.

Student_ID	Student_Name	Math_Grade	CS_Grade	History_Grade
1	"Maria García"	"A"	"B+"	"C"
2	"Satoshi Nakamoto"	"B"	"A"	"B"
3	"Mai Nguyen"	"C+"	"A-"	"B+"

- The table contains repeating groups (Math_Grade, CS_Grade, History_Grade) for each student.
- The structure is not flexible, making it difficult to add new subjects or scale the table.
- Let us create a new table where each row represents a single student-subject-grade combination.

Student_ID	Student_Name	Subject	Grade
1	"Maria García"	"Mathematics"	"A"
1	"Maria García"	"Computer Scie	"B+"
1	"Maria García"	"History"	"C"
2	"Satoshi Nakamoto"	"Mathematics"	"B"
2	"Satoshi Nakamoto"	"Computer Scie	"A"
2	"Satoshi Nakamoto"	"History"	"B"
3	"Mai Nguyen"	"Mathematics"	"C+"
3	"Mai Nguyen"	"Computer Scie	"A-"
3	"Mai Nguyen"	"History"	"B+"

Examples of the levels-NF1

1. Eliminate duplicate columns from the same table.
2. Eliminate duplicate rows from the same table.
3. Each cell must only have 1 value
4. Create separate tables for related data and identify each row with a unique column or set of columns **(primary key)**.
5. There must be no repeating groups (e.g., product_id_1, product_id_2), these must be put in another table.

Student_ID	Student_Name	Math_Grade	CS_Grade	History_Grade
1	"Maria García"	"A"	"B+"	"C"
2	"Satoshi Nakamoto"	"B"	"A"	"B"
3	"Mai Nguyen"	"C+"	"A-"	"B+"

- The table contains repeating groups (Math_Grade, CS_Grade, History_Grade) for each student.
- The structure is not flexible, making it difficult to add new subjects or scale the table.
- Let us create a new table where each row represents a single student-subject-grade combination.

Student_ID	Student_Name	Subject	Grade
1	"Maria García"	"Mathematics"	"A"
1	"Maria García"	"Computer Scie	"B+"
1	"Maria García"	"History"	"C"
2	"Satoshi Nakamoto"	"Mathematics"	"B"
2	"Satoshi Nakamoto"	"Computer Scie	"A"
2	"Satoshi Nakamoto"	"History"	"B"
3	"Mai Nguyen"	"Mathematics"	"C+"
3	"Mai Nguyen"	"Computer Scie	"A-"
3	"Mai Nguyen"	"History"	"B+"

- What's the primary key?

Examples of the levels-NF1

Student_ID	Student_Name	Subject	Grade
1	"Maria García"	"Mathematics"	"A"
1	"Maria García"	"Computer Scie	"B+"
1	"Maria García"	"History"	"C"
2	"Satoshi Nakamoto"	"Mathematics"	"B"
2	"Satoshi Nakamoto"	"Computer Scie	"A"
2	"Satoshi Nakamoto"	"History"	"B"
3	"Mai Nguyen"	"Mathematics"	"C+"
3	"Mai Nguyen"	"Computer Scie	"A-"
3	"Mai Nguyen"	"History"	"B+"

- What's the primary key?
- For 1NF, its usually a combo of columns in this case Student_ID and Subject

Examples of the levels-NF1

1. Eliminate duplicate columns from the same table.
2. Eliminate duplicate rows from the same table.
3. Each cell must only have 1 value
4. Create separate tables for related data and identify each row with a unique column or set of columns (primary key).
5. There must be no repeating groups (e.g., product_id_1, product_id_2), these must be put in another table.

Employee_ID	Employee_Name	Skills
1	"John Doe"	"Java Python SQL"
2	"Jane Smith"	"JavaScript HTML CSS"
3	"Michael Brown"	"C++ Jave Python"

- Skills column contains multiple values separated by commas.
- This violates the principle that each cell should contain only one value.

Employee_ID	Employee_Name	Skill
1	"John Doe"	"Java"
1	"John Doe"	"Python"
1	"John Doe"	"SQL"
2	"Jane Smith"	"JavaScript"
2	"Jane Smith"	"HTML"
2	"Jane Smith"	"CSS"
3	"Michael Brown"	"C++"
3	"Michael Brown"	"Java"
3	"Michael Brown"	"Python"

Examples of the levels-NF1

1. Eliminate duplicate columns from the same table.
2. Eliminate duplicate rows from the same table.
3. Each cell must only have 1 value
4. Create separate tables for related data and identify each row with a unique column or set of columns (primary key).
5. There must be no repeating groups (e.g., product_id_1, product_id_2), these must be put in another table.

Order_ID	Customer_ID	Order_Date	Order_Total
1001	101	"2023-03-01"	150
1002	102	"2023-03-02"	200
1003	101	"2023-03-01"	150

Order_ID	Customer_ID	Order_Date	Order_Total
1001	101	"2023-03-01"	150
1002	102	"2023-03-02"	200

- What's the primary key?

Examples of the levels-NF2

1. Meet all the requirements of 1NF.
2. Every non-primary key attribute/column must be fully dependent on the entire primary key.

Order_ID	Product_ID	Product_Name	Unit_Price	Quantity
1	101	"Widget A"	10.99	5
1	102	"Widget B"	15.99	3
2	101	"Widget A"	10.99	2
2	103	"Widget C"	20.99	4

- In this table, the Product_Name and Unit_Price attributes are functionally dependent on the Product_ID, not the entire composite primary key (Order_ID, Product_ID).
- This violates the Second Normal Form because these attributes depend on only part of the primary key.
- To remove the partial dependencies and bring the table to 2NF, we need to split it into two separate tables

Examples of the levels-NF2

Order_ID	Product_ID	Product_Name	Unit_Price	Quantity
1	101	"Widget A"	10.99	5
1	102	"Widget B"	15.99	3
2	101	"Widget A"	10.99	2
2	103	"Widget C"	20.99	4

- To remove the partial dependencies and bring the table to 2NF, we need to split it into two separate tables.

One of the tables will contain the following → Order_ID (Primary Key, Foreign Key referencing Orders), Product_ID (Primary Key) and Quantity.

Order_ID	Product_ID	Quantity
1	101	5
1	102	3
2	101	2
2	103	4

- Now, each non-key attribute (Quantity) is fully dependent on the entire primary key (Order_ID, Product_ID).
- The Product_Name and Unit_Price attributes are removed because they were partially dependent on the primary key and are now stored in a separate table where Product_ID serves as the primary key.

A foreign key is a primary key that also links to other tables (column shared by other tables)

Examples of the levels-NF3

1. Meet all the requirements of 2NF.
2. No non-key attribute must be dependent on another non-primary key attribute.

Employee_ID	Employee_Name	Department_ID	Department_Name	Manager_ID	Manager_Name
1	"John Doe"	101	"Marketing"	201	"Jane Smith"
2	"Jane Smith"	102	"Finance"	301	"Mark Johnson"
3	"Mark Johnson"	102	"Finance"	401	"Michael Brown"

- In this table, the Department_Name and Manager_Name attributes are functionally dependent on the Department_ID and Manager_ID, respectively.
- The Manager_Name attribute is also dependent on the Department_ID, since it's indirectly determined by the Department_ID through the Manager_ID, violating the 3NF because a non-key attribute (Manager_Name) is dependent on another non-key attribute (Department_Name).
- We need to split it into three separate tables

Examples of the levels-NF3

1. Meet all the requirements of 2NF.
2. No non-key attribute must be dependent on another non-primary key attribute.

Employee_ID	Employee_Name	Department_ID	Department_Name	Manager_ID	Manager_Name
1	"John Doe"	101	"Marketing"	201	"Jane Smith"
2	"Jane Smith"	102	"Finance"	301	"Mark Johnson"
3	"Mark Johnson"	102	"Finance"	401	"Michael Brown"

- We need to split it into three separate tables

Employee_ID	Employee_Name	Department_ID	Manager_ID
1	"John Doe"	101	201
2	"Jane Smith"	102	301
3	"Mark Johnson"	102	401

Primary key

Department_ID	Department_Name
101	"Marketing"
102	"Finance"

Primary key/
foreign key

Manager_ID	Manager_Name
201	"Jane Smith"
301	"Mark Johnson"
401	"Michael Brown"

Primary key/
foreign key

Examples of the levels-NF3

1. Meet all the requirements of 2NF.
2. No non-key attribute must be dependent on another non-primary key attribute.

- We need to split it into three separate tables

Employee_ID	Employee_Name	Department_ID	Manager_ID
1	"John Doe"	101	201
2	"Jane Smith"	102	301
3	"Mark Johnson"	102	401

Primary key

Department_ID	Department_Name
101	"Marketing"
102	"Finance"

Primary key/
foreign key

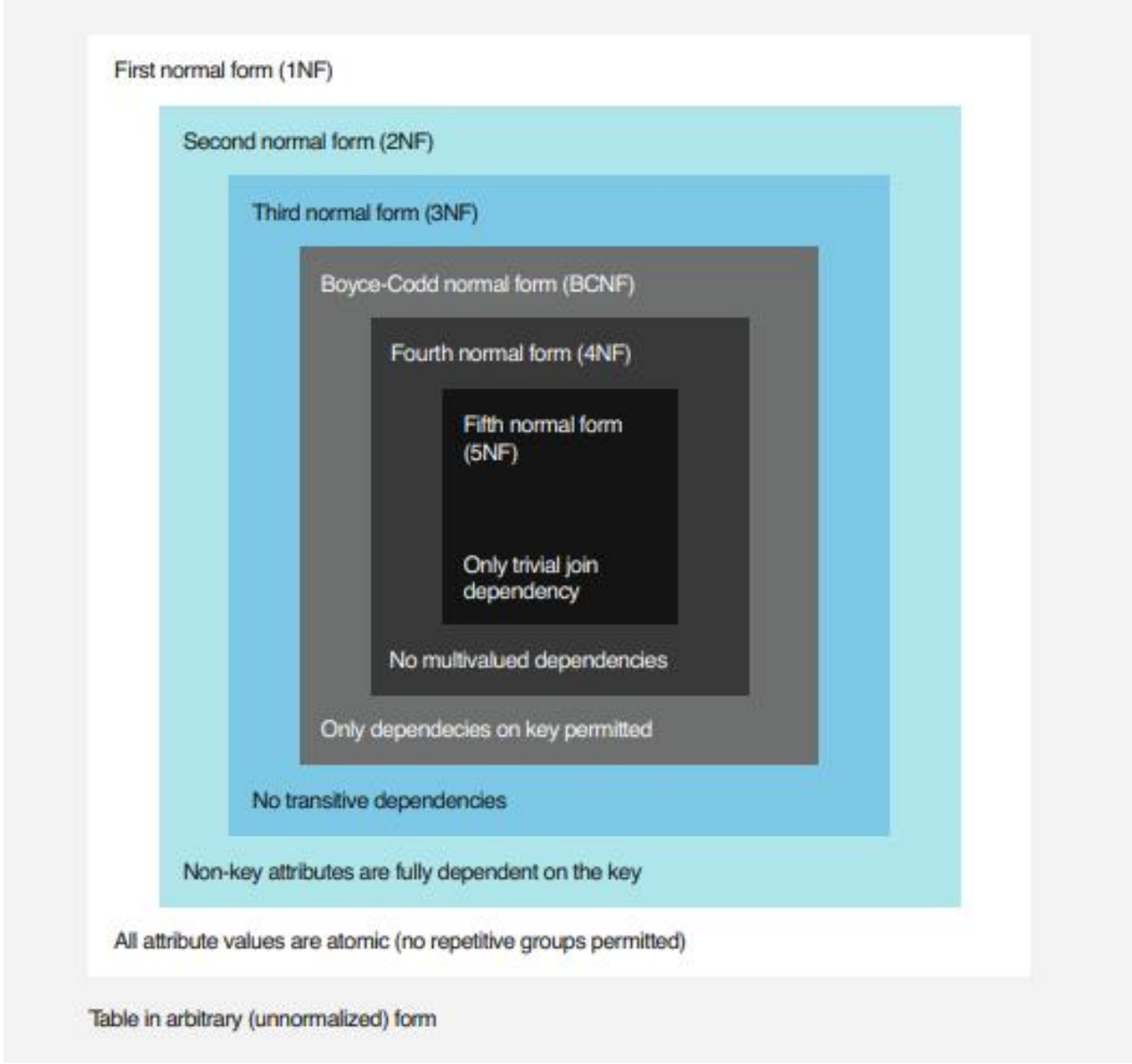
Manager_ID	Manager_Name
201	"Jane Smith"
301	"Mark Johnson"
401	"Michael Brown"

Primary key/
foreign key

3NF requirements met !

Each non-Primary key attribute (Department_Name, Manager_Name) is directly dependent only on the primary key of its respective table, and there are no dependencies present in the database schema (except with the primary key).

Summary


First normal form (1NF)
Second normal form (2NF)
Third normal form (3NF)
Boyce-Codd normal form (BCNF)
Fourth normal form (4NF)
Fifth normal form (5NF)
Only trivial join dependency
No multivalued dependencies
Only dependecies on key permitted
No transitive dependencies
Non-key attributes are fully dependent on the key
All attribute values are atomic (no repetitive groups permitted)
Table in arbitrary (unnormalized) form

MongoDB and Neo4J

- Normalisation is not enforced in NoSQL databases, given that they are already at the atomic level, e.g., MongoDB stores every entry as its document with flexible schema.

- In MongoDB, you have the option to embed related data within a single document or to reference data stored in separate documents. This decision depends on factors such as the frequency of access, data size, and relationships between entities.

- While denormalization (introducing redundancy for performance optimization) is common in MongoDB due to its document-based nature, excessive redundancy can lead to data inconsistency and maintenance challenges. It's important to strike a balance between denormalization for performance and avoiding unnecessary redundancy.

- MongoDB provides atomic operations at the document level, allowing you to update multiple fields within a document atomically. This can simplify data management and ensure consistency within individual documents.

- MongoDB allows documents to contain nested structures, including arrays and sub-documents. This enables the storage of related data within a single document, similar to how normalized tables in relational databases represent related entities. Embedding can reduce the need for joins and can result in better performance for read-heavy workloads.
- MongoDB's flexible schema allows documents within the same collection to have different structures, accommodating changes in data requirements over time. This flexibility can simplify schema evolution and reduce the need for complex migration processes often associated with schema changes in relational databases.