Normalisation and SQL CRUD operations

Key Terms

Functional Dependencies

> Functional Dependency is when one attribute determines another attribute in a DBMS

Data Normalisation

> the process of splitting relations into well-structured relations that allow users to insert, delete, and update tuples without introducing database inconsistencies

SQL data types

MySQL supports SQL data types in several categories: numeric types, date and time types, string (character and byte) types, spatial types, and the JSON data type. The following are the string and numeric types:

String types

Type	Description	Size	Range	Example
`CHAR(n)`	Fixed-length string	0-255	0-65,535	`CHAR(10)`

Type	Description	Size	Range	Example
`VARCHAR(n)`	Variable-length string	0-255	0-65,535	`VARCHAR(10)`
`TINYTEXT`	Variable-length string	0-255	0-65,535	`VARCHAR(10)`
`TEXT`	Variable-length string	0-65,535	0-4,294,967,295	`TEXT`
`MEDIUMTEXT`	Variable-length string	0-16,777,215	0-4,294,967,295	`MEDIUMTEXT`
`LONGTEXT`	Variable-length string	0-4,294,967,295	0-4,294,967,295	`LONGTEXT`

Numeric types

Type	Description	Size	Range	Example
`TINYINT`	Integer	1 byte	-128 to 127	`TINYINT`
`SMALLINT`	Integer	2 bytes	-32,768 to 32,767	`SMALLINT`
`MEDIUMINT`	Integer	3 bytes	-8,388,608 to 8,388,607	`MEDIUMINT`
`INT`	Integer	4 bytes	-2,147,483,648 to 2,147,483,647	`INT`
`BIGINT`	Integer	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	`BIGINT`
`DECIMAL`	Fixed-point number	0-65	-10^38+1 to 10^38-1	`DECIMAL(10, 2)`
`FLOAT`	Floating-point number	4 bytes	-3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38	`FLOAT`

Data Anomalies

> An anomaly is something that is unusual or unexpected; an abnormality

A very common source for issues in the database is redundancy. Apart from storage issues, having the same value present in multiple rows can lead to inconsistent data. Have a look at the following `STUDENTS` table.

<u>ID</u>	NAME	EMAIL	BATCH_ID	BATCH_NAME

Let us assume that the above table is the only table in the database. The student and batch entities are tightly coupled. What are some issues that can be caused by this?

- How do we create a new batch without students?
- How do we create a new student without a batch?

- What data do we lose if we delete a student?
- What happens if we modify a batch name but miss a record?

Anomalies are avoided by the process of normalisation The above issues or anomalies can be categorised into the following categories:

Insertion Anomalies

> The inability to insert a new tuple into a table due to missing data is known as insertion anomaly.

> An insertion anomaly occurs when data cannot be inserted into a database due to other missing data

This is most common for fields where a foreign key must not be NULL, but lacks the appropriate data Adding a student without a batch is not possible in the above schema is a batch_id is required. Whereas creating a new batch without students would require multiple null values and special handling.

Deletion Anomalies

> A deletion anomaly occurs when data is unintentionally lost due to the deletion of other data A deletion anomaly is the unintended loss of data due to deletion of other data

ID	NAME	EMAIL	BATCH_ID	BATCH_NAME
1	John Watson	j@sherlock.ed	1	Sherlock Season 6
2	Mary Watson	m@sherlock.ed	1	Sherlock Season 6
3	Kilvish	kil@vi.sh	2	Shaktimaan

In the above table, just `Kilvish` is associated with the batch `Shaktimaan`. Now, if we delete `Kilvish` from the database, we lose the data associated with the batch. This results in database inconsistencies and is an example of how combining information that does not really belong together into one table can cause problems

Updation Anomalies

> An update anomaly occurs when data is only partially updated in a database An update anomaly is a data inconsistency that results from data redundancy and a partial update

ID	NAME	EMAIL	BATCH_ID	BATCH_NAME
1	John Watson	j@sherlock.ed	1	Sherlock Season 6
2	Mary Watson	m@sherlock.ed	1	Sherlock Season 6
3	Kilvish	kil@vi.sh	2	Shaktimaan
4	Mycroft Holmes	brother@sherlock.ed	1	Sherlock Season 6

In the above table, we have three students associated with the batch `Sherlock Season 6`. If we have to update the batch name, each row will have to be updated due to redundancy. This adds an overhead and a likely source of data inconsistency. If our developer or query misses a record, the database will be in an inconsistent state.

Functional Dependencies

> a dependency FD: X → Y means that the values of Y are determined by the values of X. Two tuples sharing the same values of X will necessarily have the same values of Y.

- A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets.
- It is denoted as `X → Y`, where X is a set of attributes that is capable of determining the value of Y
- The attribute set on the left side of the arrow, X is called Determinant, while on the right side, Y is called the Dependent

> Suppose one is designing a system to track vehicles and the capacity of their engines. Each vehicle has a unique vehicle identification number (VIN). One would write **VIN → EngineCapacity** because it would be inappropriate for a vehicle's engine to have more than one capacity. On the other hand, EngineCapacity → VIN is incorrect because there could be many vehicles with the same engine capacity

ID	NAME	EMAIL	BATCH_ID	BATCH_NAME

Using the above schema, the following observations can be made:

- `ID` can be used to derive `NAME` and `EMAIL`. Hence,
  - `ID → NAME`
  - `ID → EMAIL`
- `BATCH_ID` can be used to derive `BATCH_NAME`
  - `BATCH_ID → BATCH_NAME`
- Since an `email` is a unique identifier, it can be used to derive `ID` and `NAME`
  - `EMAIL → ID`
  - `EMAIL → NAME`
- `ID` can also be used to derive `BATCH_ID`
  - `ID → BATCH_ID`

Figure out the functional dependencies for the below table

MENTOR_ID	STUDENT_ID	SESSION_ID	RATING	FEEDBACK
1	1	1	5	Very Good
1	2	1	4	Good
2	3	1	3	Average
1	1	2	4	Good

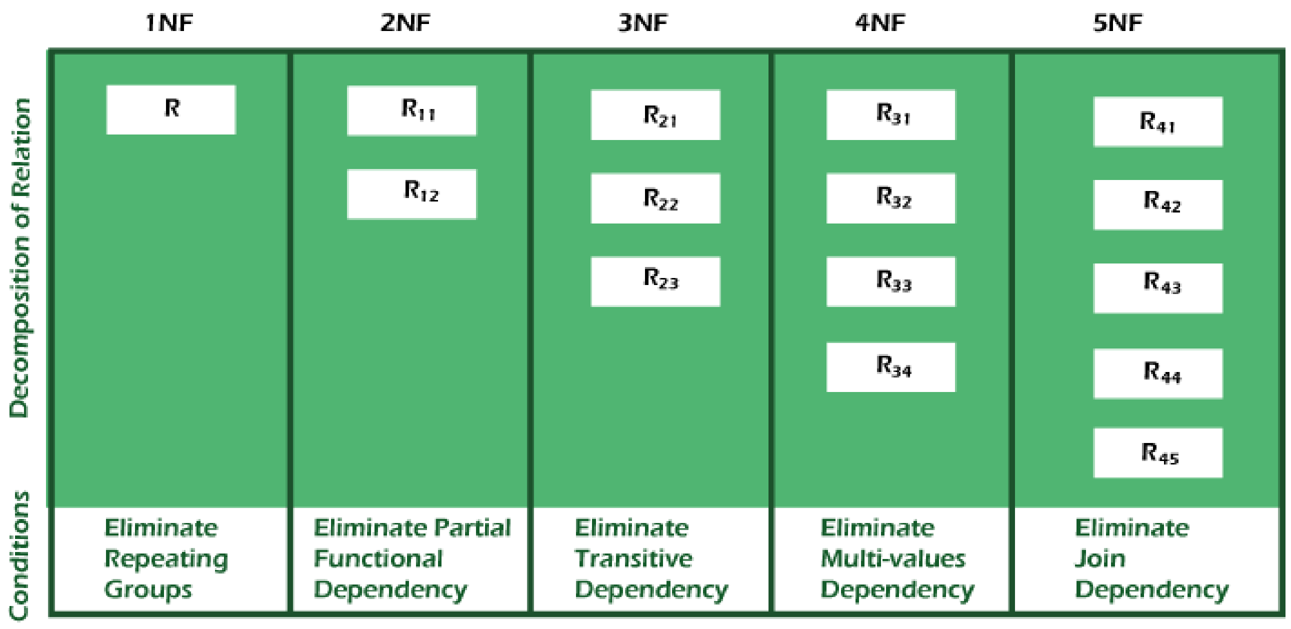
Data Normalisation

> the process of structuring a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity. Normalization entails organizing the columns (attributes) and tables (relations) of a database to ensure that their dependencies are properly enforced by database integrity constraints

The goal of normalisation is to produce a set of tables that

- Is a faithful model of the enterprise
- Is highly flexible
- Reduces redundancy-saves space and reduces inconsistency in data
- Is free of update, insertion and deletion anomalies

Following are the various normal forms:


1NF
2NF
3NF
4NF
5NF
Decomposition of Relation
Conditions
R
R11
R12
R21
R22
R23
R31
R32
R33
R34
R41
R42
R43
R44
R45
Eliminate Repeating Groups
Eliminate Partial Functional Dependency
Eliminate Transitive Dependency
Eliminate Multi-values Dependency
Eliminate Join Dependency

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.

ID	NAME	EMAIL	PHONE_NUMBERS
1	Tantia Tope	tantia@rani.bai	[123456789, 987654321]
2	Kilvish	kil@vi.sh	[987654321, 123456789]
3	John Watson	i.am@sherlock.ed	[123456789, 987654321]

The above table is not 1NF because it contains a multi-valued attribute i.e. phone numbers.

Redundant columns

NAME	EMAIL	PHONE_NUMBER_01	PHONE_NUMBER_02
Tantia Tope	tantia@rani.bai	123456789	987654321
Kilvish	kil@vi.sh	987654321	123456789
John Watson	i.am@sherlock.ed	123456789	987654321

Cons -

- Wasteful if all rows have mostly one phone number
- Hard to determine upper bound of number of phone numbers
- Querying is not efficient since multiple columns needs to be queried
- Multiple indexes

Redundant rows

ID	NAME	EMAIL	PHONE_NUMBERS
1	Tantia Tope	tantia@rani.bai	123456789
1	Tantia Tope	tantia@rani.bai	987654321
2	Kilvish	kil@vi.sh	123456789
2	Kilvish	kil@vi.sh	987654321
3	John Watson	i.am@sherlock.ed	987654321
3	John Watson	i.am@sherlock.ed	123456789

What will be primary key in the above table?

Cons -

- A lot of redundant rows which can lead to anomalies
- Primary key needs to be altered

Separate Mapping Table

The two solutions above are not ideal due to the large amount of redundant data. In order to properly convert the above table to 1NF, we need to create a separate table that maps the redundant data to the primary key. Hence, a `PHONE_NUMBER` table is created with `student_id` and `phone_number` columns. There are multiple rows for each student and the ID is used to map the redundant data. This minimises the amount of redundant data.

```
erDiagram
    STUDENT {
        int id
        string name
        string email
    }
    PHONE_NUMBER {
        int student_id
        string phone_number
    }
    STUDENT ||--|{ PHONE_NUMBER : has
```

2NF

- In the 2NF, relational must be in 1NF.
- There should be no partial dependencies.
- Every non candidate-key attribute must depend on the whole candidate key, not just part of it

ID	NAME	BATCH_ID	BATCH_NAME	PSP

Listing out the dependencies for the above table:

- `ID` and `BATCH_ID` can be used to derive `PSP`
  - `ID, BATCH_ID → PSP`
- `BATCH_ID` can be used to derive `BATCH_NAME`
  - `BATCH_ID → BATCH_NAME`

In the first dependency, `ID` and `BATCH_ID` can determine a non-candiate key attribute. However, in the second dependency just `BATCH_ID` can determine a non-candidate key. This is an example of a partial dependency and hence violates 2NF.

The above table can be normalised by creating a separate table for batch information.

```
erDiagram
    STUDENT {
        int id
        int batch_id
        string name
        float psp
    }
    BATCH {
        int batch_id
        string name
    }
    STUDENT ||--|{ BATCH : joins
```

3NF

- In the 3NF, relational must be in 2NF.
- It should also not contain any transitive dependencies.

ID	NAME	BATCH_ID	BATCH_NAME

Listing out the dependencies for the above table:

- `ID → NAME`
- `ID → BATCH_ID`
- `BATCH_ID → BATCH_NAME`
- `ID → BATCH_NAME`

It can be observed in the last three dependencies that `ID` can determine `BATCH_ID` that can be used to determine `BATCH_NAME` i.e. `ID → BATCH_ID → BATCH_NAME`. This is an example of a **transitive dependency** and hence violates 3NF.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

- X is a super key.
- Y is a prime attribute, i.e., each element of Y is part of some candidate key.

ID	NAME	PHONE	BATCH_ID	BATCH_NAME

In the above table,

- `ID -> PHONE`
- `PHONE → ID`
- `PHONE → NAME`

Do any of the above violate 3NF? **NO** Since `ID` and `PHONE` are both candidate keys. What about `BATCH_ID → BATCH_NAME`?

Yes, this violates 3NF since `BATCH_ID` is not a super key and `BATCH_NAME` is not a prime attribute.

Again, the above table can be normalised by creating a separate table for batch information.

```
erDiagram
    STUDENT {
        int id
        int batch_id
        string name
        int phone
    }
    BATCH {
        int batch_id
        string name
    }
    STUDENT ||--|{ BATCH : joins
```

Boyce-Codd Normal Form (BCNF)

- A table is in BCNF if every functional dependency X → Y, **X is the primary key of the table**.

Looking at the normalised table from 3NF

ID	NAME	PHONE	BATCH_ID

We can list the following dependencies for the above table:

- `ID → NAME`
- `ID → PHONE`
- `ID → BATCH_ID`
- `PHONE → NAME`

It can be clearly seen that the last dependency violates BCNF as phone is not a primary key. The above table can be normalised by creating a separate table for phone information.

```
erDiagram
    STUDENT {
        int id
        int batch_id
        string name
    }
    BATCH {
        int batch_id
        string name
    }
    PHONE_NUMBER {
        int student_id
        int phone_number
    }
    STUDENT ||--|{ BATCH : joins
    STUDENT ||--|{ PHONE_NUMBER : has
```

Reading List

- Normalisation
- CHAR and VARCHAR
- CHAR vs VARCHAR vs TEXT
- Why not to use floating points
- IEEE 754