3/18/2024

Database packages in Python

Sqlite
Pandasql
Mysql-connector-python (MySQL)
Psycopg2 (for PostGreSQL)
Python-sql
pymySQL
MySQL-python
PyQt5 (QtSQL)
Supersqlite
Sqlalchemy
Records
Pugsql
SQLObject
Peewww
Pony ORM

Django ORM

noSQL:
pymongo
mongodb
Others.

Normalization in relational databases.

Database normalization comes in a number of forms:

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)
- 5NF (Fifth Normal Form)
- 6NF (Sixth Normal Form)

Sixth Normal form is still being debated. Generally speaking, SQL tends work best in 3rd Normal form. This will be our goal for our database project (#2).

The requirements for first normal form are:

- Each table cell should contain a single value.
- Each record needs to be unique.

Put another way, no list elements in cells, and no duplications.

Tables in SQL databases have what are called keys. Primary keys are cell values that unique identify a record. Foreign keys will be introduced later: these are primary keys whose records are in another table, but are used to link tables together.

Primary keys are often numerical values since numbers take up less memory space than text, but primary keys don't have to be numbers, or even a single column. It's possible that you can use combinations of columns that make the record unique. These are called composite keys. However, it is preferred that this isn't done generally…

In 2nd Normal form, the database tables must satisfy all the rules of the 1st normal form AND

- Single Column Primary Key that does not functionally dependent on any subset of candidate key relation

It's at this level of normalization that we tend to start splitting our single table database into multiple tables. For instance, people information in one table, and transaction data (things they purchase) in a second table, and so on. This means that if more than one person purchases the same product, we don't have to list the product multiple times. And if the same person purchases several different products, we don't have to list the people multiple times. Instead, we can use their primary keys to refer to the rest of their information.

This is when we introduce the idea of a foreign key.

You can only use foreign keys that exist in the referential table, to prevent pointing at null records. The database will throw an error if you try this.

Let's look at 3rd normal form. To be in this form, you must follow all the rules of 2nd normal form, AND

- Has no transitive functional dependencies

This is also where we can reduce memory load, but putting commonly used values (especially text that takes up a lot of memory or is prone to spelling errors) into a separate table and use only the foreign key to reference it. Because of the way foreign keys work, this helps reduce errors because you have to point to a value in the table. If you update the reference table, all values are updated everywhere. Each record referring to that value won't have to be updated separately.

Further than this:
Boyce-Codd Normal form is sometimes referred to as 3.5 normal form adds the requirement that there is only one candidate key.

4th normal form no database table instance contains two or more, independent and multivalued data describing the relevant entity.

5th normal form follows all the rules above, but also it cannot be decomposed into any number of smaller tables without loss of data. (this is a good goal to achieve in the project #2 assignment.)

6th normal form isn't fully standardized, so we won't worry about it here.

Why normalize?

The main objectives of normalizing your product data is to achieve the following:

- To correct duplicate data and database anomalies.
- To avoid creating and updating any unwanted data connections and dependencies.
- To prevent unwanted deletions of data.
- To optimize storage space.
- To reduce the delay when new types of data need to be introduced.
- To facilitate the access and view of data to users and product tools.

Resources:

1. https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description
2. https://www.guru99.com/database-normalization.html
3. https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description
4. https://www.techtarget.com/searchdatamanagement/definition/normalization
5. https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/
6. https://www.databasestar.com/database-normalization/
7. https://www.essentialsql.com/database-normalization/
8. https://www.datanamic.com/support/database-normalization.html
9. https://www.w3schools.in/dbms/database-normalization
10. https://blog.saleslayer.com/why-is-database-normalization-so-important
11. https://www.sqlshack.com/what-is-database-normalization-in-sql-server/
12. https://towardsdatascience.com/a-complete-guide-to-database-normalization-in-sql-6b16544deb0

Extended commentary:

Normalization is a process of organizing data in a relational database to minimize data redundancy and improve data integrity. It involves breaking down a table into smaller, more manageable tables and defining relationships between them. Normalization is important in ensuring data consistency and reducing data anomalies such as insertion, deletion, and update anomalies.

There are different levels of normalization, including:

First Normal Form (1NF): A table is in 1NF if it has no repeating groups or arrays, and each column contains atomic values.

Second Normal Form (2NF): A table is in 2NF if it is in 1NF and all non-key columns are fully dependent on the primary key.

Third Normal Form (3NF): A table is in 3NF if it is in 2NF and all non-key columns are independent of each other.

Boyce-Codd Normal Form (BCNF): A table is in BCNF if it is in 3NF and every determinant is a candidate key.

Fourth Normal Form (4NF): A table is in 4NF if it is in BCNF and has no multi-valued dependencies.

Fifth Normal Form (5NF): A table is in 5NF if it is in 4NF and every join dependency is implied by the candidate keys.

Normalization has several benefits, including:

Data consistency: Normalization helps to ensure that data is consistent and accurate.

Reduced data redundancy: Normalization reduces data redundancy and improves database efficiency.

Improved data integrity: Normalization improves data integrity by reducing data anomalies.

However, normalization can also have some drawbacks, including:

Increased complexity: Normalization can increase database complexity, which may require more time and resources to implement and maintain.

Performance issues: In some cases, normalization can result in slower query performance, especially when dealing with large datasets.

Overall, normalization is an important aspect of database design that can help to improve data consistency, accuracy, and integrity.

Normalization is the process of organizing a database to reduce redundancy and improve data integrity. The main objective of normalization is to eliminate data redundancy and improve data consistency by ensuring that data is stored logically in related tables. Normalization helps to eliminate data anomalies that can occur when data is stored in a non-normalized database.

A non-normalized database can result in data inconsistencies, duplicate data, and update anomalies, such as when changing one piece of data requires updating multiple records. Normalizing a database can help to improve data quality, reduce the risk of data corruption, and make it easier to maintain and modify the database over time.

Normalization is typically accomplished by breaking up a large table into smaller, related tables, and using relationships between these tables to bring the data back together as needed. The process typically involves creating primary and foreign keys to establish relationships between tables, and reorganizing data so that it is stored in a way that avoids duplication and ensures consistency.

In a relational database, a primary key is a field or a combination of fields that uniquely identify each record in a table. It is a column or set of columns that serves as the unique identifier of a table. Primary keys enforce the entity integrity of the table, which means that they ensure that each record in the table is unique and can be identified without ambiguity.

A foreign key, on the other hand, is a field in one table that refers to the primary key of another table. It is used to establish a relationship between two tables, where the foreign key in one table refers to the primary key in another table. This relationship enables data to be retrieved from multiple tables and provides a way to maintain data integrity across tables. The foreign key in one table is linked to the primary key of another table to ensure referential integrity.