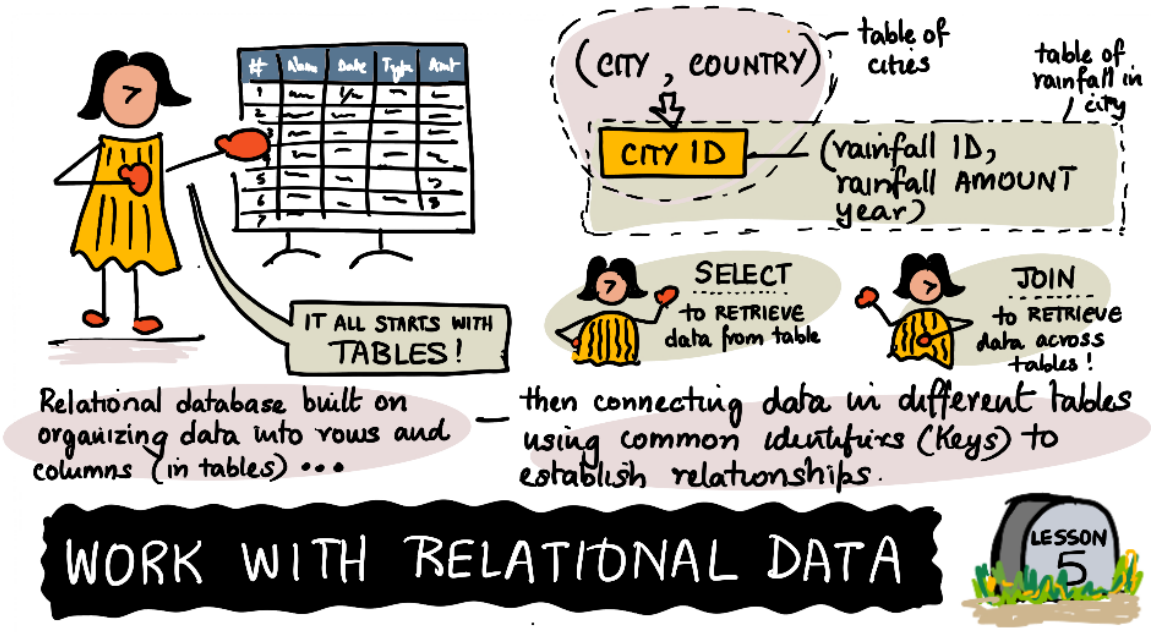
Working with Data: Relational Databases



Working With Data: Relational Databases - *Sketchnote by @nitya*

Chances are you have used a spreadsheet in the past to store information. You had a set of rows and columns, where the rows contained the information (or data), and the columns described the information (sometimes called metadata). A relational database is built upon this core principle of columns and rows in tables, allowing you to have information spread across multiple tables. This allows you to work with more complex data, avoid duplication, and have flexibility in the way you explore the data. Let's explore the concepts of a relational database.

Pre-lecture quiz

It all starts with tables

A relational database has at its core tables. Just as with the spreadsheet, a table is a collection of columns and rows. The row contains the data or information we wish to work with, such as the name of a city or the amount of rainfall. The columns describe the data they store.

Let's begin our exploration by starting a table to store information about cities. We might start with their name and country. You could store this in a table as follows:

City	Country
Tokyo	Japan
Atlanta	United States
Auckland	New Zealand

Notice the column names of **city**, **country** and **population** describe the data being stored, and each row has information about one city.

The shortcomings of a single table approach

Chances are, the table above seems relatively familiar to you. Let's start to add some additional data to our burgeoning database - annual rainfall (in millimeters). We'll focus on the years 2018, 2019 and 2020. If we were to add it for Tokyo, it might look something like this:

City	Country	Year	Amount
Tokyo	Japan	2020	1690
Tokyo	Japan	2019	1874
Tokyo	Japan	2018	1445

What do you notice about our table? You might notice we're duplicating the name and country of the city over and over. That could take up quite a bit of storage, and is largely unnecessary to have multiple copies of. After all, Tokyo has just the one name we're interested in.

OK, let's try something else. Let's add new columns for each year:

City	Country	2018	2019	2020
Tokyo	Japan	1445	1874	1690
Atlanta	United States	1779	1111	1683
Auckland	New Zealand	1386	942	1176

While this avoids the row duplication, it adds a couple of other challenges. We would need to modify the structure of our table each time there's a new year. Additionally, as our data grows having our years as columns will make it trickier to retrieve and calculate values.

This is why we need multiple tables and relationships. By breaking apart our data we can avoid duplication and have more flexibility in how we work with our data.

The concepts of relationships

Let's return to our data and determine how we want to split things up. We know we want to store the name and country for our cities, so this will probably work best in one table.

City	Country
Tokyo	Japan
Atlanta	United States
Auckland	New Zealand

But before we create the next table, we need to figure out how to reference each city. We need some form of an identifier, ID or (in technical database terms) a primary key. A primary key is a value used to identify one

specific row in a table. While this could be based on a value itself (we could use the name of the city, for example), it should almost always be a number or other identifier. We don't want the id to ever change as it would break the relationship. You will find in most cases the primary key or id will be an auto-generated number.

> ☑ Primary key is frequently abbreviated as PK

cities

city_id	City	Country
1	Tokyo	Japan
2	Atlanta	United States
3	Auckland	New Zealand

> ☑ You will notice we use the terms "id" and "primary key" interchangeably during this lesson. The concepts here apply to DataFrames, which you will explore later. DataFrames don't use the terminology of "primary key", however you will notice they behave much in the same way.

With our cities table created, let's store the rainfall. Rather than duplicating the full information about the city, we can use the id. We should also ensure the newly created table has an *id* column as well, as all tables should have an id or primary key.

rainfall

rainfall_id	city_id	Year	Amount
1	1	2018	1445
2	1	2019	1874
3	1	2020	1690
4	2	2018	1779
5	2	2019	1111
6	2	2020	1683
7	3	2018	1386
8	3	2019	942
9	3	2020	1176

Notice the **city_id** column inside the newly created **rainfall** table. This column contains values which reference the IDs in the **cities** table. In technical relational data terms, this is called a **foreign key**; it's a primary key from another table. You can just think of it as a reference or a pointer. **city_id** 1 references Tokyo.

> [!NOTE] Foreign key is frequently abbreviated as FK

Retrieving the data

With our data separated into two tables, you may be wondering how we retrieve it. If we are using a relational database such as MySQL, SQL Server or Oracle, we can use a language called Structured Query Language or SQL. SQL (sometimes pronounced sequel) is a standard language used to retrieve and modify data in a relational database.

To retrieve data you use the command `SELECT`. At its core, you **select** the columns you want to see **from** the table they're contained in. If you wanted to display just the names of the cities, you could use the following:

```
SELECT city
FROM cities;

-- Output:
-- Tokyo
-- Atlanta
-- Auckland
```

`SELECT` is where you list the columns, and `FROM` is where you list the tables.

> [NOTE] SQL syntax is case-insensitive, meaning `select` and `SELECT` mean the same thing. However, depending on the type of database you are using the columns and tables might be case sensitive. As a result, it's a best practice to always treat everything in programming like it's case sensitive. When writing SQL queries common convention is to put the keywords in all upper-case letters.

The query above will display all cities. Let's imagine we only wanted to display cities in New Zealand. We need some form of a filter. The SQL keyword for this is `WHERE`, or "where something is true".

```
SELECT city
FROM cities
WHERE country = 'New Zealand';

-- Output:
-- Auckland
```

Joining data

Until now we've retrieved data from a single table. Now we want to bring the data together from both **cities** and **rainfall**. This is done by *joining* them together. You will effectively create a seam between the two tables, and match up the values from a column from each table.

In our example, we will match the **city_id** column in **rainfall** with the **city_id** column in **cities**. This will match the rainfall value with its respective city. The type of join we will perform is what's called an *inner* join, meaning if any rows don't match with anything from the other table they won't be displayed. In our case every city has rainfall, so everything will be displayed.

Let's retrieve the rainfall for 2019 for all our cities.

We're going to do this in steps. The first step is to join the data together by indicating the columns for the seam - **city_id** as highlighted before.

```
SELECT cities.city
    rainfall.amount
FROM cities
    INNER JOIN rainfall ON cities.city_id = rainfall.city_id
```

We have highlighted the two columns we want, and the fact we want to join the tables together by the **city_id**. Now we can add the `WHERE` statement to filter out only year 2019.

```
SELECT cities.city
    rainfall.amount
FROM cities
    INNER JOIN rainfall ON cities.city_id = rainfall.city_id
WHERE rainfall.year = 2019

-- Output

-- city     | amount
-- -------- | ------
-- Tokyo    | 1874
-- Atlanta  | 1111
-- Auckland |  942
```

Summary

Relational databases are centered around dividing information between multiple tables which is then brought back together for display and analysis. This provides a high degree of flexibility to perform calculations and otherwise manipulate data. You have seen the core concepts of a relational database, and how to perform a join between two tables.

🚀 Challenge

There are numerous relational databases available on the internet. You can explore the data by using the skills you've learned above.

Post-Lecture Quiz

Post-lecture quiz

Review & Self Study

There are several resources available on Microsoft Learn for you to continue your exploration of SQL and relational database concepts

- Describe concepts of relational data

- Get Started Querying with Transact-SQL (Transact-SQL is a version of SQL)
- SQL content on Microsoft Learn

Assignment

Assignment Title