Weekend: Introduction to Databases; Database Relationships

Introduction To Databases

So far our applications have been storing data inside the **RAM** Random access memory of our computers. This is quick and easy but also temporary because every time we restart the application the data has disappeared.

This week we will learn how to persist our data inside databases, create a database, insert and query data from it. We will also learn about a handy tool called **SQLAlchemy** that allows us to interact with our database at a higher level.

There are many types of databases. But most fall under 2 categories **SQL** - Structured Query Language databases and **NoSQL** -Non Structured Query Language databases.

Task

Go online and do research on these different kinds of databases. 1. Get the definition of both. 2. How they are structured. 3. The different types of databases for each category. 4. The Benefits of using each. 5. Disadvantages of using each.

You will present what you have learned the following day during stand up.

Postgres

We will be using **PostgreSQL** database for our application. This is a type of SQL database.

SQL Databases store data in tables which are different items in an application. Tables have a fixed number of columns and a variable number of rows.

They have a special column called a primary key which is a unique identifier for each row stored in the table.

They might also have foreign keys which are a reference to the primary key of another table. This defines the relationship between those two tables.

Installation

1. Ubuntu

To Install Postgres on Ubuntu we need to run the following commands in our terminal.

```
$ sudo apt-get update
```

To install Postgres

```
$ sudo apt-get install postgresql postgresql-contrib libpq-dev
```

Enter **y** when prompted “Do you want to continue? **[Y/n]**” and wait as the installation completes.

Defining a user role

Postgres uses "roles" to aid in authentication and authorization. By default, Postgres creates a Postgres user and is the only user who can connect to the server.

We want to create our own superuser role to connect to the server.

For those running on elementary or parrot, run the following command first;

```
$ sudo service postgresql start
```

```

$ sudo -u postgres createuser --superuser $USER
```

Enter your desired password when prompted.

We then have to create a database with that `$USER` login name, this is what Postgres expects as default.

```
$ sudo -u postgres createdb $USER
```

Navigate to your home directory and enter the following command to create the .psql_history in order to save your history:

```
$ touch .psql_history
```

You can now connect to the postgres server by typing :

```
$ psql
```

2. Mac

Homebrew makes it really easy to install Postgres. Just run:

```
 $ brew install postgres
```

After it finishes installing, you'll need to configure your computer a bit. First, you need to tell Postgres where to find the database cluster where your databases will be stored:

```
 $ echo "export PGDATA=/usr/local/var/postgres" >> ~/.bash_profile
```

This command will help some programs find Postgres more easily:

```
$ echo "export PGHOST=/tmp" >> ~/.bash_profile
```

To load these configuration changes, run:

```
$ source ~/.bash_profile
```

To start the Postgres server, simply run:

```
$ postgres
```

You'll have to leave that window open while you need the server. To stop the server, press Ctrl + C (_not_ Cmd + C). If you want Postgres to boot at startup and run in the background, run:

```
 $ ln -sfv /usr/local/opt/postgresql/*.plist ~/Library/LaunchAgents
```

And to start it now (since it won't boot automatically until you restart your computer), run:

```
$ pg_ctl start
```

To prepare for the next lesson, create a default database with your computer's username:

```
$ createdb $USER
```

And you're done.

Database Relationships

Databases are like linked spreadsheets. Each spreadsheet being like single **table**.

```
Shopping list
------------------

 id |    item         | quantity
 ---+-----------------+------------
 1  |  milk(packets)  | 7
 2  |  shoe polish    | 4
 3  | jewelry         | 5
```

Here is an example of a table displaying a shopping list.

One to many Relationships

Let us assume we are running a school and want to manage the students and what courses they were taking. We would create two tables to hold the data for the courses and for the students

```
courses
---------
 id |    course
 ---+-----------------
 1  |  Advanced Mathematics
 2  |  Literature
 3  |  Visual art
 4  |  Computer Science

 students
 ---------
  id |    student        | course_id
 ---+--------------------+-----
 1   |  James Muriuki     | 2
 2   |  Christine Wasike  | 1
 3   |  Moses Okemwa      | 3
 4   |  Audrey Cheng      | 1
```

The course_id in the students table references the `id` column in the `course` table. Since a single course can be taken by multiple students we say the `course` table has a **one-to-many** relationship with the `student` table.

Many to Many Relationships

It is very rare to find a school where all the students take on one course. Let us enable our students to take more than one course.

```
courses
---------
 id |    course
 ---+-----------------
 1   |  Advanced Mathematics
 2   |  Literature
 3   |  Visual art
 4   |  Computer Science

 students
 ---------
 id |    student         | course_id
 ---+-------------------+-----
 1   |  James Muriuki     | 2
 2   |  Christine Wasike  | 1
 3   |  Moses Okemwa      | 3
 4   |  Audrey Cheng      | 1
 5   |  Moses Okemwa      | 2
 6   |  Audrey Cheng      | 3
```

Here we added more entries to our `students` table to allow the students to take on more courses. This is **NOT** advisable because we do not want duplicate entries inside our tables.

```
courses
---------
 id |    course
 ---+-----------------
 1   |  Advanced Mathematics
 2   |  Literature
 3   |  Visual art
 4   |  Computer Science

 students
 ---------
  id |    student
 ---+-------------------
 1   |  James Muriuki
 2   |  Christine Wasike
 3   |  Moses Okemwa
 4   |  Audrey Cheng


 enrollments
 ----------------
   id | student_id | course_id
 ---+-------------+-----
 1   |      1       | 2
 2   |      2       | 1
 3   |      3       | 3
 4   |      4       | 1
 5   |      3       | 2
 6   |      4       | 3
```

We Instead create a new table called a **join** table. This table will reference the Items on the other table using the **id** attribute. When naming joining tables it is advisable to name them using the names of the tables that are being joined separated by underscores. If there is another name that could accurately describe the table it can be used instead.