
SQL

WELCOME BACK

QUESTIONS?

GOALS FOR THIS SECTION

1. Intro to Databases
2. SQL & Relational Databases
3. Intro to Postgres
    - Installation
4. Basic Building & Querying Tables
    - Lab
5. node-pg
6. Integrating Node and Postgres
    - Lab

DATABASES

DATABASES

A database is an organized collection of data. It is the collection of tables, queries, reports, views, and other objects. The data is typically organized to model aspects of reality in a way that supports processes requiring information, such as modeling the availability of rooms in hotels(in a way that supports finding a hotel with vacancies).

- via wikipedia

DATABASE MANAGEMENT SYSTEM

DBMSs are software applications that interact with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase and IBM DB2.

- also via wikipedia

TYPES OF DATABASES

TYPES OF DATABASES

There are a few different kinds of DBMSs but there are really only two broad categories of DB that you're likely to run into on a project.

- Relational Database
- NoSQL Database

NOSQL

NOSQL DATABASES

'NoSQL' is a catch-all term for databases that operate outside the relational standard. NoSQL Databases have become more popular in the last few years as a lightweight alternative to relational databases. As NoSQL does not refer to a specific style of data management.

- Document: MongoDB, CouchDB
- Key-Value: Redis, CouchDB
- Graph: Allegro, InfiniteGraph

RELATIONAL DATABASE

The Relational Model organizes data into one or more tables (or "relations") of rows and columns, with a unique key for each row. Generally, each entity type described in a database has its own table, the rows representing instances of that type of entity and the columns representing values attributed to that instance.

RELATIONAL DATABASE

...Because each row in a table has its own unique key, rows in a table can be linked to rows in other tables by storing the unique key of the row to which it should be linked (where such unique key is known as a "foreign key").

- More wikipedia

ANATOMY OF A RELATIONAL TABLE

SUPERHERO

ID	Name	Hero Name	Primary Powers	Teams
1	Bruce Banner	Hulk	Strength, Durability	Avengers
2	Logan	Wolverine	Healing Factor, Adamantium Skeleton	X-men
3	Max Eisenhardt	Magneto	Elemental Control (Magnetism)	Brotherhood of Evil Mutants

DATA TYPES

Each column has a specific data type.

All data in that column has to fit that type.

DATA TYPES

- Number: `TINYINT`, `SMALLINT`, `INT`, `BIGINT`
- Number with decimals: `REAL`, `FLOAT`, `DECIMAL`
- Date: `DATE`, `TIME`, `DATETIME`, `YEAR`, `TIMESTAMP`
- String: `CHAR(LEN)`, `VARCHAR(LEN)`, `TEXT`
- Other: `BLOB`, `BYTEA`

SQL

SQL

Structured **Q**uery **L**anguage is a special purpose programming language designed to manage the data held in relational databases. SQL consists of a data definition language and a data manipulation language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control.

POSTGRES

POSTGRESQL

PostgreSQL, often simply Postgres, is RDBMS with an emphasis on extensibility and on standards-compliance. As a database server, its primary function is to store data securely, supporting best practices, and to allow for retrieval at the request of other software applications.

- Yep, Wikipedia

POSTGRESQL

Setting Up Postgres!

Download Postgres

Download pgAdmin

Do *NOT* forget your password.

SQL COMMANDS

SQL COMMANDS

SQL uses queries to interact with databases. Queries are built from a series of command words and arguments which are interpreted by the RDMS.

example:

```
SELECT * FROM customer WHERE id='12392';
```

The syntax for **keywords** isn't case sensitive. Column and table names are case-sensitive. By convention, SQL keywords words are in all caps. SQL Queries are terminated with a semicolon.

SQL COMMANDS

Common SQL Commands and what they do

CREATE TABLE

`CREATE TABLE` is used to create new tables in a database. When creating new tables, you must also define the columns and the data types for those columns.

```
CREATE TABLE fighter(id SERIAL UNIQUE PRIMARY KEY,
name VARCHAR(20),
weight_kg SMALLINT,
height_cm SMALLINT,
weight_class  VARCHAR(40),
wins SMALLINT,
losses SMALLINT,
draws SMALLINT,
birth_date DATE,
bio TEXT,
salary REAL);
```

SELECT

One of the most common database operations is to retrieve data from a table. This is achieved by using the '`SELECT ... FROM`' keywords along with some qualifying information.

```
SELECT column_name FROM table_name;
```

```
SELECT column_name
FROM table_name;
```

Sometimes you'll see queries written in this way (stacked vertically). This is just to improve readibility. It has no effect on the operation.

SQL SELECT EXAMPLE

```
SELECT hero_name FROM Avengers;
```

```
SELECT hero_name, primary_power  FROM Avengers;
```

AVENGERS

id	name	hero_name	primary_power
1	Bruce Banner	Hulk	Strength
2	Steve Rogers	Captain America	Tactician
3	Thor	Thor	Lightning
4	Tony Stark	Iron Man	Genius
5	Natasha Romanov	Black Widow	Marksman
6	Clint Barton	Hawkeye	Archer

WILDCARD

In order to return all columns, you can use the wildcard (`*`) operator.

```
SELECT * FROM Avengers;
```

WHERE

You can filter your data using the `WHERE` clause.

```
SELECT *
FROM Avengers
WHERE name='Bruce Banner';
```

You can append AND / OR operators to WHERE statements

```
SELECT *
FROM Avengers
WHERE primary_power='Strength' OR name='Clint Barton';
```

ORDER BY

ORDER BY will order our returned data in a specific way.

```
SELECT name, population
FROM cities
WHERE population > 1000000
ORDER BY population DESC;
```

INSERT

`INSERT INTO` is the command we use put data into a table.

```
INSERT INTO table_name ( column_name1, column_name2,... )
values ('value1', 'value2', ... );
```

INSERT INTO

Consider this table:

Avengers

Column Name	Data Type
name	varchar(40)
hero_name	varchar(40)
primary_power	varchar(40)

INSERT INTO

```
INSERT INTO Avengers(name, hero_name, primary_power)
values('Peter Parker', 'Spider-Man', 'Mouthing off');
```

Adds this row to the Avengers table:

Name	hero_name	primary_power
Peter Parker	Spider-Man	Mouthing off

UPDATE ... SET

If we need to modify an existing row in an existing table, we use the `UPDATE` keyword

```
UPDATE Avengers
SET primary_power='Web Slinging'
WHERE hero_name='Spider-Man';
```

The new row would now look like this.

Name	hero_name	primary_power
Peter Parker	Spider-Man	Web Slinging

DELETE FROM

If we need to delete an existing row, we use the `DELETE FROM` keywords

```
DELETE FROM Avengers
WHERE hero_name='Hawkeye';
```

Removes the entire record (all columns) from the database.

CRUD

These basic database operations (SELECT, INSERT, UPDATE, DELETE) are also frequently referred to as **C**reate, **R**ead, **U**pdate, and **D**elete - **CRUD**

Almost every app you're likely to work will be, at its core, be a CRUD app.

POP QUIZ

SQL

POP QUIZ

Get the name of the Avenger who's hero_name is Hawkeye.

POP QUIZ

List all the Avengers in alphabetical order by hero_name.

POP QUIZ

Add a new Avenger.

- name: Cluck Kent
- hero_name: Grant Chirpus
- primary_power: Finding Bugs

POP QUIZ

Change Thor's primary_power to Hammer.

POP QUIZ

Remove Captain America.

COLUMN OPERATORS

TICKETS

id	seat	price	num_sold
1	Box Level	105	4
2	Dress Circle	75	2
3	Main Floor	58	10
4	Mid Balcony	38	0
5	Upper Balcony	19	3

COUNT

```
SELECT COUNT(*) FROM Tickets;
```

id	seat	price	num_sold
1	Box Level	105	4
2	Dress Circle	75	2
3	Main Floor	58	10
4	Mid Balcony	38	0
5	Upper Balcony	19	3

COUNT

```
SELECT COUNT(*) FROM Tickets WHERE num_sold <> 0;
```

id	seat	price	num_sold
1	Box Level	105	4
2	Dress Circle	75	2
3	Main Floor	58	10
4	Mid Balcony	38	0
5	Upper Balcony	19	3

SUM

```
SELECT SUM(num_sold) FROM Tickets;
```

id	seat	price	num_sold
1	Box Level	105	4
2	Dress Circle	75	2
3	Main Floor	58	10
4	Mid Balcony	38	0
5	Upper Balcony	19	3

AVERAGE

```
SELECT AVG(price) FROM Tickets;
```

id	seat	price	num_sold
1	Box Level	105	4
2	Dress Circle	75	2
3	Main Floor	58	10
4	Mid Balcony	38	0
5	Upper Balcony	19	3

MAXIMUM

```
SELECT MAX(num_sold) FROM Tickets;
```

id	seat	price	num_sold
1	Box Level	105	4
2	Dress Circle	75	2
3	Main Floor	58	10
4	Mid Balcony	38	0
5	Upper Balcony	19	3

MINIMUM

```
SELECT MIN(price) FROM Tickets;
```

id	seat	price	num_sold
1	Box Level	105	4
2	Dress Circle	75	2
3	Main Floor	58	10
4	Mid Balcony	38	0
5	Upper Balcony	19	3

COMBO

```
SELECT SUM(num_sold) AS "Total Sold", SUM(price * num_sold) AS "Total Revenue"
FROM Tickets;
```

id	seat	price	num_sold
1	Box Level	105	4
2	Dress Circle	75	2
3	Main Floor	58	10
4	Mid Balcony	38	0
5	Upper Balcony	19	3

RELATIONSHIPS BETWEEN TABLES

TABLE ARE LINKED BY ID

Student

id	name	class_id
1	G. Washington	1
2	M. Gandhi	1
3	N. Mandela	NULL
4	Q. Victoria	2

Class

id	title
1	Front-End
2	Java
3	.NET

JOINS

```
SELECT * FROM Student
JOIN Class ON Student.class_id = Class.id
```

id	name	class_id	id	title
1	G. Washington	1	1	Front-End
2	M. Gandhi	1	1	Front-End
4	Q. Victoria	2	2	Java

INNER JOIN

```
SELECT * FROM Student
INNER JOIN Class ON Student.class_id = Class.id
```

The default join type. Where both tables match.

id	name	class_id	id	title
1	G. Washington	1	1	Front-End
2	M. Gandhi	1	1	Front-End
4	Q. Victoria	2	2	Java

LEFT JOIN

```
SELECT * FROM Student
LEFT JOIN Class ON Student.class_id = Class.id
```

Includes everything in first table, even if it doesn't have a match.

id	name	class_id	id	title
1	G. Washington	1	1	Front-End
2	M. Gandhi	1	1	Front-End
3	N. Mandela	NULL	NULL	NULL
4	Q. Victoria	2	2	Java

RIGHT JOIN

```
SELECT * FROM Student
RIGHT JOIN Class ON Student.class_id = Class.id
```

Includes everything in second table, even if it doesn't have a match.

id	name	class_id	id	title
1	G. Washington	1	1	Front-End
2	M. Gandhi	1	1	Front-End
4	Q. Victoria	2	2	Java
NULL	NULL	NULL	3	.NET

FULL JOIN

```
SELECT * FROM Student
FULL JOIN Class ON Student.class_id = Class.id
```

Includes everything in both tables, even if it doesn't have a match.

id	name	class_id	id	title
1	G. Washington	1	1	Front-End
2	M. Gandhi	1	1	Front-End
3	N. Mandela	NULL	NULL	NULL
4	Q. Victoria	2	2	Java
NULL	NULL	NULL	3	.NET

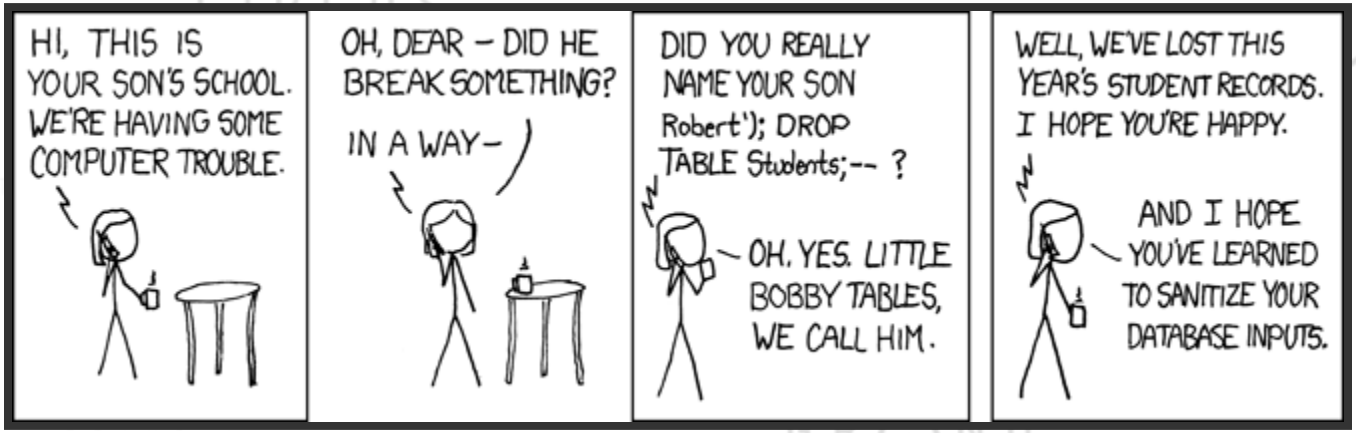
CARTESIAN JOIN

```
SELECT * FROM Student
CROSS JOIN Class
```

All the things!!

Moral of the story: You gotta have the `ON`.

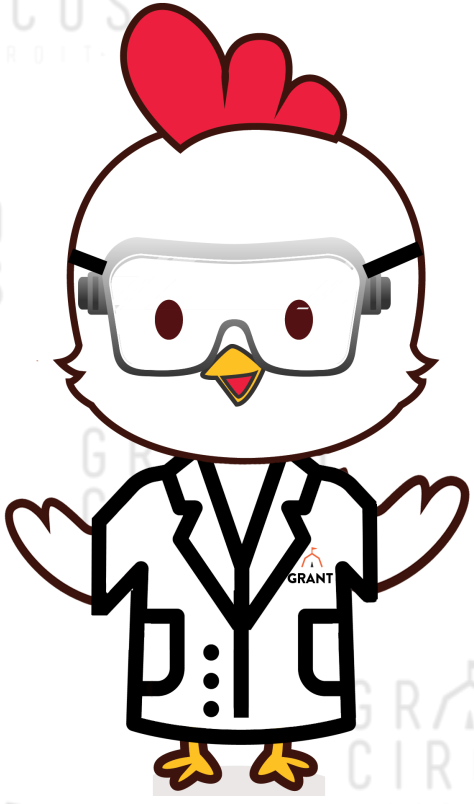
QUESTIONS?



xkcd - A webcomic you should all read.

LAB 23

NORTHWIND SQL QUERIES



LAB

First, install the Northwind database

1. In pgAdmin3, create a database called `northwind`.
2. Open up a SQL window. Copy-paste and run this file...

https://raw.githubusercontent.com/pthom/northwind_psql/master/nor

LAB

Write SQL queries for each of the questions in the workbook. Record these queries in a text document.