Code Nation Master

{CODENATION}™

Why shouldn't you ask a DBA to help you move?

They've been known to drop tables.

By the end of the day that will be hilarious

NoSQL vs SQL

The question is where do you want the pain?

It's all about reading and writing

NoSQL

Easy to write difficult to read

SQL

Easy to read difficult to write

What is SQL?

SQL is a language used for managing and accessing data held in a relational database.

That doesn't help...

Let's see

A relational database is a set of tables structured in columns in rows.

The structure allows relations between pieces of data in separate tables

Think Excel...

name	age	salary
Ross	35	50000
Rachel	33	45000
Monica	33	48000
Chandler	35	?
Phoebe	32	16500
Joey	32	20000

But a bit more complicated...

Name	Age	salary
Ross	35	50000
Rachel	33	45000
Monica	33	48000
Chandler	35	?
Phoebe	32	16500
Joey	32	20000

Name	Love_interest
Ross	Rachel
Rachel	Ross
Rachel	Russ
Monica	Pete
Phoebe	Mike
Joey	Janine

Name	**Address**
Ross	20 Apartment Block
Rachel	20 Apartment Block
Monica	20 Apartment Block
Phoebe	20 Apartment Block
Chandler	20 Apartment Block
Joey	20 Apartment Block
Rachel	19 Apartment Block
Joey	19 Apartment Block
Chandler	19 Apartment Block

Name	**job_title**	**min_salary**	**max_salary**
Chandler	?	0	∞
Monica	Chef	20000	80000
Joey	Actor	10000	1000000
Phoebe	Masseuse	10000	30000

Microsoft Access

MySQL

IBM DB2

PostgreSQL

Oracle

Microsoft SQL Server

SQLite

https://www.codecademy.com/articles/what-is-rdbms-sql

Lets start with the basics

Create our connection

1. Open MySQL Workbench

2. Create a new connection

3. Call the connection name employee and click OK

Create our schema

1. Click create new schema button

2. Name our schema employeedb

3. Click apply (twice)

Import data

1. Right click on the schema and select Table Data Import Wizard

2. Select the CSV to import

3. Make sure data types are correct and create table

Ready for the SeQueL

All a basic SQL query does is **SELECT** which **columns** we want to see **FROM** which **table(s)**. We then want to limit them **WHERE** the **rows** meet certain conditions.

Which table?

```
SELECT *
FROM current_job_detail;
```

Which columns?

```
SELECT employee_id,
       job_title
FROM current_job_detail;
```

Which rows?

This is were we get a few more options...

{CN}™

==

!=

<

<=

>

>=

BETWEEN

NOT BETWEEN

LIKE

NOT LIKE

IN

NOT IN

```
SELECT *
FROM current_job_detail
WHERE employee_id = 1000;
```

```
SELECT *
FROM current_job_detail
WHERE employee_id != 1000;
```

```
SELECT *
FROM current_job_detail
WHERE salary >= 50000;
```

```
SELECT *
FROM current_job_detail
WHERE salary BETWEEN
30000 and 50000;
```

```
SELECT *
FROM current_job_detail
WHERE job_title LIKE
('%evel%');
```

```
SELECT *
FROM current_job_detail
WHERE job_title LIKE
('Devel%');
```

```
SELECT *
FROM current_job_detail
WHERE job_title NOT LIKE
('%evel%');
```

```
SELECT *
FROM current_job_detail
WHERE job_title LIKE
('P_oduct Lead');
```

```
SELECT *
FROM current_job_detail
WHERE job_title IN ('Product
Lead', 'Marketing Lead');
```

Only one constraint isn't very helpful though

```
SELECT *
FROM current_job_detail
WHERE salary <= 35000
AND job_title = 'Developer' ;
```

1) Look in each of the tables and work out what information is in there
2) Return a table of all of the tech leads
3) Return a table of all of the female employees
4) Return a table of all the employees that name starts with an S
5) Return a table of all the employees that have ever been a developer
6) Return a table of all the laptop ids that run Ubuntu as an OS

Extension

1) Return a table of all the employees whose name starts with A or S
2) Return a table of all the employees born in the 80s

```
CREATE TABLE
my_favourite_employees (
employee_id    int PRIMARY KEY,
job_title              varchar(64));
```

DANGER ZONE

```
DROP TABLE my_favourite_employees;
```

```
INSERT INTO my_favourite_employees
SELECT employee_id,
       job_title
FROM   current_job_detail
WHERE  employee_id in (1001, 1002)
```

DANGER ZONE

```
DELETE FROM my_favourite_employees
WHERE employee_id = 1001
```

1) Create a table called great_names with 2 columns name and employee_id
2) Insert 5 employees with great names into your table (your choice*)
3) Delete one of the employees out of your table based on their job title

Extension

1) Recreate your table with an extra column called great_name_ind
2) Insert 5 employees into your table and set the value of great_name_ind to ‘Y’
3) Change one of the rows in your table so the great_name_ind = ’N'

* Sam wins brownie points

What makes relational databases a bit different to NoSQL

Relations?

Primary vs Foreign

Would student_name be a good primary key?

Would student_id* be a better primary key?

* a unique id given to each student when they start

https://en.wikipedia.org/wiki/Relational_database

```
SELECT *
FROM current_job_detail
INNER JOIN employee_detail
ON current_job_detail.employee_id =
employee_detail.employee_id
```

Eurgh...

Introducing aliases

```
SELECT *
FROM current_job_detail cjd
INNER JOIN employee_detail ed
ON cjd.employee_id = ed.employee_id
```

FROM current_job_detail cjd

DANGER ZONE

```
SELECT *
FROM current_job_detail cjd
INNER JOIN jobs_history jh
ON cjd.employee_id = jh.employee_id
```

Why is this dangerous?

One to One relationships

One to Many relationships

Many to Many relationships

Try this

```
SELECT employee_id
FROM current_job_detail cjd
INNER JOIN employee_detail ed
ON cjd.employee_id = ed.employee_id
```

```
SELECT cjd.employee_id
FROM current_job_detail cjd
INNER JOIN employee_detail ed
ON cjd.employee_id = ed.employee_id
```

1) Return a table linking laptop_detail and current_job_detail
2) Return a table of only the employees that own a Mac
3) Return a table of all the employees that were an apprentice developer but are now a developer
4) Return a table of all the employees that weren't a developer and now are

Extension

1) Return a table of all the employees that have had more then one job title (not using aggregates)
2) Look in your table, you may have duplicates. Remove them.

Aggregating

So far we've only returned the values in a table

But what if I want to know an average salary?

AVG

COUNT

SUM

MIN

MAX

```
SELECT MAX(salary)
FROM current_job_detail;
```

What about the max salary for each job title?

Introducing GROUP BY

```
SELECT job_title,
       MAX(salary)
FROM current_job_detail
GROUP BY job_title;
```

```
SELECT job_title,
       AVG(salary)
FROM current_job_detail
GROUP BY job_title;
```

```
SELECT job_title,
       MIN(salary)
FROM current_job_detail
GROUP BY job_title;
```

```
SELECT job_title,
       SUM(salary)
FROM current_job_detail
GROUP BY job_title;
```

COUNT is a little different

```
SELECT job_title,
       COUNT(*)
FROM current_job_detail
GROUP BY job_title;
```

1) Return a table of the max salary by job type
2) Return a table counting how many people have each OS
3) Return a table of the average salary of staff members that have at some point been an apprentice developer

Extension

1) Return a row of data containing the name of the person with the highest salary (don't just eyeball the table and select an employee id)
2) Do the same for the highest salary by job type