SQL Basics (1)

- so once we have a database, we need to be able to use it and make queries
- for these we use SQL - **Structured Query Language**

What is SQL?

- it is a Query langauge (not exactly a programming langauge) which was developed for asking questions about databases back in. 1974
- was standadised in the 1986/1987s
    - the most dominant langauge for quereies today still

It is not a general purpose programming lanaguge

- not Turing Complete (you couldnt encode every possible calculation in it)
- has a strange syntax - similar to English
    - case-insensitive

Standadisation

- in practise, every database engine will have differences to other database engines
    - some differences being quite substantial
    - some have differences in performace for example:
        - **SQLite is good with strings**, most others are better with numbers
- managing these differences used to be a degree/job in of itself
    - E.g. designing and managing the database

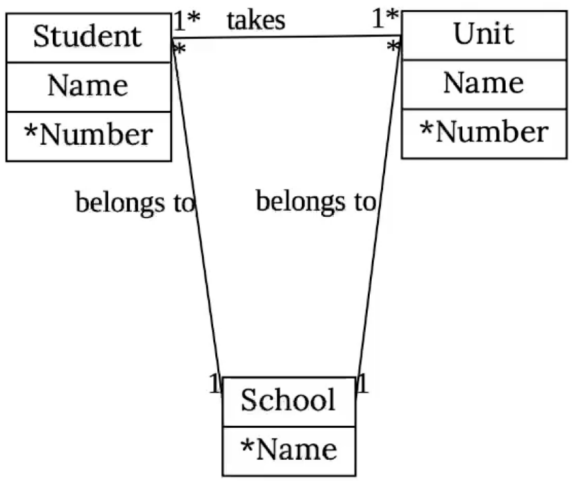
    - Database Engineers
- now software engineering has absorbed this role

Joe's lectures will stick to SQLite syntax

- different database engines will have differences...

Creating Tables

In the last lecture we had the following Entity relationship diagram:



- i.e. a student takes one or more units, and a unit takes 1 or more students
    - a student belongs to 1 school, a school has many students
    - a unit belongs to one school, a school has many units
- lets build this diagram in SQL:

```
// below creates the table for the Student:

CREATE TABLE IF NOT EXISTS student (
    name TEXT NOT NULL,  // this is going to be text and it wont
    number TEXT NOT NULL,
    PRIMARY KEY (number));

// now create the table for the unit:

CREATE TABLE IF NOT EXISTS unit (
    name TEXT NOT NULL,
    number TEXT NOT NULL,
    PRIMARY KEY (number));

// for the school:

CREATE TABLE IF NOT EXISTS school (
    name TEXT NOT NULL,
    PRIMARY KEY (name));

// for the register
// this will store which students are taking what units

CREATE TABLE IF NOT EXISTS class_register (
    student TEXT NOT NULL,
    unit TEXT NOT NULL,
    FOREIGN KEY (student) REFERENCES student(number),
    FOREIGN KEY (unit) REFERENCES unit(name),
    PRIMARY KEY (student, unit));
```

- you dont HAVE to provide foreign key relationships but it is helpful sometimes for yourself
    - many database engines ignore them (e.g. MariaDB)

Deleting Tables

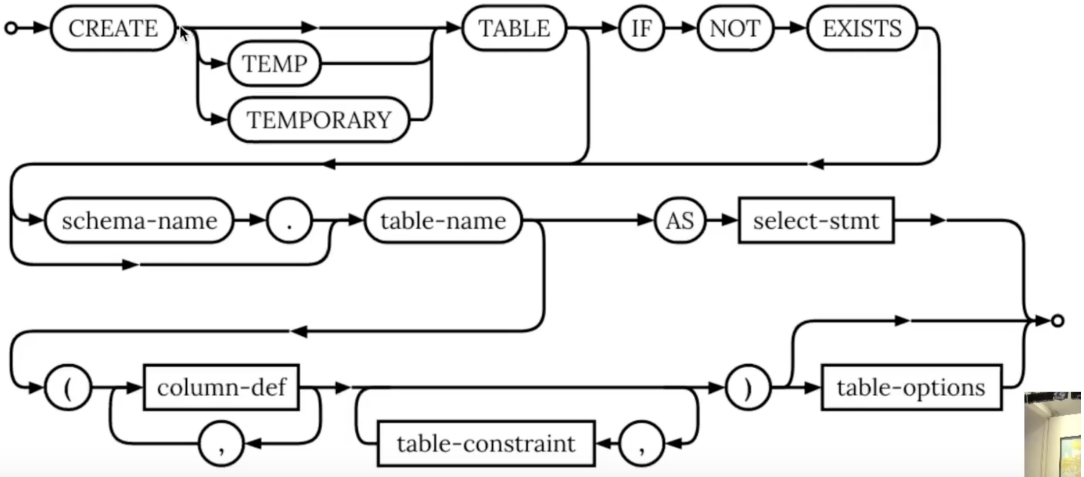
```
DROP TABLE IF EXISTS class_register;
DROP TABLE IF EXISTS student;
DROP TABLE IF EXISTS unit;
DROP TABLE IF EXISTS school;
```

SQLite Documentation Page

Syntax, syntax, syntax

If you go on the *SQLite* documentation page...

- ...you can find syntax diagrams for all of SQL!
- `https://www.sqlite.org/lang_createtable.html`



Types

When creating the fields in our database we made them all of type TEXT...

- What other types exist?

INTEGER	whole numbers
REAL	lossy decimals
BLOB	binary data (images/audio/files...)
VARCHAR(10)	a string of 10 characters
TEXT	any old text
BOOLEAN	True or false
DATE	Today
DATETIME	Today at 2pm

But really types

Databases sometimes *simplify* these types

- SQLite makes the following tweaks...

INTEGER	whole numbers
REAL	lossy decimals
BLOB	binary data (images/audio/files...)
VARCHAR(10)	*actually TEXT*
TEXT	any old text
BOOLEAN	*actually INTEGER*
DATE	*actually TEXT*
DATETIME	*actually TEXT*

(others may exist... *read the manual!*)

Table Constraints

- when we created the tables earlier we specified that some columns to be NOT NULL
- others were PRIMARY KEY or FOREIGN KEY etc
- these are known as **constraints**

Here are a few common constraints:

`NOT NULL`	can't be `NULL`
`UNIQUE`	can't be the same as another row
`CHECK`	arbitrary checking (including it conforms to a regular expression)
`PRIMARY KEY`	unique, not `NULL` and (potentially) autogenerated
`FOREIGN KEY`	(IGNORED BY MARIADB) other key must exist

- **SQLite wont actually enforce these types or constaints unless you ask it to**
  - Use **STRICT** keyword when creating the table

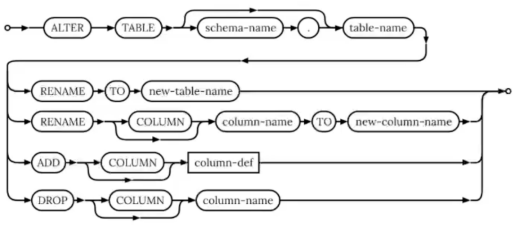
Altering tables

Can you add constraints after table creation?

- yes

Yes with the `ALTER TABLE` statement

- But often easiest just to save the table somewhere else
- Drop the table
- Reimport it



- **alter table might not be easiest with massive databases**

Adding Data to a Table

- use INSERT INTO

```
INSERT INTO unit(name, number)
VALUES ("Software Tools", "COMS100012");
```

- in the above examle, we are instering into the unit table the values “Software Tools” and the unit code “COMS10012” representing the name and number values respectively

Querying the Data

- how do we ask questions and find relationships in databases
- this example will use an old iTunes library

Selecting Rows from a Table

- basic command for this is `SELECT`

AlbumId	Title	ArtistId
1	For Those About To Rock We Salute You	1
2	Balls to the Wall	2
3	Restless and Wild	2
4	Let There Be Rock	1
5	Big Ones	3

- the `*` will select all rows form the table - can also be more specific
- `LIMIT 5` will only list 5 things - limits are completely optional

- inside the Album table you have a the AlbumID, the Title and AristId

Then if we see whats in the Artist Table:

```
SELECT * FROM artist
LIMIT 5;
```

ArtistId	Name
1	AC/DC
2	Accept
3	Aerosmith
4	Alanis Morissette
5	Alice In Chains

Ideally we would like to view these two tables as one...

Joining Tables

- use the `JOIN` statement

```
SELECT *
FROM album
JOIN artist
ON album.artistid = artist.artistid
LIMIT 5;
```

AlbumId	Title	ArtistId	ArtistId	Name
1	For Those About To Rock We Salute You	1	1	AC/DC
2	Balls to the Wall	2	2	Accept
3	Restless and Wild	2	2	Accept
4	Let There Be Rock	1	1	AC/DC
5	Big Ones	3	3	Aerosmith

- we're going to join the tables if the Album artist id is the same as the artist id

- this will match albums to artists

Reducing Unnecessary Columns

- say we dont want the id colums

```
SELECT album.title, artist.name
FROM album
JOIN artist
ON album.artistid = artist.artistid
LIMIT 5;
```

Title	Name
For Those About To Rock We Salute You	AC/DC
Balls to the Wall	Accept
Restless and Wild	Accept
Let There Be Rock	AC/DC
Big Ones	Aerosmith

Renaming Columns

- say the we need to rename the columns to provide more clarity or context
- The `AS` keyword lets you do this

```
SELECT album.title AS album,
            artist.name AS artist
FROM album
JOIN aritst
ON album.artistid = artist.artistid
LIMIT 5;
```

album	artist
For Those About To Rock We Salute You	AC/DC
Balls to the Wall	Accept
Restless and Wild	Accept
Let There Be Rock	AC/DC
Big Ones	Aerosmith

Asking Specific Questions about the Data

- say we want to find all of the albums that contain the word “Rock” in the title
- we can use a `SELECT` statement with an extra clause, the `WHERE` command:

```
SELECT album.title AS album,
            artist.name AS artist
FROM album
JOIN artist
ON album.artistid = artist.artistid
WHERE album LIKE '%Rock%'
LIMIT 5;
```

- *where the album is like the string* basically

album	artist
For Those About To Rock We Salute You	AC/DC
Let There Be Rock	AC/DC
Deep Purple In Rock	Deep Purple
Rock In Rio [CD1]	Iron Maiden
Rock In Rio [CD2]	Iron Maiden

so which artists who put out albums with Rock in the title?

```
SELECT artist.name AS artist
FROM album
JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
LIMIT 5;
```

artist
AC/DC
AC/DC
Deep Purple
Iron Maiden
Iron Maiden

However this table contains duplicate artists as they have numerous albums with rock in the title

- its showing the artists however many times thy are in the database

If we want to remove duplicates:

```
SELECT DISTINCT artist.name AS artist
FROM album
JOIN aritst
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
LIMIT 5;
```

- the `DISTINCT` keyword simply says: if there is a duplicate, dont include it in the new list

artist
AC/DC
Deep Purple
Iron Maiden
The Cult
The Rolling Stones

say we now wanted to count how many rock albums each artist has produced...

so we want to GROUP BY artist and COUNT the albums:

```
SELECT artist.name AS artist,
            COUNT(album.title) as albums

FROM album
JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
GROUP BY artist
LIMIT 5;
```

- the GROUP BY statement will select from artists, as there could be different artists who use the same album title

artist	albums
AC/DC	2
Deep Purple	1
Iron Maiden	2
The Cult	1
The Rolling Stones	1

Ordering the List

Lets group by artist and count the albums...

- And order it by album count!

```
SELECT artist.name AS artist,
       COUNT(album.title) as albums
FROM album
JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
GROUP BY artist
ORDER BY albums DESC
LIMIT 5;
```

artist	albums
Iron Maiden	2
AC/DC	2
The Rolling Stones	1
The Cult	1
Deep Purple	1

- just add another clause `ORDER BY`
  - and make it descending, can also use ASC to make it go in ascending order