SQL lite notes

Get the data.

```
mkdir swc_sql
cd swc_sql
wget http://files.software-carpentry.org/survey.db
```

Setup SQLite within the terminal.

```
select * from site;
.mode column
.header on
select * from site;
```

Motivation

In the late 1920s and early 1930s, William Dyer, Frank Pabodie, and Valentina Roerich led expeditions to the Pole of Inaccessibility in the South Pacific, and then onward to Antarctica. Two years ago, their expeditions were found in a storage locker at Miskatonic University. We have scanned and OCR'd the data they contain, and we now want to store that information in a way that will make search and analysis easy.

Lesson 1.

Objectives.

- Relational database description
- Tables that have columns (fields) and rows (data) or records

You have used relational databases all you life - e.g the internet but through an interface, and these interfaces will at some point start runnning SQL commands to access and store databases.

SQL is the language to interface with these, it's declarative for the most part. You ask for what, it does the how. Although R with all it's packages can be a little bit like this.

- Database managers, Oracle, MySQL, Microsoft access, SQLIte do their own thing, analogy with GIT.

SHOW website with the databases.

Return to missing values later.

Let's look at our tables

```
.schema
.schema site
```

Ok let's select shit from them.

```
.schema person;
SELECT family, personal FROM person;
```

It's case insensitive e.g doesn't care.

```
SeLeCT FaMILy, PeRSONal, from PERson;
```

Conventions are usually to capitilize the key words, e.g uppercase for SQL statments.

but it doesn't really matter a question at the end of this section will ask for your preference.

Order doesn't matter it's up to you to specify, here we show column order later we will show how to do row order.

```
SELECT personal, family, FROM person;
```

Can also repeat

```
SELECT ident, ident, ident from person;
```

And finally, just like bash we can use the * to match anything.

```
SELECT * from person;
```

Answers

1. select name from site;
2. Discussion of the answer, get students to articulate a preference.

Lesson 2

Motivations - What kind of quantitiy measurments were taken at each site; - Which scientists took measurments on the expidition - the sites where scientists took meausuremnts.

Brain does not store information ...

```
.schema survey;
select quant for survey;
```

Redudancy galore.

NEWCOMMAND DISTINCT

Introduce a key word DISTINCT explain how it returns only basically uniq

```
SELECT DISTINCT taken, quant from survey;
```

NEWCOMMAND ORDER BY the same as the uniq sort basically.

Next identify the scientists measurements on the expidition.

```
SELECT * from person;
SELECT * FROM person ORDER BY ident;
```

That was an ascending sort NEW WORD DESC Could also be more explicit for instance.

```
SELECT * from person ORDER BY ident desc;
SELECT * from person ORDER BY ident asc;
```

Just like in bash, we can begin to combine these ideas, to perform more complex operations

So to determine which scientists meausured which quantities at each site, we can look again at the survey table.

```
SELECT taken, person, quant FROM SURVEY ORDER BY taken ASC, person DESC;
```

Some scientists may have specialised in certain kinds of measurements, here we return to the concept of distinct.

```
SELECT quant, person FROM SURVEY ORDER by quant ASC;
SELECT DISTINCT quant, person FROM SURVEY ORDER BY quant ASC;
```

Answers

1. d

```
SELECT DISTINCT dated FROM visited;
SELECT personal, family FROM person ORDER BY family ASC;
SELECT DISTINCT taken, person, quantity FROM survey;
```

Lesson 3

So we can filter columns using SELECT decide on a table etc but what if we are only interested in certain values, we need to be blate to filter the rows.

NEWCOMMAND WHERE clause.

```
SELECT * FROM visited;
SELECT * FROM visited where site='DR-1';
```

Can filter on columns that are not displayed.

Below I will explain the order of operation interactively.

```
SELECT * FROM visited;
SELECT * from visited where site='DR-1';
SELECT ident from visited where site='DR-1';
```

The return of the boolean these can also be combined.

NEWCOMMAND AND

only if both statements are true will it continue

```
SELECT * FROM visited WHERE site='DR-1' AND dated<'1930-01-01';
```

Dates are complicated !!! very very complicated. Most DBM have them, but you will have some special support for them but SQLite doesn't. So you can store them as text a number or .

measurments by lake or roe we can use the OR

NEWCOMMAND OR

if one statment is true it will continue.

```
SELECT * FROM survey WHERE person='lake' OR person='roe';
```

Parallels with R and the IN clause found there.

```
SELECT * FROM survey WHERE person IN('lake','roe');
```

COMBINATION STYLE.

```
SELECT * FROM survey WHERE quant='sal' AND person='LAKE' or person='roe';
```

Funny problem now the order matters

```
SELECT * FROM survey WHERE quant='sal' AND (person ='lake' OR person='roe');
```

We can also do partial matches, such as

```
SELECT * FROM visited WHERE site LIKE 'DR%'
```

Then distinct and WHERE to give a second level of filtering.

COMBINATION STYLE

```
SELECT person, quant FROM survey WHERE person='lake' OR person='roe';
SELECT DISTINCT person, quant FROM survey WHERE person='lake' OR person='roe';
```

See our queries are growing and growing.

Distinct is doing stuff.

If your db is large sometimes it is best to create a test db for performing test-queries on.

Answers

```
SELECT * FROM site WHERE (lat > -60) or (lat < 60);
SELECT * FROM survey WHERE quant = 'sal' AND reading <0.0 AND reading > 1.0;
SELECT * FROM survey WHERE person = 'pb' AND (quant = 'temp' OR reading > 5) OR person IN('r
```

Lesson 4

New values.

We need to increase a number 5% errors in teh expidition logs.

```
SELECT 1.05 * reading FROM survey WHERE quant='rad';
```

We can do the usual temp conversion.

```
SELECT taken, round(5*(reading-32)/9,2) FROM survey WHERE quant='temp';
```

Can also concatenate strings, to create new fields from other fields.

```
SELECT person || ' ' || family FROM person;
```

Answers

```
SELECT * from survey where person ='val';
SELECT reading/100 from survey where person ='val';
```

Unions.

```
SELECT * FROM person WHERE ident='dyer' UNION SELECT * FROM person WHERE ident='roe';
```

Answers

```
SELECT * FROM survey WHERE person = 'pb' AND (quant = 'temp' OR reading > 5) UNION SELECT *
```

Lesson 5

- Explain how databases represent missing information
- Explain 3-valued logic that databases use when manipulating missing info
- Write queries that handle missing information correctly.

It's special it actually means nothing.

```
select * from visited;
```

It doesn't behave like other values.

```
SELECT * FROM visited WHERE dated <'1930-01-01';
```

and the greater than

```
SELECT * FROM visited WHERE dated >='1930-01-01';
```

How do we get the nulls then !?!?!?!?

```
SELECT * FROM visited WHERE dated=NULL;
SELECT * FROM visited WHERE dated != NULL;
SELECT * FROM visited WHERE dated IS NULL;
```

OR the reverse.

```
SELECT * FROM visited WHERE dated is NOT NULL;
```

Null cause problems every where they go!

Interested in salinity only

```
SELECT * FROM survey WHERE quant='sal' AND (person != 'lake' OR person IS NULL);
```

In contrast to this aggregation functions that combine multiple values, such as min, max or avg, ignore null values.

Answers

```
select * from visited order by date;
```

Lesson 6

We can also calculate ranges and averages for our data. We'll start simple as always say we know how to select all of the dates from.

```
SELECT dated from Visited;
```

But to combine them we need to aggregate them using functions.

NEWCOMMANDS

MIN and MAX.

```
SELECT min(dated) FROM visited;
SELECT max(dated) FROM visited;
```

NEWCOMMANDS

AVG, COUNT, and SUM

```
SELECT avg(reading) FROM survey WHERE quant='sal';
SELECT count(reading) FROM survey WHERE quant='sal';
SELECT sum(reading) FROM survey WHERE quant='sal';
```

Can use count on anything, just counts the rows.

```
SELECT count(*) from survey WHERE quant='sal';
```

Muliple aggregations

```
SELECT min(reading), max(reading) FROM survey WHERE quant='sal' AND reading <= 1.0;
```

Combine the aggregated results, although the output may suprise you.

```
SELECT person, count(*) FROM survey WHERE quant='sal' AND reading<=1.0;
```

When there are now values, the result is don't know rather than zero or some arbitrary values

```
SELECT person, max(reading), sum(reading) FROM survey WHERE quant='missing';
```

They totally ignore the nulls, so everything is fine.

```
SELECT date from visited;
SELECT min(dated) from visited;
```

Instead of filtering explicitly.

```
SELECT min(dated) FROM visited WHERE dated IS NOT NULL;
```

Systematic bias.

```
SELECT person, count(reading), round(avg(reading), 2)
FROM survey WHERE quant='rad';
```

Doesn't work shat the bed.

Could do individual.

But

NEWCOMMAND GROUP BY

```
SELECT person, count(reading) round(avg(reading), 2)
FROM survey WHERE quant='rad' GROUP BY person;
```

Can use mutiple

```
SELECT person, quant, count(reading), round(avg(reading), 2)
FROM survey GROUP BY person, quant;
SELECT person, quant, count(reading), round(avg(reading), 2)
FROM SURVEY
WHERE PERSON IS NOT NULL
GROUP by person, quant
ORDER by person, quant;
```

Lesson 7 - Joins

In order to submit her data to a web site that aggregates historical meteorological data, Gina needs to format it as latitude, longitude, date, quantity, and reading. However, her latitudes and longitudes are in the Site table, while the dates of measurements are in the Visited table and the readings themselves are in the Survey table. She needs to combine these tables somehow. Explain how joins work.

```
SELECT * FROM site JOIN visited;
```

It's just given us the cross-product –not interesting.

Need to filter on stuff that makes sense.

Got to find the corresponding stuff.

```
SELECT * FROM site JOIN visited ON site.name=visited.site;
SELECT Site.lat, Site.long, Visited.dated
FROM   Site JOIN Visited
ON     Site.name=Visited.site;
```

Many joins, let's make many many joins.

```
SELECT Site.lat, Site.long, Visited.dated, Survey.quant, Survey.reading
FROM   Site
JOIN   Visited ON Site.name=Visited.site
JOIN   Survey ON Visited.ident=Survey.taken
WHERE  Visited.dated IS NOT NULL;
```

PKs and FKs

person.ident and survey.person.

Answers

```
select * from visited join site on site.name=visited.site join survey on visited.ident=su
select * from visited join site on site.name=visited.site and visited.ident=survey.taken
```