REFINING SELECTIONS

DISTINCT → don't repeat eg. only give the "book title" once.

> SELECT DISTINCT auth_lastname FROM books

OR a combination eg

> SELECT DISTINCT auth_firstname, auth_lastname FROM books

ORDER BY

> SELECT name FROM books ORDER BY pubdate DESC (descending)

> SELECT title, name FROM books ORDER BY 2 → (Number refers to name)

LIMIT

> SELECT title, pubdate FROM books ORDER BY pubdate DESC LIMIT 5;

(could be LIMIT 3,7)

LIKE

→ WHERE name_first LIKE "%hom%"

→ WHERE stock_quantity LIKE "____"

So Thomas Lhomey would work. wildcards

"%_%%" to select for % itself

COUNT eg > SELECT COUNT(*) FROM books;

> SELECT COUNT(auth_fname) FROM books;

COUNT + DISTINCT

> SELECT COUNT(DISTINCT auth_fname) FROM books

GROUP BY to "blob" stuff together eg.

> SELECT title, authofirst FROM books GROUP BY authofirst

or to count the number of books written by someon with a certain ~~first~~ last name

> SELECT ~~authfirst~~ auth_last, COUNT(*) FROM books GROUP BY auth_last;

MAX & MIN eg > SELECT MIN(pubdate) FROM books

Combine SELECTS

> SELECT * FROM books WHERE pages = (SELECT MIN(pages) FROM books);

OUTPUT SAME

> SELECT * FROM books ORDER BY pages ASC LIMIT 1;

AS 'date ~~first~~ book published'

MIN + GROUP BY

> SELECT auth_fname, auth_lname, MIN(pubdate) FROM books GROUP BY auth_lname, auth_fname;

SUM > SELECT SUM(pages) FROM books;

+ group by > SELECT auth_fname, auth_lname, SUM(pages) FROM books GROUP BY (auth_lname, auth_fname)

note: This is the right way around as surname is priority

AVG average > SELECT AVG(pubdate) FROM books

> SELECT AVG(stock_quant) FROM books GROUP BY (pubdate)

DATA TYPES

CHAR(20) → Fixed, must be 20, always 20, → right padding.

Postcodes, Male/Female Flags

VARCHAR(20) * use Mostly

up to 20 Max of 255 characters

Numbers — **INT** → 31

DECIMAL eg DECIMAL(5,2) like 124.73

FLOAT 4 Bytes ~7 digits | DOUBLE 8 bytes ~15 digits

DATE YYYY-MM-DD | **TIME** HH:MM:SS | **DATETIME** YYYY-MM-DD HH:MM:SS | TIMESTAMP (posts & tweets)

CURDATE() current date | CURTIME() current time | NOW() current date/time

DAY() DAYNAME() DAYOFWEEK() DAYOFYEAR()

MySQL 5.7 reference

DATE_FORMAT(date, format)

eg > SELECT DATE_FORMAT('2009-10-04 22:12:06', '%D %y %b')

%M month name, %W weekday, etc etc.

LOGICAL OPERATORS

> SELECT title FROM books WHERE year != 2017; — != not equal

> SELECT title FROM books WHERE title NOT LIKE 'W%'; "doesnt start with W" — NOT LIKE

> SELECT * FROM book WHERE year > 2000 — > greater than

> SELECT * FROM book WHERE year >= 2000 — < less than

> SELECT 99 > 1 → TRUE (1)

> SELECT 'a' = 'A' → TRUE (1) (no capitals in SQL) — OUTPUTS

> SELECT * FROM books WHERE name = 'Bob' AND pubdate >= 2000 — AND, &&

> SELECT * FROM books WHERE name_last = 'smith' || pubdate = 1977 — ||, OR

> SELECT title, pubdate FROM books WHERE pubdate BETWEEN 2002 AND 2005 — "BETWEEN" Also "NOT BETWEEN"

> SELECT title, auth_lname FROM books WHERE auth_lname IN ('Smith', 'Baker', 'Jones') — IN (= or = or =)

also "NOT IN"

> SELECT title, stock_quantity, CASE WHEN stock_quant <= 50 THEN '*' WHEN stock_quant <= 100 THEN '**' ELSE '***' END AS stock

Structured Query Language

CREATE DATABASE & TABLES

> CREATE DATABASE ____;
> DROP DATABASE ____;
> USE ____;
> SELECT database();

Database ⇒ lots & lots of tables!

eg

Name	breed	Age
Blue	Tabby	3
Rocket	Persian	~~ten~~

'Cat' table

must be variable characters

must be an integer number

Many variable data types

eg numbers → INT, DECIMAL, FLOAT, DOUBLE, BIT etc etc.

string → CHAR, VARCHAR, BLOB, TEXT, TINY TEXT

Date types → DATE, DATETIME, TIMESTAMP, TIME, YEAR

eg Name VARCHAR(100)
Age INT

> CREATE TABLE cats
(name VARCHAR(100),
age INT
);

> SHOW TABLES;
or
> SHOW COLUMNS FROM cats;
OR
DESC cats;

> DROP TABLE cats;

> INSERT INTO cats (name, age)
VALUES ('Blue', 3)
('Rocket', 10);

> SELECT * FROM cats;

>SHOW WARNINGS

NULL ≠ ∅! & DEFAULT

>CREATE TABLE cats2
(
name VARCHAR(100) NOT NULL DEFAULT 'unnamed',
age INT NOT NULL DEFAULT 99
);

INSERTING DATA + others

> INSERT INTO cats (name, age)
VALUES ('Jetson', 7);

can change order if values order changes too.

> INSERT INTO cats (breed, age)
VALUES ('Maggy', 8)

will use default 'unamed' for name.

PRIMARY KEY → unique ID eg.

>CREATE TABLE cats3
(cat_id INT NOT NULL AUTO_INCREMENT,
name VARCHAR(100) NOT NULL DEFAULT 'unnamed',
age INT NOT NULL DEFAULT 99,
PRIMARY KEY (cat_id)
);

LOGING ON TO MYSQL

>mysql -u root -p
then enter password eg 'tom'
for XAMP default password is 'root'
to start mysql server >
for version: > mysqladmin -v

CRUD — Create Read Update Delete

create → INSERT INTO !!

Read → SELECT eg >SELECT * FROM ____;
or > SELECT name FROM cats;
Where:- SELECT * FROM cats
WHERE age = 4;

using Aliases:- SELECT cat_id AS id,
name FROM cats;
change column "label" on output

update → UPDATE cats SET breed = 'Shothair'
WHERE breed = 'Tabby';
[SELECT Before you Update to check!]

Delete DELETE FROM cats WHERE name='mo';
[again use SELECT first to check!]

SQL files

.sql → for Kate, VIM gedit etc
> SOURCE filename.sql

Concatenate

To put strings together

> CONCAT (column-x, column-y)
or
> CONCAT ('the name is ', name, 'the age is ', age);

eg > SELECT CONCAT (auth_firstname, ' ', auth_lastname) FROM books;

Substrings

> SELECT SUBSTRING ('hello World', 3, 4);
or SUBSTRING
⇓
hell

>SELECT SUBSTRING ('hello World', 7);
⇓
World

Replace

>SELECT REPLACE ('Hello World', 'Hell', '%$#0');
⇓
%$#0 World

Reverse

> SELECT REVERSE ("Hello World");
⇓
dlroW olleH

Char length

> SELECT CHAR_LENGTH ("hello cat")
⇓
11

upper Case/lower Case

> SELECT UPPER ('Hello World') ⇒ "HELLO WORLD"

~~Express~~ EXPRESS.JS

(3)

See expressjs.com → hello world example app

```
const express = require("express");
const app = express()
app.get("/", function(req, res){
  res.send("Hello World!")
});
app.listen(3000, function(){
  console.log("Listening to port 3000!")
});
```

app.js

request → get http

response

port.

The root of the home page.

Adding a "random num" page.

```
app.get("/random_num", function(req, res){
  var num = Math.floor((Math.random() * 10) + 1);
  res.send("your lucky number is " + num);
});
```

EXAMPLE — NOT COMPLETE!! (4)

```
app.get("/", function(req, res){
  var q = 'SELECT COUNT(*) AS count FROM users'
  connection.query(q, function(error, results){
    if(error) throw error;
    var msg = "We have " + results[0].count + " users";
    res.send(msg);
  });
});
```

needs to go into HTML

HTML markup — ("join us" app) (5)

```
<h1>Join Us</h1>
<p class='lead'>Enlist email to join 5(8)</p>
<form method='post' action='/register'>
  <input type='text' class='form' placeholder='Enter email'>
  <button>Join Now</button>
</form>
```

need to inject ejs here!

(6) we'll use EJS so

```
>npm install --save ejs
```

back in app.js ensure ejs is the view engine

```
app.set("view engine", "ejs");
```

Put before var connection=

also in "views" directory

views directory

EJS uses this with a file called "home.ejs"

views → create home.ejs file

join us → create directory → views → create this

app.js

node_modules

package.json

(7) NEED to also create the "join us" mysql database & "user" tables etc → can do in mysql) CREATE DATABASE joinus dcl or probably via a script through connection.query ("CREATE DATABASE ... etc etc

express + faker + ejs

app.js

```
var express = require('express');
var mysql = require('mysql');
var bodyParser  = require("body-parser");
var app = express();

app.set("view engine", "ejs");
app.use(bodyParser.urlencoded({extended: true}));
app.use(express.static(__dirname + "/public"));

var connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'learnwithcolt',
  database : 'join_us'
});

app.get("/", function(req, res){
    // Find count of users in DB
    var q = "SELECT COUNT(*) AS count FROM users";
    connection.query(q, function(err, results){
        if(err) throw err;
        var count = results[0].count;
        res.render("home", {count: count});
    });
});

app.post("/register", function(req, res){
    var person = {
        email: req.body.email
    };
    connection.query('INSERT INTO users SET ?', person,
    function(err, result) {
        if (err) throw err;
        res.redirect("/");
    });
});

app.listen(8080, function(){
    console.log("Server running on 8080!");
});
```

home.ejs

```
<head>
    <link href="https://fonts.googleapis.com/css?family=Roboto:100,300,400" rel="stylesheet">
    <link rel="stylesheet" href="/app.css">
</head>

<body>
    <div class="flex-container">
    <div class="container">
        <h1>JOIN US</h1>
        <p class="lead">Enter your email to join
        <strong><%= count %></strong> others on
        our waitlist. We are 100% not a cult. </p>
        <form method="POST" action='/register'>
            <input type="text" class="form"
            name="email" placeholder="Enter Your
            Email">
            <button>Join Now</button>
        </form>
    </div>
</div>
</body>
```

ONE TO MANY

CUSTOMER
- customer_id
- first_name
- last_name
- email

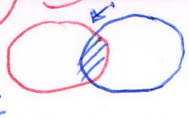
PRIMARY KEY

ORDER
- order_id — PRIMARY KEY
- order_date
- amount
- customer_id → FOREIGN KEY

so order table look like

```
CREATE TABLE orders(
  id INT AUTO_INCREMENT PRIMARY KEY,
  order_date DATE,
  customer_id INT,
  FOREIGN KEY (customer_id)
  REFERENCES (customer_id)
```

INNER JOIN



Explicit INNER JOINS

```
> SELECT * FROM customers
  JOIN orders
    ON customers.id =
       orders.customer_id;
```

LEFT JOIN

```
> SELECT * FROM customers
  LEFT JOIN orders
    ON customers.id = orders.customer_id;

  GROUP BY customers.id
  ORDER BY total_spent;
```

RIGHT JOIN also possible

```
SELECT first_name,
       Ifnull(Avg(grade), 0) AS average,
       CASE
         WHEN Avg(grade) IS NULL THEN 'FAILING'
         WHEN Avg(grade) >= 75 THEN 'PASSING'
         ELSE 'FAILING'
       end                          AS passing_status
FROM   students
       LEFT JOIN papers
              ON students.id = papers.student_id
GROUP  BY students.id
ORDER  BY average DESC;
```

```
SELECT
    first_name,
    IFNULL(title, 'MISSING'),
    IFNULL(grade, 0)
FROM students
LEFT JOIN papers
    ON students.id = papers.student_id;
```

Relationships

1 to 1	Customer to details
1 to many	Customer to order
many to many	Book to authors

MANY TO MANY

example

Setting up Tables.

Table 1

```
CREATE TABLE series(
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100),
    released_year YEAR(4),
    genre VARCHAR(100)
);
```

Table 2

```
CREATE TABLE reviewers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100)
);
```

Table 3

```
CREATE TABLE reviews (
    id INT AUTO_INCREMENT PRIMARY KEY,
    rating DECIMAL(2,1),
    series_id INT,
    reviewer_id INT,
    FOREIGN KEY(series_id) REFERENCES series(id),
    FOREIGN KEY(reviewer_id) REFERENCES reviewers(id)
);
```

Table 2 + Table 3 + Table 1

```
SELECT
    title,
    rating,
    CONCAT(first_name,' ', last_name) AS reviewer
FROM reviewers
INNER JOIN reviews
    ON reviewers.id = reviews.reviewer_id
INNER JOIN series
    ON series.id = reviews.series_id
ORDER BY title;
```

```
SELECT
    title,
    AVG(rating) as avg_rating
FROM series
JOIN reviews
    ON series.id = reviews.series_id
GROUP BY series.id
ORDER BY avg_rating;
```

Table 1 + Table 3

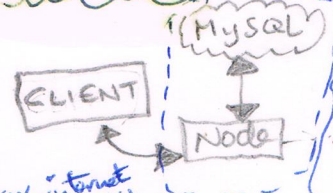
```
SELECT
    first_name,
    last_name,
    rating
FROM reviewers
INNER JOIN reviews
    ON reviewers.id = reviews.reviewer_id;
```

Table 2 + Table 3

NODE

Could be PhP / Ruby / Python

(1)



To create a new node app. (check node installed 1st)

```
> npm init
```

↳ This creates the package.json file

eg faker.internet.email()

faker: → creates dummy data

```
>npm install faker.
```

↳ check in >tail package.json

>tail package.json or cat

*BUT

To connect node to mySQL

(2)

in "app.js"

```
var mysql = require('mysql')
var connection = mysql.createConnection({
    host: 'localhost',
    user:
    database:
});
```

eg. 'jaiau'

eg. 'mysql'

also password...

--save → comes with mySQL support