CREATE DATABASE & TABLES

> CREATE DATABASE ___ ; → > USE ___ ;

> DROP DATABASE ___ ;

> SELECT database();

Database ⇒ lots & lots of tables!

eg

Name	breed	Age
Blue	Tabby	3
Rocket	Persian	ten

'Cat' tab — must be an integer number

must be variable characters

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

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

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

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

> SELECT * FROM cats;

> DROP TABLE cats;

SHOW WARNINGS

NULL ≠ ∅ ! & DEFAULT

> CREATE TABLE cats 2
(name VARCHAR(100) NOT NULL DEFAULT 'unnamed',
age INT NOT NULL DEFAULT 99
);

INSERTING DATA + others

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

> INSERT INOT cats (breed, age)
VALUES ('Moggi', 8)

Can change order if values order changes too

will use default 'unnamed' for name

PRIMARY KEY → unique ID eg.

> CREATE TABLE cats 3
(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 cat
WHERE age = 4;

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

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

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

• sql   Sql files → for Kate, VIM gedit etc

> SOURCE filename . sql

To put strings 'together' Concatenate

> 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', 1, 4);
⇓
or
SUBSTRING | hell |

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

Replace   > SELECT REPLACE
('Hello World', 'Hell', '%$#@)
| %$#@ World |

Reverse > SELECT REVERSE ("Hello World");
| dlroW olleH |

Char length > SELECT CHAR-LEGTH ("hello world")
⇒ 11

Upper Case / lower Case
> SELECT UPPER ('Hello World') → "HELLO WORLD"

Structured Query Language

REFINING SELECTIONS

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

> SELECT DISTINCT ath-last name FROM books

or a combination eg

> SELECT DISTINCT ath-firstname, ath-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;

LIKE

→ WHERE name-first LIKE "%hom%"   could be LIMIT 3,7

→ WHERE stock-quantity LIKE "____"   So Thomas & honey 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, authorfirst FROM books
    GROUP BY authorfirst

or to count the number of books written by someone with a certain last name

> SELECT auth-last,
    COUNT(*) FROM books
    GROUP BY auth-last;

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

Combine SELECTS

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

> SELECT *
    FROM books
    ORDER BY pages
    ASC LIMIT 1;
    → AS 'date 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, surname is priority

AVG average
    > SELECT AVG(pubdate) FROM books
    > SELECT AVG(stock-qnty) FROM books
    GROUP BY (pubdate)

DATA TYPES

CHAR(20)   VARCHAR(20) * use Mostly
    ↓ Fixed, must   up to 20
Postcodes be 20,   Max of 255 characters
state/ always 20!
found. → right padding.

Flags
Numbers — INT →31   DECIMAL
    DECIMAL(5,2)
    like 124.73

FLOAT ; DOUBLE
4 bytes ~7 digits ; 8 bytes ~15 digits

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

CURDATE()   CURTIME()   NOW()
current date   current time   current datetime

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

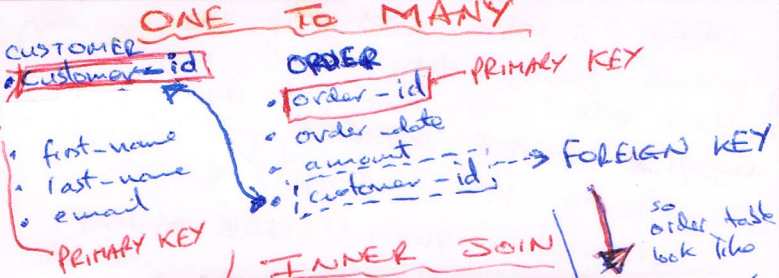|	
> SELECT title FROM books WHERE year != 2017 ;	!= not equal
> SELECT title FROM books WHERE title NOT LIKE 'W%'; 'doesn't start with 'W''	NOT LIKE
> SELECT * FROM book WHERE year > 2000 > SELECT * FROM book WHERE year >= 2000	> greater than < less than
> SELECT 99 > 1 ; > SELECT 'a' = 'A' TRUE ① ; TRUE ① OUTPUTS TRUE (NO capitals in sql)	
> 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"
> SELECT title, auth-lname FROM books WHERE auth-lname IN ('Smith', 'Baker', 'Jones')	Also 'NOT BETWEEN'
> SELECT title, stock-quantity, CASE WHEN stock-qnty <= 50 THEN * WHEN stock-qnty <= 100 THEN ** ELSE '***' END AS Check	also "NOT IN" IN (in or in)

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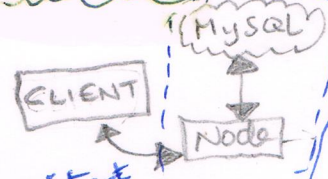
```
SELECT
    title,
    AVG(rating) as avg_rating
FROM series
JOIN reviews
    ON series.id = reviews.series_id
GROUP BY series.id
ORDER BY avg_rating;
```

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

MySQL

CLIENT → Node

To create a new node app.
(check node installed bt)

> npm init
↳ This creates (to
Package.JSON file)

eg kar internet
faker email()

faker: → creates
dummy data

> npm install faker.
↳ check is 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 'jaws'
                      - mysql
3);
also password...

--save → comes with mysql support
express framework
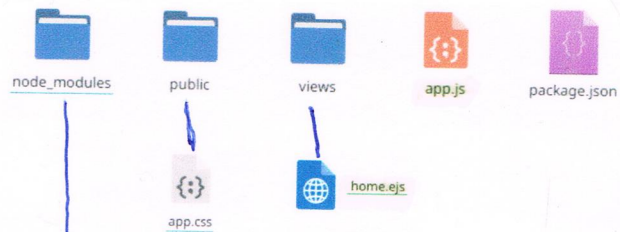
EXPRESS.js ③

See express.js.com → Hello world simple app example

```
const express = require('express')
const app = express()
app.get ("/", function (req, res) {
    res.send ('Hello world')
3)
app.listen (3000, function() {
    console.log ('listen to port 3000')
3);
```
app.js

← request
← response

↙ port.

{ The root of the "home page".

Adding a "random N°" page.

```
app.get ("/random_num", function (req, res) {
    var num = Math.floor ((Math.random()
            * 10) +1);
    res.send ("your lucky number is " + num);
3);
```

━━━ EXAMPLE — NOT COMPLETE!!! ━━━ ④

```
app.get ("/", function (req, res) {
    var q = 'SELECT count(*) AS count From users'
    connection.query (q, function (error, result) {
        if (error) throw error;
        var msg = "we have " + results[0].count +
            " users";
        res.send (msg);
3); 3);
```
count

results[0].count

needs to go into HTML

━━━ HTML markup. — (join us example app) ⑤ ━━━

```
<h1>Join us</h1>
<p class='lead'> Enlist email to join
<form method='post' action='/register'>
    <input type = 'text' class='form'
    Placeholder=' Enter email'>
    <button> Join now </button>
</form>
```
5(0) (p

need to inject ejs here!

━━━ we'll use EJS so ⑥ ━━━

→ >npm install --save ejs

back in app.js → ensure ejs is the view engine

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

also in "views" directory

Put before var connection=

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

views → create home.ejs file

join us (main directory)
views (create this)
app.js
package.js
node-modules

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

node_modules public views **app.js** package.json

{} app.css   🌐 home.ejs

express + faker + ejs

body-parser   ejs   express   faker   mysql

```javascript
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!");
});
```
app.js

home.ejs

```html
<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>
```