Lecture: SQL

Relational databases

(eg)

origin	dest	duration
NY	London	415
moscow	paris	435
	⋮	

← each row contains data about a flight

"sequel"

SQL (structured query language)

- interact with databases in an easy manner
- we'll use SQL version called PostgreSQL

Data Types: integer, decimal, serial, varchar, timestamp, boolean, enum, ...

- varchar: variable length of characters (eg) string
- serial: like an integer but counts automatically (eg: counter)
- enum: one of a finite # of discrete values

Creating Table

files called . sql

- create a table called flights
- rows and cols

create. sql

```
CREATE TABLE flights (
    id SERIAL PRIMARY KEY,
    original VARCHAR NOT NULL,
    destination VARCHAR NOT NULL,
    duration INTEGER NOT NULL
);
```

names of cols

data types of entries in cols

" CONSTRAINTS "

some properties of data entries in these cols

CONSTRAINTS

NOT NULL - Data can't be null

UNIQUE - unique for each data point col data

PRIMARY KEY - single way by which table is referenced...

DEFAULT val - if I don't tell you what to put in col, put DEFAULT val

CHECK - only allow values that pass CHECK conditions...

need a postgreSQL server

start server locally on computer

find servers hosted online

(eg) Heroku

online platform for hosting web sites and databases

psql "url of database"

to access database

\d

- shows different parts of database

INSERT DATA INTO TABLES

```
INSERT INTO flights (origin, destination, duration)
VALUES ('NY', 'London', 415);
```

table-name

names of cols

values.

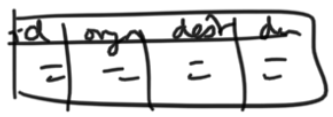
Not necc that every col should be updated...

Viewing data in a table

SELECT query

all cols ↙ ↙ table-name

```
SELECT * FROM flights;
```




```
SELECT origin, destination FROM flights;
```

restrict rows selected ↓

```
SELECT * FROM flights WHERE
id = 3;
```

---

more powerful queries

(eg) AVG = Average function, SUM

(1)
```
SELECT AVG(duration) FROM
FLIGHTS;
```

→ avg of all flights duration

(2)
```
SELECT AVG(duration) FROM
```

(2)

flights WHERE origin

= 'NewYork';

↳ avg of all flights duration from NY.

(eg) COUNT → returns how many rows returned from database

SELECT COUNT(*) FROM flights;

(eg) MIN(col), MAX(col)

---

* can use AND, OR, IN

⋮

SELECT * FROM flights WHERE origin IN

('NewYork', 'Lima');

* string matching --- (LIKE)

SELECT * FROM flights WHERE origin

LIKE '%a%';

% ... text

Updating data in a table

```
UPDATE flights
    SET duration = 430
    WHERE origin = 'New York'
    AND destination = 'London';
```

update duration col = 430 for all flights from NY → London...

```
UPDATE flights
    SET duration = 430
```

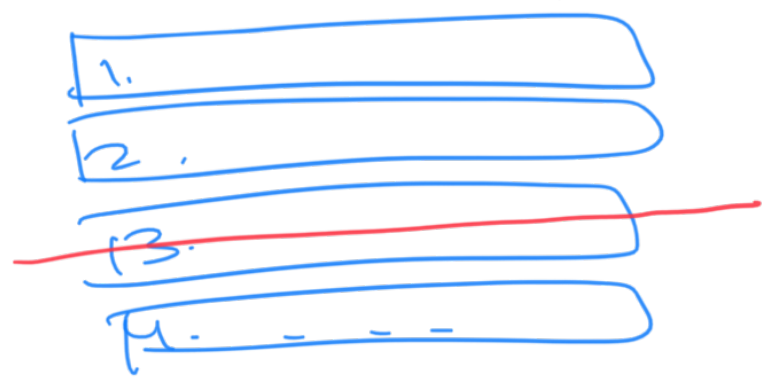
set every value in duration = 430

Delete data from a table

```
DELETE FROM flights
    WHERE destination = 'Tokyo';
```

how do delete columns?

⚠ Also if we delete a row with 'id' = 3 say, no other row will ever have 'id' = 3.



then we add in a row →

etc.

ADDITIONAL USEFUL COMMANDS

① Limit ← limits # of rows returned.

```
SELECT * FROM flights LIMIT 2;
```

② sorting

```
SELECT * FROM flights ORDER BY duration ASC;
```

duration ↑ ascending

col you want to sort by

(DESC is descending)

Combining multiple queries into a single query

most common origins ?

```
SELECT origin, COUNT(*) FROM flights
GROUP BY origin
```

what col you are grouping by

origin	count
Shangai	1
NY	2
una	1
Istanbul	1
moscow	1

```
SELECT origin, COUNT(*) FROM flights
GROUP BY origin HAVING
COUNT(*) > 1;
```

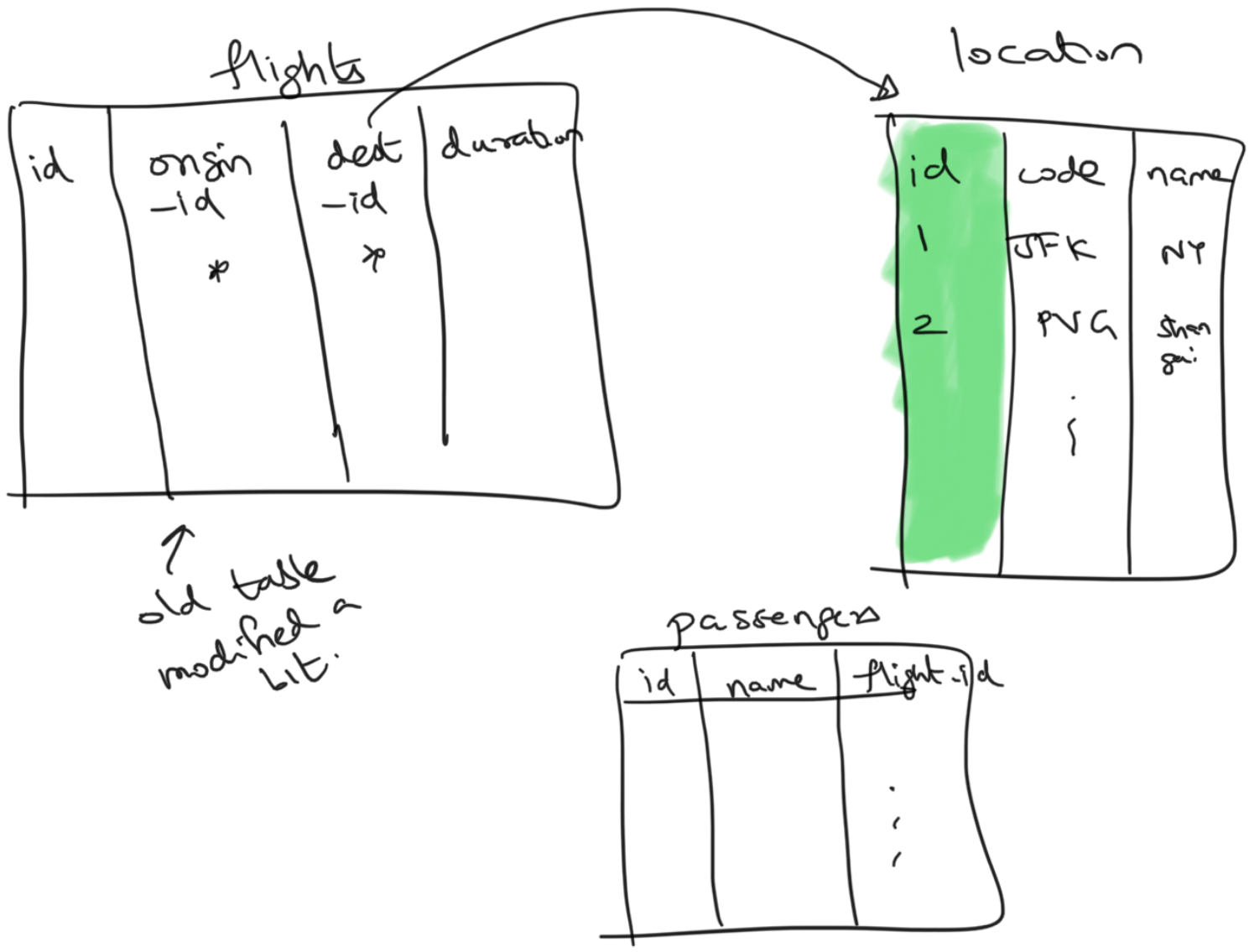
follows

what property you want data in origins to satisfy...

origin	count
NY	2

Connecting multiple tables together

Foreign key



foreign key -- " referencing key from a different table"

```
CREATE TABLE passengers (
```

implicitly

passenger_id →

id SERIAL PRIMARY KEY,

name VARCHAR NOT NULL,

flight_id INTEGER REFERENCES flights);

primary key of flights...

Join 2 related tables so that you can give a more complicated query

```
SELECT origin, destination, name
FROM flights JOIN passengers
ON passengers.flight_id = flights.id;
```

origin, destination → flights table

name → passenger

which tables joining

relationship bet the tables

Inner Join (default)

→ returns relevant rows only if there are matches bet passengers and flights table --

flights LEFT JOIN passengers

↝ matches + all rows in flights..

flights RIGHT JOIN passengers

↝ matches + all rows in passengers

To include all rows in both tables ??

Creating Index

- easy way to reference something by looking at something else

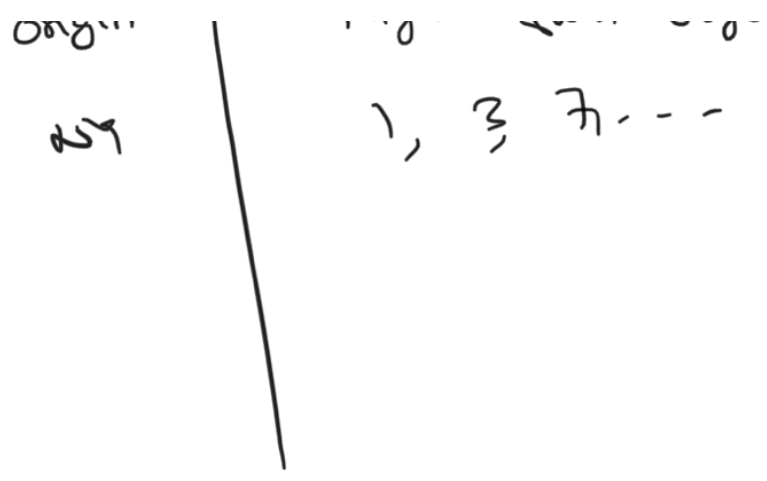
- can add 'index' to a particular col

eg if referencing flights by their origins a lot, ...

index

easy lookup

origin	flights from origin



index → takes up space
(but updating .... → takes more time)

```
CREATE INDEX name_of_index (arbitrary)
ON table_name column_name
```

nested queries

passenger

id	name	flight_id
1	A	1
2	B	2
3	C	1
4	D	3
5	E	3
6	F	4

```
SELECT flight_id FROM passengers
GROUP BY flight_id HAVING
```

COUNT(*) > 1 ;

flight_id
1
3

```
SELECT * FROM flights WHERE
id IN
(SELECT flight_id FROM passengers
GROUP BY flight_id HAVING
COUNT(*) >1) ;
```

nested query

↓ selects this -- from flights

id	origin	dest	duration
1			
2			. .
6			

SQL transactions

```
BEGIN

-- code - here

COMMIT
```

executed as a bunch together

Locks database while these are

running

Prevents concurrency conflicts
(RACE CONDITIONS)

SQL Injection

```
SELECT * FROM USERS
WHERE (username = username)
AND (password = password);
```

user_1

`1' OR `1' = `1` ← malicious hacker can input this

```
SELECT * FROM USERS
WHERE (username = ` user_1 '
AND (password = ` 1' OR `1'=1 ');
```

SQL will read this as

```
AND [(password = '1') OR ('1' = '1')]
```

WHICH WILL always be True!!

① don't directly substitute user input into SQL code directly

② sanitise input!