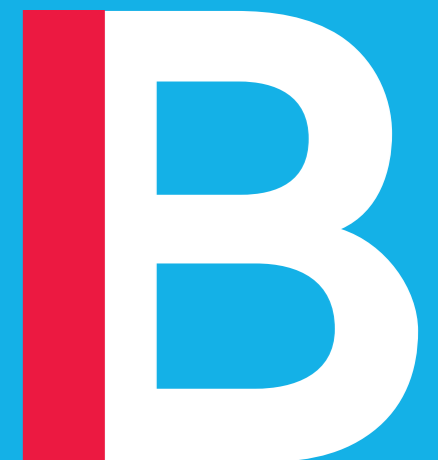


# Lecture 4

## Joins: bringing data together & the command line

Dr Fintan Nagle  
[f.nagle@imperial.ac.uk](mailto:f.nagle@imperial.ac.uk)



# Reading

Video lectures:

5.2 - Basic joins.mp4

5.3.1 - Basic joins in practice.mp4

5.3.2 - Intermediate joins.mp4

5.5 - Advanced joins.mp4

5.6 - Solving an advanced join problem.mp4

Command Line Introduction.mp4

7.2.1 - Using PSQL.mp4

What's the most beautiful SQL query you ever wrote?

<https://www.quora.com/Whats-the-most-complex-SQL-query-you-ever-wrote>

SQL outer joins for fun and profit: <https://www.slideshare.net/billkarwin/sql-outer-joins-for-fun-and-profit>



# Joins

# Joins are the opposite of normalisation

We normalise data to make it efficient and safe to store. This mainly involves splitting up tables.

This makes data harder to view and analyse.

So we have to use the JOIN operation to bring data back together for analysis.

# Joins

How do we combine information from two tables?

dog_id	name	breed
1	Rover	Great Dane
2	Fido	Dalmatian
3	Daisy	Mutt
4	Buddy	Poodle

owner_id	name	dog_name
1	Alex	Buddy
2	Brenda	Artemis
3	Charlie	Fido
4	Diana	Roger

# Joins

dogs

dog_id	name	breed
1	Rover	Great Dane
2	Fido	Dalmatian
3	Daisy	Mutt
4	Buddy	Poodle

owners

owner_id	name	dog_name
1	Alex	Buddy
2	Brenda	Artemis
3	Charlie	Fido
4	Diana	Roger

dogs **LEFT OUTER JOIN** owners  
**ON** dogs.name = owners.dog\_name

dogs columns

owners columns

dog_id	name	breed	owner_id	name	dog_name
1	Rover	Great Dane			
2	Fido	Dalmatian	3	Charlie	Fido
3	Daisy	Mutt			
4	Buddy	Poodle	1	Alex	Buddy

# Joins

owners

owner_id	name	dog_name
1	Alex	Buddy
2	Brenda	Artemis
3	Charlie	Fido
4	Diana	Roger

dogs

dog_id	name	breed
1	Rover	Great Dane
2	Fido	Dalmatian
3	Daisy	Mutt
4	Buddy	Poodle

owners **LEFT OUTER JOIN** dogs  
**ON** owners.dog\_name = dogs.name

owners columns

dogs columns

owner_id	name	dog_name	dog_id	name	breed
1	Alex	Buddy	4	Buddy	Poodle
2	Brenda	Artemis			
3	Charlie	Fido	2	Fido	Dalmatian
4	Diana	Roger			

# Joins

owners

owner_id	name	dog_name
1	Alex	Buddy
2	Brenda	Artemis
3	Charlie	Fido
4	Diana	Roger

dogs

dog_id	name	breed
1	Rover	Great Dane
2	Fido	Dalmatian
3	Daisy	Mutt
4	Buddy	Poodle

owners **INNER JOIN** dogs  
**ON** owners.dog\_name = dogs.name

owners columns

dogs columns

owner_id	name	dog_name	dog_id	name	breed
1	Alex	Buddy	4	Buddy	Poodle
3	Charlie	Fido	2	Fido	Dalmatian

# OUTER and INNER joins

A **JOIN** operation brings together two tables.

Rows from one table are matched with rows from another.

The **ON** line says which attributes to use to match up the rows.

**A LEFT OUTER JOIN B**

**ON** A.id = B.id

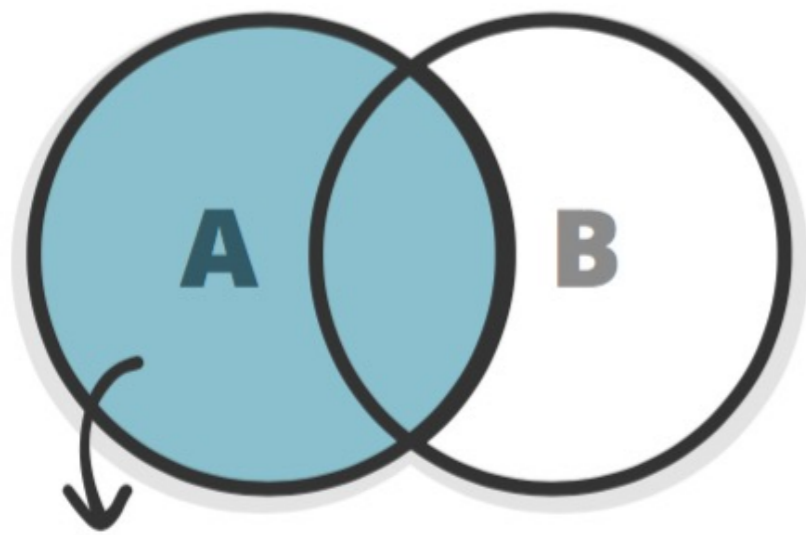
Here table A is copied into the result. Then the SQL processor works down the table and inserts rows from B where they match.

**A INNER JOIN B**

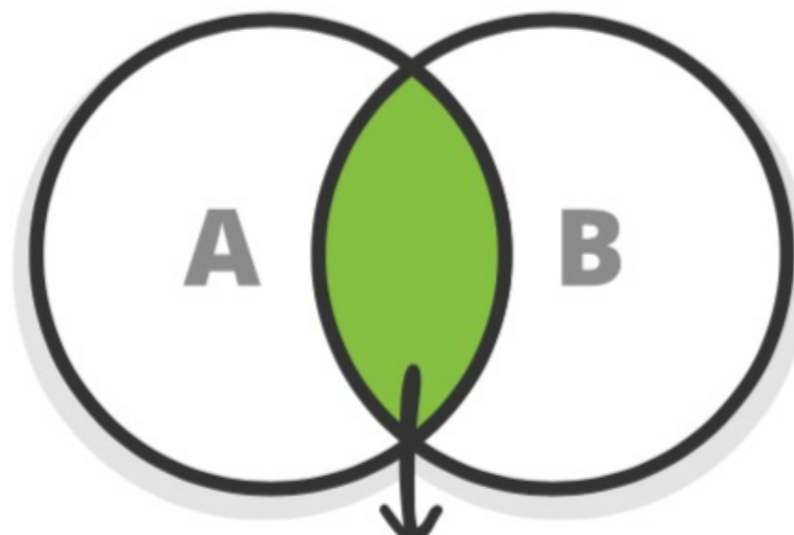
**ON** a.id = B.id

Here only the rows which were matched up appear in the result. Rows which were not matched up are skipped.

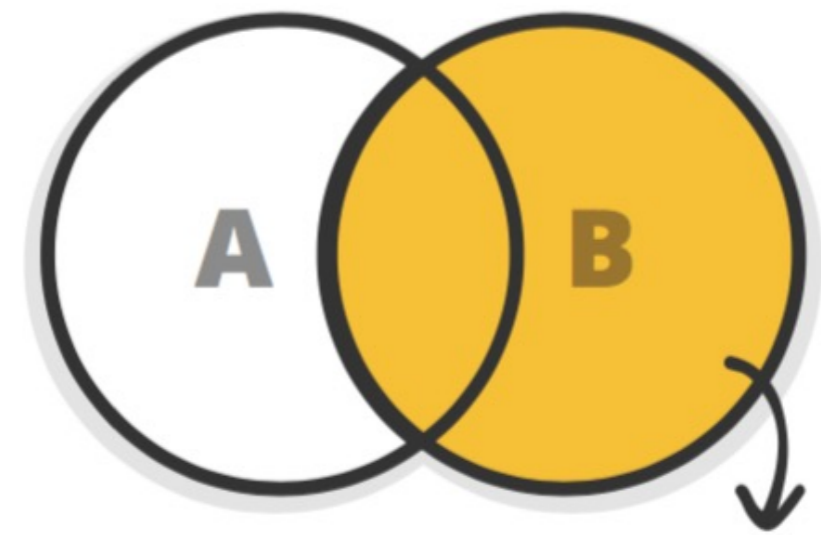
# Joins



**LEFT OUTER JOIN** - all rows from table A, even if they do not exist in table B



**INNER JOIN** - fetch the results that exist in both tables



**RIGHT OUTER JOIN** - all rows from table B, even if they do not exist in table A

# Joins

The inner join is symmetric!

```
owners INNER JOIN dogs  
ON owners.dog_name = dogs.name
```

=

```
dogs INNER JOIN owners  
ON dogs.name = owners.dog_name
```

# Joins

The outer join is antisymmetric:

```
owners LEFT OUTER JOIN dogs  
ON owners.dog_name = dogs.name
```

=

```
dogs RIGHT OUTER JOIN owners  
ON dogs.name = owners.dog_name
```

# Joins

Please note:

- No matter what kind of join you use, the columns in the result are always all the columns from the two input tables.
- As a result, we often have two columns with the same name. These can be qualified with the table name to separate them:

`dogs.name`

`owners.name`

# Ambiguous columns

```
SELECT * from rental, staff  
WHERE rental.staff_id = staff.staff_id
```

Notice that there are two staff\_id columns, so we have to specify which table we are referring to.

You can use **SELECT** rental.\* or **SELECT** staff.\* to return all columns from one table.

# How joins are processed

The SQL processor makes an intermediate table with **all** the columns from both joined tables.

Then, **AFTER** the join has been processed, the columns you asked for in the **SELECT** clause are extracted and sent to you.

The temporary joined table is then thrown away.

# Sequential joins

You can chain joins:

```
A INNER JOIN B  
INNER JOIN C  
INNER JOIN D
```

etc  
(each with an ON line)

The SQL processor does the joins sequentially, adding to the joined result table as it goes along.

Doing a JOIN is like following a relationship line on the ER diagram.

A blue-tinted photograph of a modern glass building with a person sitting on a bench in the foreground. The text "How many rows in the result?" is overlaid in white.

# How many rows in the result?

# When do joins increase the row count?

owners

owner_id	name
1	Alex
2	Brenda

dogs

dog_id	owner_id	name	breed
1	1	Rover	Great Dane
2	1	Fido	Dalmatian
3	1	Daisy	Mutt
4	2	Buddy	Poodle

owners **LEFT OUTER JOIN** dogs  
**ON** dogs.owner\_id = owners.owner\_id

owners columns

dogs columns

owner_id	name	dog_id	owner_id	name	breed
1	Alex	1	1	Rover	Great Dane
1	Alex	2	1	Fido	Dalmatian
1	Alex	3	1	Daisy	Mutt
2	Brenda	4	2	Buddy	Poodle

# When do joins increase the row count?

When we do

`table_1 INNER JOIN table_2`

or

`table_1 LEFT OUTER JOIN table_2`

and some rows in `table_1` are matched up with more than one row in `table_2`,  
the row count will increase  
(joined table will have more rows than `table_1`).

# When do joins decrease the row count?

owners

owner_id	name
1	Alex
2	Brenda
3	Charlie
4	Daisy

dogs

dog_id	owner_id	name	breed
1	3	Rover	Great Dane

owners **LEFT OUTER JOIN** dogs  
**ON** dogs.owner\_id = owners.owner\_id

owners columns

dogs columns

owner_id	name	dog_id	owner_id	name	breed
1	Alex				
2	Brenda				
3	Charlie	1	3	Rover	Great Dane
4	Daisy				

# When do joins decrease the row count?

owners

owner_id	name
1	Alex
2	Brenda
3	Charlie
4	Daisy

dogs

dog_id	owner_id	name	breed
1	3	Rover	Great Dane

owners **INNER JOIN** dogs

**ON** dogs.owner\_id = owners.owner\_id

owners columns

dogs columns

owner_id	name	dog_id	owner_id	name	breed
3	Charlie	1	3	Rover	Great Dane

# When do joins decrease the row count?

When we do

`table_1 INNER JOIN table_2`

and some rows in `table_1` do not get matched up with rows in `table_2`

(or the other way around, since the inner join is symmetric),  
the row count is decreased  
(joined table does not have all of the rows from table 1).

When we do

`table_1 LEFT JOIN table_2`

we will always see all the rows from `table_1`, since the left join includes them all.

# Both effects combined

owners

owner_id	name
1	Alex
2	Brenda
3	Charlie

dogs

dog_id	owner_id	name	breed
1	3	Rover	Great Dane
2	3	Spot	Dalmatian
3	3	Fido	Terrier

owners **INNER JOIN** dogs  
**ON** dogs.owner\_id = owners.owner\_id

owners columns

dogs columns

owner_id	name	dog_id	owner_id	name	breed
3	Charlie	1	3	Rover	Great Dane
3	Charlie	2	3	Spot	Dalmatian
3	Charlie	3	3	Fido	Terrier

Two owners rows lost; one owners row copied three times: still three rows in the final table.

# JOIN USING

**Where two tables share a column, instead of writing:**

```
select * FROM  
orders INNER JOIN order_details  
ON orders.orderid = order_details.orderid
```

**...one can write:**

```
select * FROM  
orders INNER JOIN order_details  
USING (orderid)
```

**USING works with lists of columns too (joins on all).**

# Outer joins

## **LEFT JOIN**

**is the same thing as**

## **LEFT OUTER JOIN**

**(same for right joins)**

# Outer joins

**A INNER JOIN B LEFT OUTER JOIN C**

**can be written**

**(A INNER JOIN B) LEFT OUTER JOIN C**

**or**

**A INNER JOIN B  
LEFT OUTER JOIN C**

**and is not the same as**

**A INNER JOIN (B LEFT OUTER JOIN C)**

# The cross join

- The cross join produces a table with each row associated with each other row.
- There are no join conditions – all rows are joined to all other rows. This operation is also called the **Cartesian product**.
- If table A has x rows and table B has y rows, A cross join B will have  $x * y$  rows.

**SELECT \* FROM**  
rental **CROSS JOIN** staff

This can be abbreviated with the comma (this is equivalent):  
**SELECT \* FROM** rental, staff

# Cross joins and WITH vs. normal joins and ON

Notice that

```
SELECT * from rental, staff  
WHERE rental.staff_id = staff.staff_id
```

returns the same rows as

```
SELECT * FROM  
rental INNER JOIN staff  
ON rental.staff_id = staff.staff_id
```

Checking the matching staff\_id can be done by either WHERE or ON.

Is there any difference in speed?

# The command line

A blue-tinted photograph of a modern glass-walled building. In the foreground, a person is sitting on a row of colorful beanbag chairs. The building's glass reflects the surrounding environment. The text "The command line" is overlaid in white on the left side of the image.

# The command line

The command line is a vital everyday tool for data scientists.

It allows the computer to be controlled by issuing text commands, without a GUI (graphical user interface).

The command line has not changed much since the 1970s.

# Common commands

	OS X/Linux/Unix	Windows
Show current folder	<code>pwd</code> <i>print working directory</i>	<code>cd</code> <i>by itself</i>
Change directory	<code>cd</code> <i>change directory</i>	<code>cd</code>
List files in current folder	<code>ls</code>	<code>dir</code>
Make new directory	<code>mkdir</code>	<code>mkdir</code>
Show the file in which a command is stored	<code>which</code>	<i>N/A</i>
Set environment variable (just in current command line session)	<code>export var=val</code>	<code>SET var=val</code> (or control panel)



PSQL

# PSQL

PSQL is a command line utility allowing you to

- Make queries, just like pgAdmin;
- Control the Postgres server using special PSQL commands.

# PSQL

- h host
- p port
- U user

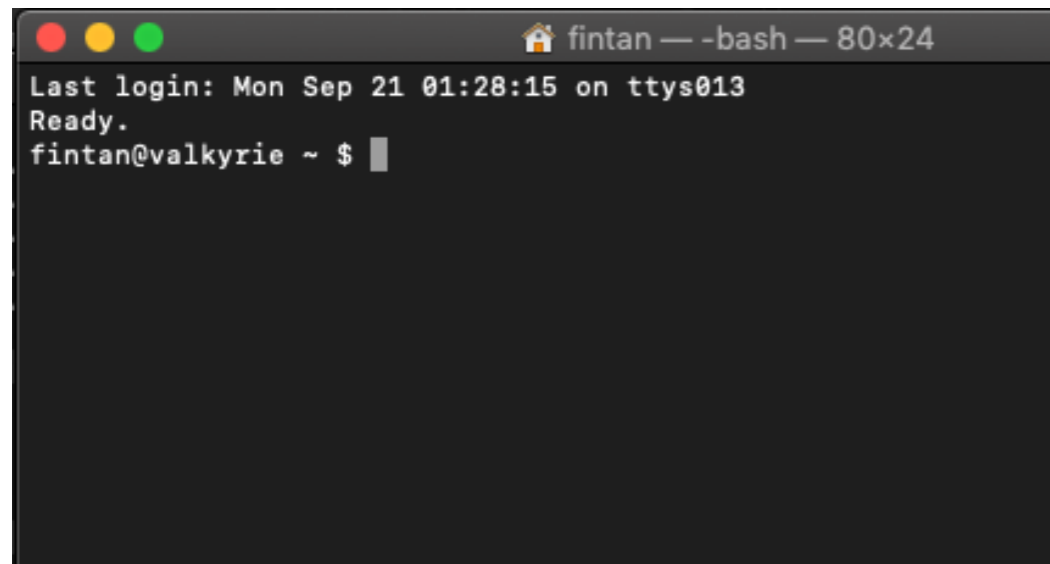
- f run a particular SQL file

- E report on what you're doing

# Which environment are you in?

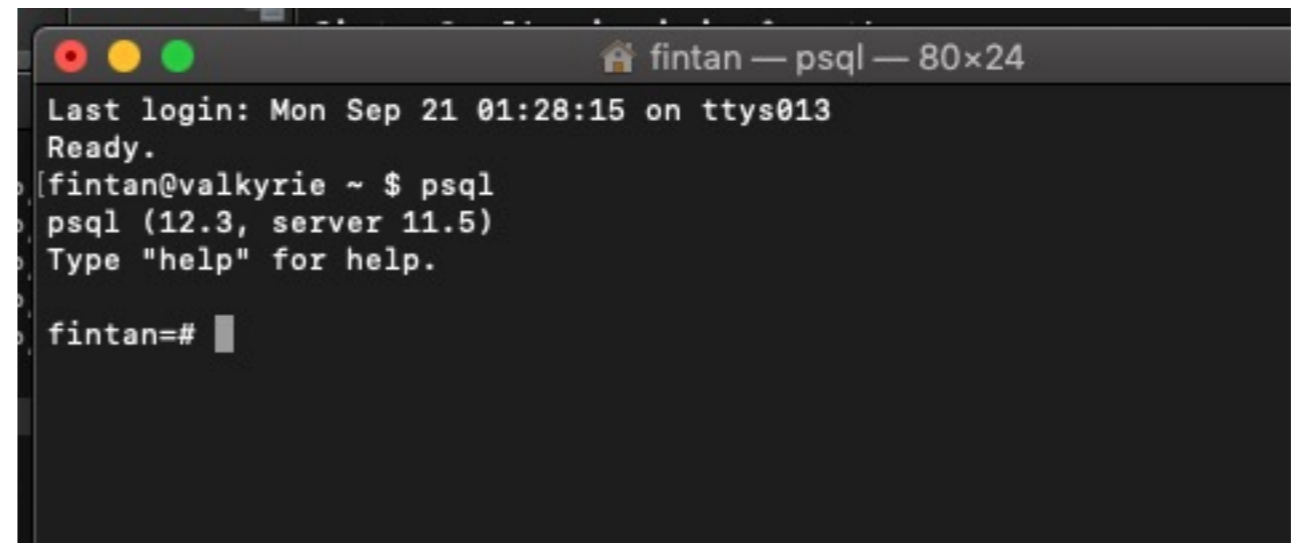
Be aware whether you are in PSQL or the normal command line.

Notice that the prompt is different in psql:



```
fintan — -bash — 80x24
Last login: Mon Sep 21 01:28:15 on ttys013
Ready.
fintan@valkyrie ~ $
```

Command line



```
fintan — psql — 80x24
Last login: Mon Sep 21 01:28:15 on ttys013
Ready.
[fintan@valkyrie ~ $ psql
psql (12.3, server 11.5)
Type "help" for help.

fintan=#
```

PSQL

Note that PSQL **requires** a closing semicolon and closed quotes, etc.  
Restart PSQL if you get stuck.

# Which environment are you in?

When we launch PSQL we need to tell it the host, port, username, PW, etc. If you just type **psql** it assumes the host is **localhost** (your computer). You don't have a Postgres server running on your computer yet (unless you installed one yourself). So PSQL can't connect. So let's supply the right arguments:

```
PGPASSWORD=imperial-fdt-online-2019-colossal-shelf;  
psql -h imperial-2020.ckp3dl3vzxoh.eu-west-  
2.rds.amazonaws.com -U imperial -d dvdrental
```

Paste this into your terminal to connect to the remote DB server.

You can then run

```
SELECT * FROM film;
```

(Don't forget the semicolon)

If this doesn't work, try just

```
psql -h imperial-2020.ckp3dl3vzxoh.eu-west-  
2.rds.amazonaws.com -U imperial -d dvdrental
```

then enter the password.

# PSQL

EXPLAIN \_\_query\_\_: see the query plan for the given query

EXPLAIN ANALYZE \_\_query\_\_: see and execute the query plan for the given query

ANALYZE [\_\_table\_\_]: collect statistics

# PSQL

\c my\_database

connect to database

\l

list databases

\lv list views

\copy (SELECT \* FROM \_\_table\_name\_\_) TO 'file\_path\_and\_name.csv'  
WITH CSV

export as CSV

\du

list users

# pg\_dump

# PSQL

```
sudo service postgresql stop  
sudo service postgresql start  
sudo service postgresql restart
```