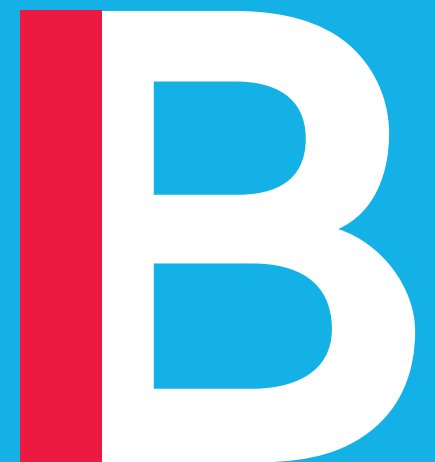


Lecture 1

Introduction to SQL

Dr Fintan Nagle
f.nagle@imperial.ac.uk



Module outline

Week 1: Basic SQL

- 1 Introduction to SQL
- 2 Our First SQL Queries

Week 2: Intermediate SQL

- 3 Normalisation: keeping our data tidy
- 4 Joins: bringing our data together; the command line

Module outline

Week 3: Advanced SQL

5 GROUP BY and window functions

6 Subqueries: splitting up queries

Week 4: Professional SQL

7 Query planning and development; CSV files

8 Query profiling, tuning and optimisation

Week 5: Big Data and the SQL Ecosystem

9 Integrating SQL with other languages

10 Big data, varieties of SQL, and noSQL

Coursework assignments

Assessment is by 5 equally-rated coursework assignments.

Learning resources

- Slides
- Video lectures
- Lecture notes (PDF)
- Ask your fellow students
- Ask me (I encourage questions **at any time** in lectures)
- Ask me and the TAs on EdStem
- Google (check it's Postgres, not MySQL etc)
- Stack Overflow

Learning resources

Please ask me if you have any questions at any time – just raise your hand to attract my attention (or post in the chat on Zoom).

Any and all questions are encouraged :)

Course reading

Video lectures:

1.3 - Naive methods.mp4

1.5 - The relational approach.mp4

2.2.1 - Downloading pgAdmin.mp4

2.2.2 - How to connect to the server.mp4

2.3 - The Entity Relationship diagram.mp4

Background reading

- The perils of unsanitised database input: <https://xkcd.com/327/>
- SQL horror stories: unfortunate mishaps
- https://www.reddit.com/r/SQL/comments/45y1z7/sql_horror_stories_unfortunate_mishaps/
- Uber complaining about Postgres: <https://eng.uber.com/mysql-migration/>
- Postgres responding to Uber complaining about Postgres: <https://news.ycombinator.com/item?id=14222721>
- Postgres at Instagram: <https://instagram-engineering.com/sharding-ids-at-instagram-1cf5a71e5a5c>
- Can Postgres actually do everything?
<http://renesd.blogspot.com/2017/02/is-postgresql-good-enough.html>

Background reading

<https://opentextbc.ca/dbdesign01/chapter/chapter-1-before-the-advent-of-database-systems/>

<https://www.quickbase.com/articles/timeline-of-database-history>

<http://www.bbc.co.uk/schools/gcsebitesize/ict/databases/2databasesrev4.shtml>
!

<http://avant.org/project/history-of-databases/>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6359709>

<https://pgsnake.blogspot.com/2014/12/the-story-of-pgadmin.html>

<https://www.postgresql.org/message-id/595393a6.1ba3df0a.a92ad.c754%40mx.google.com>

<https://www.postgresql.org/docs/10/static/tutorial-select.html>

Reading

Textbook purchase is not necessary for this course. It is generally not a good idea to buy a basic SQL textbook. The best resources to learn basic SQL, apart from this module, are online: Wikipedia, the Postgres documentation, and tutorials. Everyone learns differently, so find a tutorial which uses a style you can learn from.

These books are not necessary for the module - techniques therein are covered in our teaching materials, where required. These books are interesting if you want to go further.

- The Art of PostgreSQL, Dimitri Fontaine
- The Art of SQL, Stéphane Faroult and Peter Robson

What did we do before databases?

The history of storing data



The history of storing data

1750 BC

Clay tablet; letter from Nanni to Ea-nasir
complaining that the wrong grade of copper ore has
been delivered after a gulf voyage and about
misdirection and delay of a further delivery

[British Museum]



mid-19th Century Jacquard loom (National Museum of Scotland)

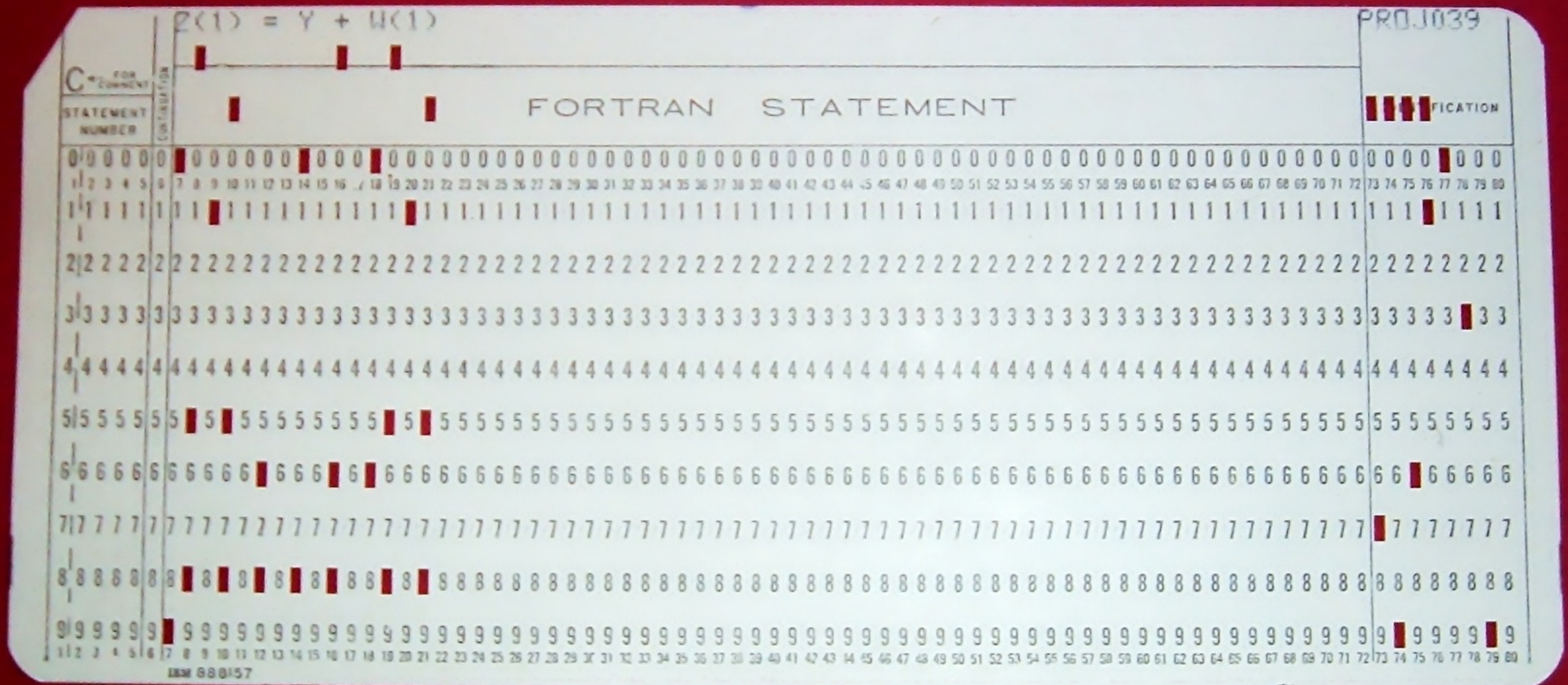


Punch card reader for a Jacquard loom
(National Museum of Scotland)



Each row of holes corresponds to a row of textile

The history of storing data



Punch card containing some of a Fortran program, late 1960s or early 1970s



How to describe our data?

Independent of storage medium

How to store our data?

Physical details of storage medium

The relational approach

How to describe our data?

Independent of storage medium

How to store our data?

Physical details of storage medium

Problems with traditional methods

- Writing takes time (sometimes it must be done by a human)
- Reading takes time (sometimes it must be done by a computer)
- Storage is fragile
- No security - data can be deleted or even faked
- Copying is hard
- The data may not be machine-processable; algorithms, queries and statistics cannot be run on it

Things can go very wrong

Case 1: AWS S3 outage:

https://www.theregister.co.uk/2017/03/01/aws_s3_outage/

<https://www.recode.net/2017/3/2/14792636/amazon-aws-internet-outage-cause-human-error-incorrect-command>

<https://aws.amazon.com/message/41926/>

<http://www.datacenterdynamics.com/content-tracks/colo-cloud/amazon-web-services-us-east-1-goes-down-errors-cause-outage/97892.article>

Case 2: Yahoo's 2013 data breach

<https://www.bbc.co.uk/news/business-41493494>

<https://www.oath.com/press/yahoo-provides-notice-to-additional-users-affected-by-previously/>

<https://help.yahoo.com/kb/account/SLN28451.html?impressions=true>

<https://www.technologyreview.com/s/603157/a-history-of-yahoo-hacks/>

<https://www.nytimes.com/2016/12/14/technology/yahoo-hack.html>

Case 3: JournalSpace backup loss

<https://techcrunch.com/2009/01/03/journalspace-drama-all-data-lost-without-backup-company-deadpooled/>

<http://www.gobitcan.com/blog/2014-10-24-the-cautionary-tale-of-journalspace-lessons-learned-in-database-backups>

Case 4: GitLab backup loss

<https://techcrunch.com/2017/02/01/gitlab-suffers-major-backup-failure-after-data-deletion-incident/>

https://www.theregister.co.uk/2017/02/01/gitlab_data_loss/?mt=1485932441853

<https://www.geek.com/news/disgruntled-employee-kills-journalspace-with-data-wipe-660762/>

JournalSpace Drama: All Data Lost Without Backup, Company Deadpooled

10 years ago

Comment



**TC Sessions:
AR/VR
Next Week!**

Los Angeles
Oct 18

Buy Now

advertisement

Startups

Apps

Gadgets

Events

Videos

—

Crunchbase

More

Search 🔍

Apple

Security

Tesla

Fundings & Exits

Log in / Sign up

GitLab suffers major backup failure after data deletion incident

Natasha Lomas @riptari / 2 years ago

Comment





NEWS

[Home](#) | [UK](#) | [World](#) | [Business](#) | [Politics](#) | [Tech](#) | [Science](#) | [Health](#) | [Family & Education](#) | [Entertainment & Arts](#) | [Stories](#) | [More](#) ▾[Business](#) | [Your Money](#) | [Market Data](#) | [Companies](#) | [Economy](#)

Yahoo 2013 data breach hit 'all three billion accounts'

🕒 3 October 2017



Share

Top Stories

Princess Eugenie marries Jack Brooksbank

The Queen's granddaughter marries her long-term partner in front of royals and celebrities in Windsor.

🕒 43 minutes ago

Fracking to start as legal challenge

Data Centre ► **Cloud**

AWS's S3 outage was so bad Amazon couldn't get into its own dashboard to warn the world

Websites, apps, security cams, IoT gear knackered

By [Shaun Nichols](#) in [San Francisco](#) 1 Mar 2017 at 03:00

122

SHARE ▼

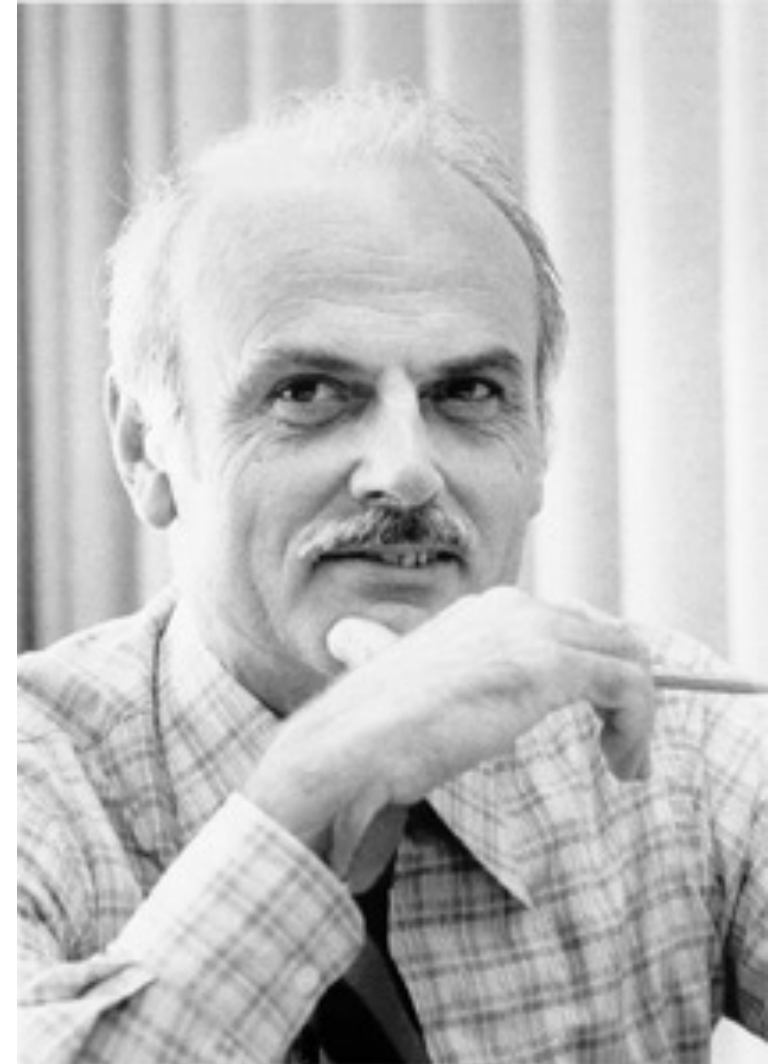


Relational databases

A blue-tinted photograph of a modern glass-walled building. The building's facade is composed of large glass panels that reflect the surrounding environment. In the foreground, a person is sitting on a white, rounded bench. The ground is paved with light-colored tiles. The overall scene is a modern, urban setting.

The history of relational databases

- The relational paradigm was invented in the late 1960s by English computer scientist Edgar F. Codd, working at IBM.

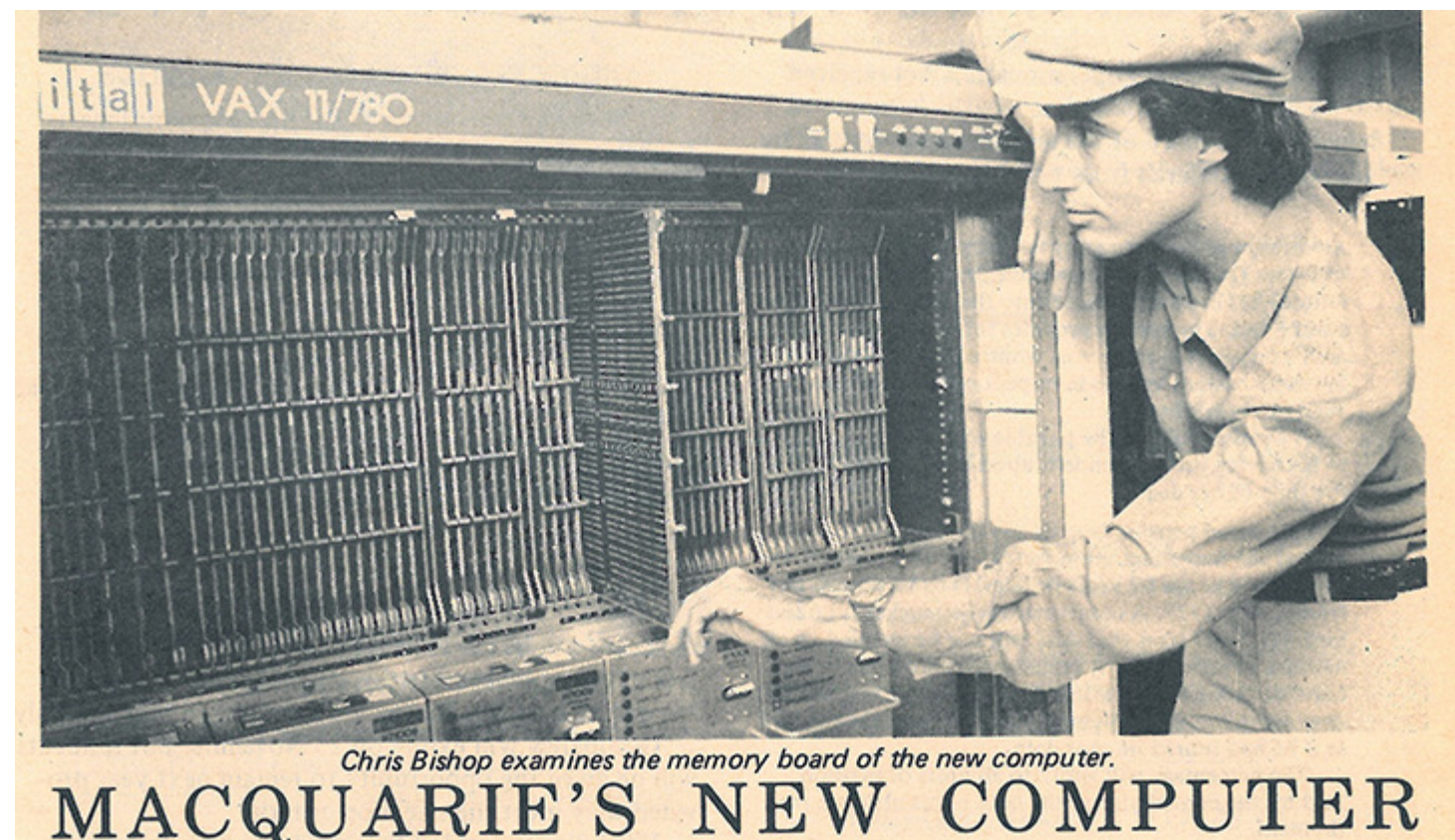


Edgar F. Codd
1923 - 2003

"The key, the whole key, and nothing but the key, so help me Codd."

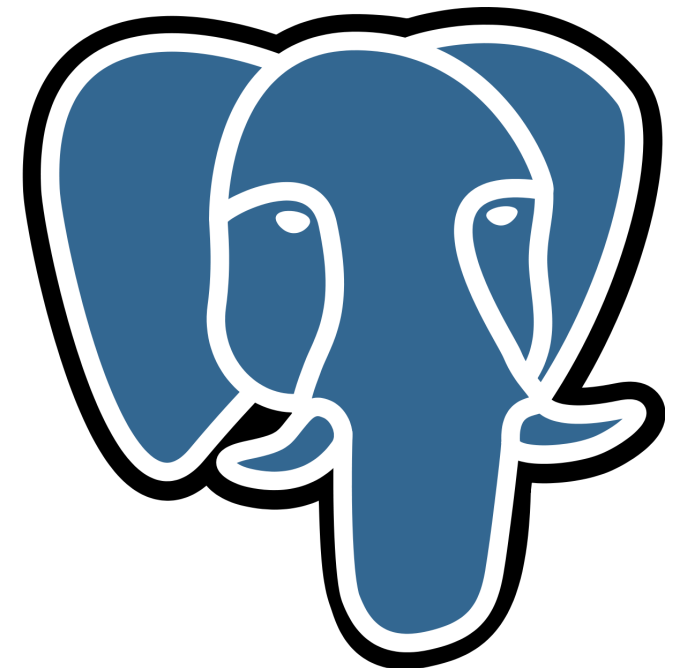
The history of SQL

- SQL is the only widely used declarative programming language (you specify what you want, not how to do it).
- Initially developed as SEQEL (Structured English QuEry Language) in the 1970s, by Chamberlin & Boyce at IBM
- The first commercially available implementation was Oracle V2 in 1979
- There are now several versions of SQL (PostgreSQL, MySQL) and many languages inspired by SQL (GraphQL, SPARQL, Apache Spark...). They all have slightly different syntax and features.



PostgreSQL

- Here we will focus on the most common (and powerful) version, PostgreSQL, which is open-source.
- Started at Berkeley in 1985
- Post INGRES
- INGRES was a research database at Berkeley in the 1970s:
Interactive **G**raphics **R**etrieval **S**ystem
- 1994: Berkely students install SQL as Postgres's query language
- Today: an industry-standard database, winner of many awards and used by many large enterprises



The relational approach

A blue-tinted photograph of a modern glass building with a person sitting on a bench in the foreground. The building's glass facade reflects the surrounding environment, creating a complex pattern of light and shadow. The person is sitting on a light-colored, modular bench, looking down at something in their hands. The foreground is a paved area with a grid pattern.

The relational model

- The basic data structure is the table (or **relation**)
- Tables have rows (or **records**) and columns (or **attributes**)
- “Relational” refers to the relation (i.e. the table)
- An attribute that uniquely identifies a row is called a **primary key**.
- There are also **relationships** (not relations) between tables
- Relationships are encoded simply by seeing the same value in two different tables

Keys

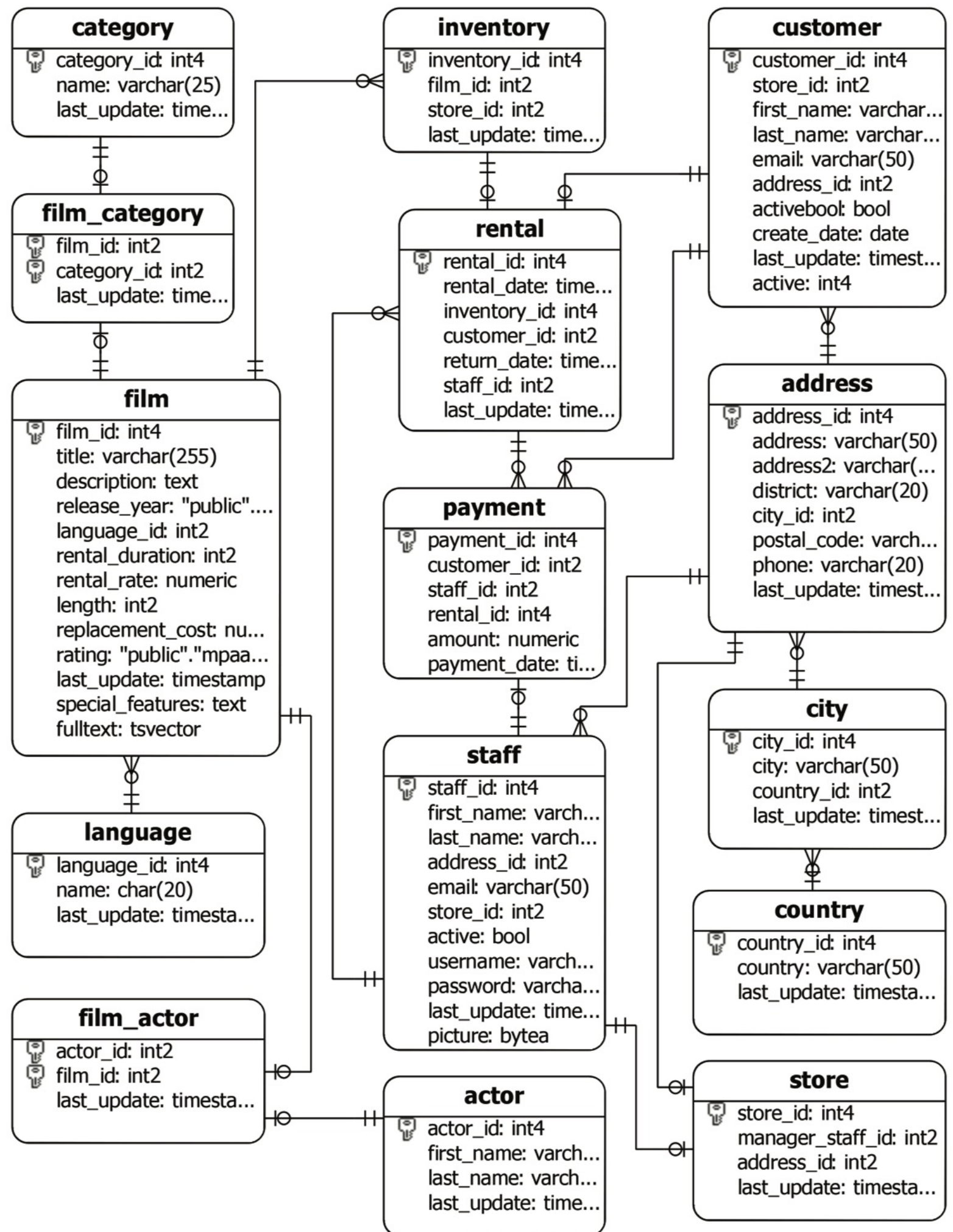
A **key** is an attribute, *or set of attributes*, which uniquely identifies a row.

A **candidate key** is a **minimal** set of attributes which uniquely identifies a row (this means we can't remove any of the attributes, and still identify the row).

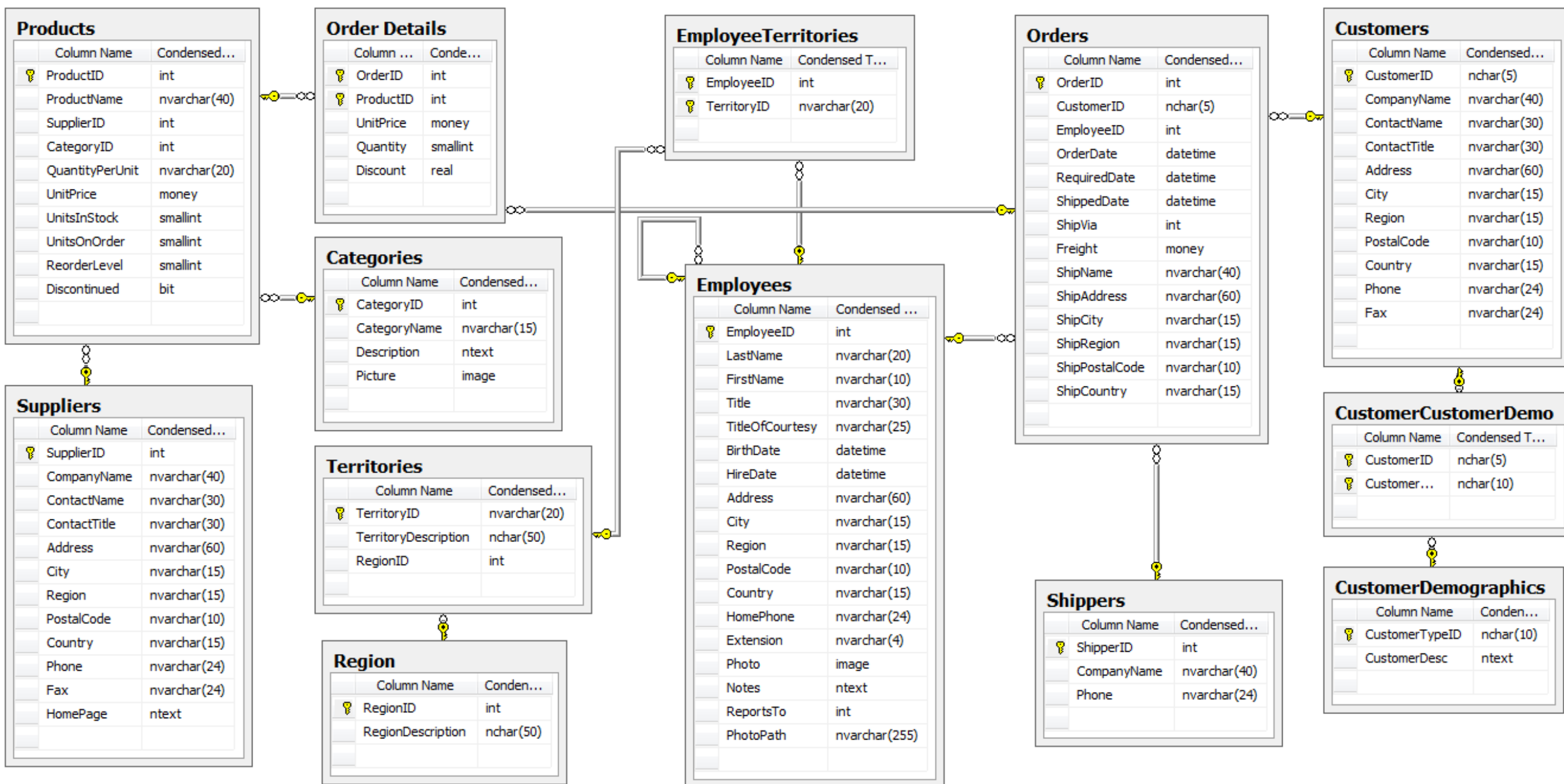
A **primary key** is one of the candidate keys which has been chosen, by the database developer, as the most useful candidate key.

Entity relationship diagrams

The dvdrental database



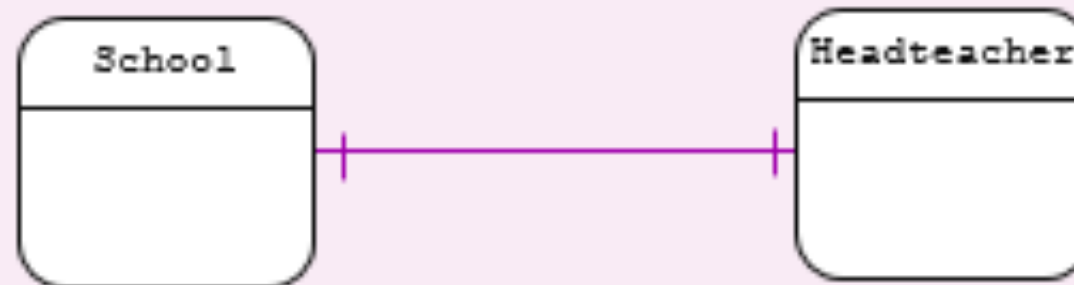
The Northwind database



Relationships

3 Types of Relationships:

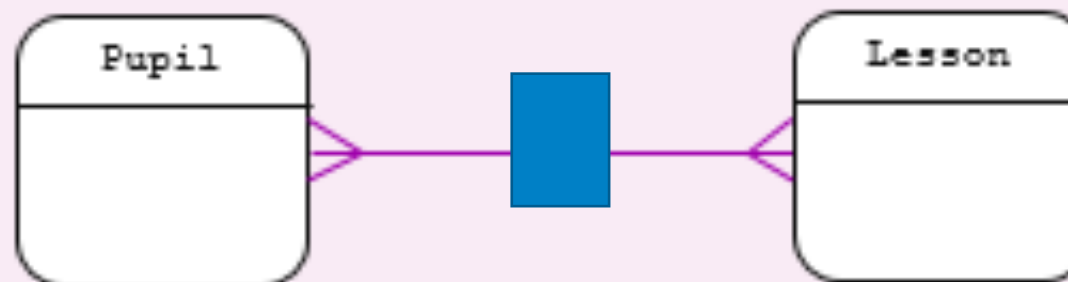
One-to-one:



One-to-many:



Many-to-many:



Relationships

One-to-one

There is one entity at each end of the relationship.

One-to-many

(or many-to-one – it's the same thing)

There is one entity at one end of the relationship. The other end can have many entities.

Many-to-many

There can be many entities at both ends of the relationship.

Cardinality indicators and relationships

Cardinality indicators are **not** the same as relationship types.
Don't get confused.

- We first have to look at **the whole relationship** (one-to-one, many-to-one or many-to-many)
- Then we have to look at **each end separately** (there is one cardinality indicator at each end - one, many, only one, zero or one, one or many, zero or many...)

A relationship has two cardinality indicators, one at each end.
Relationships are built out of cardinality indicators.

There are three types of relationship and at least four cardinality indicators.

Cardinality indicators and relationships

Notation for cardinality indicators is not always standard!

There should always be a legend.

You need to include a legend, too.

Cardinality indicators



One



Many

(Hardly ever used)



One (and only one) (Hardly ever used, same as “one”)



Zero or one

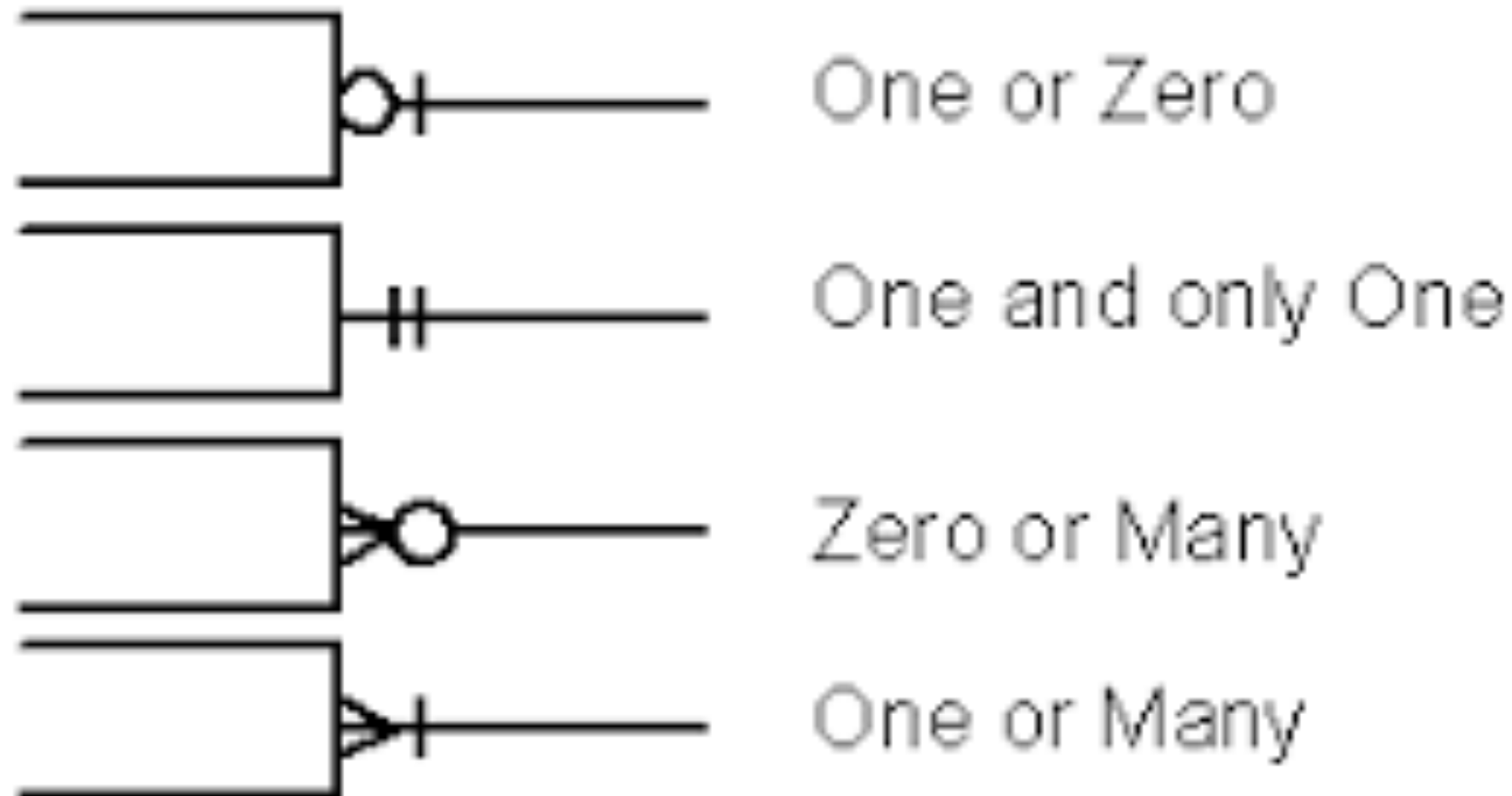


One or many



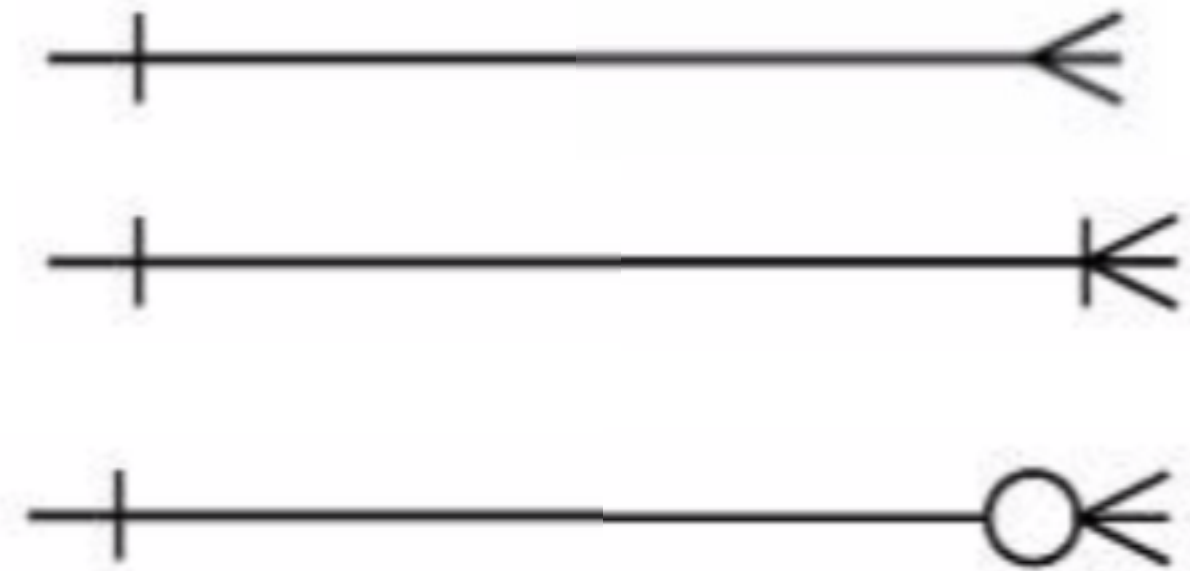
Zero or many

Summary of Crow's Foot Notation

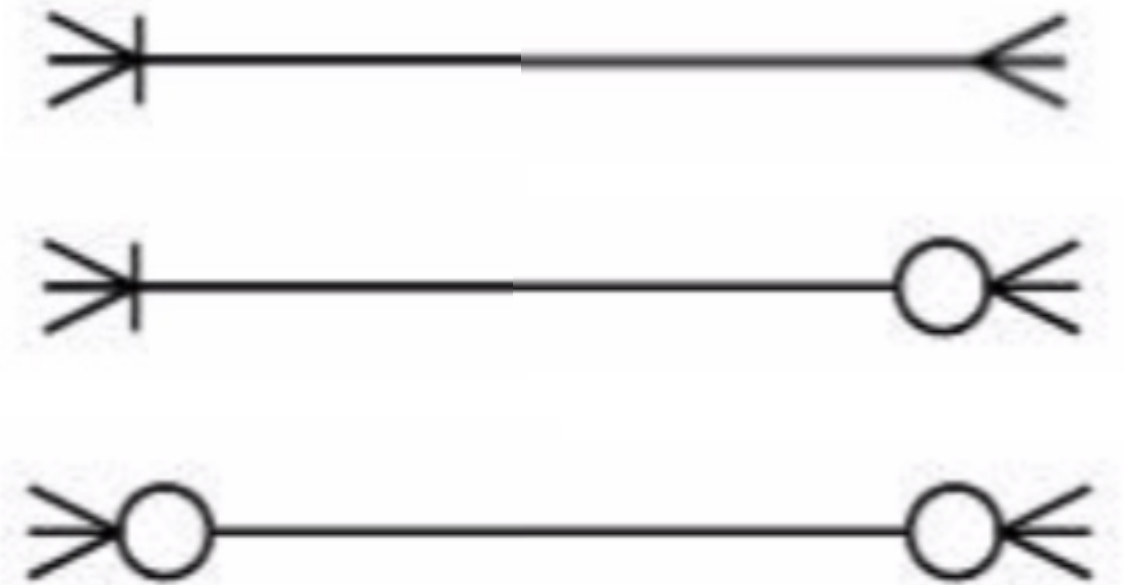


Relationships

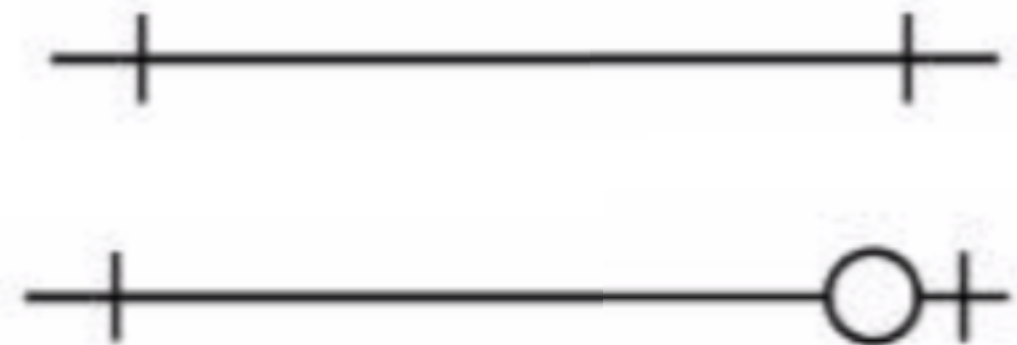
These are all one-to-many:



These are all many-to-many:

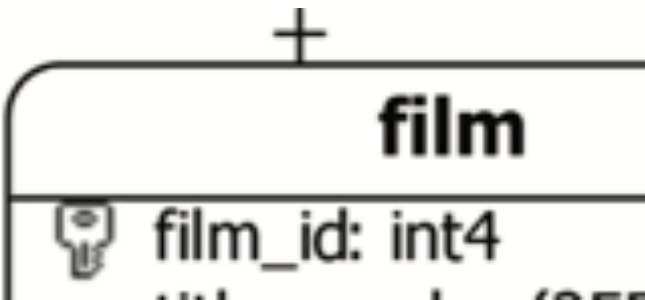


These are all one-to-one:



...etc, etc, etc

Relationships



Key



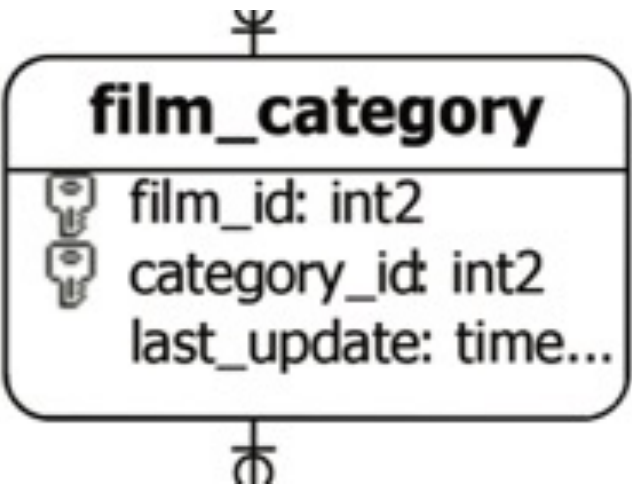
This end: one or none



This end: exactly one



This end: one or many



Join table (junction table): has two keys

Relationships

Orders	
	Column Name
	OrderID

Key

EmployeeTerritories		
	Column Name	Condensed T...
	EmployeeID	int
	TerritoryID	nvarchar(20)

Join table (junction table): has two keys

Relationships

One-to-one

Employee IDs to employee social security numbers

Dogs to current bookings

Husbands to wives

One-to-many

Employee ID numbers to employee names

Dogs to past bookings

Husbands to ex-wives

Many-to-many

Employees to projects

Dogs to buildings

Dating partners to dating partners

Cloud services

A blue-tinted photograph of a modern glass-walled building. The building's facade is composed of large glass panels that reflect the surrounding environment. In the foreground, a person is sitting on a white, rounded, modular bench. The ground is paved with light-colored tiles. The overall scene is modern and urban.

Our practice databases:
Stored in the AWS London datacentre



Amazon RDS

Metrics

×

Metric

DB Connec... ▼

Statistic

Average ▼

Time Range

Last 12 Ho... ▼

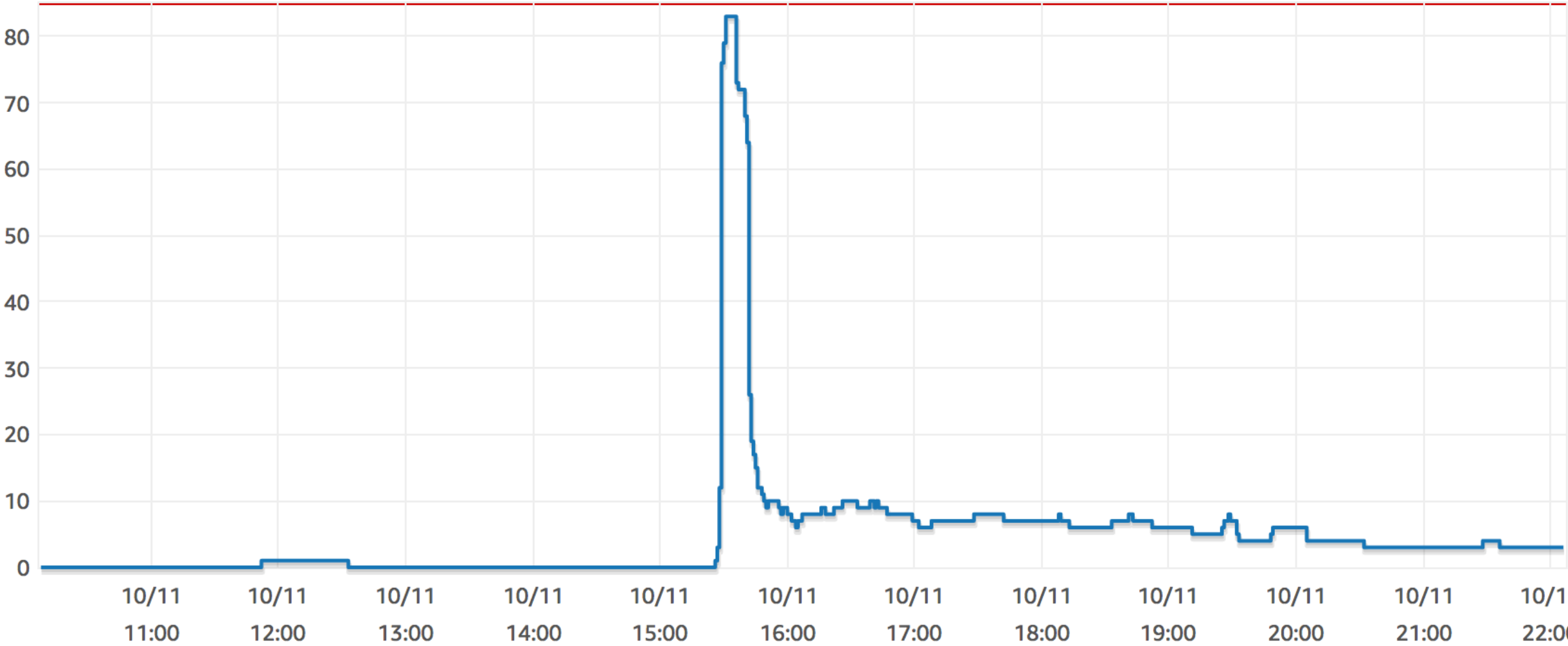
Period

1 Minute ▼

Legend: imperial

Refresh

Zoom out



Metrics

×

Metric

Statistic

Time Range

Period

CPU Utiliz... ▼

Average ▼

Last 12 Ho... ▼

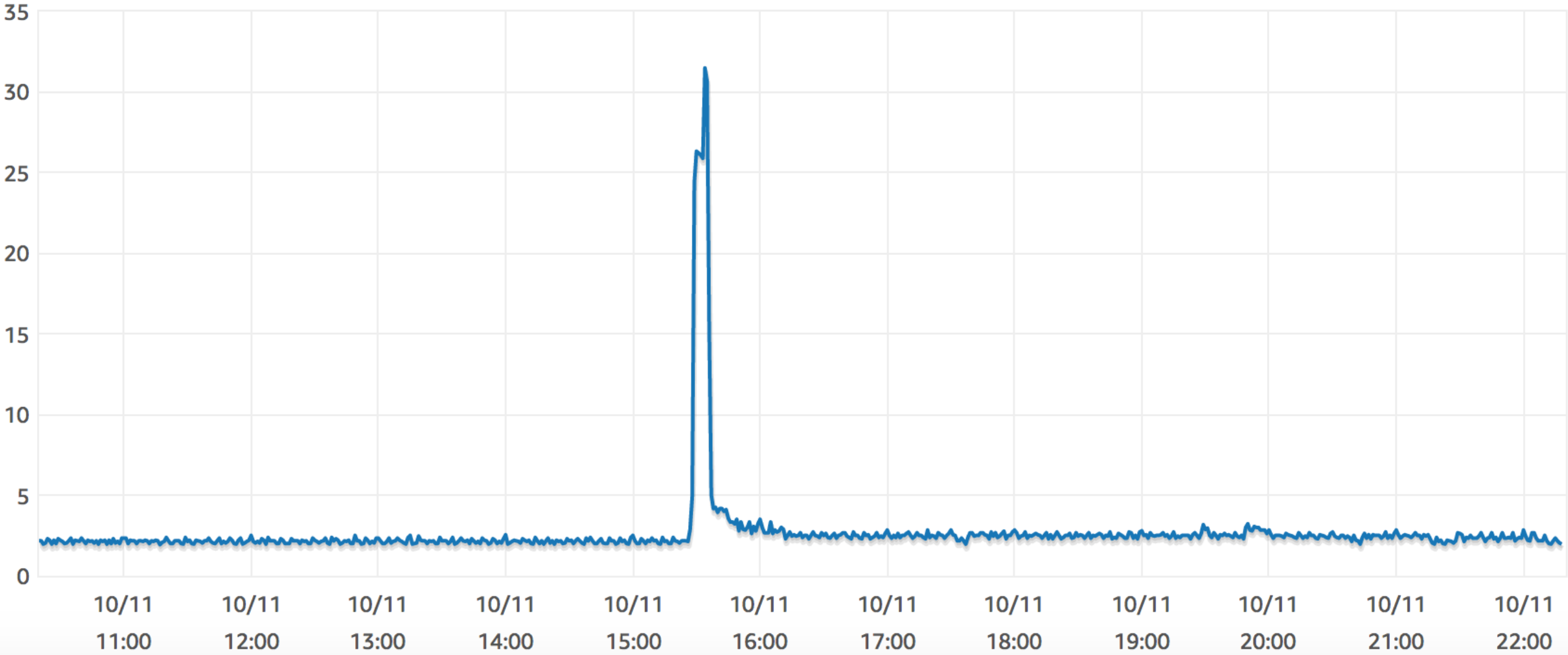
1 Minute ▼

Legend:

imperial

Refresh

Zoom out



[RDS](#) > [Instances](#) > [imperial](#) > [Modify](#)

Modify DB Instance: imperial

Instance specifications

DB engine version

Version number of the database engine to be used for this instance.

PostgreSQL 9.6.6-R1 ▾

DB instance class

Contains the compute and memory capacity of the DB instance.

db.t2.large — 2 vCPU, 8 GiB RAM ▾

Multi-AZ deployment

Specifies if the DB instance should have a standby deployed in another availability zone.

☐ Yes

☒ No

Storage type

General Purpose (SSD) ▾

Instance specifications

DB engine version

Version number of the database engine to be used for this instance.

PostgreSQL 9.6.6-R1



DB instance class

Contains the compute and memory capacity of the DB instance.

db.t2.large — 2 vCPU, 8 GiB RAM



db.t2.large — 2 vCPU, 8 GiB RAM

db.t2.xlarge — 4 vCPU, 16 GiB RAM

db.t2.2xlarge — 8 vCPU, 32 GiB RAM

db.m4.large — 2 vCPU, 8 GiB RAM

db.m4.xlarge — 4 vCPU, 16 GiB RAM

db.m4.2xlarge — 8 vCPU, 32 GiB RAM

db.m4.4xlarge — 16 vCPU, 64 GiB RAM

db.m4.10xlarge — 40 vCPU, 160 GiB RAM

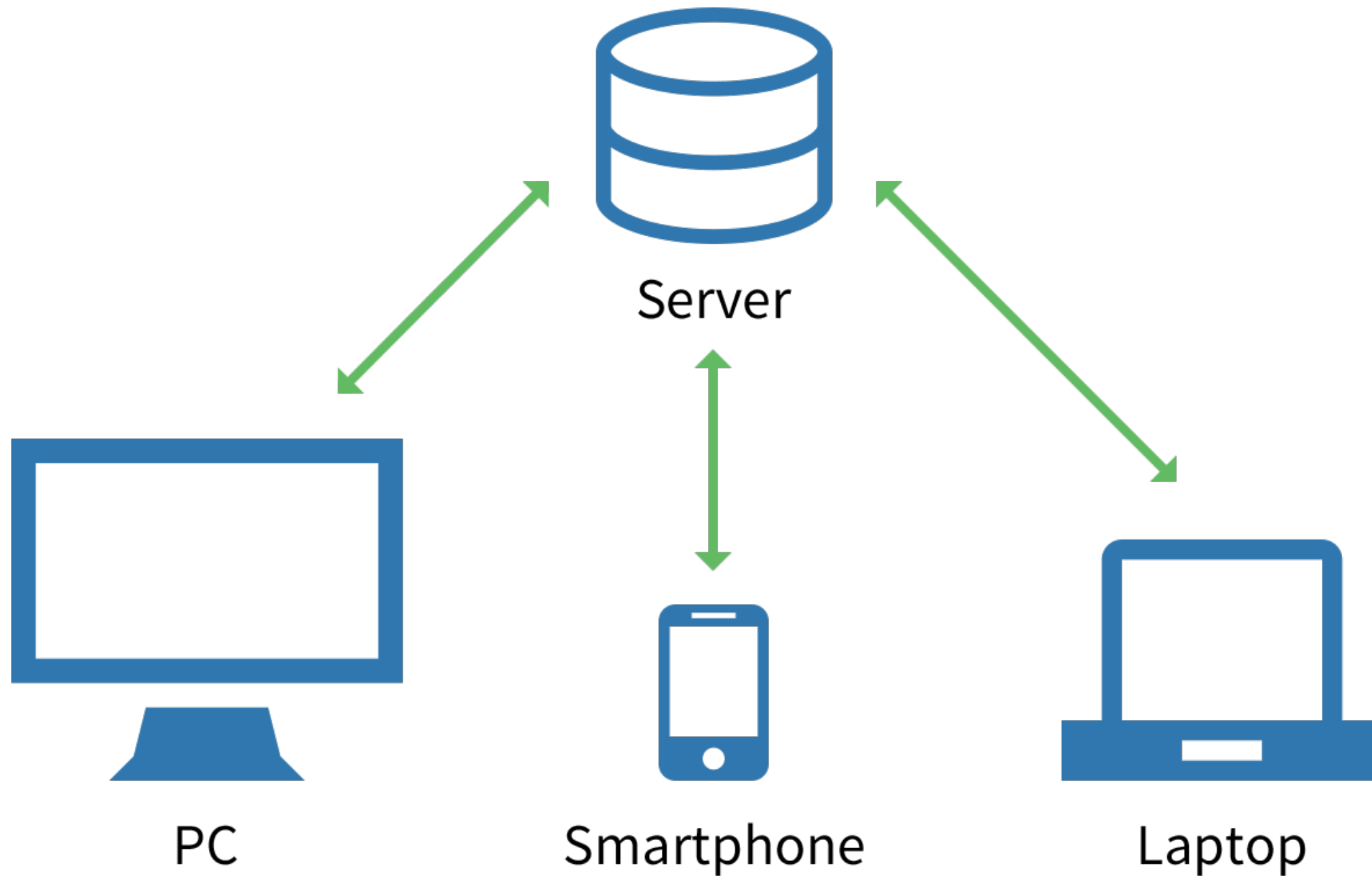
db.m4.16xlarge — 64 vCPU, 256 GiB RAM

This instance supports multiple storage ranges between 40 and 16384 GiB. [See all](#)



pgAdmin

Client-Server Model



Don't confuse the client and the server!

- Who stores the data?
- Who do you interact with directly?
- Who does the hard work?
- Who provides a text box for your query text?
- Who processes your query text?
- Who sorts the results?
- Who stores your results so that you can see them?
- Who lets you scroll up and down in your results?

Don't confuse the client and the server!

- Who stores the data? **THE SERVER**
- Who do you interact with directly? **THE CLIENT**
- Who does the hard work? **THE SERVER**
- Who provides a text box for your query text? **THE CLIENT**
- Who processes your query text? **THE SERVER**
- Who sorts the results? **THE SERVER**
- Who stores your results so that you can see them? **THE CLIENT**
- Who lets you scroll up and down in your results? **THE CLIENT**

Installing pgAdmin

- Get the software from www.pgadmin.org
- The latest version (4) runs in a browser; if you don't get on with this, you can download the previous version (3).



Our practice databases

- Host:

`imperial2023.cxtfduh3lk4q.eu-west-1.rds.amazonaws.com`

- Username: `postgres`
- Password: `imperial2023`
- Port: `5432`
- Database (maintenance DB): `dvdrental`

Getting to know a new database

- Find out what all the tables are called (*here we only have one*)
- In each table, look at the rows: what do they represent?
- In *most* tables, each row represents an entity, person, or object or some kind (*but this is not always true*)
- In each table, look at the columns: what do they represent?
- Keep the tables, rows and columns (the **schema**) where you can see them easily

Columns: attributes (fixed for each table)

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

Rows: entities or objects (any number of these)

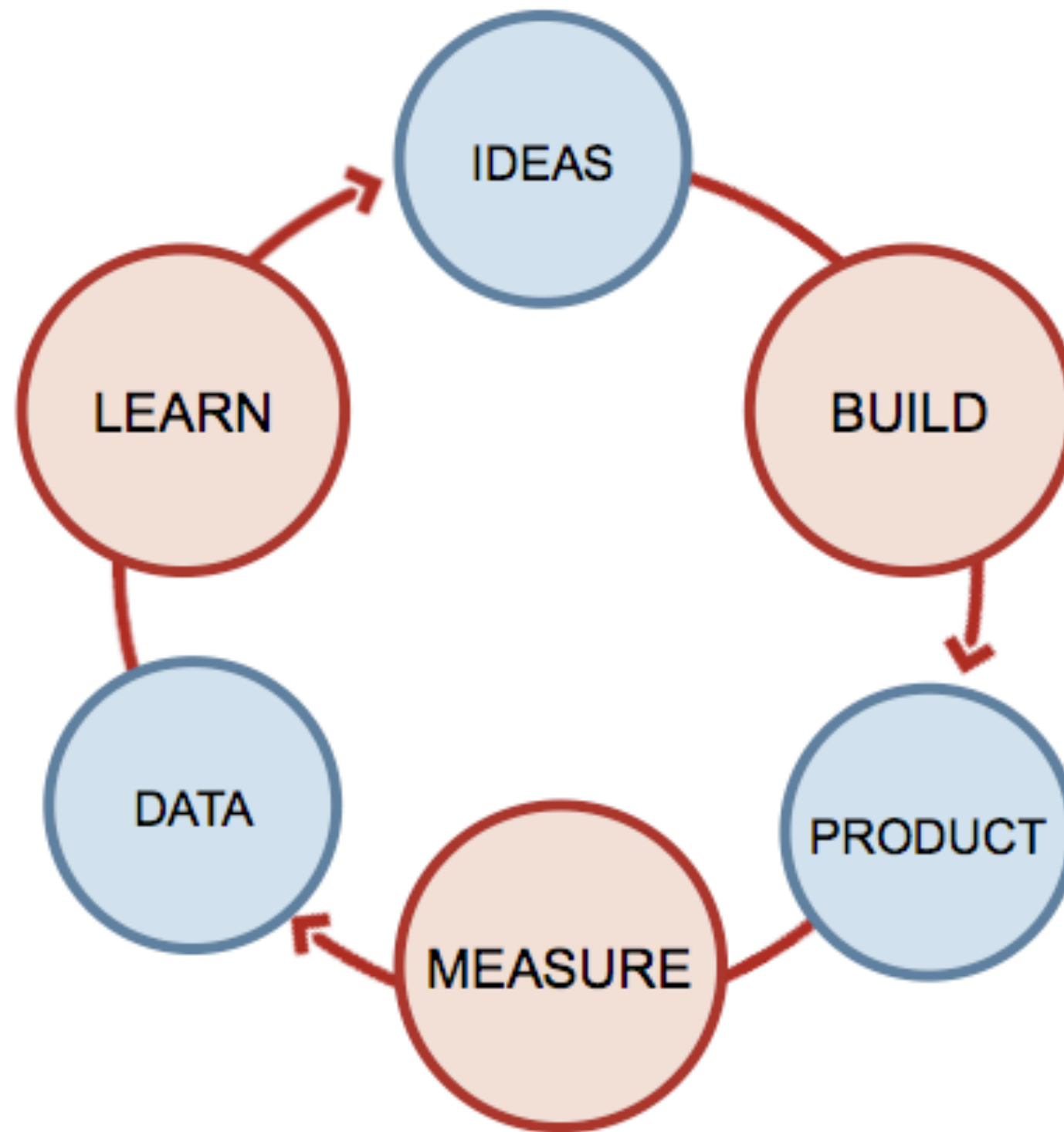
The Movies database

- **director_name** text
- **num_critic_for_reviews** integer
- **duration** integer
- **director_facebook_likes** integer
- **actor_3_facebook_likes** integer
- **actor_2_name** text
- **actor_1_facebook_likes** integer
- **gross** integer
- **genres** text
- **actor_1_name** text
- **movie_title** text
- **num_voted_users** integer
- **cast_total_facebook_likes** integer
- **actor_3_name** text
- **facenumber_in_poster** integer
- **plot_keywords** text
- **movie_imdb_link** text
- **num_user_for_reviews** integer
- **language** text
- **country** text
- **content_rating** text
- **budget** bigint
- **title_year** text
- **actor_2_facebook_likes** integer
- **imdb_score** real
- **aspect_ratio** text
- **movie_facebook_likes** integer

How to write a query

1. Which tables contain the information you need?
Check you understand what all the tables are for.
2. What result do I want exactly?
What columns are present in the result?
Approximately how many rows do you expect to return?
What does each row represent?

The development loop



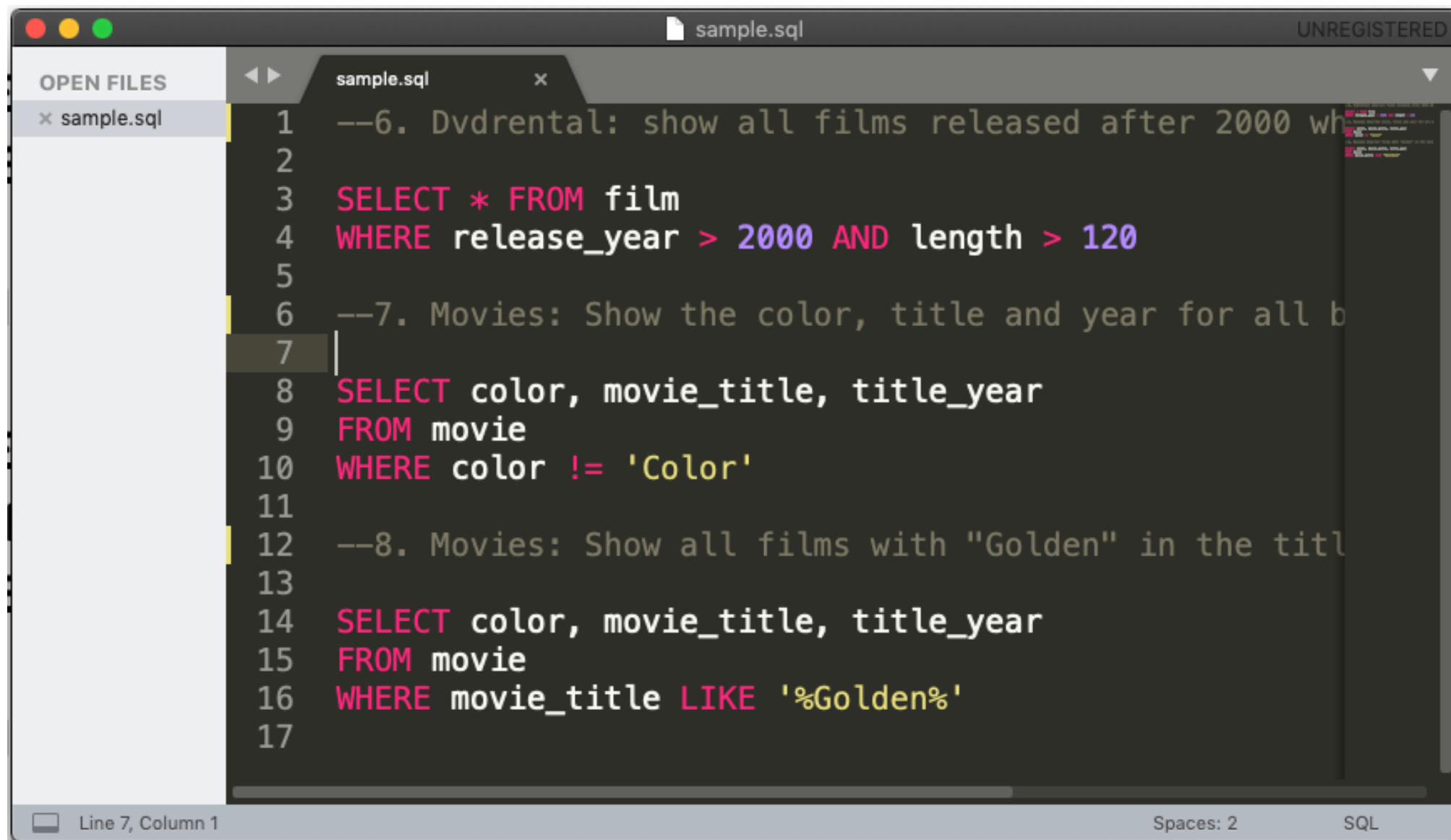
Saving and editing queries

You should use a **plain text editor** (as usually used for programming) to save a .sql file to keep track of your queries (not to run them).

An .sql file is just a normal text file!

You should open it in a good editor like Sublime Text (free to use) to benefit from syntax highlighting.

Comments are done with two dashes

A screenshot of a text editor window titled 'sample.sql'. The editor shows three SQL queries with syntax highlighting. The first query is a comment: '--6. Dvdrental: show all films released after 2000 wh'. The second query is 'SELECT * FROM film WHERE release_year > 2000 AND length > 120'. The third query is a comment: '--7. Movies: Show the color, title and year for all b'. The fourth query is 'SELECT color, movie_title, title_year FROM movie WHERE color != 'Color''. The fifth query is a comment: '--8. Movies: Show all films with "Golden" in the titl'. The sixth query is 'SELECT color, movie_title, title_year FROM movie WHERE movie_title LIKE '%Golden%'''. The editor has a sidebar on the left with 'OPEN FILES' and 'sample.sql'. The status bar at the bottom shows 'Line 7, Column 1', 'Spaces: 2', and 'SQL'.

The most basic query

```
SELECT * FROM table_name;
```

```
SELECT * FROM movies;
```

*The semicolon is usually optional.
However, some software, like PSQL,
requires that we type the semicolon.*

The structure of a query

A simple query:

- SELECT
- FROM
- WHERE
- ORDER BY
- LIMIT

A more complex query:

- SELECT
- FROM
- **JOINS, each with an ON**
- WHERE
- **GROUP BY**
- ORDER BY
- LIMIT

What do rows represent?

Rows can represent:

- People
- Real objects
- Imaginary objects
- Concepts
- Events (sales, rentals)
- Contracts
- Facts
- Debts
- ... etc

Building your own databases

Postgres data types:

text	string of any length	
integer	4-byte signed	-2147483648 to +2147483647
bigint	8-byte signed	-9223372036854775808 to 9223372036854775807
serial	4-byte autoincrementing	
real	4-byte floating point	
double	8-byte floating point	
money	currency	
bool or boolean	true/false	

`varchar(size)` length-limited string example: `varchar(10)` allows ten characters

DO NOT USE VARCHAR unless there is a **BUSINESS CONSTRAINT** imposing the limit (it does not save space and it is extremely annoying for users)

Buiding your own databases

SQL only specifies the integer types integer (or int), smallint, and bigint. The type names int2, int4, and int8 are extensions, which are also used by some other SQL database systems.

Postgres date/time types:

date

timestamp

timestamp with timezone

time

time with timezone

(Not exhaustive, see documentation)

Creating a new table

```
CREATE TABLE table_name(  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
    columnN datatype  
);
```


Buiding your own databases

```
CREATE TABLE students(  
  id serial,  
  first_name text,  
  last_name text,  
  gender text,  
  age integer,  
  favourite_colour text,  
  height real,  
  favourite_animal text)
```

Buiding your own databases

```
INSERT INTO students VALUES
```

```
('joe', 'bloggs', 'm', 22, 'Britain', 'Europe', 'dog')
```

```
INSERT INTO films (code, title, did, date_prod, kind) VALUES ('T_601', 'Yojimbo', 106,  
'1961-06-16', 'Drama');
```

```
INSERT INTO students (first_name, last_name, gender, age, favourite_colour, height,  
favourite_animal) VALUES ('Fintan', 'Nagle', 'M', 33, 'ultraviolet', 182, 'dog');
```

Insert subset of values:

```
INSERT INTO students (first_name, last_name, gender, age) VALUES ('Fintan', 'Nagle',  
'M', 33
```

Buiding your own databases

Insert multiple rows at once:

```
INSERT INTO films (code, title, did, date_prod, kind) VALUES ('B6717', 'Tampopo', 110, '1985-02-10', 'Comedy'), ('HG120', 'The Dinner Game', 140, DEFAULT, 'Comedy');
```