Introduction to PostgreSQL

Sample manual - first two chapters

Wise Owl Training

TABLE OF CONTENTS (1 of 5)

TABLE OF CONTENTS (2 of 5)

TABLE OF CONTENTS (3 of 5)

TABLE OF CONTENTS (4 of 5)

TABLE OF CONTENTS (5 of 5)

CHAPTER 1 - DESIGNING DATABASES

The world runs on relational databases. If you understand the principles upon which these are built, you'll find it much easier to write SQL to get information out of them!

Wise Owl's Hint

*This manual gives an overview only of database design principles. If you want to delve deeper, try Googling phrases like **Third Normal Form**, **Database Normalisation** or **Entity Diagram**. If nothing else, this will give you an impressive search history in your browser!*

1.1 The Four Stages of Database Design

There are four stages to designing a relational database, shown below (using the example of creating a simple database to hold films; or movies, if you must).

Stage 1 – Deciding what to Include

A good way to do this is to create a spreadsheet of the data you want to include for each film:

Title	Oscars	Director	Date of birth	Studio
Armageddon	0	Michael Bay	17/02/1965	Touchstone Pictures
Bad Boys	0	Michael Bay	17/02/1965	Jerry Bruckheimer Films
Bad Boys II	0	Michael Bay	17/02/1965	Jerry Bruckheimer Films
Dead Poets Society	1	Peter Weir	21/08/1944	Touchstone Pictures
Master and Commander ...	2	Peter Weir	21/08/1944	20th Century Fox
Pearl Harbor	1	Michael Bay	17/02/1965	Touchstone Pictures
The Rock	0	Michael Bay	17/02/1965	Hollywood Pictures
The Truman Show	0	Peter Weir	21/08/1944	Scott Rudin Productions

We want to assign each film to a director, but we don't want to have to type each director's name in over and over again!

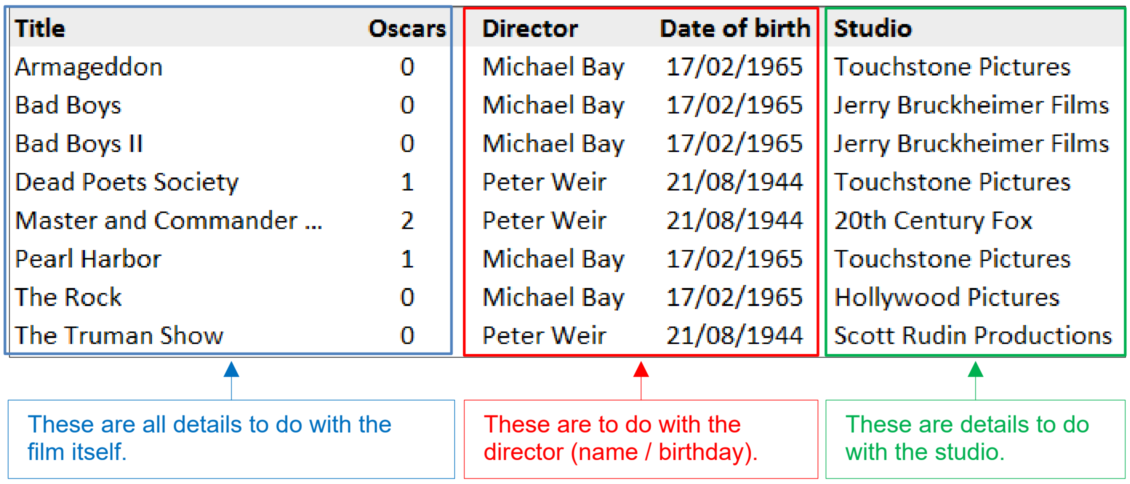
The aim of designing a relational database is to ensure that you don't hold information twice:

Title	Oscars	Director
Dead Poets Society	1	Peter Weir
Master and Commander ..	2	Peter Wier
The Truman Show	0	Peter Weird

Not only is holding duplicate information inefficient, but it also means that spelling mistakes will creep in. Here listing out films directed by **Peter Weir** would miss out the last two films, as his name has been misspelt.

Stage 2 – Dividing Data into Tables

Having decided what data you want to include, the next stage of database design is to decide which table each bit of information belongs to:



Title	Oscars	Director	Date of birth	Studio
Armageddon	0	Michael Bay	17/02/1965	Touchstone Pictures
Bad Boys	0	Michael Bay	17/02/1965	Jerry Bruckheimer Films
Bad Boys II	0	Michael Bay	17/02/1965	Jerry Bruckheimer Films
Dead Poets Society	1	Peter Weir	21/08/1944	Touchstone Pictures
Master and Commander ...	2	Peter Weir	21/08/1944	20th Century Fox
Pearl Harbor	1	Michael Bay	17/02/1965	Touchstone Pictures
The Rock	0	Michael Bay	17/02/1965	Hollywood Pictures
The Truman Show	0	Peter Weir	21/08/1944	Scott Rudin Productions

These are all details to do with the film itself.

These are to do with the director (name / birthday).

These are details to do with the studio.

Wise Owl's Hint

There's no magic wand to make this easier, other than bitter experience of getting it wrong and having to start again! A good guideline is that if you find yourself typing in something twice, it probably belonged in a different table.

For our example above, there are clearly 3 separate entities: films, the directors who made them and the studios which produced them. Here are the fields that each table could contain:

Table	Fields
Film	**Title** and **Oscars Won**, plus something to identify which director and which studio made it
Director	**Director name** and **Date of birth**, plus some unique identifier for the director
Studio	**Studio name**, plus some unique identifier for the studio

What you need to do next is to decide what form these unique identifiers should take.

Stage 3 – Choosing a Primary Key for each Table

The *primary key* for a table is a field which tells you exactly which record you're considering (for example, if you know a film's **director_id** you can look up all of the director's other details).

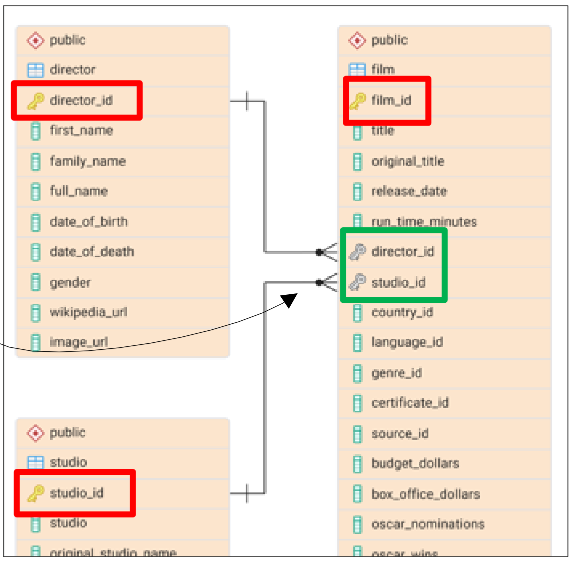
Wise Owl's Hint

From the above definition, it follows that two records in a table can't have the same value for the primary key field – the field is unique.

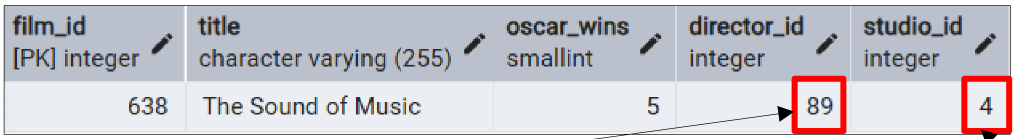
For our example, we could use the director and studio names as our primary keys, but *PostgreSQL* works most efficiently if the primary key is as short as possible, so we'll create new fields instead:

The primary key for each table is a number which uniquely identifies which studio, film or director we're looking at.

Each film contains fields specifying who directed the film, and which studio made it.

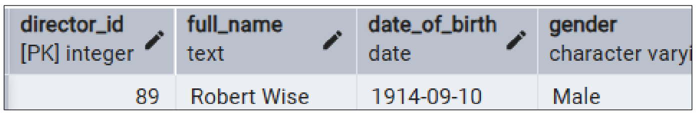


Here's what **The Sound of Music** would now look like:



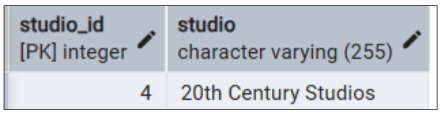
film_id [PK] integer	title character varying (255)	oscar_wins smallint	director_id integer	studio_id integer
638	The Sound of Music	5	89	4

Including the director's unique number allows us to look up all their other details:



director_id [PK] integer	full_name text	date_of_birth date	gender character varyi
89	Robert Wise	1914-09-10	Male

Including the studio's unique number allows us to look up its name:



studio_id [PK] integer	studio character varying (255)
4	20th Century Studios

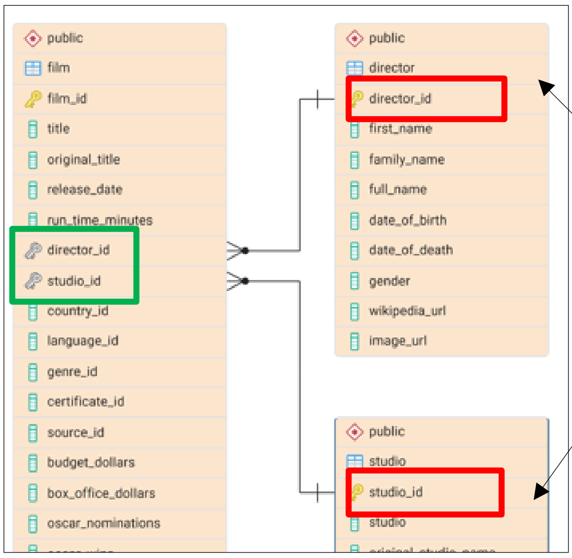
Wise Owl's Hint

*If you're beginning to think that relational databases are just like a lot of spreadsheets joined together with a more efficient version of a **VLOOKUP** or **XLOOKUP** formula in Excel, you're absolutely right!*

Stage 4 – Creating Relationships and a Database Diagram

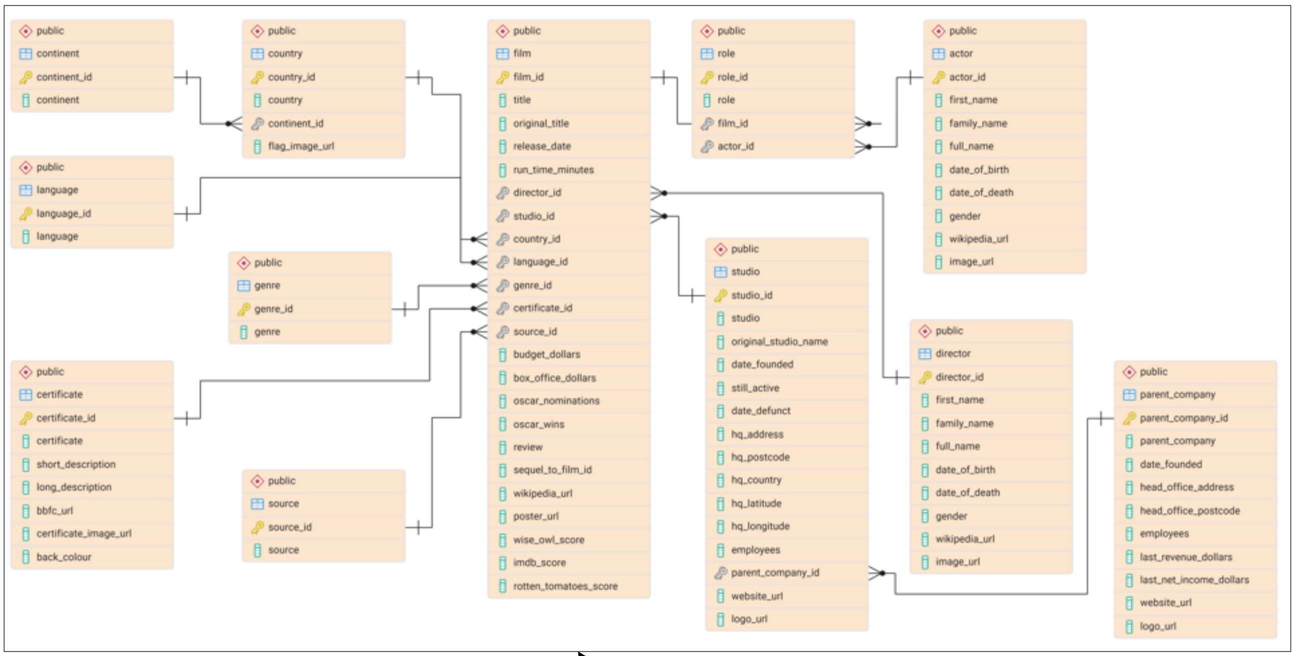
The last step in designing a database is to decide for each relationship that you create whether it is *one-to-many* or *many-to-one* (*parent-child* or *child-parent*):

These are the *foreign keys* in each relationship (the fields at the child end of the line). Each director id, for example, can appear once and once only in the director table, but can and will be repeated many times in the **Film** table (once for each film that a director has made).



These are the *primary keys* in the relationship (the field at the parent end of the line). Each director can make many films, and likewise each studio can release many films.

Database diagrams often involve hundreds of tables:



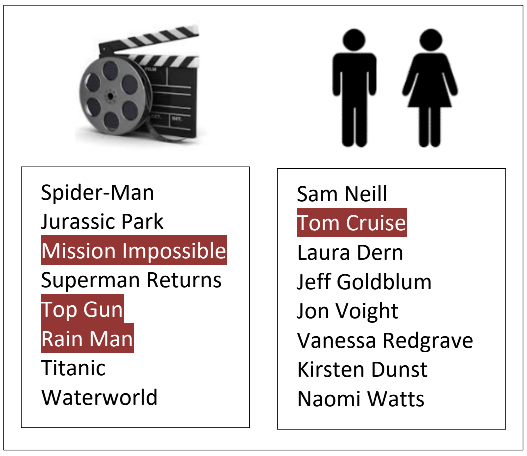
Our **Movies** database contains just 12 tables, all of which are shown above, and hence is untypically simple.

Wise Owl's Hint

The above diagrams were created using the ERD Entity-Relationship Diagram facility in PG Admin (read on for how to install this).

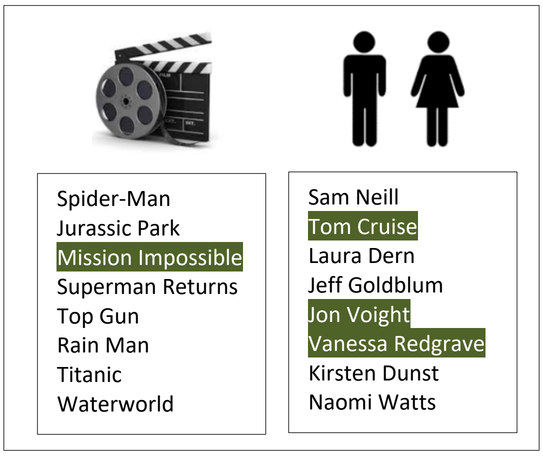
1.2 Many-to-Many Relationships

There's no such thing as a many-to-many relationship in *PostgreSQL*, but they do exist in real life:



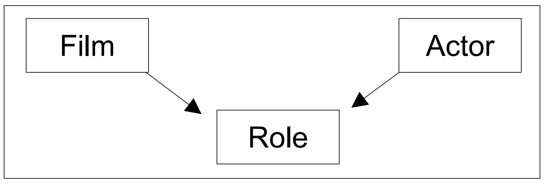
Spider-Man	Sam Neill
Jurassic Park	Tom Cruise
Mission Impossible	Laura Dern
Superman Returns	Jeff Goldblum
Top Gun	Jon Voight
Rain Man	Vanessa Redgrave
Titanic	Kirsten Dunst
Waterworld	Naomi Watts

Tom Cruise *has appeared in lots of films, but equally* ***Mission: Impossible*** *has lots of actors in it.*

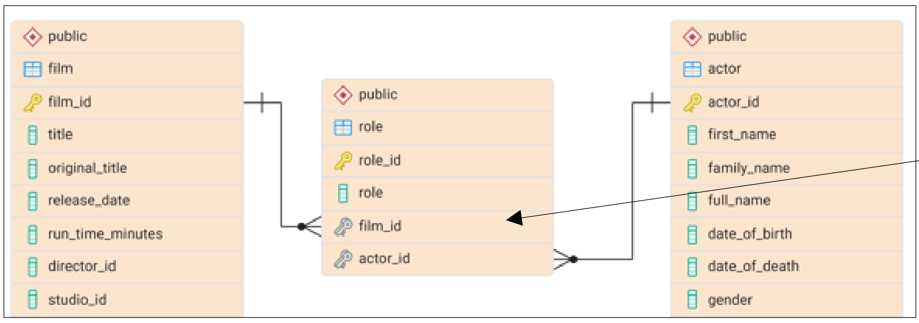


Spider-Man	Sam Neill
Jurassic Park	Tom Cruise
Mission Impossible	Laura Dern
Superman Returns	Jeff Goldblum
Top Gun	Jon Voight
Rain Man	Vanessa Redgrave
Titanic	Kirsten Dunst
Waterworld	Naomi Watts

The solution to this problem is to create a table that is a child to both of the two parent tables, as here:



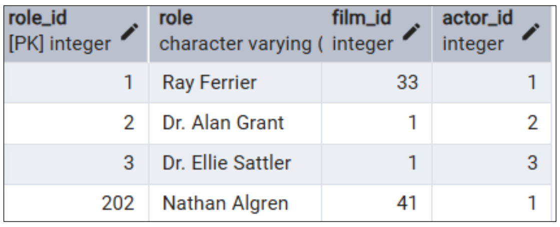
Here's what the database would look like:



The **Role** table links the **Film** and **Actor** tables. Each film can contain many roles (otherwise a film could only have a single actor), but likewise each actor can have many roles (otherwise they would never work again after completing their first film).

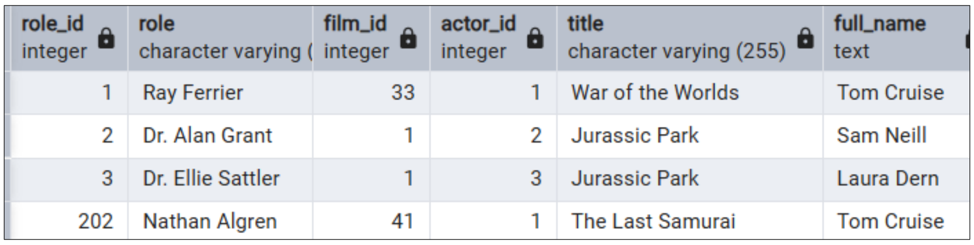
Here are 3 rows from the **Role** table:

Film number 1 appears twice in this list, as does actor number 1.



role_id [PK] integer	role character varying (	film_id integer	actor_id integer
1	Ray Ferrier	33	1
2	Dr. Alan Grant	1	2
3	Dr. Ellie Sattler	1	3
202	Nathan Algren	41	1

Here are the films and actors who are represented by these rows of data (the duplicate film name was **Jurassic Park**, and duplicated actor turns out to be **Tom Cruise**).



role_id integer	role character varying (	film_id integer	actor_id integer	title character varying (255)	full_name text
1	Ray Ferrier	33	1	War of the Worlds	Tom Cruise
2	Dr. Alan Grant	1	2	Jurassic Park	Sam Neill
3	Dr. Ellie Sattler	1	3	Jurassic Park	Laura Dern
202	Nathan Algren	41	1	The Last Samurai	Tom Cruise

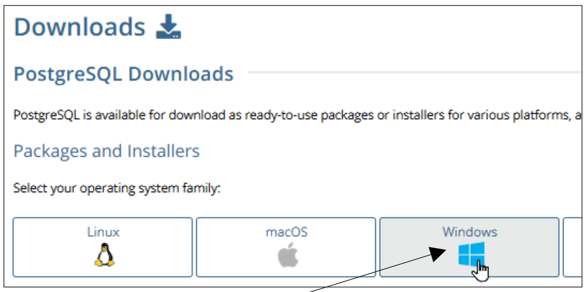
CHAPTER 2 - GETTING STARTED WITH POSTGRESQL

2.1 Getting Started with PostgreSQL

Postgres was created in 1986 as the successor to the older *INGRES* database system (hence the name *Postgres*, although it's also often called *PostgreSQL*). *Postgres* usually comes in two parts.

The Command Line Interface

You first need to install the core *PostgreSQL* database from https://www.postgresql.org/download/ :



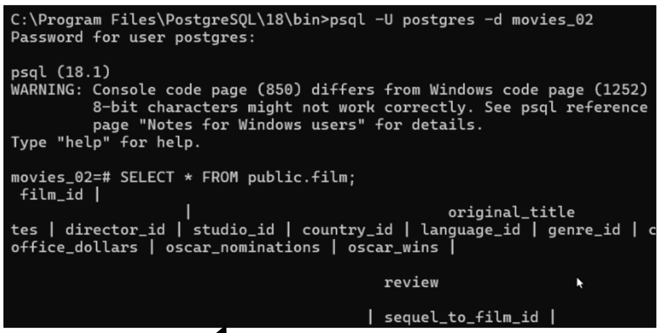
Downloads

PostgreSQL Downloads

PostgreSQL is available for download as ready-to-use packages or installers for various platforms, a

Packages and Installers

Select your operating system family:



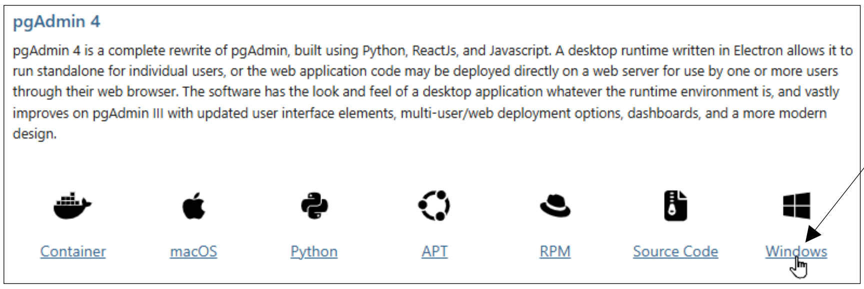
```
C:\Program Files\PostgreSQL\18\bin>psql -U postgres -d movies_02
Password for user postgres:

psql (18.1)
WARNING: Console code page (850) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

movies_02=# SELECT * FROM public.film;
 film_id |
                   |                               original_title
tes | director_id | studio_id | country_id | language_id | genre_id | c
office_dollars | oscar_nominations | oscar_wins |

                                         review

                                       | sequel_to_film_id |
```

Installing *PostgreSQL* from the official website is simple – and free!

An example of using the command line interface to run *PostgreSQL* commands. There is no reason the author can think why you would ever want to do this!

The GUI User Interface

There are a few graphical interfaces available for using Postgres, but by far the most common (and hence the best choice) is *PG Admin*:



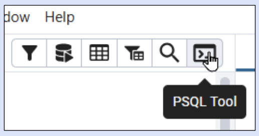
pgAdmin 4

pgAdmin 4 is a complete rewrite of pgAdmin, built using Python, ReactJs, and Javascript. A desktop runtime written in Electron allows it to run standalone for individual users, or the web application code may be deployed directly on a web server for use by one or more users through their web browser. The software has the look and feel of a desktop application whatever the runtime environment is, and vastly improves on pgAdmin III with updated user interface elements, multi-user/web deployment options, dashboards, and a more modern design.

PG Admin is also simple to install, and also free! It gives you this icon if you add it to your desktop:

Wise Owl's Hint

Should you ever want to get at the PostgreSQL *command line interface from within PG Admin, you can. Just click on this tool on the Object Explorer toolbar:*

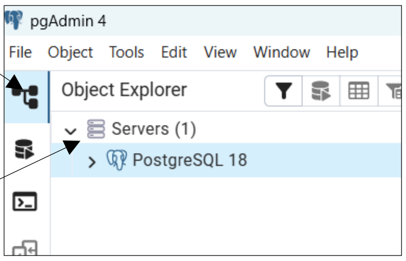


2.2 Object Explorer

You will spend nearly all your time in *PostgreSQL* using *Object Explorer*:

Should you ever accidentally leave Object Explorer, you can return to it by clicking on this icon.

Theoretically you could use Object Explorer to navigate between different installed instances or versions of *PostgreSQL*, but in practice you'll only ever have one instance.

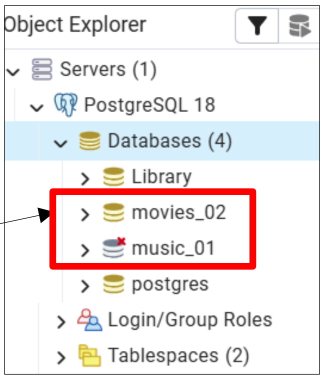


Listing Databases

You can expand the *PostgreSQL* server to see the databases created:

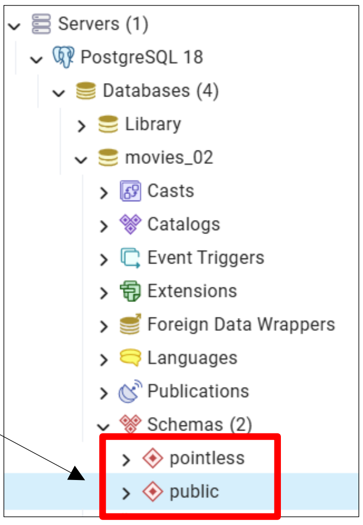
The **Library** and **Postgres** databases are system ones, which you should ignore. The other two databases are as follows:

Database	What it contains
movies_02	The films, directors, studios, actors and roles in the Wise Owl movies database, referenced throughout this courseware.
music_01	The artists, albums, songs and tours referenced by the exercises used in the Wise Owl SQL courses.



Listing Schemas

Expanding a database further will give you a list of the *schemas* it contains:



This database contains two schemas, although only one (**public**) is used. Creating different schemas serves two purposes:

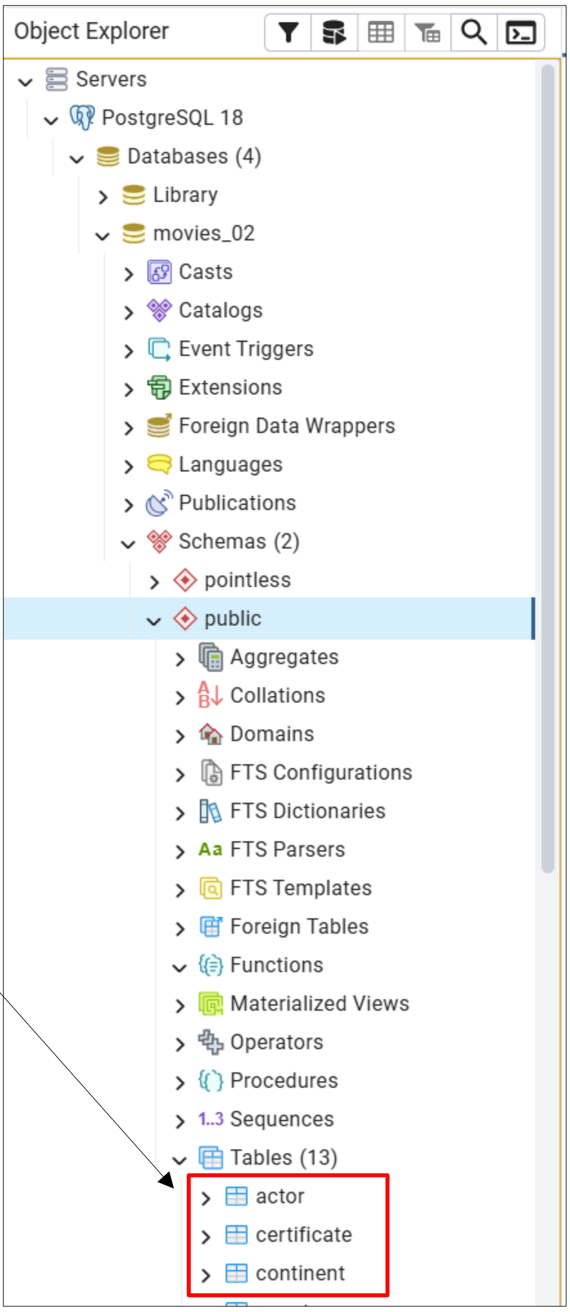
- It lets you group tables into different sets; and
- It allows you to apply different security rules to different sets of tables (for example, people in HR might be able to see the **hr** schema tables but not the **accounts** ones).

Wise Owl's Hint

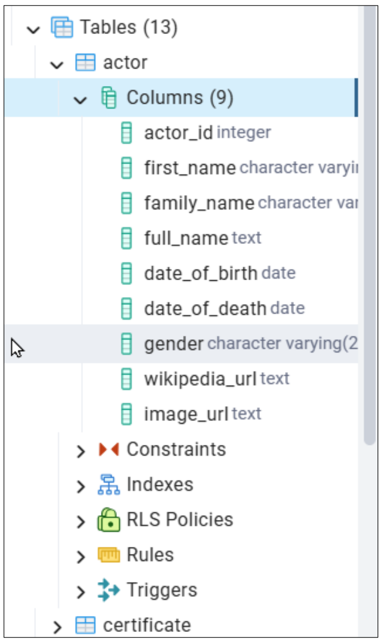
When you create a new table it will (by default) join the public schema. Usually the best thing to do it to accept this default, as it means that you can find all of your tables in the same place.

Listing Tables

If you keep expanding the Object Explorer nodes for long enough, you will eventually get to the tables in a database:

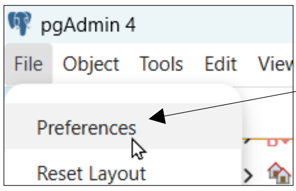


The first few tables in this database. You can expand any table further to see the columns that it contains. Here for example are the columns in the **actor** table:

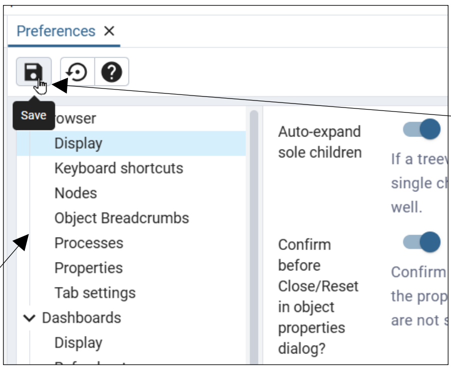


2.3 Setting Preferences

Everybody will have their own preferred way of working, but the author will always change 3 preferences in PG Admin before doing anything else.



a) Choose this **Preferences** option from the PG Admin **File** menu.

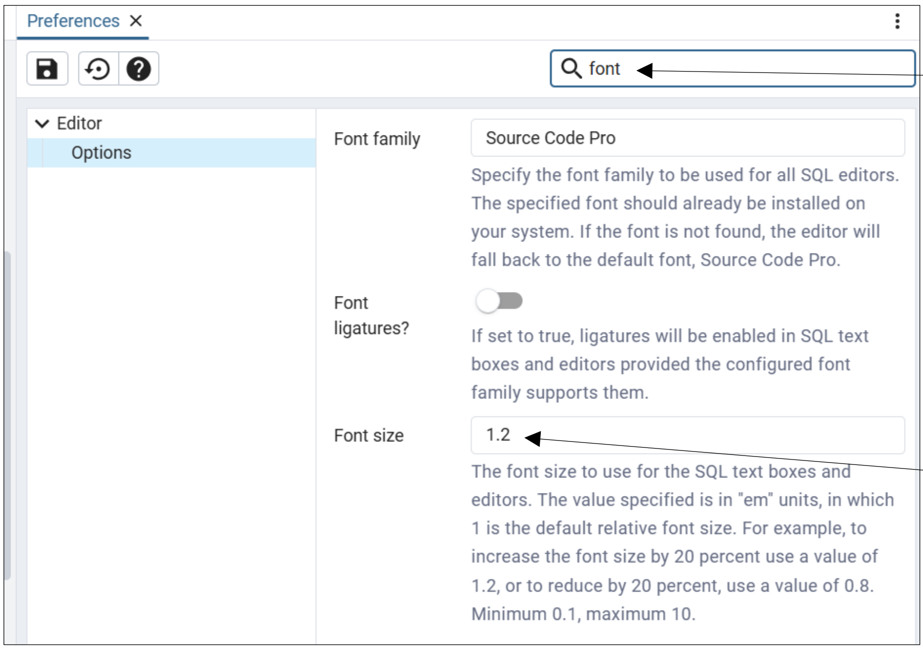


c) Crucially, remember to click on the save button to save your changes!

b) Choose the preference category on the left that you want to work with, then change one or more properties within it.

Change 1: Setting the Default Font Size

This owl finds that SQL in PG Admin is much more readable at 20% bigger than the default size!



Font family: Source Code Pro

Specify the font family to be used for all SQL editors. The specified font should already be installed on your system. If the font is not found, the editor will fall back to the default font, Source Code Pro.

Font ligatures?

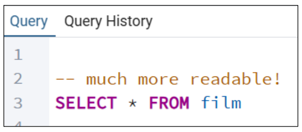
If set to true, ligatures will be enabled in SQL text boxes and editors provided the configured font family supports them.

Font size: 1.2

The font size to use for the SQL text boxes and editors. The value specified is in "em" units, in which 1 is the default relative font size. For example, to increase the font size by 20 percent use a value of 1.2, or to reduce by 20 percent, use a value of 0.8. Minimum 0.1, maximum 10.

a) Type **font** into the search box to find properties to do with fonts.

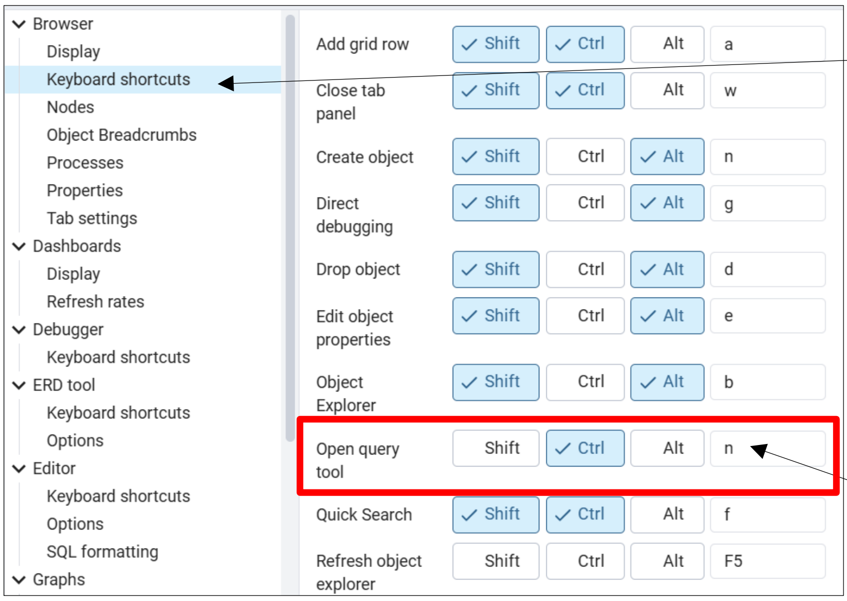
b) Change the font size to a multiple of 1 (here we've gone for 1.2, or 120% of the default size) to get much more readable SQL!



```

-- much more readable!
SELECT * FROM film
```

Change 2: Changing the New Query Short-Cut

You will create many new queries in PG Admin. The default short-cut key is Ctrl + Alt + Q which is hard to remember – this owl changes this to Ctrl + N :



- Browser
  - Display
  - Keyboard shortcuts
  - Nodes
  - Object Breadcrumbs
  - Processes
  - Properties
  - Tab settings
- Dashboards
  - Display
  - Refresh rates
- Debugger
  - Keyboard shortcuts
- ERD tool
  - Keyboard shortcuts
  - Options
- Editor
  - Keyboard shortcuts
  - Options
  - SQL formatting
- Graphs

Add grid row	✓ Shift	✓ Ctrl	Alt	a
Close tab panel	✓ Shift	✓ Ctrl	Alt	w
Create object	✓ Shift	Ctrl	✓ Alt	n
Direct debugging	✓ Shift	Ctrl	✓ Alt	g
Drop object	✓ Shift	Ctrl	✓ Alt	d
Edit object properties	✓ Shift	Ctrl	✓ Alt	e
Object Explorer	✓ Shift	Ctrl	✓ Alt	b
Open query tool	Shift	✓ Ctrl	Alt	n
Quick Search	✓ Shift	✓ Ctrl	Alt	f
Refresh object explorer	Shift	Ctrl	Alt	F5

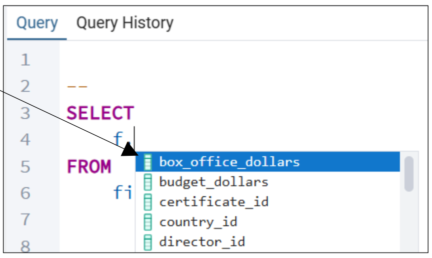
a) Choose the **Keyboard shortcuts** category within the **Browser** tab.

b) Tick the combination of Shift , Ctrl and Alt keys which you want to have to press to create a new query and type in the letter to accompany them (here n).

Change 3: Enabling Intellisense Automatically

For some strange reason the automatic completion of commands shown on the right doesn't appear by default in PG Admin.

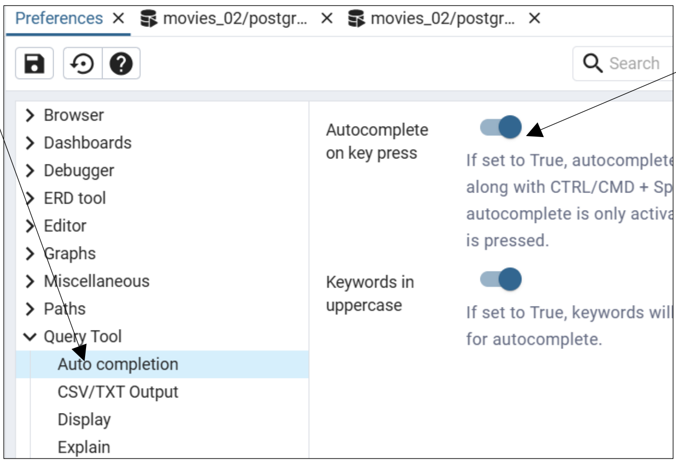
Normally you have to press Ctrl + Space Bar to get this list of possible columns to appear.



```
--
SELECT
    f
FROM
    fi
```

Here's how to change this:

In preferences select the **Auto completion** tab in the **Query Tool** category.



Autocomplete on key press

If set to True, autocomplete along with CTRL/CMD + Sp autocomplete is only activa is pressed.

Keywords in uppercase

If set to True, keywords wil for autocomplete.

b) Turn this feature on. You'll thank yourself (and us)!

What we do!

		Basic training	Advanced training	Systems / consultancy
Office	Microsoft Excel	✓	✓	✓
	VBA macros	✓	✓	✓
	Office Scripts	✓		
	Microsoft Access			✓
Power BI, etc	Power BI and DAX	✓	✓	✓
	Power Apps	✓		
	Power Automate (both)	✓	✓	
SQL Server	SQL	✓	✓	✓
	Reporting Services	✓	✓	✓
	Report Builder	✓	✓	✓
	Integration Services	✓	✓	✓
	Analysis Services	✓		
Coding	Visual C#	✓	✓	✓
	VB programming			✓
	MySQL	✓		
	Python	✓	✓	✓

Wise Owl Training