Introduction to PostgreSQL

Sample manual - first two chapters

Wise Owl Training

TABLE OF CONTENTS (1 of 5)

1	DESIGNING DATABASES	Page
1.1	The Four Stages of Database Design	7
	Stage 1 – Deciding what to Include	7
	Stage 2 – Dividing Data into Tables	8
	Stage 3 – Choosing a Primary Key for each Table	9
	Stage 4 – Creating Relationships and a Database Diagram	10
1.2	Many-to-Many Relationships	11

2	GETTING STARTED WITH POSTGRESQL	Page
2.1	Getting Started with PostgreSQL	12
	The Command Line Interface	12
	The GUI User Interface	12
2.2	Object Explorer	13
	Listing Databases	13
	Listing Schemas	13
	Listing Tables	14
2.3	Setting Preferences	15
	Change 1: Setting the Default Font Size	15
	Change 2: Changing the New Query Short-Cut	16
	Change 3: Enabling Intellisense Automatically	16

3	CREATING TABLES	Page
3.1	Creating our Sample Database	17
3.2	Creating a Table	18
3.3	Creating Columns	19
	Adding Columns to a Table while Creating It	19
	Adding Columns to an Existing Table	20
	Setting the Primary Key	21
	Setting Identity Columns	21
	Setting Default Values	22
	Allowing or Disallowing Nulls	22
3.4	Postgres Data Types	23
	Data Types Explained in this Chapter	23
	Integer Field Types	23
	Boolean Field Type	24
	Decimal and Numeric Field Types	24
	Double Precision and Real Data Types	24
	String Data Types	25
	Date/Time Data Types	25
3.5	Entering Data in Tables	26

4	RELATIONSHIPS BETWEEN TABLES	Page
4.1	Understanding Relationships	27
4.2	Creating a Relationship	28
4.3	ERD Diagrams	29
	Creating an ERD Diagram from a Database	29
	Creating an ERD Diagram from Scratch	30
	Adding Relationships	31
	Other Options Available	32

5	CREATING QUERIES	Page
5.1	Creating Queries	33
	Viewing a Query's Database	34
	Managing Query Windows	34
5.2	Basic SELECT Statements	35
	Where to Put your Commas	35
	Using Autocomplete	36
5.3	Running Queries	37
	Executing a Query	37
	Cancelling a Running Query	37
	Explaining a Query	38
	The Data Output Window	38
	Running Multiple SQL Commands in the Same Query	39
	Dealing with Errors	39
	Copying the Rows Returned from a Query	40
5.4	Saving, Opening and Closing Queries	41
	Saving Queries	41
	Opening Queries	41
5.5	Using Upper and Lower Case	42
	Changing the Default Case of SQL Keywords	42
5.6	Indentation and Word Wrap	43
	Changing Indentation	43
	Word Wrap	43
5.7	Comments	44
	Commenting Out Blocks of Code	44
5.8	Colours in SQL	45
5.9	Auto-formatting SQL	46

© Wise Owl Business Solutions Ltd

TABLE OF CONTENTS (2 of 5)

6	THE SELECT STATEMENT	Page
6.1	SELECT Statement Syntax	47
	Mnemonic for Order of Commands	47
6.2	Qualified Tables and Columns	48
	Dragging Tables/Columns onto a Query	48
	Schemas	48
6.3	Table Aliases	48
	Reason 1 for Aliases – Easier to Refer to Field Names	49
	Reason 2 for Aliases – Joins	50
	Changing a Table Alias	50
6.4	Column Aliases	51
	Basic Column Aliases	51
	The Word AS is Optional	51
	Aliases in WHERE and ORDER BY Clauses	52
6.5	Ordering Rows	53
	Simple Sorting	53
	Sorting by Multiple Columns	53
	Sorting Nulls	54
6.6	Miscellaneous SELECT Tricks	55
	Selecting All Columns Using the Asterisk	55
	Selecting Unique Rows	55
	Showing Top and Bottom Rows	56
	Including Ties	56
6.7	Using UNION to Combine Results	57

7	CRITERIA USING WHERE	Page
7.1	The WHERE Clause	58
	Relational Operators	58
7.2	Criteria with Numbers	59
	Using Comparisons	59
	Finding Numbers in a Given Range	59
7.3	Criteria using Text	60
	Exact Matches	60
	Wildcard Matches using LIKE and ILIKE	60
	Using Special Characters with LIKE	61
	Ranges and Regular Expressions	61
	Using Similar To	62
7.4	Criteria for Dates	63
	Using Dates in Criteria	63
7.5	Combining Criteria	64
	Example using AND	64
	Example using OR	64
	Criteria Explained using Venn Diagrams	65
	Using Parentheses	65
7.6	Nulls	66
	An Example of Testing for Nulls	66

8	IMPORTING AND EXPORTING DATA	Page
8.1	Copying and Pasting	67
8.2	Exporting Rows for Tables	68
	Starting the Export Process	68
	Choosing to Include Columns	69
	Choose Columns to Include	69
	Finishing the Export	70
	Viewing Errors	70
8.3	Exporting Queries	71
	Exporting a Query Via a Table	71
	Exporting a Query by Pasting its SQL	71
8.4	Loading Data into Excel or Power BI	73
	Installing the Driver	73
	Add an ODBC Data Source	73
	Loading Data into Excel	74

© Wise Owl Business Solutions Ltd

TABLE OF CONTENTS (3 of 5)

9	CALCULATIONS	Page
9.1	Creating Calculated Columns	75
	Giving Calculated Columns Aliases	75
	Using Column Aliases in ORDER BY Clauses	75
	Column Aliases Don't Work in WHERE Criteria	76
9.2	Using SQL Functions	77
	Typing an SQL Function	77
	Getting Help on PostgreSQL Functions	77
9.3	Casting Data Types	78
	The Need for Casting	78
	Casting Using the CAST Function	78
	Casting using the Cast Operator	79
9.4	Numerical Calculations	80
	Mathematical Symbols and BODMAS	80
	The Modulus Operator (%)	80
	Powers and Roots	81
	Rounding	81
	Other Useful Mathematical Functions	81
	The Importance of Casting Numbers for Calculations	82
	A Short-Cut to Forcing the Right Number Type	82
9.5	Text Calculations	83
	Concatenating Text	83
	Changing the Case of Text	83
	Getting Numerical Information	84
	Extracting Text	84
	Manipulating Text	85
9.6	Examples of Text Functions	86
	Example: Counting Words	86
	Example: Getting the Last Word (1)	86
	Example: Getting the Last Word (2)	87
9.7	Dealing with Nulls – the COALESCE Function	88

10	THE CASE EXPRESSION	Page
10.1	The Searched Case Expression	89
	Example: Film Bands	89
	Example: Film Era	90
	Using CASE Expressions in WHERE Criteria	91
10.2	The Simple Case Statement	92
10.3	Nested CASE Statements	93

11	DATE CALCULATIONS	Page
11.1	How Dates and Times Work	94
	How PostgreSQL Stores Dates	94
	How PostgreSQL Stores Times	94
11.2	Basic Dates and Times	94
	Date and Time Types	95
	Getting the Current Date/Time	95
11.3	Representing Dates (and Times)	96
	The Main Components of a Date	96
	The FM Modifier	97
	Ordinal Suffices	97
	Putting Everything Together	97
	Time Format Codes	98
11.4	Extracting Parts of Dates	99
	The Extract Function	99
	Keywords for Extract	100
	The Date_Part Function	100
	Truncating Dates/Times	101
11.5	Getting the Interval Between Dates	102
	Simple Date Subtraction	102
	Introducing Intervals	102
	Date/Time Subtraction Returning Intervals	102
	Getting True Ages	103
	Using Extract with Intervals	104
11.6	Adding and Subtracting Dates	105
	Adding and Subtracting Days	105
	Adding and Subtracting Other Time Intervals	105
	Using Date_Add and Date_Subtract	106
	Generating Intervals from Numbers	107

TABLE OF CONTENTS (4 of 5)

12	JOINS	Page
12.1	Overview of Joins	108
	What is a Join?	108
	The Types of Join	108
12.2	Understanding your Database	109
	How Relationships Work (Reminder)	109
12.3	Inner Joins	110
	The Syntax of an Inner Join	110
	Our Example – Joining the Film and Director Tables	110
	Joining more than One Table	111
	Variations on Inner Join Syntax	112
	Composite Joins	113
	Joining by Expressions	113
12.4	Outer Joins	115
	Creating Outer Joins in SQL	115
	Right Outer Joins	115
	Picking Out Unmatched Rows	116
	Full Outer Joins	116
12.5	Cross Joins	117
	A Practical Example of Cross Joins	117
12.6	Self-Joins	118

13	SUMMARISING DATA	Page
13.1	Simple Summarising	119
	Syntax of a Simple Summary	119
13.2	Counting	120
	Counting All of a Table's Rows	120
	Counting Non-Null Columns	120
	Counting Unique Values	121
	Using Filter to Count a Subset of Rows	121
13.3	Grouping	122
	Why you Need GROUP BY	122
	The GROUP BY Clause	122
	Grouping by Multiple Columns	123
	Grouping by Expressions	123
	Grouping without Aggregating	124
13.4	Filtering Results using HAVING	125
13.5	Dealing with Nulls	126
	The Default Treatment of Nulls	126
	Forcing SQL to Include Nulls	126
13.6	The CUBE and ROLLUP Operators	127
	Using CUBE to Show All Combinations	127
	Using ROLLUP to Show Most Combinations	128
13.7	String Aggregation	129

14	VIEWS	Page
14.1	What Views Are	130
14.2	Why Views are Useful	131
	Use 1: Pre-Joining Tables	131
	Use 2: Virtually Renaming Columns	132
14.3	Creating Views	133
14.4	Changing Views	134
	Scripting Changes	134
	Why Changes Rarely Work	134
	Solution 1 – Delete and Recreate the View	135
	Solution 2 – Amend the Existing View	135
14.5	Sorting in Views	136
14.6	Joining to Views	137

15	CTES AND DERIVED TABLES	Page
15.1	Multi-Stage Queries	138
15.2	Derived Tables	139
	Single Derived Tables	139
	Multiple Derived Tables	140
15.3	Single CTEs (Common Table Expressions)	141
	Syntax of Single CTEs	141
	The CTE for our Example	141
15.4	Multiple CTEs	142
	Syntax of Multiple CTEs	142
	Example of a Multiple CTE	143

16	SUBQUERIES	Page
16.1	Single-Value Subqueries	144
	Example: Showing the Name of the Longest Film	144
16.2	ANY, ALL, IN and NOT IN	145
16.3	Correlated Subqueries	146
	Definition and Example	146
	Alternatives to Correlated Subqueries	146
	Using EXISTS to Check whether Rows are Returned	147

© Wise Owl Business Solutions Ltd

TABLE OF CONTENTS (5 of 5)

17	POSTGRES AND AI	Page
17.1	Choosing an AI Tool	148
17.2	Blind Queries	149
17.3	Querying a Specific Database	150
	Creating a Backup of a Schema	150
	Uploading your Schema File	151
	Requesting a Query	151
17.4	Tidying Up Queries	152
17.5	Optimising Queries	153
17.6	Debugging SQL	154

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!

*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.

© Copyright 2026

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.

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.

© Copyright 2026

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).

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 vary
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

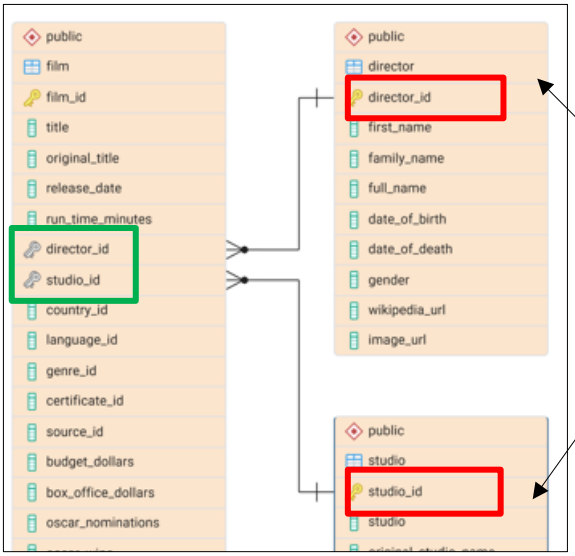
*If you're beginning to think that relational databases are just like a lot of spreadsheets joined together with a more efficient version of a **VLOOKUP** or **XLOOKUP** formula in Excel, you're absolutely right!*

© Copyright 2026

Stage 4 – Creating Relationships and a Database Diagram

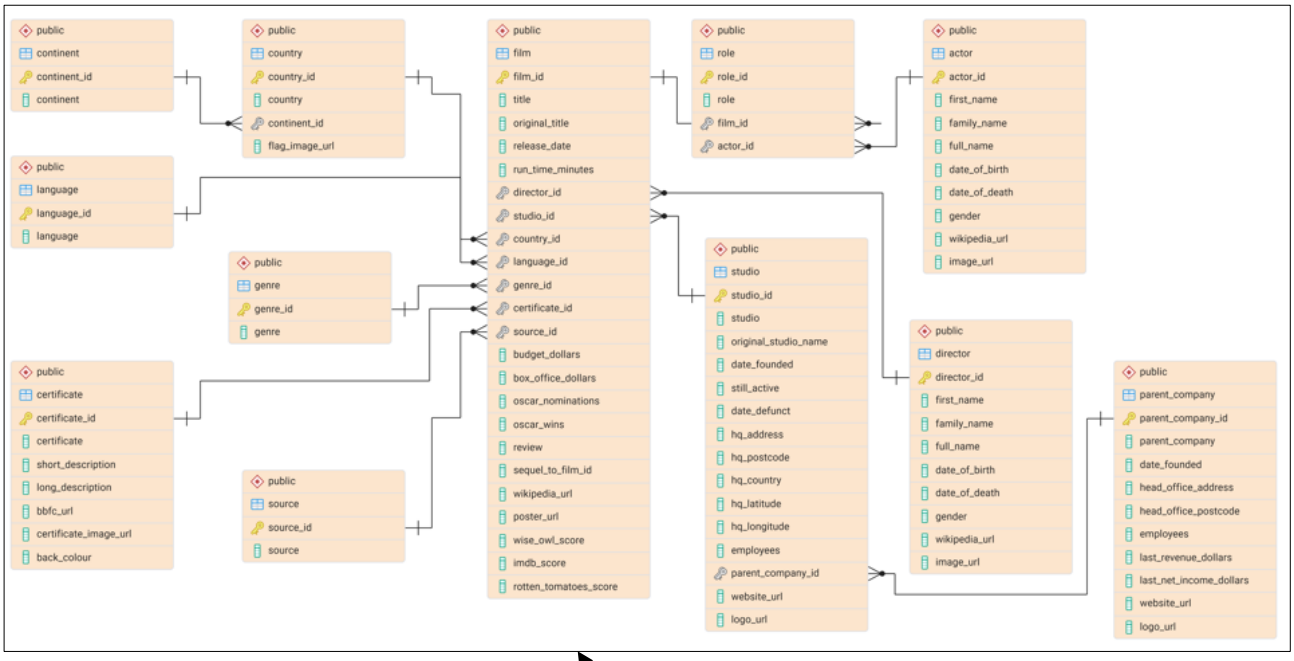
The last step in designing a database is to decide for each relationship that you create whether it is *one-to-many* or *many-to-one* (*parent-child* or *child-parent*):

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:



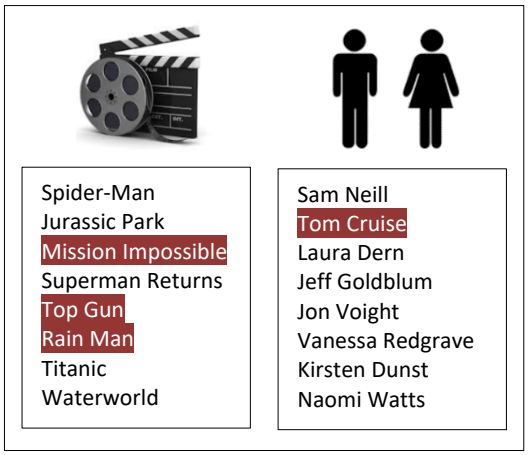
Our **Movies** database contains just 12 tables, all of which are shown above, and hence is untypically simple.

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

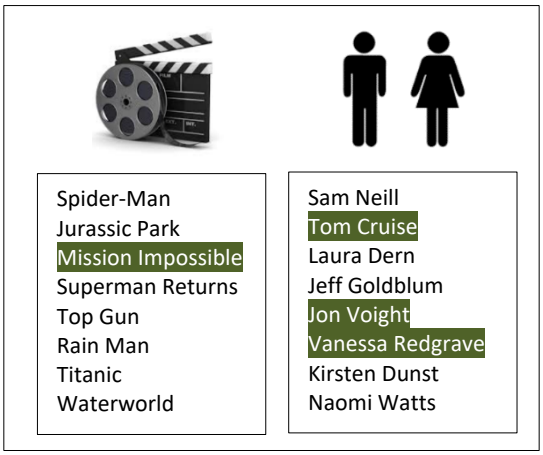
© Copyright 2026

1.2 Many-to-Many Relationships

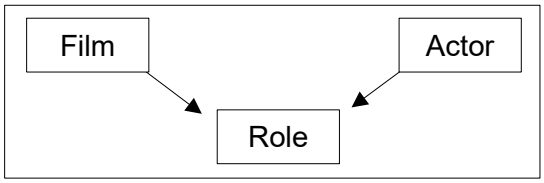
There's no such thing as a many-to-many relationship in *PostgreSQL*, but they do exist in real life:



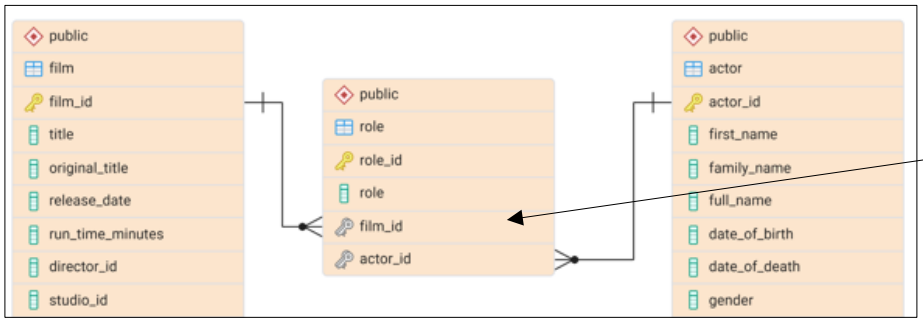
***Tom Cruise** has appeared in lots of films, but equally **Mission: Impossible** has lots of actors in it.*



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:

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

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

© Copyright 2026

CHAPTER 2 - GETTING STARTED WITH POSTGRESQL

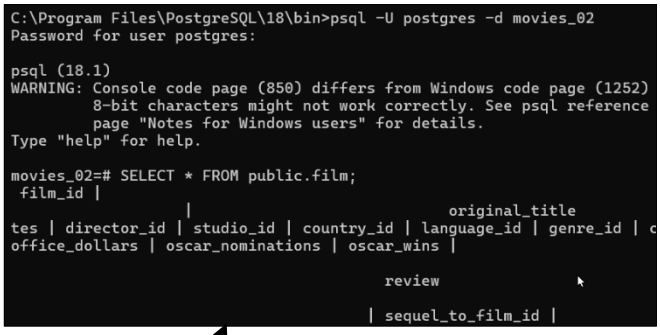
2.1 Getting Started with PostgreSQL

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/ :

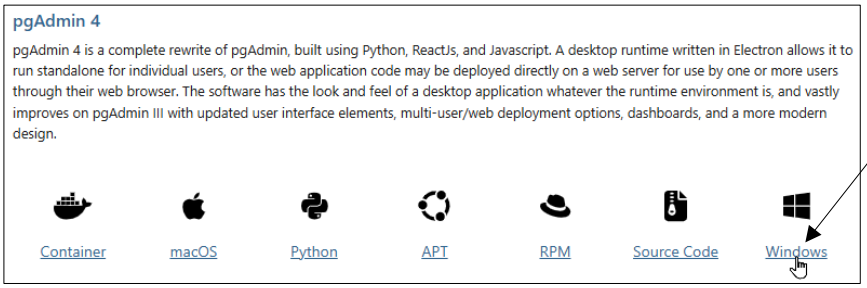
Installing *PostgreSQL* from the official website is simple – and free!



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

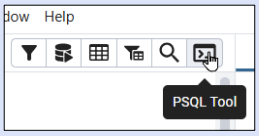
There are a few graphical interfaces available for using Postgres, but by far the most common (and hence the best choice) is *PG Admin*:



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:*



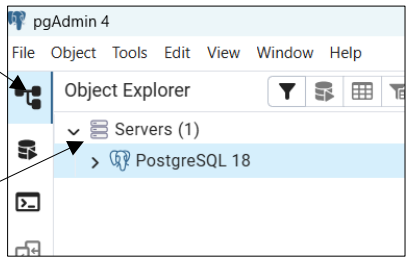
© Copyright 2026

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.

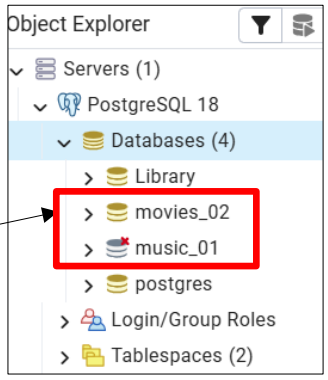


Listing Databases

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.

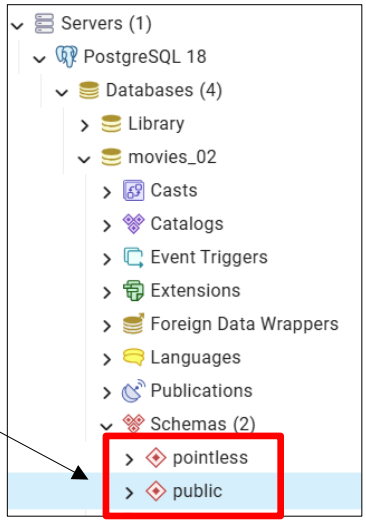


Listing Schemas

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).

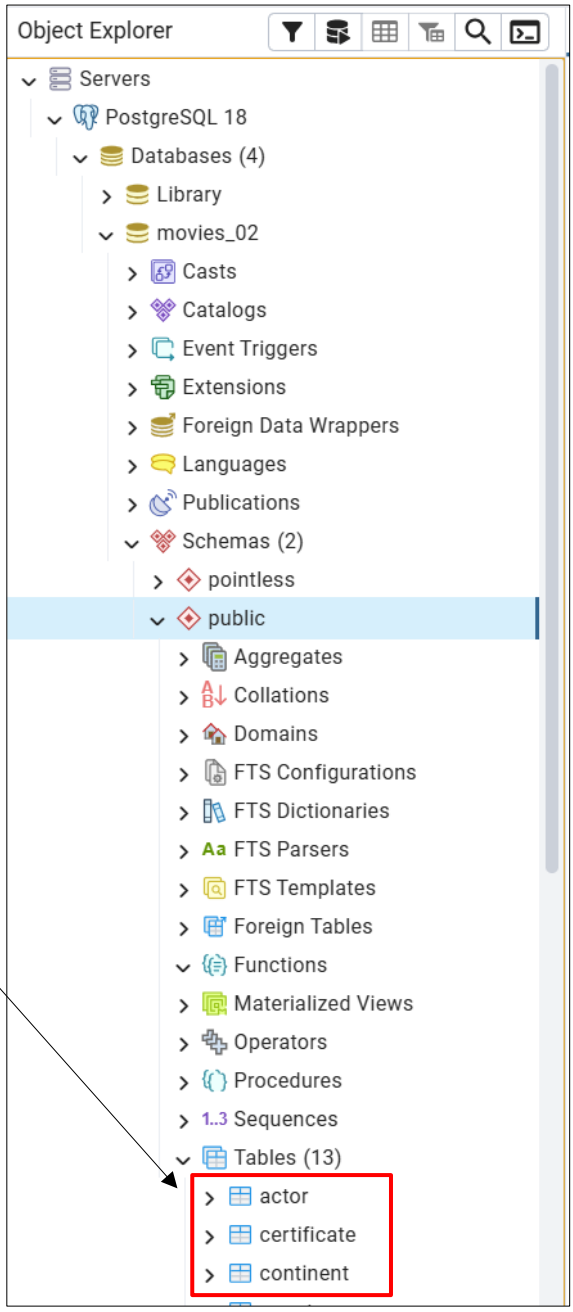


When you create a new table it will (by default) join the public schema. Usually the best thing to do it to accept this default, as it means that you can find all of your tables in the same place.

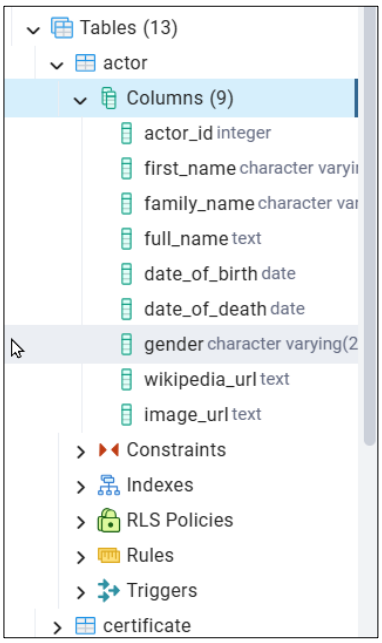
© Copyright 2026

Listing Tables

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



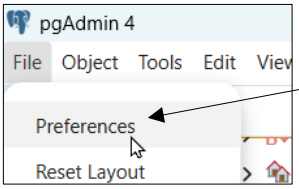
The first few tables in this database. You can expand any table further to see the columns that it contains. Here for example are the columns in the **actor** table:



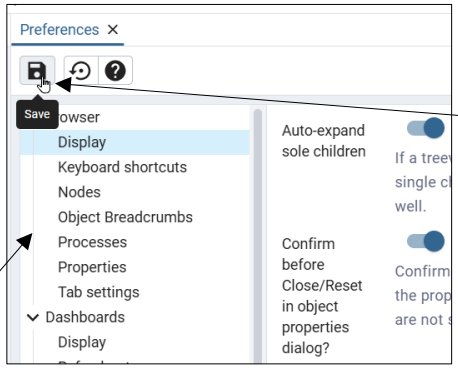
© Copyright 2026

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.

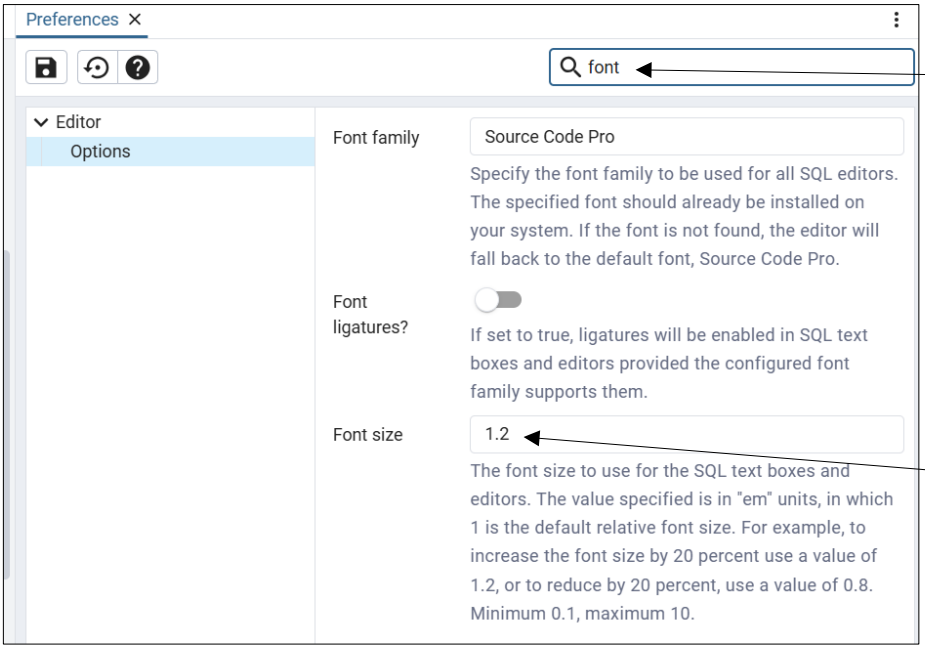


b) Choose the preference category on the left that you want to work with, then change one or more properties within it.

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

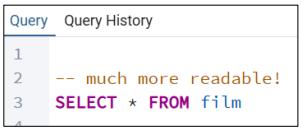
Change 1: Setting the Default Font Size

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



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

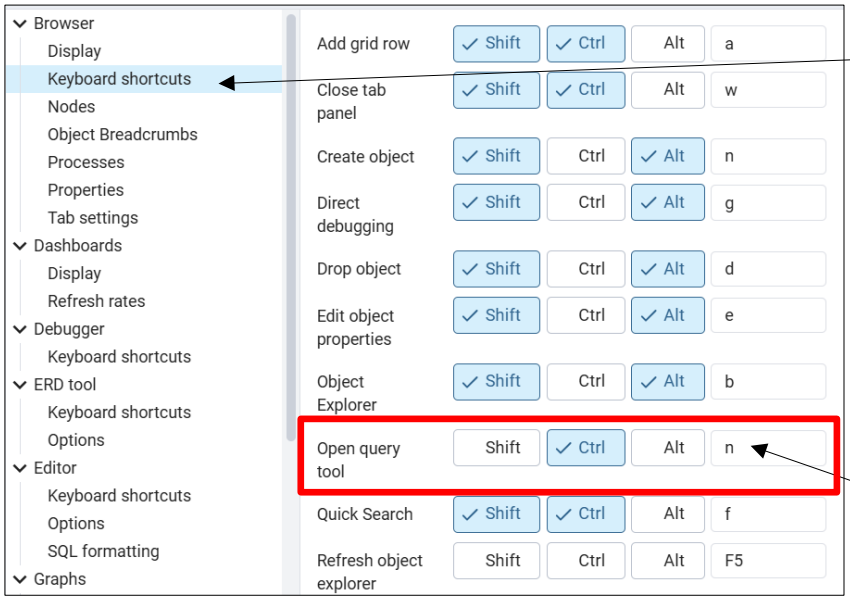
b) Change the font size to a multiple of 1 (here we've gone for 1.2, or 120% of the default size) to get much more readable SQL!



© Copyright 2026

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 :



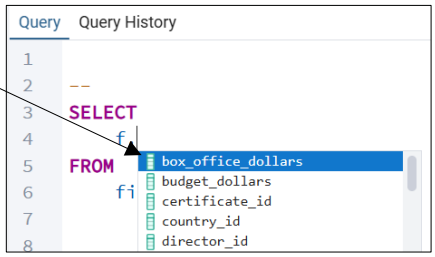
a) Choose the **Keyboard shortcuts** category within the **Browser** tab.

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

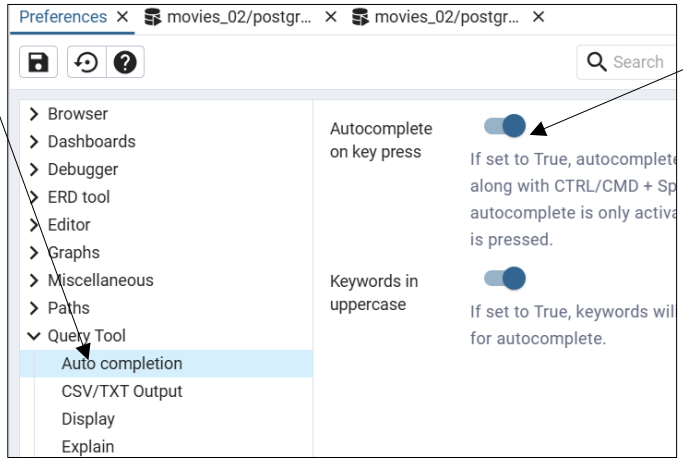
For some strange reason the automatic completion of commands shown on the right doesn't appear by default in PG Admin.

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



Here's how to change this:

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



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

© Copyright 2026

Our training and consultancy services

		Basic training	Advanced training	Systems / consultancy
Desktop	Microsoft Excel			
	VBA macros			
	Microsoft Access			
Business Intelligence	Power BI			
	Power Apps			
	Power Automate			
SQL Server	SQL			
	Reporting Services			
	Report Builder			
	Integration Services			
	Analysis Services			
Coding	Visual C# programming			
	VB programming			
	DAX			
	Python			