Simplify SQL for Absolute Beginners

Learn to Query Databases Step by Step · Preview Edition: Chapters 1–3

SIMPLIFY DATABASE

Contents

Get the Files

Chapter 1 What a Database Is (and Why Not Excel)

Chapter 2 Set Up Your Practice Database

Chapter 3 SELECT, WHERE, ORDER BY and LIMIT

Appendix C Answers to the Exercises

Get the Files

Every example in this book runs on **ShopDB**, a small practice database for an online electronics shop. Download it free from:

simplifydatabase.com/shopdb-sample-database

The download is a zip with three files:

File	Use it with
`shopdb_mysql.sql`	MySQL 8 or MariaDB
`shopdb_sqlserver.sql`	SQL Server 2016 or later
`README.md`	A short guide to the tables

Chapter 2 shows you how to install a free database server and load ShopDB. Questions or mistakes? Write to simplifydatabase@gmail.com.

How to use this book

- **Type the queries yourself.** Copying is faster, but typing is how SQL sticks.
- **Compare your result with the book.** Every result printed in this book is the real output of the query on ShopDB, so yours should match row for row.
- **Do the practice questions** at the end of each chapter before you look at Appendix C.
- **Break things.** Run the ShopDB script again any time to get the original data back.

1

What a Database Is (and Why Not Excel)

IN THIS CHAPTER

- What a database is, in plain words
- Why a growing business moves from a notebook to a spreadsheet to a database
- How tables, rows, columns and keys fit together
- The difference between a database and a DBMS
- When Excel is the right tool, and when SQL is

Every time you check your bank balance, order something online or look up a contact on your phone, a database answers you. You never see it, but it's there, quietly storing information and handing back exactly the piece you asked for.

This chapter explains what a database is in plain language, using one small shop as the example from start to finish: ShopDB, the practice database you'll use for the rest of the book. You don't need any technical background. By the end, you'll know the words people use (tables, rows, keys, SQL), and you'll be ready to set up your own copy in Chapter 2.

> **In one sentence:** A database is an organised collection of information, stored on a computer so it can be searched, updated and shared quickly and safely. Most databases keep their data in tables made of rows and columns, and you talk to them with a language called SQL.

A database in everyday life

The contacts list on your phone is a tiny database. Each contact has the same kinds of information (a name, a number, maybe an email), and you can find "Ahmed" in a second even if you have 500 contacts. You don't scroll through all of them. You search, and the phone finds him for you.

A real database does the same thing on a much bigger scale: millions of records, many people using it at the same time, and rules that stop the information from becoming a mess.

From a notebook to a database: a small shop's story

ShopDB belongs to a small online electronics shop in Manama. It sells laptops, phones, monitors and the cables and mice that go with them. Here's how its records grew up.

Stage 1: the notebook

At first, the owner wrote every order in a notebook: the date, the customer, what they bought and the price. It worked for a few

weeks. Then she wanted to know how many USB-C cables she sold last month, and she spent an evening counting by hand.

Stage 2: the spreadsheet

So she moved to Excel. Now she could sort and add up totals in seconds. Much better. But as the shop grew, new problems appeared:

- The same customer was typed three different ways ("Sara Ahmed", "S. Ahmed", "sara ahmed"), so reports counted her as three people.
- Her assistant opened an older copy of the file, added orders to it, and now there were two versions of the truth.
- Someone typed "two" in the quantity column, and the monthly total broke.
- The file got big and slow, and one day it wouldn't open.

Stage 3: the database

Finally, the shop moved its records into a database. Each customer is stored once. The quantity column only accepts numbers. Two people can record orders at the same time without overwriting each other. And "how many USB-C cables did we sell last month?" is answered in a moment, even after years of orders.

That's really what a database is for: keeping information correct, shared and fast to find as it grows.

How a database organises data

Most databases you'll meet are **relational databases**. They store data in **tables**, which look a lot like spreadsheets:

- Each **column** is one kind of information, like a name or a price. Every value in a column has the same type: all numbers, all dates, or all text.
- Each **row** is one record: one customer, one product, one order.

Here are the first rows of ShopDB's Customers table:

CUSTOMERS 15 rows · first 3 shown

	CustomerID	FirstName	LastName	Phone	City
1	1	Ali	Hassan	+973 3300 1001	Manama
2	2	Sara	Ahmed	+966 50 200 2002	Riyadh
3	3	Omar	Khalid	NULL	Dubai

And here are the first rows of the Orders table:

ORDERS 48 rows · first 6 shown

	OrderID	CustomerID	OrderDate
1	1001	2	2025-01-08
2	1002	2	2025-01-17
3	1003	2	2025-01-31
4	1004	5	2025-03-25
5	1005	6	2025-03-29
6	1006	1	2025-04-07

Notice that the Orders table doesn't repeat Sara's name and phone. It just stores her number, `CustomerID = 2`. That number links each

order to the right customer. If Sara changes her phone number, you update one row in Customers, and every order still points to the correct, up-to-date person.

(Omar's phone says NULL. That's how a database writes "no value here". You'll meet NULL many times in this book.)

This idea of separate tables connected by numbers is where the word *relational* comes from. The tables are related to each other.

The words you'll hear most

Database people use a handful of words all the time. Here they are in plain English:

Word	What it means	In ShopDB
Table	A set of records of the same kind	Customers, Orders, Products
Row (record)	One item in a table	Ali's customer details
Column (field)	One kind of information	Phone
Primary key	A column that identifies each row uniquely	CustomerID in Customers
Foreign key	A column that points to a row in another table	CustomerID in Orders
Query	A question you ask the database	"Show Ali's orders"
SQL	The language you write queries in	`SELECT ... FROM Orders`
DBMS	The software that runs the database	SQL Server, MySQL, PostgreSQL

The two kinds of key get a whole chapter of their own: Chapter 10.

Database vs DBMS: a common mix-up

People often say "MySQL is a database". Strictly speaking, MySQL is a **database management system** (DBMS): the program that stores the data, checks the rules and answers your questions. The database is the data itself: the shop's customers, products and orders.

Think of a music player and your playlist. The player is the software (the DBMS). The playlist is your data (the database). One DBMS can look after many databases at the same time. In everyday conversation people use the two words loosely, and that's fine, but knowing the difference makes documentation much easier to read.

The most common DBMSs are:

- **Microsoft SQL Server**: very common in companies, especially with Windows and Microsoft tools.
- **MySQL** (and its close cousin **MariaDB**): popular for websites.
- **PostgreSQL**: free, powerful and growing fast.
- **SQLite**: a tiny database inside a single file, used in phones and small apps.

This book shows every example in MySQL and SQL Server. Chapter 2 helps you install one of them.

How do you talk to a database? With SQL

You don't open a database and scroll through it like a spreadsheet. You ask it questions in a language called **SQL** (Structured Query Language). Here's the shop owner asking for Ali's orders:

SQL

```sql
SELECT OrderID, OrderDate, Status
FROM Orders
WHERE CustomerID = 1;
```

RESULT 4 rows

	OrderID	OrderDate	Status
1	1006	2025-04-07	Delivered
2	1009	2025-04-21	Delivered
3	1029	2026-01-13	Delivered
4	1040	2026-04-23	Delivered

Read it out loud: "select the order ID, date and status from Orders where the customer is number 1." That's SQL. It reads almost like English, which is why it's one of the easiest technical skills to start with. The same query works in SQL Server, MySQL and PostgreSQL. You'll write it yourself in Chapter 3.

SQL and Excel: different jobs

If you work with data, you almost certainly use Excel. It's on every office computer, everyone understands it, and you can get an answer in a few clicks. So it's fair to ask: what can SQL do that Excel can't?

I work with finance and operations teams who live in Excel, and I write SQL every day, so I see both sides. Here's the main difference in one sentence: **in Excel, you do the work** (click, filter, drag formulas, copy and paste); **in SQL, you describe the result you want, and the database does the work for you**, on ten rows or ten million, as many times as you like.

	Excel	Database + SQL
Best for	Small data, hands-on analysis, charts	Growing or shared data, repeatable reports
Size	Up to about 1 million rows per sheet, slow well before that	Millions of rows are normal
Linking lists	VLOOKUP / XLOOKUP, which break quietly	JOIN on keys (Chapter 11)
Rules	Anyone can type anything in any cell	Columns accept only the right type; rules stop bad data
Many users	Copies and conflicts	Built for it
Repeating a report	Repeat the clicks	Run the same query again

Excel is the right tool for a household budget, a list of class grades or a one-off price comparison. Anything a business depends on every day, with several people entering data, belongs in a database.

The same task, both ways

Say you want ShopDB's number of customers per country, biggest first.

In Excel: open the file and wait for it to load, insert a PivotTable, drag Country to Rows and CustomerID to Values, make sure it says Count, then sort the column from largest to smallest. It works. But next month, with new customers, you do it all again, and you need to be sure everyone has the latest file.

In SQL:

```sql
SELECT Country, COUNT(*) AS Customers
FROM Customers
GROUP BY Country
ORDER BY Customers DESC, Country;
```

RESULT 11 rows · first 3 shown

	Country	Customers
1	Bahrain	3
2	Saudi Arabia	2
3	UK	2

Four lines. Next month you run exactly the same query, and the new customers are included automatically. Chapter 8 explains every word of it.

From Excel to SQL

If you already know Excel, you know more SQL ideas than you think:

In Excel you...	In SQL you write...	Chapter
Pick some columns to show	`SELECT FirstName, City FROM Customers`	3
Use a filter	`WHERE Country = 'Bahrain'`	3
Sort a column	`ORDER BY UnitPrice DESC`	3
Remove duplicates	`SELECT DISTINCT Country`	4
Use an IF formula	`CASE WHEN ... THEN ... END`	7
Build a pivot table	`GROUP BY` with SUM, COUNT or AVG	8
Use VLOOKUP / XLOOKUP	`JOIN`	11

In most companies, the two work as a team: SQL pulls exactly the rows you need from millions and summarises them, and Excel presents the small result as charts and shared files. Excel can even connect straight to a database (Data > Get Data > From Database) and refresh the result with one click.

Other kinds of databases

Relational databases are the most common, but not the only kind:

- **Document databases** such as MongoDB store each record as a flexible document, a bit like a form that can have different fields each time.
- **Key-value stores** such as Redis are like a giant, very fast dictionary: give it a key, get back a value. They're often used to make websites faster.
- **Cloud databases** are usually the same kinds of databases, run for you by a provider like Microsoft Azure, Amazon or Google, so you don't manage the server yourself.

As a beginner, start with relational databases and SQL. They're used almost everywhere, and the ideas carry over to everything else.

Key points

- A database keeps information correct, shared and fast to find as it grows.
- Relational databases store data in tables: columns are kinds of information, rows are records.
- Tables are linked by keys. Orders stores a CustomerID instead of copying the customer's name and phone.
- The DBMS (SQL Server, MySQL, PostgreSQL) is the software; the database is your data.
- You ask a database questions in SQL. You describe the result; the database does the work.
- Excel is great for small, hands-on work. Shared, growing data and repeatable reports belong in a database.

Practice

Answer these from memory before you look at the answers.

1.1 In ShopDB, what is one row in the Orders table?

1.2 Why does the Orders table store CustomerID instead of the customer's name and phone?

1.3 Is MySQL a database or a DBMS?

1.4 Name two problems the shop had with its spreadsheet that a database solves.

1.5 Excel, SQL or both? (a) Planning a family holiday budget. (b) A monthly sales report from 2 million orders. (c) A chart of this year's sales by region for a presentation. (d) An online shop's website saving every new order.

Answers are in Appendix C.

Enough reading about databases. In the next chapter, you'll install a free database server and load ShopDB, so you can run every query in this book yourself.

2

Set Up Your Practice Database

IN THIS CHAPTER

- Choose between MySQL and SQL Server (both are free)
- Install the database server and a tool to write queries in
- Load ShopDB, the practice database used in every chapter
- Check that everything worked, and fix the usual setup problems
- Take a tour of ShopDB's six tables

You learn SQL by typing it. Reading a query is like reading about swimming: it helps, but you only really get it in the water. So before we write any more SQL, let's give you your own database to practise on.

This chapter takes about 30 minutes. You'll install a free database server, load ShopDB with one script, and run your first query. If anything goes wrong, the section near the end covers the problems almost everyone meets.

Step 1: choose MySQL or SQL Server

Every example in this book works on both. The basics are almost the same everywhere, so this choice matters less than you'd think. Pick the one that fits you:

	MySQL	SQL Server
Made by	Oracle (open source)	Microsoft
Cost for learning	Free (Community Edition)	Free (Developer or Express edition)
Runs on	Windows, Mac, Linux	Windows, Linux (Mac through Docker)
Where you'll meet it	Websites, WordPress, PHP and Laravel apps	Company systems: accounting, ERP, payroll, stock
Tool to write queries	MySQL Workbench	SQL Server Management Studio (SSMS)
"Top 3 rows"	`... LIMIT 3`	`SELECT TOP (3) ...`

My advice: if your workplace uses Microsoft tools, or you have a Windows PC and want a job in business software, choose SQL Server. If you're on a Mac, build websites, or aren't sure, choose MySQL. I use SQL Server every day at work and MySQL for web projects, and moving between them takes an afternoon.

Step 2a: install MySQL

Install **MySQL 8.4 LTS**. LTS means "long-term support": it gets bug and security fixes for years. You'll also see 9.x "Innovation"

releases, which add features quickly but are supported only for a short time. They're not needed for this book.

On Windows

1. Download **MySQL Community Server 8.4** from the official MySQL website (dev.mysql.com).
2. Run the installer and keep the default settings. When it asks, choose a password for the **root** user and write it down.
3. Download and install **MySQL Workbench** from the same site.
4. Open Workbench and click the **Local instance** connection. Enter your root password.

On a Mac

Use the installer from the MySQL website (pick the file for your chip, Apple or Intel), or run `brew install mysql` if you use Homebrew. Then install MySQL Workbench.

On Ubuntu Linux

Run `sudo apt install mysql-server`, then `sudo mysql_secure_installation` to set a password and remove the test settings.

> **Already have XAMPP?** Its control panel says "MySQL", but it actually installs MariaDB, MySQL's close cousin. That's fine: every query in this book works on MariaDB too.

Step 2b: install SQL Server

For learning, install the free **Developer** edition. It has every feature, but you may only use it for learning, development and testing. The free **Express** edition is the one you can also use for small live applications; it works for this book too.

On Windows

1. Download SQL Server **Developer** from Microsoft's SQL Server downloads page.
2. Run the installer and choose **Basic**. It takes about ten minutes.
3. Download and install **SQL Server Management Studio (SSMS)**. It's a separate free download.
4. Open SSMS. In **Server name**, type `localhost` (or `.\SQLEXPRESS` if you installed Express). Choose **Windows Authentication** and click **Connect**.

On a Mac or Linux

SQL Server runs natively on Linux; Microsoft's Linux quickstarts walk you through the install. There's no Mac version, so on a Mac you run the Linux version in a Docker container (Microsoft's "Run SQL Server Linux container images with Docker" quickstart has the exact command) and connect with **VS Code** and its **MSSQL** extension.

> **Watch out:** Older tutorials use Azure Data Studio. Microsoft retired it in 2026 and recommends VS Code with the MSSQL extension instead.

Step 3: get the ShopDB scripts

Download the book's files from the address on the "Get the files" page at the front of this book. The zip contains:

File	Use it with
`shopdb_mysql.sql`	MySQL 8 or MariaDB
`shopdb_sqlserver.sql`	SQL Server 2016 or later
`README.md`	A short guide to the tables

Each script creates a database called ShopDB, builds its six tables and fills them with data. If a ShopDB database already exists, the script **drops it first** and starts again. That's useful: whenever you break something while practising, just run the script again to get back to the original data.

Step 4: run the script

In MySQL Workbench: choose **File > Open SQL Script**, pick `shopdb_mysql.sql`, then click the lightning-bolt button (**Execute**) above the editor. It runs the whole script.

In SSMS: choose **File > Open > File**, pick `shopdb_sqlserver.sql`, then click **Execute** (or press **F5**).

The script finishes in a second or two. Its last query counts the rows it loaded:

```sql
SELECT 'Categories' AS TableName, COUNT(*) AS RowsLoaded
FROM Categories
UNION ALL SELECT 'Products',   COUNT(*) FROM Products
UNION ALL SELECT 'Customers',  COUNT(*) FROM Customers
UNION ALL SELECT 'Employees',  COUNT(*) FROM Employees
UNION ALL SELECT 'Orders',     COUNT(*) FROM Orders
UNION ALL SELECT 'OrderItems', COUNT(*) FROM OrderItems;
```

RESULT 6 rows

	TableName	RowsLoaded
1	Categories	7
2	Products	21
3	Customers	15
4	Employees	8
5	Orders	48
6	OrderItems	95

If your numbers match, you're ready. Don't worry about understanding this query yet; you'll learn every part of it by Chapter 13.

Step 5: run your first query

Open a new query tab (**File > New Query Tab** in Workbench, **New Query** in SSMS). Make sure ShopDB is the database you're working in:

MYSQL VS SQL SERVER

MYSQL

```
USE ShopDB;
```

SQL SERVER

```
USE ShopDB;
GO
```

`USE` chooses the database for the queries that follow. (In SSMS you can also pick ShopDB from the drop-down list in the toolbar. `GO` tells SSMS to send everything above it before carrying on.)

Now type this and run it:

SQL

```
SELECT *
FROM Categories;
```

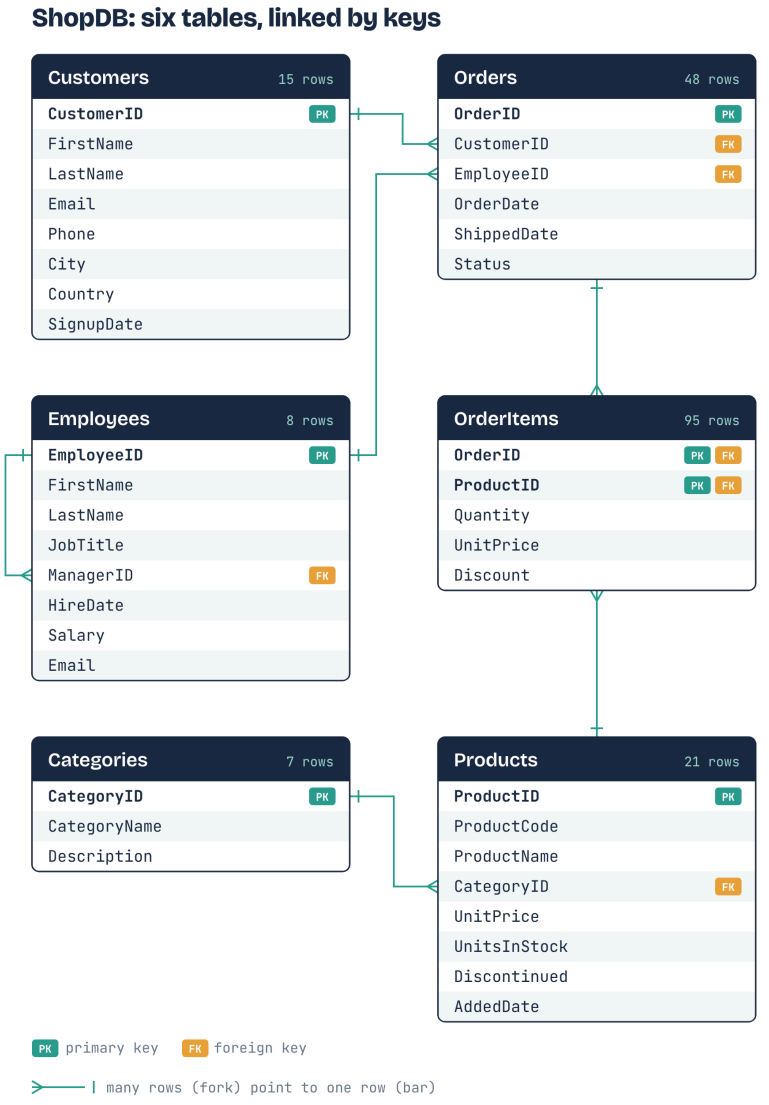
RESULT 7 rows

	CategoryID	CategoryName	Description
1	1	Laptops	Notebook computers for work and study
2	2	Phones	Smartphones
3	3	Accessories	Cables, mice, keyboards and other small items
4	4	Monitors	Computer screens
5	5	Printers	Printers and ink
6	6	Software	Software licences
7	7	Gaming	NULL

That's it: you've asked a database a question and got an answer. `SELECT *` means "all columns", and `FROM Categories` names the table. Gaming has no description yet, so the database shows NULL.

A tour of ShopDB

ShopDB is a small online electronics shop. Here are its six tables and how they connect:



ShopDB's six tables. Each line joins a foreign key (FK) to the primary key (PK) it points to.

Table	One row is…	Linked to
Categories	a product group, such as Laptops	Products
Products	something the shop sells, with its price and stock	Categories, OrderItems
Customers	a person who signed up	Orders
Employees	a member of staff	Orders, and their own manager
Orders	one order: who placed it, when, and its status	Customers, Employees, OrderItems
OrderItems	one product inside an order, with quantity and price	Orders, Products

A few things worth knowing before you start:

- **One order, many items.** An order's products live in OrderItems, one row per product. Order 1001, for example, has one row: product 5, quantity 1, price 899.00 with a 5% discount.
- **Employees have managers.** Each employee's ManagerID points to another row in the same table. Hana, the CEO, has no manager, so her ManagerID is NULL.
- **Some orders have no salesperson.** Online orders have NULL in EmployeeID.
- **The data has a few surprises on purpose:** a customer with no orders, a product nobody has bought, products with the same price, and some empty values. Each one is there to teach something, and you'll meet them in later chapters.

Exploring a database

These commands help you look around any database:

What you want	MySQL	SQL Server
List the databases	`SHOW DATABASES;`	`SELECT name FROM sys.databases;`
Choose one	`USE ShopDB;`	`USE ShopDB;`
List its tables	`SHOW TABLES;`	`SELECT name FROM sys.tables;`
See a table's columns	`DESCRIBE Products;`	`EXEC sp_help 'Products';`

You can also browse: the **Navigator** panel in Workbench and **Object Explorer** in SSMS show every database, table and column. Click the arrow next to a table to open its list of columns.

Common setup problems

"Can't connect" or "server not found." The database server isn't running. On Windows, open **Services**, find **MySQL84** or **SQL Server (MSSQLSERVER)** (or **SQLEXPRESS**) and click **Start**.

SSMS can't find localhost. If you installed Express, the server name is `.\SQLEXPRESS`, not `localhost`.

MySQL says "Access denied for user 'root'". The password is wrong. Retype it carefully; it's case-sensitive.

The script stopped halfway with an error. Run the whole script again. It drops ShopDB first, so a second run always starts clean.

Your MySQL table names show in lowercase. On Windows, MySQL stores table names in lowercase by default, so `SHOW TABLES` lists `orderitems`. That's fine: queries still work with `OrderItems`.

Key points

- Choose MySQL (8.4 LTS) or SQL Server (free Developer or Express edition). Every example works on both.
- Write queries in MySQL Workbench or SSMS (or VS Code with the MSSQL extension).
- Run `shopdb_mysql.sql` or `shopdb_sqlserver.sql` to create ShopDB. Run it again any time to reset the data.
- `USE ShopDB;` chooses the database you're working in.
- ShopDB has six tables: Categories, Products, Customers, Employees, Orders and OrderItems, linked by keys.

Practice

2.1 List the tables in ShopDB using the command for your database.

2.2 Show every row and every column of the Employees table. How many employees are there?

2.3 Look at the diagram. Which column links Orders to Customers, and which table is it a primary key in?

2.4 Which table records how many units of each product were in an order?

2.5 Change something on purpose (or not), then put ShopDB back exactly as it was. How?

Answers are in Appendix C.

Your database is ready. In the next chapter, you'll start asking it real questions with SELECT, WHERE, ORDER BY and LIMIT.

3

SELECT, WHERE, ORDER BY and LIMIT

IN THIS CHAPTER

- Pick the columns you want with SELECT
- Keep only the rows you need with WHERE, AND, OR and IS NULL
- Sort the result with ORDER BY
- Take the top rows with LIMIT (MySQL) or TOP (SQL Server), and page through results with OFFSET
- Avoid the three traps that catch everyone: ties, NULLs and unstable paging

When I first learned SQL, `SELECT * FROM Orders` felt like magic. Then I ran it on a real table with 80,000 rows, and my screen filled with a wall of data I couldn't use. That's when I realised the useful part of SQL isn't getting data out. It's getting the *right* data out.

Three clauses do most of that work: WHERE, ORDER BY and LIMIT. Once you're comfortable with them, you can answer most everyday questions about your data in one short query. ShopDB is small, but every query in this chapter works exactly the same on a table with millions of rows.

You write them in this order:

```sql
SELECT ... FROM ... WHERE ... ORDER BY ... LIMIT ...
```

SELECT: choose your columns

What are the names and cities of our customers?

```sql
SELECT FirstName, LastName, City
FROM Customers;
```

RESULT 15 rows · first 5 shown

	FirstName	LastName	City
1	Ali	Hassan	Manama
2	Sara	Ahmed	Riyadh
3	Omar	Khalid	Dubai
4	Lina	Yousef	Amman
5	Fatima	Saleh	Muharraq

SELECT lists the columns you want, separated by commas. FROM names the table. You get every row, but only the columns you asked for.

`SELECT *` returns every column. It's handy for a quick look at a table, but in real queries, name your columns. The result is easier to read, and your query won't change when someone adds a column later.

WHERE: keep only the rows you care about

WHERE is a filter. The database checks every row against your condition and keeps only the ones where it is true.

SQL

```sql
SELECT FirstName, LastName
FROM Customers
WHERE City = 'Manama';
```

RESULT 2 rows

	FirstName	LastName
1	Ali	Hassan
2	Mona	Farouk

Everyone else is dropped before you ever see them. Text and dates go inside single quotes; numbers don't.

These are the operators you'll use all the time:

Operator	Meaning	Example
`=`	equal to	`City = 'Manama'`
`<>`	not equal to	`Status <> 'Cancelled'`
`>` `<`	greater / less than	`UnitPrice > 500`
`>=` `<=`	greater / less than or equal	`OrderDate >= '2026-01-01'`
`IS NULL`	has no value	`Phone IS NULL`

Chapter 5 adds three more: LIKE for text patterns, IN for lists and BETWEEN for ranges.

> **Tip:** Write dates as `'YYYY-MM-DD'`, for example `'2026-06-01'`. MySQL and SQL Server both read this format the same way, whatever the server's language settings.

Combining conditions with AND and OR

Real questions usually have more than one condition. "Show me the accessories that cost more than $40" becomes:

SQL

```sql
SELECT ProductName, UnitPrice
FROM Products
WHERE CategoryID = 3
  AND UnitPrice > 40;
```

RESULT 2 rows

	ProductName	UnitPrice
1	Mechanical Keyboard	89.00
2	Laptop Backpack	45.00

AND means both conditions must be true. OR means at least one must be true.

Here's a mistake I made more than once. Say you want laptops (category 1) or phones (category 2) that cost less than $600:

SQL

```sql
SELECT ProductName, CategoryID, UnitPrice
FROM Products
WHERE CategoryID = 1 OR CategoryID = 2 AND UnitPrice < 600;
```

RESULT 5 rows

	ProductName	CategoryID	UnitPrice
1	ProBook 14 Laptop	1	899.00
2	UltraSlim 13 Laptop	1	1199.00
3	Student 15 Laptop	1	549.00
4	Galaxy Lite Phone	2	299.00
5	Basic 4G Phone	2	79.00

Two of those laptops cost far more than $600. SQL checks AND before OR, just as maths does multiplication before addition, so it read the condition as "any laptop, or a phone under $600". Brackets make your intent clear:

SQL

```sql
SELECT ProductName, CategoryID, UnitPrice
FROM Products
WHERE (CategoryID = 1 OR CategoryID = 2)
  AND UnitPrice < 600;
```

RESULT 3 rows

	ProductName	CategoryID	UnitPrice
1	Student 15 Laptop	1	549.00
2	Galaxy Lite Phone	2	299.00
3	Basic 4G Phone	2	79.00

When you mix AND and OR, just use brackets. It costs nothing and saves you from confusing results.

A quick word on NULL

NULL means “unknown”, and it doesn’t behave like a normal value. `WHERE Phone = NULL` returns nothing, ever. You have to write IS NULL (or IS NOT NULL):

SQL

```
SELECT FirstName, LastName, Phone
FROM Customers
WHERE Phone IS NULL;
```

RESULT 4 rows

	FirstName	LastName	Phone
1	Omar	Khalid	NULL
2	John	Smith	NULL
3	Yusuf	Rahman	NULL
4	David	Lee	NULL

Almost everyone trips on this at least once. Chapter 6 covers NULL in full.

ORDER BY: put the results in a useful order

Without ORDER BY, the database returns rows in whatever order is convenient for it. That order can change, so never rely on it. If the order matters, ask for it.

Which monitors and printers do we sell, most expensive first?

SQL

```sql
SELECT ProductName, UnitPrice
FROM Products
WHERE CategoryID = 4 OR CategoryID = 5
ORDER BY UnitPrice DESC;
```

RESULT 5 rows

	ProductName	UnitPrice
1	27-inch 4K Monitor	379.00
2	Office Laser Printer	289.00
3	24-inch Full HD Monitor	159.00
4	Home Inkjet Printer	99.00
5	Black Ink Cartridge	29.00

DESC means largest first. ASC (smallest first) is the default, so you can leave it out.

You can sort by several columns. SQL sorts by the first one, then uses the next to break ties:

SQL

```
SELECT City, FirstName, LastName
FROM Customers
WHERE Country = 'Bahrain' OR Country = 'UK'
ORDER BY City, LastName;
```

RESULT 5 rows

	City	FirstName	LastName
1	London	John	Smith
2	Manama	Mona	Farouk
3	Manama	Ali	Hassan
4	Manchester	Emma	Brown
5	Muharraq	Fatima	Saleh

Cities are listed alphabetically, and inside Manama the two customers are sorted by last name. This is handy for reports where you want groups that are easy to scan.

LIMIT: only take the first few rows

LIMIT tells the database to stop after a certain number of rows. On its own it isn't very meaningful, but paired with ORDER BY it answers "top N" questions. What are our three most expensive products?

MYSQL VS SQL SERVER

MYSQL

```mysql
SELECT ProductName, UnitPrice
FROM Products
ORDER BY UnitPrice DESC
LIMIT 3;
```

SQL SERVER

```sql
SELECT TOP (3) ProductName, UnitPrice
FROM Products
ORDER BY UnitPrice DESC;
```

RESULT 3 rows

	ProductName	UnitPrice
1	UltraSlim 13 Laptop	1199.00
2	ProBook 14 Laptop	899.00
3	Pixel Pro Phone	899.00

Not every database spells it LIMIT:

Database	"Top 3" syntax
MySQL, PostgreSQL, SQLite	`... ORDER BY UnitPrice DESC LIMIT 3`
SQL Server	`SELECT TOP (3) ... ORDER BY UnitPrice DESC`
Standard SQL (SQL Server 2012+, PostgreSQL, Oracle)	`... ORDER BY UnitPrice DESC OFFSET 0 ROWS FETCH FIRST 3 ROWS ONLY`

If you work with SQL Server a lot, like I do, you'll be typing TOP far more often than LIMIT.

Paging with OFFSET

OFFSET skips rows before LIMIT starts counting. It's how websites show "page 2" of a list. With 5 products per page, page 2 means "skip 5, take 5":

MYSQL VS SQL SERVER

MYSQL

```
SELECT ProductID, ProductName, UnitPrice
FROM Products
ORDER BY UnitPrice DESC, ProductID
LIMIT 5 OFFSET 5;
```

SQL SERVER

```
SELECT ProductID, ProductName, UnitPrice
FROM Products
ORDER BY UnitPrice DESC, ProductID
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY;
```

RESULT 5 rows

	ProductID	ProductName	UnitPrice
1	4	Galaxy Lite Phone	299.00
2	17	Office Laser Printer	289.00
3	14	24-inch Full HD Monitor	159.00
4	16	Home Inkjet Printer	99.00
5	19	Office Suite 1-Year	99.00

Why the extra `ProductID` in ORDER BY? That's the third detail below.

Three details that matter

1. Ties: who gets the last place?

Look at the top 3 again. The ProBook laptop and the Pixel phone both cost 899.00. Now ask for the top **2**: which one do you get? Either. The database is free to choose, and the choice can even change between runs.

If you want both tied rows, SQL Server can include them:

SQL SERVER

```
SELECT TOP (2) WITH TIES ProductName, UnitPrice
FROM Products
ORDER BY UnitPrice DESC;
-- UltraSlim 1199.00, ProBook 899.00, Pixel 899.00
```

MySQL has no WITH TIES. If you want exactly 2 rows, add a tie-breaker so the result is always the same: `ORDER BY UnitPrice DESC,`

`ProductID`.

2. Where do NULLs go?

Customer 14, David, has no country. Sort customers by country and see where he lands:

MYSQL

```
SELECT FirstName, LastName, Country
FROM Customers
ORDER BY Country, LastName
LIMIT 4;
```

RESULT 4 rows

	FirstName	LastName	Country
1	David	Lee	NULL
2	Mona	Farouk	Bahrain
3	Ali	Hassan	Bahrain
4	Fatima	Saleh	Bahrain

MySQL and SQL Server put NULLs **first** when sorting ASC. PostgreSQL and Oracle put them last. With DESC, everything flips. If the position of empty values matters in a report, sort by a CASE expression first (Chapter 7) rather than relying on the default.

3. Paging needs a unique sort order

When you page with OFFSET, the sort order must be unique, or a row can appear on two pages, or on none. Sorting by price alone isn't unique (two products cost 899.00 and two cost 99.00), so always add the primary key at the end, as we did above.

WHERE, ORDER BY and LIMIT in real projects

In a business application, these three clauses sit behind almost every list on the screen: the last ten invoices, today's orders, the top customers this month. Three habits keep those screens fast and safe as the tables grow:

- **"Latest N" screens need an index.** A screen that shows a customer's ten newest orders runs the same query every time it opens. An index on `(CustomerID, OrderDate)` lets the database read just those ten rows instead of every order the customer ever placed. Appendix D points you to indexes.
- **Filter dates with a range, not a function.** `WHERE YEAR(OrderDate) = 2026` makes the database calculate YEAR() for every row. `WHERE OrderDate >= '2026-01-01' AND OrderDate < '2027-01-01'` returns the same rows and can use an index.
- **Pass filter values as parameters.** When the city or date comes from a search box, never paste the user's text into the SQL string. Use parameters (`@City` in SQL Server, `:City` in Delphi's FireDAC, `?` in Laravel). It protects you from SQL injection.

Putting all three together

Here's a query you might actually write at work: "Which were the two most recent delivered orders this year?"

MYSQL

```
SELECT OrderID, OrderDate, Status
FROM Orders
WHERE Status = 'Delivered'
  AND OrderDate >= '2026-01-01'
ORDER BY OrderDate DESC, OrderID DESC
LIMIT 2;
```

RESULT 2 rows

	OrderID	OrderDate	Status
1	1043	2026-05-23	Delivered
2	1042	2026-05-23	Delivered

Three orders were delivered on 23 May 2026. Without `OrderID DESC`, LIMIT 2 could return any two of them. Short, readable, and it would work just as well on millions of rows.

The order SQL actually runs things in

You write the clauses in one order, but the database processes them in another:

1. **FROM** picks the table
2. **WHERE** filters the rows
3. **SELECT** picks the columns

4. **ORDER BY** sorts what's left
5. **LIMIT / TOP** cuts it down to size

Knowing this explains a lot. LIMIT without ORDER BY gives you *some* rows, not the *top* ones, because the cut needs a sort first. And in Chapter 4 you'll see why a column alias works in ORDER BY but not in WHERE.

Common mistakes

> **Watch out:** `WHERE Phone = NULL` returns nothing. Always write IS NULL or IS NOT NULL.

- Using LIMIT or TOP without ORDER BY and expecting the "top" rows. You'll just get some rows.
- Mixing AND and OR without brackets.
- Putting quotes around numbers (`UnitPrice > '50'`). It often works, but it can force slow conversions or odd text comparisons.
- Forgetting that text comparisons may be case-sensitive, depending on the database settings.

Key points

Clause	What it does	Example
WHERE	Keeps only rows that match	`WHERE City = 'Manama'`
AND / OR	Combines conditions (use brackets when mixing)	`WHERE (a OR b) AND c`
ORDER BY	Sorts the result	`ORDER BY UnitPrice DESC`
LIMIT / TOP	Keeps the first N rows	`LIMIT 3` / `SELECT TOP (3)`
OFFSET	Skips rows first, for paging	`LIMIT 5 OFFSET 5`

Practice

Write each query yourself before you look at the answers. If you get a different result, that's good news: comparing the two is where the learning happens.

3.1 Show the accessories (category 3), cheapest first. Make sure the order never changes.

3.2 Show the products over $100 that are not laptops, most expensive first.

3.3 Show the two most recent orders.

3.4 Which customers have no email address?

3.5 Show only the third most expensive product. (*Careful: there's a tie.*)

Answers are in Appendix C.

WHERE decides which rows you keep, ORDER BY decides how they're arranged, and LIMIT decides how many you see. In the next chapter, you'll remove duplicate rows with DISTINCT and give your columns clearer names with aliases.

C

Answers to the Exercises

Chapter 1

1.1 One order: for example, order 1001, placed by customer 2 (Sara) on 2025-01-08.

1.2 So each customer's details are stored only once. If Sara's phone changes, one row in Customers is updated, and every order still points to her.

1.3 A DBMS: the software that runs databases. In everyday speech, people call it a database too.

1.4 Any two of: the same customer typed different ways, two copies of the file, text typed into a number column, the file getting slow or breaking, people overwriting each other's work.

1.5 (a) Excel: small, personal, one-off. (b) SQL: big data and a report that repeats every month. (c) Both: SQL to get the totals, Excel to make the chart. (d) SQL: the website writes to the database; a spreadsheet can't do this job.

Chapter 2

2.1 MySQL: `SHOW TABLES;` SQL Server: `SELECT name FROM sys.tables;` Either way you see six tables: Categories, Customers,

Employees, OrderItems, Orders and Products (on Windows, MySQL shows them in lowercase).

2.2

```sql
SELECT *
FROM Employees;
```

There are 8 employees, from Hana Saeed (the CEO, employee 1) to Salma Idris (support agent, employee 8).

2.3 CustomerID. It is a foreign key in Orders and the primary key of Customers.

2.4 OrderItems, in its Quantity column. Each row is one product in one order.

2.5 Run the ShopDB script again (`shopdb_mysql.sql` or `shopdb_sqlserver.sql`). It drops the database and builds it again with the original data.

Chapter 3

3.1

```sql
SELECT ProductName, UnitPrice
FROM Products
WHERE CategoryID = 3
ORDER BY UnitPrice, ProductID;
```

RESULT 8 rows

	ProductName	UnitPrice
1	100% Cotton Screen Cloth	4.99
2	USB-C Cable 1m	12.50
3	USB-C Cable 2m	15.00
4	USB 2.0 Hub	19.00
5	Wireless Mouse	25.00
6	Wireless Presenter	35.00
7	Laptop Backpack	45.00
8	Mechanical Keyboard	89.00

No two accessories cost the same today, but ProductID keeps the order fixed if one ever does.

3.2

SQL

```sql
SELECT ProductName, CategoryID, UnitPrice
FROM Products
WHERE UnitPrice > 100
  AND CategoryID <> 1
ORDER BY UnitPrice DESC;
```

RESULT 5 rows

	ProductName	CategoryID	UnitPrice
1	Pixel Pro Phone	2	899.00
2	27-inch 4K Monitor	4	379.00
3	Galaxy Lite Phone	2	299.00
4	Office Laser Printer	5	289.00
5	24-inch Full HD Monitor	4	159.00

The Gift Card has no category (NULL), so `CategoryID <> 1` is not true for it. It costs only 50.00 anyway, but remember this: a NULL never passes `<>` either.

3.3

MYSQL

```
SELECT OrderID, OrderDate
FROM Orders
ORDER BY OrderDate DESC, OrderID DESC
LIMIT 2;
```

SQL SERVER

```
SELECT TOP (2) OrderID, OrderDate
FROM Orders
ORDER BY OrderDate DESC, OrderID DESC;
```

RESULT 2 rows

	OrderID	OrderDate
1	1048	2026-06-15
2	1047	2026-06-14

3.4

SQL

```sql
SELECT FirstName, LastName
FROM Customers
WHERE Email IS NULL;
```

RESULT 1 row

	FirstName	LastName
1	Noor	Ibrahim

3.5

MYSQL

```sql
SELECT ProductID, ProductName, UnitPrice
FROM Products
ORDER BY UnitPrice DESC, ProductID
LIMIT 1 OFFSET 2;
```

SQL SERVER

```sql
SELECT ProductID, ProductName, UnitPrice
FROM Products
ORDER BY UnitPrice DESC, ProductID
OFFSET 2 ROWS FETCH NEXT 1 ROWS ONLY;
```

RESULT 1 row

	ProductID	ProductName	UnitPrice
1	5	Pixel Pro Phone	899.00

OFFSET 2 skips the two most expensive products and LIMIT 1 keeps the next one. Without the ProductID tie-breaker, the ProBook laptop and the Pixel phone (both 899.00) could swap places, and the answer could change between runs.