SIMPLIFY DATABASE

```
SELECT essentials
FROM sql
WHERE you_know_sql = 'already';
```

RESULT 1 row

1

Simplify SQL

Quick Reference

MySQL	SQL Server
LIMIT 5	TOP (5)
LOCATE('@', s)	CHARINDEX('@', s)
CURDATE()	CAST(GETDATE() AS DATE)
GROUP_CONCAT(x)	STRING_AGG(x, ', ')
AUTO_INCREMENT	IDENTITY(1,1)

Every topic on a page, plus the complete cheat sheet

---

20 topics: syntax, key points, traps

MySQL and **SQL Server**, side by side

Simplify SQL: Quick Reference

Every Topic on a Page, plus the MySQL and SQL Server Cheat Sheet

SIMPLIFY DATABASE

Contents

About This Quick Reference

ShopDB at a Glance

Chapter 3 SELECT, WHERE, ORDER BY and LIMIT

Chapter 4 DISTINCT and Aliases

Chapter 5 LIKE, IN and BETWEEN

Chapter 6 Working with NULL

Chapter 7 The CASE Expression

Chapter 8 GROUP BY and Aggregate Functions

Chapter 9 HAVING vs WHERE

Chapter 10 Primary Keys and Foreign Keys

Chapter 11 JOINs

Chapter 12 SELF JOIN

Chapter 13 UNION and UNION ALL

Chapter 14 Subqueries and CTEs

Chapter 15 Data Types

Chapter 16 Constraints

Chapter 17 INSERT, UPDATE and DELETE

Chapter 18 Transactions

Chapter 19 Normalization (1NF, 2NF, 3NF)

Chapter 20 String Functions

Chapter 21 Date Functions

Chapter 22 A First Look at Window Functions

The Cheat Sheet: MySQL and SQL Server

Want the Full Version?

About This Quick Reference

This is the short version of *Simplify SQL for Absolute Beginners*, for people who already write some SQL and want the essentials on one page per topic: what each feature does, the syntax in both MySQL and SQL Server, and the mistakes that cause wrong results.

Each topic page has three parts:

- **Covers:** what the topic is about.
- **Key points:** the syntax you need, in a table.
- **Watch out:** the mistakes I see most often, in real projects and from beginners alike.

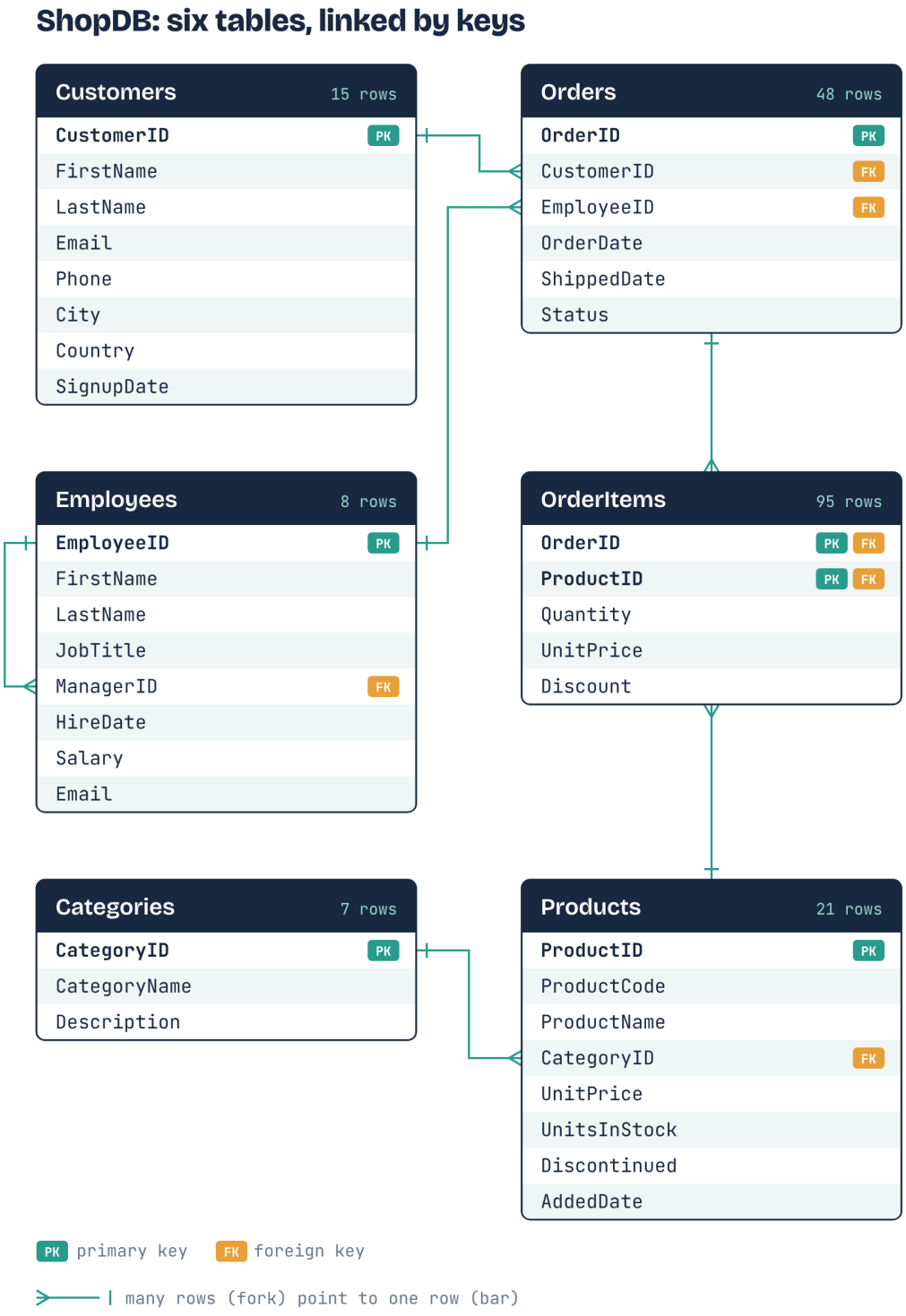
The chapter numbers match the full book, so if a page is too short, the full chapter is where to go next. The examples use ShopDB, a small practice database for an online shop; download it free from **simplifydatabase.com/shopdb-sample-database**.

The last part is the complete cheat sheet: every task in MySQL and SQL Server side by side, and the twelve traps worth remembering.

ShopDB at a Glance

Every example uses ShopDB, a small online electronics shop with six tables:

Table	One row is…	Key columns
Categories	a product group, such as Laptops	CategoryID
Products	something the shop sells, with price and stock	ProductID, CategoryID
Customers	a person who signed up	CustomerID
Employees	a member of staff; ManagerID points to another employee	EmployeeID, ManagerID
Orders	one order: customer, salesperson (NULL = online), dates, status	OrderID, CustomerID, EmployeeID
OrderItems	one product in an order, with quantity, price and discount	(OrderID, ProductID)



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

A sale's value is `Quantity * UnitPrice * (1 - Discount)` from OrderItems. The data has planned surprises: a customer with no orders, a product never sold, an empty category, a product with no category, price ties, and NULLs in phones, emails, countries and ship dates.

3

SELECT, WHERE, ORDER BY and LIMIT

Covers:

- 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

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`

Watch out

> **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.

4

DISTINCT and Aliases

Covers:

- Show each value only once with DISTINCT
- Understand why DISTINCT works on whole rows, not one column
- Count different values with COUNT(DISTINCT ...)
- Give columns and tables clear names with aliases
- Know where you can use an alias, and where you can't

Key points

You want to	Write
Each value once	`SELECT DISTINCT Status FROM Orders`
Each combination once	`SELECT DISTINCT Country, City FROM Customers`
How many different values	`SELECT COUNT(DISTINCT Country) FROM Customers`
Rename a column	`Salary * 12 AS AnnualSalary`
Sort by the new name	`ORDER BY AnnualSalary`
Filter on a calculation	`WHERE Salary * 12 > 60000` (not the alias)
A short table name	`FROM Employees AS e`, then `e.FirstName`

Watch out

- Expecting `SELECT DISTINCT Country, City` to give one row per country. DISTINCT applies to every selected column together.
- Adding DISTINCT to hide duplicates you don't understand.
- Using a column alias in WHERE. Repeat the calculation instead.
- Forgetting a comma, so a column silently becomes an alias. Always write AS.

5

LIKE, IN and BETWEEN

Covers:

- Search inside text with LIKE and its two wildcards
- Match any value in a list with IN
- Keep a range of numbers or dates with BETWEEN
- Avoid the trap each one hides: escaping, NULL in NOT IN, and dates

Key points

You want	Write	Watch out for
Text starts with X	`LIKE 'X%'`	Case sensitivity depends on collation
Text contains X	`LIKE '%X%'`	Can't use an index; slow on big tables
A real % or _	`LIKE '%!%%' ESCAPE '!'`	Without escaping, it matches everything
One of several values	`IN ('a', 'b', 'c')`	NOT IN and NULL return too few rows
Several text patterns	`LIKE 'a%' OR LIKE 'b%'`	IN doesn't accept wildcards
A number range	`BETWEEN 10 AND 20`	Both ends included; smaller value first
A date range	`>= '2026-03-01' AND < '2026-04-01'`	BETWEEN misses times on the last day

Watch out

- Searching for a real % or _ without ESCAPE, and matching every row.
- Forgetting that NULL passes neither LIKE nor NOT LIKE, nor NOT IN.
- Writing the larger value first in BETWEEN.
- Using BETWEEN on a date-and-time column and losing the last day.
- Expecting IN to understand wildcards. Use LIKE … OR LIKE … instead.

6

Working with NULL

Covers:

- What NULL really means, and what it doesn't
- Why you must write IS NULL, never = NULL
- How "unknown" quietly removes rows from your results
- Replace NULL with something readable using COALESCE
- How COUNT treats NULL

Key points

You want to	Write
Find missing values	`WHERE Phone IS NULL`
Find filled-in values	`WHERE Phone IS NOT NULL`
Include NULL rows in a "not equal"	`WHERE EmployeeID <> 4 OR EmployeeID IS NULL`
Show a fallback	`COALESCE(Phone, 'No phone')`
The first value that exists	`COALESCE(Phone, Email, 'No contact')`
Count rows / filled-in values	`COUNT(*)` / `COUNT(Phone)`

Watch out

> **Watch out:** `WHERE Phone = NULL` returns nothing, every time. Write `IS NULL`.

- Using `<>` and forgetting the NULL rows, so totals don't add up.
- Treating NULL as zero or as an empty string. It's neither.
- Expecting `COUNT(column)` to count every row. It skips NULL; `COUNT(*)` doesn't.

7

The CASE Expression

Covers:

- Add if/then/else logic to a query with CASE
- Use the searched form for ranges and the simple form for exact values
- Count several things in one pass with CASE inside SUM
- Sort in your own custom order
- Avoid the classic CASE mistakes: order of WHENs, NULL and mixed data types

Key points

You want	Write
Labels for ranges	`CASE WHEN UnitPrice >= 500 THEN 'Premium' WHEN ... ELSE 'Budget' END`
Labels for exact values	`CASE Discontinued WHEN 0 THEN 'Available' WHEN 1 THEN ... END`
Count matching rows	`SUM(CASE WHEN Status = 'Delivered' THEN 1 ELSE 0 END)`
A custom sort order	`ORDER BY CASE Status WHEN 'Pending' THEN 1 ... END`
A NULL test	`CASE WHEN EmployeeID IS NULL THEN 'Online' ... END`

The first true WHEN wins, so put the strictest condition first.

Watch out

> **Watch out:** Putting a broad condition first. `WHEN UnitPrice >= 100` before `WHEN UnitPrice >= 500` labels every expensive product "Mid-range", and nothing ever reaches the second line.

- Forgetting END. Every CASE needs one.
- Leaving out ELSE, then being surprised by NULL results.
- Writing `WHEN column = NULL`. Use `WHEN column IS NULL`.
- Returning a number or date in one branch and text in another.

8

GROUP BY and Aggregate Functions

Covers:

- Turn many rows into one answer with COUNT, SUM, AVG, MIN and MAX
- Get one result row per group with GROUP BY
- Follow the one rule every grouped query must obey
- Group by more than one column, and by a calculation
- Take a first look at HAVING, which filters groups

Key points

Part	What it does	Example
COUNT(*)	Counts rows	`COUNT(*)`
SUM()	Adds up values	`SUM(Quantity)`
AVG()	Averages values	`ROUND(AVG(UnitPrice), 2)`
MIN() / MAX()	Smallest / largest value	`MAX(UnitPrice)`
GROUP BY	One result row per group	`GROUP BY Status`
HAVING	Filters the groups	`HAVING COUNT(*) >= 5`

Watch out

- Selecting a column that isn't grouped. Every column must be in GROUP BY or inside an aggregate.
- Putting an aggregate in WHERE. `WHERE COUNT(*) >= 5` fails; counts and totals belong in HAVING.
- Forgetting that COUNT(column) skips NULLs. Use COUNT(*) when you mean "how many rows".
- Whole-number averages in SQL Server. AVG of an INT column drops the decimals.
- Expecting grouped results to come out sorted. Add ORDER BY.

9

HAVING vs WHERE

Covers:

- Know when to filter rows with WHERE and when to filter groups with HAVING
- Understand why SUM and COUNT can't go in WHERE
- Use WHERE and HAVING together, and read the result correctly
- Find duplicate values with GROUP BY and HAVING
- Keep row filters out of HAVING

Key points

	WHERE	HAVING
Filters	Individual rows	Groups
Runs	Before GROUP BY	After GROUP BY
Can use COUNT, SUM, AVG...	No	Yes
Needs GROUP BY?	No	Usually (without it, the whole result is one group)
Can use an index	Yes	Not for the aggregate condition

Watch out

- Putting COUNT, SUM or AVG in WHERE. Aggregates go in HAVING.
- Using a SELECT alias in HAVING. MySQL allows it; SQL Server doesn't. Repeat the expression.
- Putting plain row conditions in HAVING. It works, but it's slower and harder to read.
- Forgetting that WHERE runs first, so HAVING counts only the rows WHERE kept.

10

Primary Keys and Foreign Keys

Covers:

- Understand what a primary key is and the two rules it enforces
- See how a foreign key links one table to another
- Read ShopDB's keys, including a composite key and a table that points to itself
- Know what the database refuses to do, and why that protects you
- Choose what happens when a parent row is deleted

Key points

	Primary key	Foreign key
Job	Identifies each row in its own table	Links a row to a row in another table
Unique?	Yes, always	No, values can repeat
Can be NULL?	Never	Yes, unless you add NOT NULL
How many per table?	Exactly one (may have several columns)	As many as you need
ShopDB example	Customers.CustomerID	Orders.CustomerID

A primary key answers "which row is this?" and a foreign key answers "which row does this belong to?".

Watch out

- Creating a table without a primary key. Give every table one, no exceptions.
- Leaving out foreign keys "to keep things simple". The mess shows up years later.
- Using CASCADE on a shared parent like Customers or Products.
- Giving the two linked columns different data types. Make the foreign key the same type as the key it points to (INT to INT).

11

JOINs

Covers:

- Put matching rows from two tables side by side with INNER JOIN
- Keep every row from one table with LEFT JOIN, and find what's missing
- Know what RIGHT JOIN and FULL JOIN do (and why you rarely need them)
- Join three or four tables in one query
- Combine JOIN with GROUP BY, and avoid the LEFT JOIN trap that catches everyone

Key points

You want	Use
Only rows that match in both tables	INNER JOIN (or just JOIN)
Every row from the first table, matched or not	LEFT JOIN
Rows in A with no match in B	LEFT JOIN + `WHERE b.key IS NULL`
Every row from both tables	FULL JOIN (SQL Server; not in MySQL)
Data from three tables	One extra JOIN … ON per table
A filter on the right table that keeps all left rows	Put it in ON, not WHERE
A count that shows 0 for no matches	LEFT JOIN + COUNT(right_table.key)

For most everyday work you'll only need INNER JOIN and LEFT JOIN. Get comfortable with those two first.

Watch out

- A condition on the right table in WHERE turns a LEFT JOIN into an INNER JOIN. Put it in ON.
- Joining on the wrong column runs without error and returns nonsense. Join a foreign key to the key it points to.
- Two tables with the same column name: write `p.CategoryID`, or you get "ambiguous column".
- Joining two lists (orders and payments) multiplies the rows, so totals are too big. Total each list first.
- DISTINCT hides duplicate rows from a bad join; fix the join instead.

12

SELF JOIN

Covers:

- Join a table to itself with two aliases
- Put each employee's manager's name next to them
- Keep the top of the tree with LEFT JOIN
- Compare a row with a related row, and find pairs that share a value
- Climb more than one level of a hierarchy

Key points

You want	Write
Each row with its parent row	`FROM Employees e JOIN Employees m ON e.ManagerID = m.EmployeeID`
Keep rows with no parent	LEFT JOIN instead of JOIN
Compare with the parent	`WHERE e.Salary > m.Salary`
Pairs that share a value, once each	`ON a.Country = b.Country AND a.CustomerID < b.CustomerID`
Rows that are nobody's parent	LEFT JOIN + `WHERE e.EmployeeID IS NULL`

Watch out

- **No aliases.** `FROM Employees JOIN Employees` is an error. Give each copy its own meaningful name.
- **The ON condition backwards.** `e.ManagerID = m.EmployeeID` is "employee to manager". Swap it and every column means the opposite of its name.
- **Losing the top row.** An inner join drops rows whose ManagerID is NULL. Use LEFT JOIN when everyone must appear.
- **Self-pairs and double pairs** when matching on a shared value. Add `a.ID < b.ID`.

13

UNION and UNION ALL

Covers:

- Stack the results of two queries with UNION and UNION ALL
- Know which one removes duplicates, and which one to use by default
- Follow the four rules every UNION must obey
- Build a FULL JOIN in MySQL
- Meet UNION's relatives, INTERSECT and EXCEPT

Key points

	UNION	UNION ALL
Duplicates	Removed	Kept
Speed	Slower (sort or hash)	Faster
Row order	Not guaranteed	Not guaranteed
Safe for totals	No, may drop real rows	Yes
Use when	You want a distinct list	Almost every other time

INTERSECT keeps rows in both results; EXCEPT keeps rows in the first but not the second.

Watch out

- Using UNION when you mean UNION ALL, and silently losing rows that look the same.
- A different number of columns in each query.
- Columns in a different order in each query. They're matched by position, so `FirstName, LastName` and `LastName, FirstName` mix up the names without any error.
- An ORDER BY in the middle. Only one is allowed, at the end.
- Relying on the order of the rows without ORDER BY.

14

Subqueries and CTEs

Covers:

- Use one query's result inside another with a subquery
- Filter with IN and EXISTS, and avoid the NOT IN trap
- Build a query in steps with a derived table, then with a CTE
- Chain several CTEs, each with its own name
- Fix the "totals too big" JOIN problem from Chapter 11

Key points

You want	Write
Compare with one calculated value	`WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products)`
Match against a list from another table	`WHERE CustomerID IN (SELECT CustomerID FROM Orders ...)`
Rows with no match, NULL-safe	`WHERE NOT EXISTS (SELECT 1 FROM Orders o WHERE o.EmployeeID = e.EmployeeID)`
A query in named steps	`WITH Step1 AS (...), Step2 AS (...) SELECT ... FROM Step2`
Correct totals from several lists	One CTE per list, then join the CTEs

Watch out

- A subquery after `=` or `>` that returns more than one row. Use IN, or an aggregate.
- NOT IN with a subquery whose column can contain NULL. Use NOT EXISTS.
- Forgetting the alias on a subquery in FROM.
- Forgetting the semicolon before WITH in SQL Server.
- Joining several lists, then adding up the multiplied rows. Total each list in its own CTE first.

15

Data Types

Covers:

- Understand why every column has a data type, and what the type protects you from
- Read the types of any table from INFORMATION_SCHEMA
- Pick the right type for whole numbers, money, text, dates and yes/no values
- Avoid the classic traps: FLOAT money, whole-number division and dates stored as text
- Convert from one type to another with CAST

Key points

You store	Use
IDs, quantities, counts	INT
Money	DECIMAL(10,2), or DECIMAL(12,3) for three-decimal currencies
Measurements	FLOAT is fine
Short text	VARCHAR(n) / NVARCHAR(n) with a real limit
Dates	DATE (with time: DATETIME2 / DATETIME)
Yes/no	BIT / BOOLEAN
Convert a value	CAST(value AS type)

Watch out

- FLOAT for money. Use DECIMAL.
- Dates or numbers stored as text. They sort wrongly and can't be checked.
- VARCHAR(255) for everything. The limit is free validation.
- VARCHAR for Arabic text in SQL Server. Use NVARCHAR and N'...'.
- Dividing whole numbers in SQL Server and losing the decimals.

16

Constraints

Covers:

- Know the six kinds of constraint and what each one guarantees
- Create a table with NOT NULL, UNIQUE, CHECK and DEFAULT, and watch them work
- Understand how NULL behaves in UNIQUE and CHECK
- Add a constraint to a table that already has data
- Name your constraints so their errors make sense

Key points

Constraint	Write	Note
NOT NULL	`CompanyName VARCHAR(100) NOT NULL`	A property of the column
UNIQUE	`CONSTRAINT UQ_Suppliers_Code UNIQUE (SupplierCode)`	NULLs: many in MySQL, one in SQL Server
CHECK	`CONSTRAINT CK_... CHECK (PaymentDays BETWEEN 0 AND 120)`	NULL passes; MySQL 8.0.16+
DEFAULT	`PaymentDays INT NOT NULL DEFAULT 30`	Used only when no value is given
Add later	`ALTER TABLE ... ADD CONSTRAINT ...`	Existing data must pass first

Watch out

- Leaving columns nullable “just in case”. Make them NOT NULL unless a missing value really means something.
- Expecting CHECK to reject NULL. Add NOT NULL as well.
- Expecting UNIQUE to allow several NULLs in SQL Server. Use a filtered unique index.
- Adding a UNIQUE constraint before checking for duplicates.
- Leaving constraints unnamed, then trying to work out what CK__Orders__Status__3D5E1FD2 means.

17

INSERT, UPDATE and DELETE

Covers:

- Add rows with INSERT, and let the database pick the ID
- Change rows with UPDATE, using their current values
- Remove rows with DELETE, in the order the foreign keys need
- Update one table from another, and insert-or-update in one step
- Build the habits that stop one missing WHERE from ruining your day

Key points

Task	Command
Add a row	`INSERT INTO t (a, b) VALUES (1, 'x');`
Add many rows	`INSERT INTO t (a, b) VALUES (1, 'x'), (2, 'y');`
Copy rows from a query	`INSERT INTO t (a, b) SELECT a, b FROM s WHERE ...;`
Change values	`UPDATE t SET b = 'z' WHERE a = 1;`
Change using the current value	`UPDATE t SET stock = stock - 2 WHERE a = 1;`
Remove rows	`DELETE FROM t WHERE a = 1;`
Remove all rows fast	`TRUNCATE TABLE t;`

Watch out

- UPDATE or DELETE without WHERE.
- INSERT without column names.
- Inserting order lines before the order, or deleting the order before its lines.
- Typing IDs yourself when the column generates them.
- Using REPLACE INTO as an upsert and losing columns.

18

Transactions

Covers:

- Group several changes into one all-or-nothing unit
- Use START TRANSACTION, COMMIT and ROLLBACK
- Undo a mistake before anyone sees it
- Understand the four ACID promises
- Handle an error in the middle of a transaction correctly

Key points

You want	Write
Start a group of changes	`START TRANSACTION;` (SQL Server: `BEGIN TRANSACTION;`)
Save them all	`COMMIT;`
Undo them all	`ROLLBACK;`
Roll back on any error (SQL Server)	`BEGIN TRY ... COMMIT; END TRY BEGIN CATCH ROLLBACK; THROW; END CATCH`

A transaction bundles several changes into one all-or-nothing unit. Next time you're about to run an important UPDATE or DELETE, type START TRANSACTION first.

Watch out

- Forgetting that every statement commits on its own (autocommit) when you didn't start a transaction.
- Committing after an error, and saving half the work.
- Leaving a transaction open, and blocking other users.
- Expecting a ROLLBACK to undo CREATE or ALTER TABLE in MySQL.

19

Normalization (1NF, 2NF, 3NF)

Covers:

- See the three problems that duplicated data causes
- Fix a spreadsheet-style table step by step, through 1NF, 2NF and 3NF
- Recognise each step in ShopDB's own design
- Spot normalization problems in your own tables with five questions
- Know when copying data on purpose is the right answer

Key points

Step	Ask	ShopDB result
1NF	Is every cell a single value?	One row per product on each order
2NF	Does every column need the whole key?	Orders, Products and OrderItems
3NF	Does any column describe another column instead of the key?	Customers and Categories split out
View	How do I see it all together?	JOIN the tables back

Watch out

- Storing lists in a cell ("Monitors, Phones") or in numbered columns.
- Copying a customer's or product's details into every order.
- Using names as keys. Names change; IDs don't.
- Treating the order's price as a copy to remove. It's a snapshot, and it belongs there.
- Splitting tables so far that every query needs eight joins.

20

String Functions

Covers:

- Join text together with CONCAT and CONCAT_WS, and avoid the NULL trap
- Measure, trim and change the case of text
- Cut out part of a string with LEFT, RIGHT and SUBSTRING
- Find, replace and pad text
- Turn many rows into one list with GROUP_CONCAT and STRING_AGG

Key points

Task	SQL Server	MySQL
Join text	`CONCAT(a, b)`, `a + b`	`CONCAT(a, b)`
Join, skipping NULLs	`CONCAT_WS(' ', a, b)`	`CONCAT_WS(' ', a, b)`
Characters	`LEN(s)`	`CHAR_LENGTH(s)`
Upper / lower case	`UPPER(s)`, `LOWER(s)`	`UPPER(s)`, `LOWER(s)`
Remove spaces	`TRIM(s)`	`TRIM(s)`
First / last n characters	`LEFT(s, n)`, `RIGHT(s, n)`	`LEFT(s, n)`, `RIGHT(s, n)`
Part of a string	`SUBSTRING(s, start, len)`	`SUBSTRING(s, start, len)`
Find position	`CHARINDEX(find, s)`	`LOCATE(find, s)`
Replace	`REPLACE(s, old, new)`	`REPLACE(s, old, new)`
Pad with zeros	`RIGHT('0000' + CAST(n AS VARCHAR), 4)`	`LPAD(n, 4, '0')`
Rows to one list	`STRING_AGG(col, ', ')`	`GROUP_CONCAT(col SEPARATOR ', ')`

Watch out

- MySQL's CONCAT returns NULL if any part is NULL. Use CONCAT_WS.
- MySQL's LENGTH counts bytes, not characters. Use CHAR_LENGTH (SQL Server: LEN, which ignores trailing spaces).
- Positions start at 1, and "not found" is 0.
- Hidden spaces from imports break comparisons. TRIM on the way in.
- A function on the column in WHERE (`LEFT(LastName, 2) = 'Ha'`) stops the index being used. Use `LIKE 'Ha%'`.
- Never build SQL by joining strings with user input (SQL injection). Use parameters.

21

Date Functions

Covers:

- Get today's date, and take a date apart into year, month and day
- Group orders and sales by month
- Add to a date, and count the days between two dates
- Find the first and last day of a month, and turn a date into text
- Avoid the two date mistakes that cause wrong totals and slow queries

Key points

Task	SQL Server	MySQL
Today	`CAST(GETDATE() AS DATE)`	`CURDATE()`
Year, month, day	`YEAR(d)`, `MONTH(d)`, `DAY(d)`	same
Month / day name	`DATENAME(month, d)`	`MONTHNAME(d)`, `DAYNAME(d)`
Add 7 days	`DATEADD(day, 7, d)`	`DATE_ADD(d, INTERVAL 7 DAY)`
Days between	`DATEDIFF(day, start, end)`	`DATEDIFF(end, start)`
Last day of month	`EOMONTH(d)`	`LAST_DAY(d)`
Date to text	`CONVERT(VARCHAR(10), d, 103)`, `FORMAT`	`DATE_FORMAT(d, '%d/%m/%Y')`
One month in WHERE	`d >= '2026-03-01' AND d < '2026-04-01'`	same

Watch out

- MySQL's `DATEDIFF(end, start)` and SQL Server's `DATEDIFF(day, start, end)` take the dates in opposite order.
- SQL Server's DATEDIFF counts boundaries: one day from 31 Dec to 1 Jan is "1 year".
- BETWEEN on a date-and-time column misses most of the last day. Use `>= start AND < day after end`.
- `YEAR(OrderDate) = 2026` in WHERE stops the index being used. Compare the bare column with a range.
- GROUP BY can't show months with no rows. Use a calendar table.
- Write dates as 'YYYY-MM-DD'; '03/04/2026' means different days in different countries.

22

A First Look at Window Functions

Covers:

- Add totals and averages to every row with OVER, without losing the rows
- Calculate per group with PARTITION BY
- Number and rank rows with ROW_NUMBER, RANK and DENSE_RANK
- Find the top row in each group
- Build running totals and compare each row with the previous one using LAG

Key points

You want	Write
A total next to every row	`SUM(x) OVER ()`
A total per group, next to every row	`SUM(x) OVER (PARTITION BY g)`
Number the rows	`ROW_NUMBER() OVER (ORDER BY x, id)`
Rank with ties	`RANK()` or `DENSE_RANK() OVER (ORDER BY x)`
Top row per group	`ROW_NUMBER() OVER (PARTITION BY g ORDER BY x)` in a CTE, then `WHERE rn = 1`
Running total	`SUM(x) OVER (ORDER BY date)`
Previous row's value	`LAG(x) OVER (ORDER BY date)`

Watch out

- Using a window function in WHERE. Calculate it in a CTE or subquery and filter outside.
- Forgetting ORDER BY inside OVER for running totals, LAG and ranking. Without it the order is undefined.
- Expecting ROW_NUMBER to be stable when there are ties. Add a tie-breaker column.
- Forgetting that the window only sees the rows that passed WHERE.
- Trying them on MySQL 5.7 or older, which doesn't support them.

The Cheat Sheet: MySQL and SQL Server

Everything in this book on a few pages, for when you know *what* you want and just need the syntax. Where MySQL and SQL Server agree, there's one line; where they differ, both are shown. The chapter number tells you where to read more.

The shape of a query

```sql
SELECT   columns, expressions AS alias  -- 5. what to show
FROM     table t                        -- 1. where from
JOIN     other o ON o.key = t.key       --    + joined tables
WHERE    row conditions                 -- 2. which rows
GROUP BY columns                        -- 3. make groups
HAVING   group conditions               -- 4. which groups
ORDER BY columns                        -- 6. sort
LIMIT    n;                             -- 7. how many
```

The numbers are the order the database runs the clauses in (Chapter 9). SQL Server writes `SELECT TOP (n)` instead of LIMIT. An alias made in SELECT works in ORDER BY, not in WHERE or GROUP BY.

Reading and filtering *(Chapters 3–5)*

Task	MySQL	SQL Server
All columns	`SELECT * FROM t`	same
First 5 rows	`... ORDER BY x LIMIT 5`	`SELECT TOP (5) ... ORDER BY x`
Rows 41–60	`... ORDER BY x LIMIT 20 OFFSET 40`	`... ORDER BY x OFFSET 40 ROWS FETCH NEXT 20 ROWS ONLY`
Different values only	`SELECT DISTINCT a, b`	same
Several values	`WHERE x IN (1, 2, 3)`	same
A range (both ends included)	`WHERE x BETWEEN 10 AND 20`	same
Starts with	`WHERE s LIKE 'Lap%'`	same
Exactly one character	`WHERE code LIKE 'LAP-00_'`	same
A real % or _	`LIKE '%100!%%' ESCAPE '!'`	same
AND with OR	`WHERE a = 1 AND (b = 2 OR c = 3)`	same (always use brackets)

NULL *(Chapter 6)*

Task	Write
Is missing / is not missing	`x IS NULL`, `x IS NOT NULL` (never `= NULL`)
Default for NULL	`COALESCE(x, 0)` (MySQL also `IFNULL`, SQL Server also `ISNULL`)
Turn a value into NULL	`NULLIF(x, 0)` (avoids division by zero)
"Not equal" that keeps NULLs	`WHERE x <> 5 OR x IS NULL`

CASE *(Chapter 7)*

```sql
CASE WHEN UnitPrice >= 500 THEN 'Premium'  -- first true WHEN wins
     WHEN UnitPrice >= 100 THEN 'Mid-range'
     ELSE 'Budget' END
```

Task	MySQL	SQL Server
Two-way shortcut	`IF(cond, a, b)`	`IIF(cond, a, b)`
Count matching rows	`SUM(CASE WHEN cond THEN 1 ELSE 0 END)`	same
Custom sort order	`ORDER BY CASE Status WHEN 'Pending' THEN 1 ... END`	same

Grouping *(Chapters 8–9)*

Task	Write
Count rows / non-NULL values / different values	`COUNT(*)`, `COUNT(x)`, `COUNT(DISTINCT x)`
Total, average, smallest, largest	`SUM(x)`, `AVG(x)`, `MIN(x)`, `MAX(x)`
One row per group	`GROUP BY a, b` (every selected column grouped or aggregated)
Filter groups	`HAVING COUNT(*) > 1`
Find duplicates	`GROUP BY x HAVING COUNT(*) > 1`
Average of an INT without losing decimals	`AVG(x * 1.0)` (SQL Server needs it)

Joins *(Chapters 10–12)*

Task	Write
Matching rows only	`FROM a JOIN b ON b.a_id = a.id`
Keep all rows of a	`FROM a LEFT JOIN b ON b.a_id = a.id`
Rows of a with no match	`LEFT JOIN b ... WHERE b.id IS NULL`
Filter b, keep all of a	put the condition in `ON`, not `WHERE`
Count that shows 0	`COUNT(b.id)` after a LEFT JOIN
Table joined to itself	`FROM Employees e JOIN Employees m ON e.ManagerID = m.EmployeeID`
Pairs once each	`ON a.x = b.x AND a.id < b.id`
Everything from both sides	SQL Server: `FULL JOIN`; MySQL: LEFT JOIN `UNION` RIGHT JOIN

Combining results *(Chapter 13)*

Task	Write
Stack, keep everything	`SELECT ... UNION ALL SELECT ...`
Stack, remove duplicates	`SELECT ... UNION SELECT ...`
In both / in first only	`INTERSECT`, `EXCEPT` (MySQL 8.0.31+)
Rules	same number of columns, matched by position; names from the first query; one ORDER BY at the end

Subqueries and CTEs *(Chapter 14)*

```SQL
WITH Totals AS (                        -- a named step
    SELECT CustomerID, SUM(x) AS Total
    FROM ... GROUP BY CustomerID
)
SELECT * FROM Totals
WHERE Total > (SELECT AVG(Total) FROM Totals);
```

Task	Write
Compare with one calculated value	`WHERE x > (SELECT AVG(x) FROM t)`
Match a list from another table	`WHERE id IN (SELECT ... )`
No match, NULL-safe	`WHERE NOT EXISTS (SELECT 1 FROM b WHERE b.a_id = a.id)`
Totals from two lists	one CTE per list, then join the CTEs

Window functions *(Chapter 22, MySQL 8+)*

Task	Write
Total next to every row	`SUM(x) OVER ()`
Per group, keep rows	`AVG(x) OVER (PARTITION BY g)`
Number / rank rows	`ROW_NUMBER()`, `RANK()`, `DENSE_RANK() OVER (ORDER BY x DESC, id)`
Top row per group	`ROW_NUMBER() OVER (PARTITION BY g ORDER BY x DESC)` in a CTE, then `WHERE rn = 1`
Running total	`SUM(x) OVER (ORDER BY d)`
Previous / next row	`LAG(x) OVER (ORDER BY d)`, `LEAD(x) OVER (ORDER BY d)`

Text *(Chapter 20)*

Task	MySQL	SQL Server
Join text	`CONCAT(a, ' ', b)` (NULL if any part is NULL)	`CONCAT(a, ' ', b)` or `a + ' ' + b`
Join, skipping NULLs	`CONCAT_WS(', ', a, b)`	same (2017+)
Length in characters	`CHAR_LENGTH(s)`	`LEN(s)`
Upper / lower, trim	`UPPER(s)`, `LOWER(s)`, `TRIM(s)`	same
First / last n, middle	`LEFT(s, 3)`, `RIGHT(s, 3)`, `SUBSTRING(s, 5, 3)`	same
Find position	`LOCATE('@', s)`	`CHARINDEX('@', s)`
Replace	`REPLACE(s, ' ', '')`	same
Pad with zeros	`LPAD(n, 4, '0')`	`RIGHT('0000' + CAST(n AS VARCHAR(4)), 4)`
Rows to one list	`GROUP_CONCAT(x ORDER BY x SEPARATOR ', ')`	`STRING_AGG(x, ', ') WITHIN GROUP (ORDER BY x)`

Dates *(Chapter 21)*

Task	MySQL	SQL Server
Today	`CURDATE()`	`CAST(GETDATE() AS DATE)`
Now	`NOW()`	`GETDATE()`, `SYSDATETIME()`
Year, month, day	`YEAR(d)`, `MONTH(d)`, `DAY(d)`	same
Month / day name	`MONTHNAME(d)`, `DAYNAME(d)`	`DATENAME(month, d)`, `DATENAME(weekday, d)`
Add 7 days	`DATE_ADD(d, INTERVAL 7 DAY)`	`DATEADD(day, 7, d)`
Days between	`DATEDIFF(end, start)`	`DATEDIFF(day, start, end)`
Full years between	`TIMESTAMPDIFF(YEAR, start, end)`	`DATEDIFF(month, start, end) / 12`
First of month	`DATE_FORMAT(d, '%Y-%m-01')`	`DATEFROMPARTS(YEAR(d), MONTH(d), 1)`
Last of month	`LAST_DAY(d)`	`EOMONTH(d)`
Date as text	`DATE_FORMAT(d, '%d/%m/%Y')`	`CONVERT(VARCHAR(10), d, 103)`
One month in WHERE	`d >= '2026-03-01' AND d < '2026-04-01'`	same

Types and tables *(Chapters 15–16)*

Task	MySQL	SQL Server
Auto-number ID	`INT AUTO_INCREMENT PRIMARY KEY`	`INT IDENTITY(1,1) PRIMARY KEY`
Money	`DECIMAL(10,2)`	same (never FLOAT)
Text in any language	`VARCHAR(100)` with utf8mb4	`NVARCHAR(100)`, values as `N'...'`
Yes/no	`BOOLEAN` (TINYINT(1))	`BIT`
Convert a type	`CAST(x AS DECIMAL(10,2))`	same
Rules	`NOT NULL`, `DEFAULT 0`, `CONSTRAINT UQ_t_c UNIQUE (c)`, `CONSTRAINT CK_t_c CHECK (c > 0)`	same
Foreign key	`CONSTRAINT FK_child_parent FOREIGN KEY (col) REFERENCES parent (id)`	same
Unique, but many NULLs	plain `UNIQUE`	`CREATE UNIQUE INDEX ... WHERE c IS NOT NULL`
List a table's columns	`DESCRIBE t` or `INFORMATION_SCHEMA.COLUMNS`	`EXEC sp_help 't'` or `INFORMATION_SCHEMA.COLUMNS`

Changing data *(Chapters 17–18)*

Task	MySQL	SQL Server
Add rows	`INSERT INTO t (a, b) VALUES (1, 'x'), (2, 'y')`	same
Copy rows	`INSERT INTO t (a, b) SELECT a, b FROM s WHERE ...`	same
New ID just created	`LAST_INSERT_ID()`	`SCOPE_IDENTITY()`
Change rows	`UPDATE t SET b = b + 1 WHERE a = 1`	same
Update from another table	`UPDATE t JOIN s ON ... SET t.x = s.x WHERE ...`	`UPDATE t SET t.x = s.x FROM t JOIN s ON ... WHERE ...`
Insert or update	`INSERT ... ON DUPLICATE KEY UPDATE x = x + 1`	`UPDATE`, then `IF @@ROWCOUNT = 0 INSERT`
Remove rows	`DELETE FROM t WHERE a = 1` (children first)	same
Empty / remove a table	`TRUNCATE TABLE t`, `DROP TABLE t`	same
Transaction	`START TRANSACTION; ... COMMIT;` or `ROLLBACK;`	`BEGIN TRANSACTION; ... COMMIT;` or `ROLLBACK;`

The traps, in one list

1. `= NULL` is never true. Use `IS NULL`. (6)
2. `NOT IN (... NULL ...)` returns nothing. Use `NOT EXISTS`. (5, 14)
3. A condition on the right table in WHERE turns a LEFT JOIN into an INNER JOIN. Put it in ON. (11)
4. Joining two lists multiplies the rows, and every SUM is wrong. Total each list first. (11, 14)
5. UNION silently drops rows that look the same. Use UNION ALL for totals. (13)
6. SQL Server: whole number ÷ whole number drops the decimals (`7 / 2 = 3`). (8, 15)
7. MySQL: `CONCAT` with one NULL gives NULL. Use `CONCAT_WS`. (20)
8. `BETWEEN` on a date-and-time column misses the last day. Use `>= start AND < next day`. (21)
9. A function on a column in WHERE stops the index being used. Compare the bare column. (20, 21)
0. ROW_NUMBER and LIMIT with ties give a random order. Add a tie-breaker column. (3, 22)
1. UPDATE or DELETE without WHERE changes every row. SELECT first; use a transaction. (17, 18)
2. A failed statement doesn't end the transaction. ROLLBACK, don't COMMIT. (18)

Want the Full Version?

This quick reference is the summary of **Simplify SQL for Absolute Beginners: Learn to Query Databases Step by Step**: 23 chapters with every query explained, real results from ShopDB, 110 practice questions with answers, a 15-question final project and 25 interview questions.

The free articles behind it, and the next books in the series, are on **SimplifyDatabase.com**.