



DataDesk Tools
DATA TOOLKITS AND WORKFLOW ASSETS

SQL Query Toolkit

100+ Ready-to-Use SQL Queries for Analysts, Reporting & Business Data

Dialect note: These examples use PostgreSQL/Snowflake-style syntax where date functions appear (for example DATE_TRUNC, CURRENT_DATE, and INTERVAL). For SQL Server, adapt date logic with DATEADD, DATEDIFF, EOMONTH, or DATEFROMPARTS. For MySQL, adapt date logic with DATE_FORMAT, DATE_SUB, EXTRACT, or equivalent functions. Always replace sample table names, field names, date filters, and business thresholds before using a query in production.

by Data Desk Tools

Published by Data Desk Tools. © 2026 Data Desk Tools. All rights reserved. Licensed for personal use only. Redistribution, resale, sharing, sublicensing, or uploading to marketplaces is not permitted.

Table of Contents

1. Introduction
2. Practical SQL Basics
3. Most Used SQL Functions for Work
4. SQL Dialect Notes
5. Query Navigation by Role and Difficulty
6. Data Cleaning Queries
7. Sales Analysis Queries
8. Finance and Reporting Queries
9. Inventory and Product Data Queries
10. Customer Analysis Queries
11. Data Quality Control Queries
12. Advanced Practical SQL Queries
13. SQL Debugging Guide
14. Final SQL Cheat Sheet

1. Introduction

The SQL Query Toolkit is a practical collection of ready-to-use SQL queries for people who work with data in real business environments.

It is designed for analysts, Power BI users, advanced Excel users, ERP users, reporting specialists, and office professionals who need to extract, clean, summarize, validate, and analyze data with SQL.

This is not a theoretical SQL course. The goal is to help users save time by copying, adapting, and reusing practical query patterns in daily work.

Every query includes SQL syntax, a realistic sample output, a useful variant, and a variant sample output. The sample result columns are aligned with the actual SQL output aliases.

2. Practical SQL Basics

SELECT

Practical pattern used throughout the toolkit.

```
SELECT customer_id, customer_name FROM customers;
```

WHERE

Practical pattern used throughout the toolkit.

```
SELECT * FROM sales WHERE sales_amount > 1000;
```

ORDER BY

Practical pattern used throughout the toolkit.

```
SELECT * FROM sales ORDER BY sales_amount DESC;
```

GROUP BY

Practical pattern used throughout the toolkit.

```
SELECT customer_id, SUM(sales_amount) AS total_sales FROM sales GROUP BY customer_id;
```

HAVING

Practical pattern used throughout the toolkit.

```
SELECT customer_id, SUM(sales_amount) AS total_sales FROM sales GROUP BY customer_id HAVING SUM(sales_amount) > 10000;
```

JOIN

Practical pattern used throughout the toolkit.

```
SELECT s.order_id, c.customer_name FROM sales s LEFT JOIN customers c ON s.customer_id = c.customer_id;
```

CASE WHEN

Practical pattern used throughout the toolkit.

```
SELECT order_id, CASE WHEN sales_amount >= 10000 THEN 'High' ELSE 'Normal' END AS segment FROM sales;
```

CTE

Practical pattern used throughout the toolkit.

```
WITH x AS (SELECT customer_id, SUM(sales_amount) AS total_sales FROM sales GROUP BY customer_id) SELECT customer_id, total_sales FROM x;
```

Window functions

Practical pattern used throughout the toolkit.

```
SELECT customer_id, order_date, SUM(sales_amount) OVER (PARTITION BY customer_id ORDER BY order_date) AS running_total FROM sales;
```

3. Most Used SQL Functions for Work

This section is a practical quick reference for the SQL functions used most often in daily reporting, data cleaning, finance analysis, sales analysis, Power BI preparation, and data quality checks.

Function group	Most used functions	Best for	Typical business use
Aggregate functions	COUNT(), SUM(), AVG(), MIN(), MAX()	Use them for totals, counts, averages, maximum dates, minimum dates, and grouped business reporting.	Aggregate functions summarize many rows into a single number, date, or value.

Function group	Most used functions	Best for	Typical business use
Text functions	TRIM(), UPPER(), LOWER(), LEFT(), RIGHT(), SUBSTRING(), REPLACE(), LENGTH(), LEN(), CONCAT()	Use them when imported data has spaces, inconsistent capitalization, long product codes, prefixes, suffixes, or values that need standardization before joins.	Text functions clean, standardize, extract, and combine text values such as codes, names, emails, and categories.
Numeric functions	ROUND(), ABS(), CEIL(), FLOOR()	Use them for margin percentages, variance calculations, rounded KPI values, absolute differences, and report-ready numeric outputs.	Numeric functions format, round, and normalize numeric values for analysis and reporting.
NULL handling functions	COALESCE(), NULLIF(), ISNULL(), IFNULL()	Use them when reports show blanks, calculations return NULL, costs are missing, or percentages need safe division.	NULL handling functions replace missing values, prevent calculation errors, and avoid divide-by-zero problems.
Date functions	CURRENT_DATE, DATE_TRUNC(), EXTRACT(), YEAR(), MONTH(), DATEDIFF(), DATEADD()	Use them for monthly reports, YTD analysis, rolling periods, customer inactivity, aging reports, and period comparisons.	Date functions filter, group, compare, and calculate periods for reporting.
Conditional logic	CASE WHEN	Use it for customer segmentation, product status flags, risk labels, margin buckets, and report-friendly classifications.	CASE WHEN creates business labels, flags, and categories directly inside a query.
Conversion functions	CAST(), TRY_CAST(), TO_DATE(), TO_NUMBER()	Use them when data comes from Excel, CSV, ERP exports, or legacy systems where dates and numbers are stored as text.	Conversion functions change values from one data type to another, such as text to date or text to number.
Window functions	ROW_NUMBER(), RANK(), DENSE_RANK(), LAG(), LEAD()	Use them for latest price selection, deduplication, rankings, period comparisons, running totals, and moving averages.	Window functions calculate rankings, previous values, next values, latest records, and running calculations without collapsing rows like GROUP BY.

Aggregate functions

Common functions: COUNT(), SUM(), AVG(), MIN(), MAX()

What they do: Aggregate functions summarize many rows into a single number, date, or value.

When to use them: Use them for totals, counts, averages, maximum dates, minimum dates, and grouped business reporting.

SQL example:

```
SELECT
  customer_id,
  COUNT(order_id) AS number_of_orders,
  SUM(sales_amount) AS total_sales,
  AVG(sales_amount) AS average_order_value,
  MAX(order_date) AS last_order_date
FROM sales
GROUP BY customer_id;
```

Sample result:

customer_id	number_of_orders	total_sales	average_order_value	last_order_date
CUST001	12	18,450.00	1,537.50	2026-05-28
CUST002	7	9,200.00	1,314.29	2026-05-19

Common mistake: Do not mix aggregated and non-aggregated columns without GROUP BY.

Text functions

Common functions: TRIM(), UPPER(), LOWER(), LEFT(), RIGHT(), SUBSTRING(), REPLACE(), LENGTH(), LEN(), CONCAT()

What they do: Text functions clean, standardize, extract, and combine text values such as codes, names, emails, and categories.

When to use them: Use them when imported data has spaces, inconsistent capitalization, long product codes, prefixes, suffixes, or values that need standardization before joins.

SQL example:

```
SELECT
    product_code,
    LEFT(product_code, 3) AS product_family,
    RIGHT(product_code, 4) AS product_suffix,
    UPPER(TRIM(product_name)) AS product_name_clean
FROM products;
```

Sample result:

product_code	product_family	product_suffix	product_name_clean
ABC-1001	ABC	1001	INDUSTRIAL SENSOR
MTR-2045	MTR	2045	MOTOR CONTROLLER

Common mistake: LEN() is common in SQL Server, while LENGTH() is common in Snowflake, PostgreSQL, MySQL, Oracle, and DB2.

Numeric functions

Common functions: ROUND(), ABS(), CEIL(), FLOOR()

What they do: Numeric functions format, round, and normalize numeric values for analysis and reporting.

When to use them: Use them for margin percentages, variance calculations, rounded KPI values, absolute differences, and report-ready numeric outputs.

SQL example:

```
SELECT
    product_id,
    revenue,
    cost_amount,
    ROUND((revenue - cost_amount) / NULLIF(revenue, 0) * 100, 2) AS margin_percentage
FROM sales;
```

Sample result:

product_id	revenue	cost_amount	margin_percentage
P001	1,200.00	780.00	35.00
P002	950.00	700.00	26.32

Common mistake: When calculating percentages, protect divisions with NULLIF(value, 0) to avoid divide-by-zero errors.

NULL handling functions

Common functions: COALESCE(), NULLIF(), ISNULL(), IFNULL()

What they do: NULL handling functions replace missing values, prevent calculation errors, and avoid divide-by-zero problems.

When to use them: Use them when reports show blanks, calculations return NULL, costs are missing, or percentages need safe division.

SQL example:

```
SELECT
    product_id,
    unit_price,
    unit_cost,
```

```
COALESCE(unit_cost, 0) AS unit_cost_clean
FROM products;
```

Sample result:

product_id	unit_price	unit_cost	unit_cost_clean
P001	120.00	80.00	80.00
P002	95.00	NULL	0.00

Common mistake: Do not automatically replace NULL with zero unless zero is really the correct business meaning.

Date functions

Common functions: CURRENT_DATE, DATE_TRUNC(), EXTRACT(), YEAR(), MONTH(), DATEDIFF(), DATEADD()

What they do: Date functions filter, group, compare, and calculate periods for reporting.

When to use them: Use them for monthly reports, YTD analysis, rolling periods, customer inactivity, aging reports, and period comparisons.

SQL example:

```
SELECT
    DATE_TRUNC('month', order_date) AS sales_month,
    SUM(sales_amount) AS total_sales
FROM sales
GROUP BY DATE_TRUNC('month', order_date)
ORDER BY sales_month;
```

Sample result:

sales_month	total_sales
2026-01-01	125,000.00
2026-02-01	138,500.00

Common mistake: DATE_TRUNC() works in Snowflake/PostgreSQL. SQL Server, MySQL, Oracle, and DB2 may require different date syntax.

Conditional logic

Common functions: CASE WHEN

What they do: CASE WHEN creates business labels, flags, and categories directly inside a query.

When to use them: Use it for customer segmentation, product status flags, risk labels, margin buckets, and report-friendly classifications.

SQL example:

```
SELECT
    customer_id,
    total_sales,
    CASE
        WHEN total_sales >= 50000 THEN 'High Value'
        WHEN total_sales >= 10000 THEN 'Medium Value'
        ELSE 'Low Value'
    END AS customer_segment
FROM customer_sales;
```

Sample result:

customer_id	total_sales	customer_segment
CUST001	67,500.00	High Value
CUST002	18,500.00	Medium Value

customer_id	total_sales	customer_segment
CUST003	4,200.00	Low Value

Common mistake: Order your CASE WHEN conditions carefully. SQL stops at the first true condition.

Conversion functions

Common functions: CAST(), TRY_CAST(), TO_DATE(), TO_NUMBER()

What they do: Conversion functions change values from one data type to another, such as text to date or text to number.

When to use them: Use them when data comes from Excel, CSV, ERP exports, or legacy systems where dates and numbers are stored as text.

SQL example:

```
SELECT
  order_id,
  order_date_text,
  CAST(order_date_text AS DATE) AS order_date
FROM orders;
```

Sample result:

order_id	order_date_text	order_date
SO-1001	2026-01-15	2026-01-15
SO-1002	2026-01-28	2026-01-28

Common mistake: Check the original date format before converting. 01/02/2026 can mean different dates depending on locale.

Window functions

Common functions: ROW_NUMBER(), RANK(), DENSE_RANK(), LAG(), LEAD()

What they do: Window functions calculate rankings, previous values, next values, latest records, and running calculations without collapsing rows like GROUP BY.

When to use them: Use them for latest price selection, deduplication, rankings, period comparisons, running totals, and moving averages.

SQL example:

```
SELECT
  customer_id,
  order_date,
  sales_amount,
  LAG(sales_amount) OVER (PARTITION BY customer_id ORDER BY order_date) AS previous_sales_amount
FROM sales;
```

Sample result:

customer_id	order_date	sales_amount	previous_sales_amount
CUST001	2026-01-10	1,200.00	NULL
CUST001	2026-02-12	1,450.00	1,200.00

Common mistake: Always define the correct PARTITION BY and ORDER BY. Without them, previous/next values and rankings can be misleading.

4. SQL Dialect Notes

SQL Server uses TOP 10. PostgreSQL, MySQL, and Snowflake often use LIMIT 10. Oracle and DB2 support FETCH FIRST 10 ROWS ONLY.

COALESCE works in most SQL systems. SQL Server also has ISNULL, and MySQL has IFNULL.

DATE_TRUNC is common in Snowflake and PostgreSQL, but SQL Server usually requires YEAR, MONTH, or DATEFROMPARTS alternatives.

Always test date and conversion functions in your own SQL environment before using them in production.

5. Query Navigation by Role and Difficulty

This section helps users find the right queries faster based on their job role or SQL skill level.

Query Index

Query	Category	Title	Difficulty	Best for
Q001	Data Cleaning	Find Duplicate Records	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q002	Data Cleaning	Find Duplicate Business Keys	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q003	Data Cleaning	Remove Leading and Trailing Spaces	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q004	Data Cleaning	Standardize Text to Uppercase	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q005	Data Cleaning	Standardize Text to Lowercase	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q006	Data Cleaning	Replace Blank Strings with NULL	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q007	Data Cleaning	Replace NULL Values with Default Values	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q008	Data Cleaning	Find Empty Mandatory Fields	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q009	Data Cleaning	Find Invalid Email Formats	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q010	Data Cleaning	Find Invalid Date Values	Beginner	Data Analyst, BI Developer, Excel Power User, Data Manager
Q011	Data Cleaning	Convert Text Values to Dates	Intermediate	Data Analyst, BI Developer, Excel Power User, Data Manager
Q012	Data Cleaning	Convert Text Values to Numbers	Intermediate	Data Analyst, BI Developer, Excel Power User, Data Manager
Q013	Data Cleaning	Find Abnormal Numeric Values	Intermediate	Data Analyst, BI Developer, Excel Power User, Data Manager
Q014	Data Cleaning	Find Negative Amounts	Intermediate	Data Analyst, BI Developer, Excel Power User, Data Manager
Q015	Data Cleaning	Find Records with Missing Mandatory Fields	Intermediate	Data Analyst, BI Developer, Excel Power User, Data Manager
Q016	Sales Analysis	Total Sales by Month	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q017	Sales Analysis	Total Sales by Customer	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q018	Sales Analysis	Total Sales by Product	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q019	Sales Analysis	Top 10 Products by Revenue	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q020	Sales Analysis	Top 10 Customers by Revenue	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Query	Category	Title	Difficulty	Best for
Q021	Sales Analysis	Sales by Country or Region	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q022	Sales Analysis	Monthly Sales Trend	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q023	Sales Analysis	Year-over-Year Sales Comparison	Advanced Practical	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q024	Sales Analysis	Percentage Sales Growth	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q025	Sales Analysis	Average Order Value	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q026	Sales Analysis	Products with No Sales	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q027	Sales Analysis	Customers with No Recent Sales	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q028	Sales Analysis	Sales Ranking by Customer	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q029	Sales Analysis	Sales Ranking by Product	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q030	Sales Analysis	Sales Contribution Percentage by Product	Intermediate	Sales Analyst, Business Analyst, Power BI User, Commercial Controller
Q031	Finance and Reporting	Monthly Revenue	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q032	Finance and Reporting	Monthly Cost	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q033	Finance and Reporting	Monthly Gross Margin	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q034	Finance and Reporting	Gross Margin Percentage	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q035	Finance and Reporting	Budget vs Actual Comparison	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q036	Finance and Reporting	Budget Variance Percentage	Advanced Practical	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q037	Finance and Reporting	Revenue by Category	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q038	Finance and Reporting	Cost by Category	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q039	Finance and Reporting	Profit by Category	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q040	Finance and Reporting	Monthly Trend by Business Unit	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q041	Finance and Reporting	Year-to-Date Revenue	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager

Query	Category	Title	Difficulty	Best for
Q042	Finance and Reporting	Year-to-Date Margin	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q043	Finance and Reporting	Rolling 12 Months Revenue	Advanced Practical	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q044	Finance and Reporting	Month-over-Month Variance	Advanced Practical	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q045	Finance and Reporting	Report-Ready Summary Table	Intermediate	Finance Analyst, Controller, Reporting Specialist, Data Manager
Q046	Inventory and Product Data	Available Stock by Product	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q047	Inventory and Product Data	Products with Zero Stock	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q048	Inventory and Product Data	Products Below Safety Stock	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q049	Inventory and Product Data	Products Without Price	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q050	Inventory and Product Data	Products Without Cost	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q051	Inventory and Product Data	Products with Price Lower than Cost	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q052	Inventory and Product Data	Active Products	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q053	Inventory and Product Data	Discontinued Products	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q054	Inventory and Product Data	Duplicate Product Codes	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q055	Inventory and Product Data	Products Missing in Price Table	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q056	Inventory and Product Data	Products Missing in Cost Table	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q057	Inventory and Product Data	Products Missing in Stock Table	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q058	Inventory and Product Data	Inventory Value by Product	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q059	Inventory and Product Data	Inventory Value by Category	Intermediate	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q060	Inventory and Product Data	Slow-Moving Products	Advanced Practical	Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User
Q061	Customer Analysis	New Customers	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q062	Customer Analysis	Lost Customers	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Query	Category	Title	Difficulty	Best for
Q063	Customer Analysis	Returning Customers	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q064	Customer Analysis	Customer Purchase Frequency	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q065	Customer Analysis	Average Order Value by Customer	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q066	Customer Analysis	Top Customers by Revenue	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q067	Customer Analysis	Customers with Declining Sales	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q068	Customer Analysis	Customers with Growing Sales	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q069	Customer Analysis	Customers by Number of Orders	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q070	Customer Analysis	Customer Segmentation by Revenue	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q071	Customer Analysis	Customer Segmentation by Frequency	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q072	Customer Analysis	Customers Inactive for 90 Days	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q073	Customer Analysis	Customers Active This Month	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q074	Customer Analysis	First Purchase Date by Customer	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q075	Customer Analysis	Last Purchase Date by Customer	Intermediate	CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst
Q076	Data Quality Control	Find Duplicated Primary Keys	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q077	Data Quality Control	Find Missing Foreign Keys	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q078	Data Quality Control	Records in Table A but Not in Table B	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q079	Data Quality Control	Records in Table B but Not in Table A	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q080	Data Quality Control	Check Join Multiplication	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q081	Data Quality Control	Count Rows Before and After Join	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q082	Data Quality Control	Find Mismatched Customer Codes	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q083	Data Quality Control	Find Mismatched Product Codes	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner

Query	Category	Title	Difficulty	Best for
Q084	Data Quality Control	Find Invalid Negative Quantities	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q085	Data Quality Control	Find Invalid Negative Prices	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q086	Data Quality Control	Find Dates in the Future	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q087	Data Quality Control	Find Dates Before Allowed Minimum Date	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q088	Data Quality Control	Find Orphan Records	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q089	Data Quality Control	Validate Mandatory Fields	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q090	Data Quality Control	Compare Totals Between Two Tables	Intermediate	Data Manager, BI Developer, ERP Key User, Reporting Owner
Q091	Advanced Practical SQL	ROW_NUMBER for Ranking Records	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q092	Advanced Practical SQL	RANK for Ranking with Ties	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q093	Advanced Practical SQL	DENSE_RANK for Compact Ranking	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q094	Advanced Practical SQL	LAG to Compare with Previous Row	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q095	Advanced Practical SQL	LEAD to Compare with Next Row	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q096	Advanced Practical SQL	Running Total	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q097	Advanced Practical SQL	Moving Average	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q098	Advanced Practical SQL	Deduplicate Using ROW_NUMBER	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q099	Advanced Practical SQL	Get Latest Record per Customer	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q100	Advanced Practical SQL	Get Latest Price per Product	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q101	Advanced Practical SQL	Pivot with CASE WHEN	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q102	Advanced Practical SQL	Current Month vs Previous Month	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q103	Advanced Practical SQL	Current Year vs Previous Year	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder
Q104	Advanced Practical SQL	Find Gaps in Numeric Sequences	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Query	Category	Title	Difficulty	Best for
Q105	Advanced Practical SQL	Categorize Records Using CASE WHEN	Advanced Practical	Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Recommended Paths by Role

Data Analyst / BI Developer: Start with Data Cleaning, Data Quality Control, and Advanced Practical SQL.

Business Analyst / Sales Analyst: Start with Sales Analysis, Customer Analysis, and the ranking/trend queries.

Finance Analyst / Controller: Start with Finance and Reporting, month-over-month, YTD, rolling 12 months, margin, and variance queries.

Supply Chain / Product Data User: Start with Inventory and Product Data, product master checks, missing prices/costs, and slow-moving product queries.

Excel / Power BI Advanced User: Start with report-ready summary tables, pivots with CASE WHEN, date grouping, and data quality checks.

Recommended Paths by Skill Level

Beginner: Start with SELECT, WHERE, GROUP BY, NULL handling, duplicate checks, missing fields, and simple sales summaries.

Intermediate: Move to joins, budget vs actual, customer segmentation, product/inventory checks, and data quality controls.

Advanced Practical: Use window functions, ROW_NUMBER, RANK, LAG, LEAD, running totals, moving averages, and deduplication patterns.

6. Data Cleaning Queries

Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports.

QUERY N. 1 - Find Duplicate Records

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to find duplicate records.

When to use it: Use this when you suspect repeated keys or transactions and need to prove whether duplicated records are inflating totals or breaking joins. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, product_id, order_date, sales_amount, COUNT(*) AS duplicate_count
FROM sales
GROUP BY customer_id, product_id, order_date, sales_amount
HAVING COUNT(*) > 1;
```

Sample result:

customer_id	product_id	order_date	sales_amount	duplicate_count
C1001	P-10045	2026-02-14	1,250.00	2
C1045	P-22010	2026-02-18	860.00	3

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before joining a master table to a transaction table, a data analyst checks whether the supposedly unique key appears more than once. Example request: 'Can you check the source data and apply find duplicate records before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT order_id, COUNT(*) AS duplicate_count
FROM sales
GROUP BY order_id
HAVING COUNT(*) > 1;
```

Variant sample result:

order_id	duplicate_count
SO-100451	2
SO-100915	2

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 2 - Find Duplicate Business Keys

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to find duplicate business keys.

When to use it: Use this when you suspect repeated keys or transactions and need to prove whether duplicated records are inflating totals or breaking joins. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, COUNT(*) AS record_count
FROM products
GROUP BY product_id
HAVING COUNT(*) > 1;
```

Sample result:

product_id	record_count
P-10045	2
P-33012	3

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before joining a master table to a transaction table, a data analyst checks whether the supposedly unique key appears more than once. Example request: 'Can you check the source data and apply find duplicate business keys before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, COUNT(*) AS record_count
FROM customers
GROUP BY customer_id
HAVING COUNT(*) > 1;
```

Variant sample result:

customer_id	record_count
C1045	2
C1188	2

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 3 - Remove Leading and Trailing Spaces

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to remove leading and trailing spaces.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, TRIM(product_id) AS product_id_clean
FROM products;
```

Sample result:

product_id	product_id_clean
P-10045	P-10045
P-22010	P-22010

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: an analyst needs to compare each month, order, or status change with the previous or next record. Example request: 'Can you check the source data and apply remove leading and trailing spaces before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_name, TRIM(customer_name) AS customer_name_clean
FROM customers;
```

Variant sample result:

customer_name	customer_name_clean
ACME Italy Srl	ACME Italy Srl

customer_name	customer_name_clean
Nordic Retail AB	Nordic Retail AB

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 4 - Standardize Text to Uppercase

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to standardize text to uppercase.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT country, UPPER(TRIM(country)) AS country_clean
FROM customers;
```

Sample result:

country	country_clean
italy	ITALY
Germany	GERMANY

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you check the source data and apply standardize text to uppercase before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, UPPER(TRIM(product_id)) AS product_id_clean
FROM products;
```

Variant sample result:

product_id	product_id_clean
p-10045	P-10045
p-22010	P-22010

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 5 - Standardize Text to Lowercase

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to standardize text to lowercase.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT email, LOWER(TRIM(email)) AS email_clean
FROM customers;
```

Sample result:

email	email_clean
John.Smith@ACME.COM	john.smith@acme.com
INFO@NORDICRETAIL.SE	info@nordicretail.se

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you check the source data and apply standardize text to lowercase before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT LOWER(TRIM(email)) AS email_clean, COUNT(*) AS record_count
FROM customers
GROUP BY LOWER(TRIM(email))
HAVING COUNT(*) > 1;
```

Variant sample result:

email_clean	record_count
john.smith@acme.com	2
finance@globex.com	3

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 6 - Replace Blank Strings with NULL

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to replace blank strings with null.

When to use it: Use this when a report, upload, join, or calculation depends on mandatory fields being populated and you need to identify incomplete records quickly. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, NULLIF(TRIM(customer_name), '') AS customer_name_clean
FROM customers;
```

Sample result:

customer_id	customer_name_clean
C1001	ACME Italy Srl
C1099	NULL

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before refreshing a Power BI dataset, the reporting owner checks which records are missing required fields that would create blank visuals or failed joins. Example request: 'Can you check the source data and apply replace blank strings with null before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, NULLIF(TRIM(category), '') AS category_clean
FROM products;
```

Variant sample result:

product_id	category_clean
P-10045	Sensors
P-99999	NULL

Common mistakes to avoid: Do not assume blanks and NULLs are the same. Check both conditions when validating imported data.

QUERY N. 7 - Replace NULL Values with Default Values

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to replace null values with default values.

When to use it: Use this when a report, upload, join, or calculation depends on mandatory fields being populated and you need to identify incomplete records quickly. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, product_name, COALESCE(unit_cost, 0) AS unit_cost_clean
FROM products;
```

Sample result:

product_id	product_name	unit_cost_clean
P-10045	Photoelectric Sensor	42.50
P-88910	Legacy Cable	0.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before refreshing a Power BI dataset, the reporting owner checks which records are missing required fields that would create blank visuals or failed joins. Example request: 'Can you check the source data and apply replace null values with default values before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, COALESCE(country, 'Unknown') AS country_clean
FROM customers;
```

Variant sample result:

customer_id	country_clean
C1001	Italy
C1880	Unknown

Common mistakes to avoid: Do not assume blanks and NULLs are the same. Check both conditions when validating imported data.

QUERY N. 8 - Find Empty Mandatory Fields

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to find empty mandatory fields.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, customer_name, country
FROM customers

WHERE customer_id IS NULL OR customer_name IS NULL OR TRIM(customer_name) = '' OR country IS NULL OR
TRIM(country) = '';
```

Sample result:

customer_id	customer_name	country
C1001	ACME Italy Srl	Italy
C1022		Germany

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you check the source data and apply find empty mandatory fields before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, product_name, category
```

```
FROM products

WHERE product_id IS NULL OR product_name IS NULL OR TRIM(product_name) = '' OR category IS NULL OR TRIM(category) = '';
```

Variant sample result:

product_id	product_name	category
P-10045	Photoelectric Sensor	Sensors
P-77881		Relays

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 9 - Find Invalid Email Formats

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to find invalid email formats.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, email
FROM customers
WHERE email IS NULL OR email NOT LIKE '%@%' OR email NOT LIKE '%.%.%';
```

Sample result:

customer_id	email
C1009	john.email.com
C1045	purchasing@

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you check the source data and apply find invalid email formats before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, email
FROM customers
WHERE email IS NOT NULL AND TRIM(email) <> ''
AND (email NOT LIKE '%@%' OR email NOT LIKE '%.%.%');
```

Variant sample result:

customer_id	email
C1105	sales globex.com
C1120	support@

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 10 - Find Invalid Date Values

Difficulty: Beginner

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to find invalid date values.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_id, order_date
FROM orders
WHERE order_date IS NULL OR order_date > CURRENT_DATE OR order_date < DATE '2000-01-01';
```

Sample result:

order_id	order_date
SO-100451	2099-12-31
SO-100991	1995-04-12

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you check the source data and apply find invalid date values before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT order_id, shipment_date
FROM orders
WHERE shipment_date > CURRENT_DATE;
```

Variant sample result:

order_id	shipment_date
SO-100785	2030-01-15
SO-100812	2029-11-02

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 11 - Convert Text Values to Dates

Difficulty: Intermediate

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to convert text values to dates.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_id, order_date_text, CAST(order_date_text AS DATE) AS order_date
FROM orders;
```

Sample result:

order_id	order_date_text	order_date
SO-100451	2026-01-31	2026-01-31
SO-100452	2026-02-14	2026-02-14

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you check the source data and apply convert text values to dates before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT order_id, order_date_text, TRY_CAST(order_date_text AS DATE) AS order_date
FROM orders;
```

Variant sample result:

order_id	order_date_text	order_date
SO-100711	31/02/2026	NULL
SO-100712	2026-03-05	2026-03-05

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 12 - Convert Text Values to Numbers

Difficulty: Intermediate

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to convert text values to numbers.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, price_text, CAST(TRIM(price_text) AS DECIMAL(18,2)) AS price_number
FROM prices;
```

Sample result:

product_id	price_text	price_number
P-10045	125.50	125.50
P-22010	89.90	89.90

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you check the source data and apply convert text values to numbers before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, price_text, CAST(REPLACE(TRIM(price_text), ',', '' ) AS DECIMAL(18,2)) AS price_number
FROM prices;
```

Variant sample result:

product_id	price_text	price_number
P-10045	1,250.50	1250.50
P-22010	980.00	980.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 13 - Find Abnormal Numeric Values

Difficulty: Intermediate

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to find abnormal numeric values.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_id, customer_id, sales_amount
FROM sales
WHERE sales_amount > 100000 OR sales_amount < 0;
```

Sample result:

order_id	customer_id	sales_amount
SO-100880	C1001	250,000.00
SO-100915	C1045	-850.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you check the source data and apply find abnormal numeric values before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT order_id, product_id, quantity
FROM order_lines
WHERE quantity > 10000 OR quantity < 0;
```

Variant sample result:

order_id	product_id	quantity
SO-100880	P-10045	15,000
SO-100915	P-22010	-4

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 14 - Find Negative Amounts

Difficulty: Intermediate

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to find negative amounts.

When to use it: Use this before loading or joining data, especially after CSV/Excel imports, ERP exports, manual uploads, or table refreshes where inconsistent values can break reports. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_id, customer_id, sales_amount
FROM sales
WHERE sales_amount < 0;
```

Sample result:

order_id	customer_id	sales_amount
SO-100915	C1045	-850.00
SO-100944	C1188	-120.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you check the source data and apply find negative amounts before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, unit_price
FROM prices
WHERE unit_price < 0;
```

Variant sample result:

product_id	unit_price
P-10045	-12.50
P-99110	-1.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 15 - Find Records with Missing Mandatory Fields

Difficulty: Intermediate

Best for: Data Analyst, BI Developer, Excel Power User, Data Manager

Problem: You need to clean or validate a source table before it feeds joins, totals, uploads, or reports. This query helps reveal issues related to find records with missing mandatory fields.

When to use it: Use this when a report, upload, join, or calculation depends on mandatory fields being populated and you need to identify incomplete records quickly. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_id, order_date, customer_id, product_id, quantity, sales_amount
FROM sales

WHERE order_id IS NULL OR order_date IS NULL OR customer_id IS NULL OR product_id IS NULL OR quantity IS NULL OR
sales_amount IS NULL;
```

Sample result:

order_id	order_date	customer_id	product_id	quantity	sales_amount
SO-100701	2026-02-01	NULL	P-10045	12	268410.00
SO-100715	NULL	C1045	P-22010	4	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before refreshing a Power BI dataset, the reporting owner checks which records are missing required fields that would create blank visuals or failed joins. Example request: 'Can you check the source data and apply find records with missing mandatory fields before we refresh the report?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT order_id, CASE
WHEN order_date IS NULL THEN 'Missing order date'
WHEN customer_id IS NULL THEN 'Missing customer ID'
WHEN product_id IS NULL THEN 'Missing product ID'
ELSE 'Other missing field' END AS data_quality_issue
FROM sales

WHERE order_date IS NULL OR customer_id IS NULL OR product_id IS NULL;
```

Variant sample result:

order_id	data_quality_issue
SO-100701	Missing customer ID
SO-100715	Missing order date

Common mistakes to avoid: Do not assume blanks and NULLs are the same. Check both conditions when validating imported data.

7. Sales Analysis Queries

Use this when Sales, Commercial, or Management asks for a quick revenue view, performance ranking, monthly trend, or customer/product sales breakdown.

QUERY N. 16 - Total Sales by Month

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce total sales by month for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT DATE_TRUNC('month', order_date) AS sales_month, SUM(sales_amount) AS total_sales
FROM sales
GROUP BY DATE_TRUNC('month', order_date)
ORDER BY sales_month;
```

Sample result:

sales_month	total_sales
2026-01	245,880.00
2026-02	268,410.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a business analyst receives a request to show the latest monthly trend in a management meeting or recurring sales report. Example request: 'Can you prepare total sales by month for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT DATE_TRUNC('month', order_date) AS sales_month, COUNT(DISTINCT order_id) AS order_count, SUM(sales_amount)
AS total_sales
FROM sales
GROUP BY DATE_TRUNC('month', order_date)
ORDER BY sales_month;
```

Variant sample result:

sales_month	order_count	total_sales
2026-01	420	245,880.00
2026-02	455	268,410.00

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 17 - Total Sales by Customer

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce total sales by customer for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when Sales, Commercial, or Management asks for a quick revenue view, performance ranking, monthly trend, or customer/product sales breakdown. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, SUM(sales_amount) AS total_sales
FROM sales
GROUP BY customer_id
ORDER BY total_sales DESC;
```

Sample result:

customer_id	total_sales
C1001	182,450.00
C1045	145,220.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare total sales by customer for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT s.customer_id, c.customer_name, SUM(s.sales_amount) AS total_sales
FROM sales s LEFT JOIN customers c ON s.customer_id = c.customer_id
GROUP BY s.customer_id, c.customer_name
ORDER BY total_sales DESC;
```

Variant sample result:

customer_id	customer_name	total_sales
C1001	ACME Italy Srl	182,450.00
C1045	Nordic Retail AB	145,220.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 18 - Total Sales by Product

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce total sales by product for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when Sales, Commercial, or Management asks for a quick revenue view, performance ranking, monthly trend, or customer/product sales breakdown. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, SUM(sales_amount) AS total_sales
FROM sales
GROUP BY product_id
ORDER BY total_sales DESC
;
```

Sample result:

product_id	total_sales
P-10045	96,500.00
P-22010	82,350.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare total sales by product for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT s.product_id, p.product_name, p.category, SUM(s.sales_amount) AS total_sales
FROM sales s LEFT JOIN products p ON s.product_id = p.product_id
GROUP BY s.product_id, p.product_name, p.category
ORDER BY total_sales DESC;
```

Variant sample result:

product_id	product_name	category	total_sales
P-10045	Photoelectric Sensor	Sensors	96,500.00
P-22010	Compact Relay	Relays	82,350.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 19 - Top 10 Products by Revenue

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce top 10 products by revenue for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when stakeholders ask for the highest-performing customers, products, categories, or business units in a clear ranked list. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```

SELECT product_id, SUM(sales_amount) AS total_sales
FROM sales
GROUP BY product_id
ORDER BY total_sales DESC
FETCH FIRST 10 ROWS ONLY;

```

Sample result:

product_id	total_sales
P-10045	96,500.00
P-22010	82,350.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: the sales team asks which products generated the highest revenue in the last closed period. Example request: 'Can you prepare top 10 products by revenue for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```

SELECT s.product_id, p.product_name, p.category, SUM(s.sales_amount) AS total_sales
FROM sales s LEFT JOIN products p ON s.product_id = p.product_id
GROUP BY s.product_id, p.product_name, p.category
ORDER BY total_sales DESC;

```

Variant sample result:

product_id	product_name	category	total_sales
P-10045	Photoelectric Sensor	Sensors	96,500.00
P-22010	Compact Relay	Relays	82,350.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 20 - Top 10 Customers by Revenue

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce top 10 customers by revenue for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when stakeholders ask for the highest-performing customers, products, categories, or business units in a clear ranked list. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```

SELECT * FROM sales;

```

Sample result:

result
Example A
Example B

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: account managers want a list of the most valuable customers before a quarterly business review. Example request: 'Can you prepare top 10 customers by revenue for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT * FROM sales WHERE order_date >= DATE '2026-01-01';
```

Variant sample result:

result
Example A
Example B

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 21 - Sales by Country or Region

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce sales by country or region for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when Sales, Commercial, or Management asks for a quick revenue view, performance ranking, monthly trend, or customer/product sales breakdown. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT c.country, c.region, SUM(s.sales_amount) AS total_sales
FROM sales s LEFT JOIN customers c ON s.customer_id = c.customer_id
GROUP BY c.country, c.region
ORDER BY total_sales DESC;
```

Sample result:

country	region	total_sales
Italy	South Europe	245,880.00
Germany	Central Europe	219,400.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare sales by country or region for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT COALESCE(c.region, 'Unknown') AS region, SUM(s.sales_amount) AS total_sales
FROM sales s LEFT JOIN customers c ON s.customer_id = c.customer_id
GROUP BY COALESCE(c.region, 'Unknown')
ORDER BY total_sales DESC;
```

Variant sample result:

region	total_sales
South Europe	510,300.00
Unknown	8,900.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 22 - Monthly Sales Trend

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce monthly sales trend for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT DATE_TRUNC('month', order_date) AS sales_month, SUM(sales_amount) AS total_sales
FROM sales
GROUP BY DATE_TRUNC('month', order_date)
ORDER BY sales_month;
```

Sample result:

sales_month	total_sales
2026-01	245,880.00
2026-02	268,410.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a business analyst receives a request to show the latest monthly trend in a management meeting or recurring sales report. Example request: 'Can you prepare monthly sales trend for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT DATE_TRUNC('month', order_date) AS sales_month, COUNT(DISTINCT order_id) AS order_count, SUM(sales_amount)
AS total_sales
FROM sales
GROUP BY DATE_TRUNC('month', order_date)
ORDER BY sales_month;
```

Variant sample result:

sales_month	order_count	total_sales
2026-01	420	245,880.00
2026-02	455	268,410.00

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 23 - Year-over-Year Sales Comparison

Difficulty: Advanced Practical

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce year-over-year sales comparison for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH yearly_sales AS (
    SELECT EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales
    FROM sales GROUP BY EXTRACT(YEAR FROM order_date)
)
SELECT sales_year, total_sales, LAG(total_sales) OVER (ORDER BY sales_year) AS previous_year_sales
FROM yearly_sales;
```

Sample result:

sales_year	total_sales	previous_year_sales
2025	2,850,000.00	2,540,000.00
2026	3,120,000.00	2,850,000.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: management wants to compare current year performance with last year using the same business logic. Example request: 'Can you prepare year-over-year sales comparison for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH yearly_sales AS (
    SELECT EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales
    FROM sales GROUP BY EXTRACT(YEAR FROM order_date)
)
SELECT sales_year, total_sales, ROUND((total_sales - LAG(total_sales) OVER (ORDER BY sales_year)) /
    NULLIF(LAG(total_sales) OVER (ORDER BY sales_year),0) * 100, 2) AS growth_pct
FROM yearly_sales;
```

Variant sample result:

sales_year	total_sales	growth_pct
2025	2,850,000.00	12.20
2026	3,120,000.00	9.47

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 24 - Percentage Sales Growth

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce percentage sales growth for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when Sales, Commercial, or Management asks for a quick revenue view, performance ranking, monthly trend, or customer/product sales breakdown. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH yearly_sales AS (
  SELECT EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales
  FROM sales GROUP BY EXTRACT(YEAR FROM order_date)
)
SELECT sales_year, total_sales, LAG(total_sales) OVER (ORDER BY sales_year) AS previous_year_sales
FROM yearly_sales;
```

Sample result:

sales_year	total_sales	previous_year_sales
2025	2,850,000.00	2,540,000.00
2026	3,120,000.00	2,850,000.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare percentage sales growth for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH yearly_sales AS (
  SELECT EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales
  FROM sales GROUP BY EXTRACT(YEAR FROM order_date)
)
SELECT sales_year, total_sales, ROUND((total_sales - LAG(total_sales) OVER (ORDER BY sales_year)) /
  NULLIF(LAG(total_sales) OVER (ORDER BY sales_year),0) * 100, 2) AS growth_pct
FROM yearly_sales;
```

Variant sample result:

sales_year	total_sales	growth_pct
2025	2,850,000.00	12.20
2026	3,120,000.00	9.47

Common mistakes to avoid: Use NULLIF in divisions to avoid divide-by-zero errors.

QUERY N. 25 - Average Order Value

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce average order value for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when Sales, Commercial, or Management asks for a quick revenue view, performance ranking, monthly trend, or customer/product sales breakdown. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT COUNT(DISTINCT order_id) AS order_count, SUM(sales_amount) AS total_sales, SUM(sales_amount) /
NULLIF(COUNT(DISTINCT order_id),0) AS average_order_value
FROM sales;
```

Sample result:

order_count	total_sales	average_order_value
1,245	3,120,000.00	2,506.02
1,245	3,120,000.00	2,506.02

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare average order value for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, COUNT(DISTINCT order_id) AS order_count, SUM(sales_amount) AS total_sales, SUM(sales_amount)
/ NULLIF(COUNT(DISTINCT order_id),0) AS average_order_value
FROM sales
GROUP BY customer_id
ORDER BY average_order_value DESC;
```

Variant sample result:

customer_id	order_count	total_sales	average_order_value
C1001	42	182,450.00	4,344.05
C1045	36	145,220.00	4,033.89

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 26 - Products with No Sales

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce products with no sales for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when Sales, Commercial, or Management asks for a quick revenue view, performance ranking, monthly trend, or customer/product sales breakdown. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.product_id, p.product_name
FROM products p LEFT JOIN sales s ON p.product_id = s.product_id
WHERE s.product_id IS NULL;
```

Sample result:

product_id	product_name
P-77881	Legacy Switch
P-99110	Old Connector

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare products with no sales for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.product_id, p.product_name
FROM products p LEFT JOIN sales s ON p.product_id = s.product_id AND s.order_date >= DATE '2026-01-01'
WHERE s.product_id IS NULL AND p.status = 'Active';
```

Variant sample result:

product_id	product_name
P-77881	Legacy Switch
P-99110	Old Connector

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 27 - Customers with No Recent Sales

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce customers with no recent sales for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when Sales, Commercial, or Management asks for a quick revenue view, performance ranking, monthly trend, or customer/product sales breakdown. Adapt the table and field names to your database, then review the returned

columns before using the result.

SQL:

```
SELECT c.customer_id, c.customer_name, MAX(s.order_date) AS last_order_date
FROM customers c LEFT JOIN sales s ON c.customer_id = s.customer_id
GROUP BY c.customer_id, c.customer_name
HAVING MAX(s.order_date) < CURRENT_DATE - INTERVAL '90 days' OR MAX(s.order_date) IS NULL;
```

Sample result:

customer_id	customer_name	last_order_date
C1880	Westline GmbH	2025-09-12
C1915	Blue Ocean Ltd	NULL

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare customers with no recent sales for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT c.customer_id, c.customer_name, MAX(s.order_date) AS last_order_date
FROM customers c LEFT JOIN sales s ON c.customer_id = s.customer_id
WHERE c.status = 'Active'
GROUP BY c.customer_id, c.customer_name
HAVING MAX(s.order_date) < CURRENT_DATE - INTERVAL '180 days' OR MAX(s.order_date) IS NULL;
```

Variant sample result:

customer_id	customer_name	last_order_date
C1880	Westline GmbH	2025-06-20
C1915	Blue Ocean Ltd	NULL

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 28 - Sales Ranking by Customer

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce sales ranking by customer for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when stakeholders ask for the highest-performing customers, products, categories, or business units in a clear ranked list. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, SUM(sales_amount) AS total_sales, RANK() OVER (ORDER BY SUM(sales_amount) DESC) AS sales_rank
FROM sales
GROUP BY customer_id
ORDER BY sales_rank;
```

Sample result:

customer_id	total_sales	sales_rank
C1001	182,450.00	1
C1045	145,220.00	2

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare sales ranking by customer for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT region, customer_id, SUM(sales_amount) AS total_sales, RANK() OVER (PARTITION BY region ORDER BY
SUM(sales_amount) DESC) AS regional_rank

FROM sales

GROUP BY region, customer_id

ORDER BY region, regional_rank;
```

Variant sample result:

region	customer_id	total_sales	regional_rank
South Europe	C1001	182,450.00	1
Nordics	C1045	145,220.00	1

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 29 - Sales Ranking by Product

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce sales ranking by product for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when stakeholders ask for the highest-performing customers, products, categories, or business units in a clear ranked list. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, SUM(sales_amount) AS total_sales, RANK() OVER (ORDER BY SUM(sales_amount) DESC) AS sales_rank

FROM sales

GROUP BY product_id

ORDER BY sales_rank;
```

Sample result:

product_id	total_sales	sales_rank
P-10045	96,500.00	1
P-22010	82,350.00	2

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare sales ranking by product for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT category, product_id, SUM(sales_amount) AS total_sales, RANK() OVER (PARTITION BY category ORDER BY
SUM(sales_amount) DESC) AS category_rank

FROM sales

GROUP BY category, product_id

ORDER BY category, category_rank;
```

Variant sample result:

category	product_id	total_sales	category_rank
Sensors	P-10045	96,500.00	1
Relays	P-22010	82,350.00	1

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 30 - Sales Contribution Percentage by Product

Difficulty: Intermediate

Best for: Sales Analyst, Business Analyst, Power BI User, Commercial Controller

Problem: You need a fast business query to produce sales contribution percentage by product for a sales report, dashboard, management request, or commercial analysis.

When to use it: Use this when Sales, Commercial, or Management asks for a quick revenue view, performance ranking, monthly trend, or customer/product sales breakdown. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, SUM(sales_amount) AS product_sales, ROUND(SUM(sales_amount) / NULLIF(SUM(SUM(sales_amount))
OVER (),0) * 100, 2) AS contribution_pct

FROM sales

GROUP BY product_id

ORDER BY contribution_pct DESC;
```

Sample result:

product_id	product_sales	contribution_pct
P-10045	96,500.00	12.85
P-22010	82,350.00	10.96

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a sales analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you prepare sales contribution percentage by product for the sales review using the latest closed period?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT category, product_id, SUM(sales_amount) AS product_sales, ROUND(SUM(sales_amount) /  
NULLIF(SUM(SUM(sales_amount)) OVER (PARTITION BY category),0) * 100, 2) AS category_contribution_pct  
FROM sales  
GROUP BY category, product_id  
ORDER BY category, category_contribution_pct DESC;
```

Variant sample result:

category	product_id	product_sales	category_contribution_pct
Sensors	P-10045	96,500.00	34.20
Relays	P-22010	82,350.00	41.10

Common mistakes to avoid: Use NULLIF in divisions to avoid divide-by-zero errors.

8. Finance and Reporting Queries

Use this when preparing month-end reporting, management packs, financial dashboards, budget checks, margin analysis, or recurring finance extracts.

QUERY N. 31 - Monthly Revenue

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for monthly revenue that can support month-end reporting, controlling, or budget review.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT DATE_TRUNC('month', posting_date) AS report_group, SUM(revenue) AS total_revenue  
FROM actuals  
GROUP BY DATE_TRUNC('month', posting_date)  
ORDER BY total_revenue DESC;
```

Sample result:

report_group	total_revenue
2026-02-01	268410.00
2026-03-01	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a business analyst receives a request to show the latest monthly trend in a management meeting or recurring sales report. Example request: 'Can you build a report-ready extract for monthly revenue for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_category, DATE_TRUNC('month', posting_date) AS period_month, SUM(revenue) AS total_revenue
FROM actuals
GROUP BY product_category, DATE_TRUNC('month', posting_date)
ORDER BY product_category, period_month;
```

Variant sample result:

product_category	period_month	total_revenue
Example A	2026-02-01	268410.00
Example B	2026-03-01	302950.00

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 32 - Monthly Cost

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for monthly cost that can support month-end reporting, controlling, or budget review.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT DATE_TRUNC('month', posting_date) AS report_group, SUM(cost_amount) AS total_cost
FROM actuals
GROUP BY DATE_TRUNC('month', posting_date)
ORDER BY total_cost DESC;
```

Sample result:

report_group	total_cost
2026-02-01	268410.00
2026-03-01	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a business analyst receives a request to show the latest monthly trend in a management meeting or recurring sales report. Example request: 'Can you build a report-ready extract for monthly cost for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.


```

SELECT product_category, DATE_TRUNC('month', posting_date) AS period_month, SUM(cost_amount) AS total_cost
FROM actuals
GROUP BY product_category, DATE_TRUNC('month', posting_date)
ORDER BY product_category, period_month;

```

Variant sample result:

product_category	period_month	total_cost
Example A	2026-02-01	268410.00
Example B	2026-03-01	302950.00

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 33 - Monthly Gross Margin

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for monthly gross margin that can support month-end reporting, controlling, or budget review.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```

SELECT DATE_TRUNC('month', posting_date) AS report_group, SUM((revenue - cost_amount)) AS profit
FROM actuals
GROUP BY DATE_TRUNC('month', posting_date)
ORDER BY profit DESC;

```

Sample result:

report_group	profit
2026-02-01	268410.00
2026-03-01	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a business analyst receives a request to show the latest monthly trend in a management meeting or recurring sales report. Example request: 'Can you build a report-ready extract for monthly gross margin for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```

SELECT product_category, DATE_TRUNC('month', posting_date) AS period_month, SUM((revenue - cost_amount)) AS profit
FROM actuals
GROUP BY product_category, DATE_TRUNC('month', posting_date)
ORDER BY product_category, period_month;

```

Variant sample result:

product_category	period_month	profit
Sensors	2026-02-01	39,300.00
Relays	2026-03-01	28,250.00

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 34 - Gross Margin Percentage

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for gross margin percentage that can support month-end reporting, controlling, or budget review.

When to use it: Use this when Finance or Management needs a report-ready comparison of actual performance, budget, cost, revenue, margin, or variance. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT DATE_TRUNC('month', posting_date) AS margin_month, SUM(revenue) AS total_revenue, SUM(cost_amount) AS
total_cost, ROUND((SUM(revenue)-SUM(cost_amount)) / NULLIF(SUM(revenue),0) * 100,2) AS gross_margin_pct
FROM actuals
GROUP BY DATE_TRUNC('month', posting_date)
ORDER BY margin_month;
```

Sample result:

margin_month	total_revenue	total_cost	gross_margin_pct
2026-01	245,880.00	158,900.00	35.38
2026-02	268,410.00	171,500.00	36.10

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a controller needs to verify whether revenue and cost data produce a reasonable gross margin before publishing the report. Example request: 'Can you build a report-ready extract for gross margin percentage for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_category, SUM(revenue) AS total_revenue, SUM(cost_amount) AS total_cost,
ROUND((SUM(revenue)-SUM(cost_amount)) / NULLIF(SUM(revenue),0) * 100,2) AS gross_margin_pct
FROM actuals
GROUP BY product_category
ORDER BY gross_margin_pct DESC;
```

Variant sample result:

product_category	total_revenue	total_cost	gross_margin_pct
Sensors	96,500.00	57,200.00	40.73
Relays	82,350.00	54,100.00	34.31

Common mistakes to avoid: Use NULLIF in divisions to avoid divide-by-zero errors.

QUERY N. 35 - Budget vs Actual Comparison

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for budget vs actual comparison that can support month-end reporting, controlling, or budget review.

When to use it: Use this when Finance or Management needs a report-ready comparison of actual performance, budget, cost, revenue, margin, or variance. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT a.period, a.business_unit, SUM(a.actual_amount) AS actual_amount, SUM(b.budget_amount) AS budget_amount,
SUM(a.actual_amount) - SUM(b.budget_amount) AS variance_amount
FROM actuals a LEFT JOIN budget b ON a.period = b.period AND a.business_unit = b.business_unit
GROUP BY a.period, a.business_unit
ORDER BY a.period, a.business_unit;
```

Sample result:

period	business_unit	actual_amount	budget_amount	variance_amount
2026-01	Industrial	260,100.00	250,000.00	10,100.00
2026-01	Components	150,400.00	160,000.00	-9,600.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Finance asks for a quick actual-vs-budget check to explain where the monthly deviation is coming from. Example request: 'Can you build a report-ready extract for budget vs actual comparison for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT a.period, a.business_unit, ROUND((SUM(a.actual_amount) - SUM(b.budget_amount)) /
NULLIF(SUM(b.budget_amount),0) * 100, 2) AS variance_pct
FROM actuals a LEFT JOIN budget b ON a.period = b.period AND a.business_unit = b.business_unit
GROUP BY a.period, a.business_unit;
```

Variant sample result:

period	business_unit	variance_pct
2026-01	Industrial	4.04
2026-01	Components	-6.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 36 - Budget Variance Percentage

Difficulty: Advanced Practical

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for budget variance percentage that can support month-end reporting, controlling, or budget review.

When to use it: Use this when Finance or Management needs a report-ready comparison of actual performance, budget, cost, revenue, margin, or variance. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT a.period, a.business_unit, SUM(a.actual_amount) AS actual_amount, SUM(b.budget_amount) AS budget_amount,
SUM(a.actual_amount) - SUM(b.budget_amount) AS variance_amount

FROM actuals a LEFT JOIN budget b ON a.period = b.period AND a.business_unit = b.business_unit

GROUP BY a.period, a.business_unit

ORDER BY a.period, a.business_unit;
```

Sample result:

period	business_unit	actual_amount	budget_amount	variance_amount
2026-01	Industrial	260,100.00	250,000.00	10,100.00
2026-01	Components	150,400.00	160,000.00	-9,600.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Finance asks for a quick actual-vs-budget check to explain where the monthly deviation is coming from. Example request: 'Can you build a report-ready extract for budget variance percentage for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT a.period, a.business_unit, ROUND((SUM(a.actual_amount) - SUM(b.budget_amount)) /
NULLIF(SUM(b.budget_amount),0) * 100, 2) AS variance_pct

FROM actuals a LEFT JOIN budget b ON a.period = b.period AND a.business_unit = b.business_unit

GROUP BY a.period, a.business_unit;
```

Variant sample result:

period	business_unit	variance_pct
2026-01	Industrial	4.04
2026-01	Components	-6.00

Common mistakes to avoid: Use NULLIF in divisions to avoid divide-by-zero errors.

QUERY N. 37 - Revenue by Category

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for revenue by category that can support month-end reporting, controlling, or budget review.

When to use it: Use this when preparing month-end reporting, management packs, financial dashboards, budget checks, margin analysis, or recurring finance extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_category AS report_group, SUM(revenue) AS total_revenue

FROM actuals

GROUP BY product_category

ORDER BY total_revenue DESC;
```

Sample result:

report_group	total_revenue
2026-02-01	268410.00
2026-03-01	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a finance analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you build a report-ready extract for revenue by category for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_category, DATE_TRUNC('month', posting_date) AS period_month, SUM(revenue) AS total_revenue
FROM actuals
GROUP BY product_category, DATE_TRUNC('month', posting_date)
ORDER BY product_category, period_month;
```

Variant sample result:

product_category	period_month	total_revenue
Example A	2026-02-01	268410.00
Example B	2026-03-01	302950.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 38 - Cost by Category

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for cost by category that can support month-end reporting, controlling, or budget review.

When to use it: Use this when preparing month-end reporting, management packs, financial dashboards, budget checks, margin analysis, or recurring finance extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_category AS report_group, SUM(cost_amount) AS total_cost
FROM actuals
GROUP BY product_category
ORDER BY total_cost DESC;
```

Sample result:

report_group	total_cost
2026-02-01	268410.00
2026-03-01	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Product Data or Finance needs to find items with missing or suspicious commercial values before a price list update. Example request: 'Can you build a report-ready extract for cost by category for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_category, DATE_TRUNC('month', posting_date) AS period_month, SUM(cost_amount) AS total_cost
FROM actuals
GROUP BY product_category, DATE_TRUNC('month', posting_date)
ORDER BY product_category, period_month;
```

Variant sample result:

product_category	period_month	total_cost
Example A	2026-02-01	268410.00
Example B	2026-03-01	302950.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 39 - Profit by Category

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for profit by category that can support month-end reporting, controlling, or budget review.

When to use it: Use this when preparing month-end reporting, management packs, financial dashboards, budget checks, margin analysis, or recurring finance extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_category AS report_group, SUM((revenue - cost_amount)) AS profit
FROM actuals
GROUP BY product_category
ORDER BY profit DESC;
```

Sample result:

report_group	profit
2026-02-01	39,300.00
2026-03-01	28,250.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a finance analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you build a report-ready extract for profit by category for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_category, DATE_TRUNC('month', posting_date) AS period_month, SUM((revenue - cost_amount)) AS profit
FROM actuals
GROUP BY product_category, DATE_TRUNC('month', posting_date)
ORDER BY product_category, period_month;
```

Variant sample result:

product_category	period_month	profit
Sensors	2026-02	13,800.00
Relays	2026-02	9,450.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 40 - Monthly Trend by Business Unit

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for monthly trend by business unit that can support month-end reporting, controlling, or budget review.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT business_unit AS report_group, SUM((revenue - cost_amount)) AS profit
FROM actuals
GROUP BY business_unit
ORDER BY profit DESC;
```

Sample result:

report_group	profit
2026-02-01	268410.00
2026-03-01	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a business analyst receives a request to show the latest monthly trend in a management meeting or recurring sales report. Example request: 'Can you build a report-ready extract for monthly trend by business unit for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```

SELECT product_category, DATE_TRUNC('month', posting_date) AS period_month, SUM((revenue - cost_amount)) AS profit
FROM actuals
GROUP BY product_category, DATE_TRUNC('month', posting_date)
ORDER BY product_category, period_month;

```

Variant sample result:

product_category	period_month	profit
Example A	2026-02-01	268410.00
Example B	2026-03-01	302950.00

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 41 - Year-to-Date Revenue

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for year-to-date revenue that can support month-end reporting, controlling, or budget review.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```

SELECT DATE_TRUNC('month', posting_date) AS period_month, SUM(revenue) AS monthly_amount, SUM(SUM(revenue)) OVER
(ORDER BY DATE_TRUNC('month', posting_date)) AS ytd_revenue
FROM actuals
WHERE posting_date >= DATE '2026-01-01' AND posting_date < DATE '2027-01-01'
GROUP BY DATE_TRUNC('month', posting_date)
ORDER BY period_month;

```

Sample result:

period_month	monthly_amount	ytd_revenue
2026-02-01	2026-02-01	245,880.00
2026-03-01	2026-03-01	514,290.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a finance analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you build a report-ready extract for year-to-date revenue for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```

SELECT business_unit, DATE_TRUNC('month', posting_date) AS period_month, SUM(revenue) AS monthly_amount,
SUM(SUM(revenue)) OVER (PARTITION BY business_unit ORDER BY DATE_TRUNC('month', posting_date)) AS ytd_revenue
FROM actuals
WHERE posting_date >= DATE '2026-01-01' AND posting_date < DATE '2027-01-01'
GROUP BY business_unit, DATE_TRUNC('month', posting_date)

```



```
ORDER BY business_unit, period_month;
```

Variant sample result:

business_unit	period_month	monthly_amount	ytd_revenue
Industrial Automation	2026-02-01	2026-02-01	268410.00
Safety Components	2026-03-01	2026-03-01	302950.00

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 42 - Year-to-Date Margin

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for year-to-date margin that can support month-end reporting, controlling, or budget review.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT DATE_TRUNC('month', posting_date) AS period_month, SUM((revenue - cost_amount)) AS monthly_amount,
SUM(SUM((revenue - cost_amount))) OVER (ORDER BY DATE_TRUNC('month', posting_date)) AS ytd_margin
FROM actuals
WHERE posting_date >= DATE '2026-01-01' AND posting_date < DATE '2027-01-01'
GROUP BY DATE_TRUNC('month', posting_date)
ORDER BY period_month;
```

Sample result:

period_month	monthly_amount	ytd_margin
2026-02-01	2026-02-01	104210.00
2026-03-01	2026-03-01	122740.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a controller needs to verify whether revenue and cost data produce a reasonable gross margin before publishing the report. Example request: 'Can you build a report-ready extract for year-to-date margin for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT business_unit, DATE_TRUNC('month', posting_date) AS period_month, SUM((revenue - cost_amount)) AS
monthly_amount, SUM(SUM((revenue - cost_amount))) OVER (PARTITION BY business_unit ORDER BY DATE_TRUNC('month',
posting_date)) AS ytd_margin
FROM actuals
WHERE posting_date >= DATE '2026-01-01' AND posting_date < DATE '2027-01-01'
GROUP BY business_unit, DATE_TRUNC('month', posting_date)
ORDER BY business_unit, period_month;
```

Variant sample result:

business_unit	period_month	monthly_amount	ytd_margin
Industrial Automation	2026-02-01	2026-02-01	86,980.00
Safety Components	2026-03-01	2026-03-01	183,890.00

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 43 - Rolling 12 Months Revenue

Difficulty: Advanced Practical

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for rolling 12 months revenue that can support month-end reporting, controlling, or budget review.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT DATE_TRUNC('month', order_date) AS sales_month, SUM(sales_amount) AS monthly_sales, SUM(SUM(sales_amount))
OVER (ORDER BY DATE_TRUNC('month', order_date) ROWS BETWEEN 11 PRECEDING AND CURRENT ROW) AS
rolling_12_month_sales

FROM sales

GROUP BY DATE_TRUNC('month', order_date)

ORDER BY sales_month;
```

Sample result:

sales_month	monthly_sales	rolling_12_month_sales
2026-02-01	2026-02-01	2026-02-01
2026-03-01	2026-03-01	2026-03-01

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a finance analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you build a report-ready extract for rolling 12 months revenue for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_category, DATE_TRUNC('month', order_date) AS sales_month, SUM(sales_amount) AS monthly_sales,
SUM(SUM(sales_amount)) OVER (PARTITION BY product_category ORDER BY DATE_TRUNC('month', order_date) ROWS BETWEEN
11 PRECEDING AND CURRENT ROW) AS rolling_12_month_sales

FROM sales

GROUP BY product_category, DATE_TRUNC('month', order_date)

ORDER BY product_category, sales_month;
```

Variant sample result:

product_category	sales_month	monthly_sales	rolling_12_month_sales
Example A	2026-02-01	2026-02-01	2026-02-01
Example B	2026-03-01	2026-03-01	2026-03-01

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 44 - Month-over-Month Variance

Difficulty: Advanced Practical

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for month-over-month variance that can support month-end reporting, controlling, or budget review.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH monthly_sales AS (SELECT DATE_TRUNC('month', order_date) AS sales_month, SUM(sales_amount) AS total_sales
FROM sales GROUP BY DATE_TRUNC('month', order_date))

SELECT sales_month, total_sales, LAG(total_sales) OVER (ORDER BY sales_month) AS previous_month_sales,
total_sales - LAG(total_sales) OVER (ORDER BY sales_month) AS difference

FROM monthly_sales;
```

Sample result:

sales_month	total_sales	previous_month_sales	difference
2026-02-01	268410.00	2026-02-01	22530.00
2026-03-01	302950.00	2026-03-01	34540.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Finance asks for a quick actual-vs-budget check to explain where the monthly deviation is coming from. Example request: 'Can you build a report-ready extract for month-over-month variance for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH monthly_sales AS (SELECT DATE_TRUNC('month', order_date) AS sales_month, SUM(sales_amount) AS total_sales
FROM sales GROUP BY DATE_TRUNC('month', order_date))

SELECT sales_month, total_sales, ROUND((total_sales - LAG(total_sales) OVER (ORDER BY sales_month)) /
NULLIF(LAG(total_sales) OVER (ORDER BY sales_month),0) * 100, 2) AS difference_pct

FROM monthly_sales;
```

Variant sample result:

sales_month	total_sales	difference_pct
2026-02-01	268410.00	12.45
2026-03-01	302950.00	8.72

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 45 - Report-Ready Summary Table

Difficulty: Intermediate

Best for: Finance Analyst, Controller, Reporting Specialist, Data Manager

Problem: You need a repeatable SQL pattern for report-ready summary table that can support month-end reporting, controlling, or budget review.

When to use it: Use this when preparing month-end reporting, management packs, financial dashboards, budget checks, margin analysis, or recurring finance extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT DATE_TRUNC('month', order_date) AS report_month, region, product_category, COUNT(DISTINCT order_id) AS
order_count, SUM(sales_amount) AS total_sales, SUM(cost_amount) AS total_cost, SUM(sales_amount)-SUM(cost_amount)
AS gross_margin

FROM sales

GROUP BY DATE_TRUNC('month', order_date), region, product_category;
```

Sample result:

report_month	region	product_category	order_count	total_sales	total_cost	gross_margin
2026-02-01	South Europe	Example A	42	268410.00	268410.00	104210.00
2026-03-01	DACH	Example B	36	302950.00	302950.00	122740.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a finance analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you build a report-ready extract for report-ready summary table for the month-end pack?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT DATE_TRUNC('month', order_date) AS report_month, region, product_category, SUM(sales_amount) AS
total_sales, ROUND((SUM(sales_amount)-SUM(cost_amount))/NULLIF(SUM(sales_amount),0)*100,2) AS margin_pct

FROM sales

GROUP BY DATE_TRUNC('month', order_date), region, product_category;
```

Variant sample result:

report_month	region	product_category	total_sales	margin_pct
2026-02-01	South Europe	Example A	268410.00	12.45
2026-03-01	DACH	Example B	302950.00	8.72

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

9. Inventory and Product Data Queries

Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts.

QUERY N. 46 - Available Stock by Product

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that available stock by product can be reviewed before using it in operational reports.

When to use it: Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, SUM(stock_quantity) AS available_stock
FROM inventory
GROUP BY product_id
ORDER BY product_id;
```

Sample result:

product_id	available_stock
P-10045	245
P-22010	0

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Supply Chain asks for a stock exception list to review items that are unavailable, overstocked, or below safety level. Example request: 'Can you run available stock by product before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT warehouse_id, product_id, SUM(stock_quantity) AS available_stock
FROM inventory
GROUP BY warehouse_id, product_id
ORDER BY warehouse_id, product_id;
```

Variant sample result:

warehouse_id	product_id	available_stock
ID-1001	P-10045	245
ID-1002	P-22010	0

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 47 - Products with Zero Stock

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that products with zero stock can be reviewed before using it in operational reports.

When to use it: Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, SUM(stock_quantity) AS available_stock
FROM inventory
```

```
GROUP BY product_id
HAVING SUM(stock_quantity) = 0
ORDER BY product_id;
```

Sample result:

product_id	available_stock
P-77881	245
P-33012	0

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Supply Chain asks for a stock exception list to review items that are unavailable, overstocked, or below safety level. Example request: 'Can you run products with zero stock before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT warehouse_id, product_id, SUM(stock_quantity) AS available_stock
FROM inventory
GROUP BY warehouse_id, product_id
ORDER BY warehouse_id, product_id;
```

Variant sample result:

warehouse_id	product_id	available_stock
ID-1001	P-33012	245
ID-1002	P-44501	0

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 48 - Products Below Safety Stock

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that products below safety stock can be reviewed before using it in operational reports.

When to use it: Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.product_id, p.product_name, p.safety_stock, SUM(i.stock_quantity) AS available_stock
FROM products p LEFT JOIN inventory i ON p.product_id = i.product_id
GROUP BY p.product_id, p.product_name, p.safety_stock
HAVING SUM(i.stock_quantity) < p.safety_stock;
```

Sample result:

product_id	product_name	safety_stock	available_stock
P-77881	Legacy Switch	100	245

product_id	product_name	safety_stock	available_stock
P-33012	Safety Controller	50	0

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Supply Chain asks for a stock exception list to review items that are unavailable, overstocked, or below safety level. Example request: 'Can you run products below safety stock before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.product_id, p.product_name, p.safety_stock, COALESCE(SUM(i.stock_quantity),0) AS available_stock,
p.safety_stock - COALESCE(SUM(i.stock_quantity),0) AS replenishment_gap
FROM products p LEFT JOIN inventory i ON p.product_id = i.product_id
GROUP BY p.product_id, p.product_name, p.safety_stock
HAVING COALESCE(SUM(i.stock_quantity),0) < p.safety_stock;
```

Variant sample result:

product_id	product_name	safety_stock	available_stock	replenishment_gap
P-33012	Safety Controller	25	245	Example A
P-44501	Photo Sensor	30	0	Example B

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 49 - Products Without Price

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that products without price can be reviewed before using it in operational reports.

When to use it: Use this when a report, upload, join, or calculation depends on mandatory fields being populated and you need to identify incomplete records quickly. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.product_id, p.product_name
FROM products p LEFT JOIN prices pr ON p.product_id = pr.product_id
WHERE pr.product_id IS NULL;
```

Sample result:

product_id	product_name
P-99110	Old Connector
P-88910	Legacy Cable

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before refreshing a Power BI dataset, the reporting owner checks which records are missing required fields that would create blank visuals or failed joins. Example request: 'Can you run products without price before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.product_id, p.product_name, p.category
FROM products p LEFT JOIN prices pr ON p.product_id = pr.product_id AND pr.valid_to >= CURRENT_DATE
WHERE pr.product_id IS NULL AND p.status = 'Active';
```

Variant sample result:

product_id	product_name	category
P-88910	Legacy Cable	Sensors
P-99110	Old Connector	Relays

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 50 - Products Without Cost

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that products without cost can be reviewed before using it in operational reports.

When to use it: Use this when a report, upload, join, or calculation depends on mandatory fields being populated and you need to identify incomplete records quickly. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.product_id, p.product_name
FROM products p LEFT JOIN costs c ON p.product_id = c.product_id
WHERE c.product_id IS NULL;
```

Sample result:

product_id	product_name
P-99110	Old Connector
P-88910	Legacy Cable

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before refreshing a Power BI dataset, the reporting owner checks which records are missing required fields that would create blank visuals or failed joins. Example request: 'Can you run products without cost before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.product_id, p.product_name, p.category
FROM products p LEFT JOIN costs c ON p.product_id = c.product_id AND c.cost_type = 'Standard'
WHERE c.product_id IS NULL AND p.status = 'Active';
```

Variant sample result:

product_id	product_name	category
P-88910	Legacy Cable	Sensors
P-99110	Old Connector	Relays

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 51 - Products with Price Lower than Cost

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that products with price lower than cost can be reviewed before using it in operational reports.

When to use it: Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.product_id, p.product_name, pr.unit_price, c.unit_cost
FROM products p JOIN prices pr ON p.product_id = pr.product_id JOIN costs c ON p.product_id = c.product_id
WHERE pr.unit_price < c.unit_cost;
```

Sample result:

product_id	product_name	unit_price	unit_cost
P-99110	Old Connector	NULL	8.40
P-88910	Legacy Cable	7.20	9.10

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Product Data or Finance needs to find items with missing or suspicious commercial values before a price list update. Example request: 'Can you run products with price lower than cost before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.product_id, p.product_name, pr.unit_price, c.unit_cost, pr.unit_price - c.unit_cost AS
price_cost_difference
FROM products p JOIN prices pr ON p.product_id = pr.product_id JOIN costs c ON p.product_id = c.product_id
WHERE pr.unit_price < c.unit_cost
ORDER BY price_cost_difference;
```

Variant sample result:

product_id	product_name	unit_price	unit_cost	price_cost_difference
P-88910	Legacy Cable	7.20	9.10	22530.00
P-99110	Old Connector	NULL	8.40	34540.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 52 - Active Products

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that active products can be reviewed before using it in operational reports.

When to use it: Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, product_name, category, status
FROM products
WHERE status = 'Active';
```

Sample result:

product_id	product_name	category	status
P-10045	Photoelectric Sensor	Sensors	Active
P-22010	Compact Relay	Relays	Discontinued

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a supply chain analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you run active products before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, product_name, category, status
FROM products
WHERE status = 'Active'
ORDER BY category, product_id;
```

Variant sample result:

product_id	product_name	category	status
P-10045	Photoelectric Sensor	Sensors	Active
P-22010	Compact Relay	Relays	Discontinued

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 53 - Discontinued Products

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that discontinued products can be reviewed before using it in operational reports.

When to use it: Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, product_name, category, status
FROM products
WHERE status = 'Discontinued';
```

Sample result:

product_id	product_name	category	status
P-10045	Photoelectric Sensor	Sensors	Active
P-22010	Compact Relay	Relays	Discontinued

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a supply chain analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you run discontinued products before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, product_name, category, status
FROM products
WHERE status = 'Discontinued'
ORDER BY category, product_id;
```

Variant sample result:

product_id	product_name	category	status
P-10045	Photoelectric Sensor	Sensors	Active
P-22010	Compact Relay	Relays	Discontinued

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 54 - Duplicate Product Codes

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that duplicate product codes can be reviewed before using it in operational reports.

When to use it: Use this when you suspect repeated keys or transactions and need to prove whether duplicated records are inflating totals or breaking joins. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, COUNT(*) AS record_count
FROM products
GROUP BY product_id
HAVING COUNT(*) > 1;
```

Sample result:

product_id	record_count
P-10045	2
P-22010	3

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before joining a master table to a transaction table, a data analyst checks whether the supposedly unique key appears more than once. Example request: 'Can you run duplicate product codes before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, COUNT(DISTINCT product_name) AS different_names
FROM products
GROUP BY product_id
HAVING COUNT(DISTINCT product_name) > 1;
```

Variant sample result:

product_id	different_names
P-10045	ACME Italy Srl
P-22010	Nordic Retail AB

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 55 - Products Missing in Price Table

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that products missing in price table can be reviewed before using it in operational reports.

When to use it: Use this when a report, upload, join, or calculation depends on mandatory fields being populated and you need to identify incomplete records quickly. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.product_id, p.product_name
FROM products p LEFT JOIN prices pr ON p.product_id = pr.product_id
```

```
WHERE pr.product_id IS NULL;
```

Sample result:

product_id	product_name
P-99110	Old Connector
P-77881	Legacy Switch

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before refreshing a Power BI dataset, the reporting owner checks which records are missing required fields that would create blank visuals or failed joins. Example request: 'Can you run products missing in price table before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.product_id, p.product_name, p.category
FROM products p LEFT JOIN prices pr ON p.product_id = pr.product_id AND pr.valid_to >= CURRENT_DATE
WHERE pr.product_id IS NULL AND p.status = 'Active';
```

Variant sample result:

product_id	product_name	category
P-99110	Old Connector	Sensors
P-77881	Legacy Switch	Relays

Common mistakes to avoid: Do not assume blanks and NULLs are the same. Check both conditions when validating imported data.

QUERY N. 56 - Products Missing in Cost Table

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that products missing in cost table can be reviewed before using it in operational reports.

When to use it: Use this when a report, upload, join, or calculation depends on mandatory fields being populated and you need to identify incomplete records quickly. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.product_id, p.product_name
FROM products p LEFT JOIN costs c ON p.product_id = c.product_id
WHERE c.product_id IS NULL;
```

Sample result:

product_id	product_name
P-99110	Old Connector
P-77881	Legacy Switch

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before refreshing a Power BI dataset, the reporting owner checks which records are missing required fields that would create blank visuals or failed joins. Example request: 'Can you run products missing in cost table before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.product_id, p.product_name, p.category
FROM products p LEFT JOIN costs c ON p.product_id = c.product_id AND c.cost_type = 'Standard'
WHERE c.product_id IS NULL AND p.status = 'Active';
```

Variant sample result:

product_id	product_name	category
P-99110	Old Connector	Sensors
P-77881	Legacy Switch	Relays

Common mistakes to avoid: Do not assume blanks and NULLs are the same. Check both conditions when validating imported data.

QUERY N. 57 - Products Missing in Stock Table

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that products missing in stock table can be reviewed before using it in operational reports.

When to use it: Use this when a report, upload, join, or calculation depends on mandatory fields being populated and you need to identify incomplete records quickly. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.product_id, p.product_name
FROM products p LEFT JOIN inventory i ON p.product_id = i.product_id
WHERE i.product_id IS NULL;
```

Sample result:

product_id	product_name
P-99110	Old Connector
P-77881	Legacy Switch

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before refreshing a Power BI dataset, the reporting owner checks which records are missing required fields that would create blank visuals or failed joins. Example request: 'Can you run products missing in stock table before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.product_id, p.product_name, p.category
FROM products p LEFT JOIN inventory i ON p.product_id = i.product_id
WHERE i.product_id IS NULL AND p.status = 'Active';
```

Variant sample result:

product_id	product_name	category
P-99110	Old Connector	Sensors
P-77881	Legacy Switch	Relays

Common mistakes to avoid: Do not assume blanks and NULLs are the same. Check both conditions when validating imported data.

QUERY N. 58 - Inventory Value by Product

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that inventory value by product can be reviewed before using it in operational reports.

When to use it: Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT i.product_id, SUM(i.stock_quantity) AS available_stock, c.unit_cost, SUM(i.stock_quantity) * c.unit_cost
AS inventory_value
FROM inventory i JOIN costs c ON i.product_id = c.product_id
GROUP BY i.product_id, c.unit_cost
ORDER BY inventory_value DESC;
```

Sample result:

product_id	available_stock	unit_cost	inventory_value
P-10045	245	42.50	10,412.50
P-22010	0	52.50	6,300.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Supply Chain asks for a stock exception list to review items that are unavailable, overstocked, or below safety level. Example request: 'Can you run inventory value by product before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT i.product_id, SUM(i.stock_quantity) AS available_stock, COALESCE(c.unit_cost,0) AS unit_cost,
SUM(i.stock_quantity)*COALESCE(c.unit_cost,0) AS inventory_value
FROM inventory i LEFT JOIN costs c ON i.product_id = c.product_id
GROUP BY i.product_id, COALESCE(c.unit_cost,0)
ORDER BY inventory_value DESC;
```

Variant sample result:

product_id	available_stock	unit_cost	inventory_value
P-10045	245	72.40	124,800.00
P-22010	0	54.10	96,450.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 59 - Inventory Value by Category

Difficulty: Intermediate

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that inventory value by category can be reviewed before using it in operational reports.

When to use it: Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.category, SUM(i.stock_quantity * c.unit_cost) AS inventory_value
FROM inventory i JOIN products p ON i.product_id = p.product_id JOIN costs c ON i.product_id = c.product_id
GROUP BY p.category
ORDER BY inventory_value DESC;
```

Sample result:

category	inventory_value
Sensors	10,412.50
Relays	6,300.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Supply Chain asks for a stock exception list to review items that are unavailable, overstocked, or below safety level. Example request: 'Can you run inventory value by category before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.category, i.warehouse_id, SUM(i.stock_quantity * c.unit_cost) AS inventory_value
FROM inventory i JOIN products p ON i.product_id = p.product_id JOIN costs c ON i.product_id = c.product_id
GROUP BY p.category, i.warehouse_id
ORDER BY p.category, inventory_value DESC;
```

Variant sample result:

category	warehouse_id	inventory_value
Sensors	ID-1001	124,800.00
Relays	ID-1002	96,450.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 60 - Slow-Moving Products

Difficulty: Advanced Practical

Best for: Supply Chain Analyst, Product Data Specialist, Inventory Planner, ERP User

Problem: You need to check product, stock, price, or cost data so that slow-moving products can be reviewed before using it in operational reports.

When to use it: Use this when checking product master data, price/cost completeness, stock availability, replenishment issues, or inventory reporting extracts. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT p.product_id, p.product_name, SUM(i.stock_quantity) AS available_stock, MAX(s.order_date) AS
last_sale_date

FROM products p LEFT JOIN inventory i ON p.product_id = i.product_id LEFT JOIN sales s ON p.product_id =
s.product_id

GROUP BY p.product_id, p.product_name

HAVING SUM(i.stock_quantity) > 0 AND (MAX(s.order_date) < CURRENT_DATE - INTERVAL '180 days' OR MAX(s.order_date)
IS NULL);
```

Sample result:

product_id	product_name	available_stock	last_sale_date
P-77881	Legacy Switch	245	2025-07-14
P-99110	Old Connector	0	2025-08-02

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a supply chain analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you run slow-moving products before the product/inventory data is sent to operations?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT p.product_id, p.product_name, SUM(i.stock_quantity) AS available_stock, SUM(i.stock_quantity *
COALESCE(c.unit_cost,0)) AS stock_value, MAX(s.order_date) AS last_sale_date

FROM products p LEFT JOIN inventory i ON p.product_id = i.product_id LEFT JOIN costs c ON p.product_id =
c.product_id LEFT JOIN sales s ON p.product_id = s.product_id

GROUP BY p.product_id, p.product_name

HAVING SUM(i.stock_quantity) > 0 AND (MAX(s.order_date) < CURRENT_DATE - INTERVAL '180 days' OR MAX(s.order_date)
IS NULL)

ORDER BY stock_value DESC;
```

Variant sample result:

product_id	product_name	available_stock	stock_value	last_sale_date
P-77881	Photoelectric Sensor	245	268410.00	2026-02-14
P-99110	Safety Relay	0	302950.00	2026-03-05

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

10. Customer Analysis Queries

Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation.

QUERY N. 61 - New Customers

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for new customers, without rebuilding the logic from scratch each time.

When to use it: Use this when Sales or CRM teams need to identify customer lifecycle status: new, active, returning, inactive, or lost. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH first_orders AS (SELECT customer_id, MIN(order_date) AS first_order_date FROM sales GROUP BY customer_id)
SELECT customer_id, first_order_date
FROM first_orders
WHERE first_order_date >= DATE '2026-01-01' AND first_order_date < DATE '2026-02-01';
```

Sample result:

customer_id	first_order_date
C1001	2026-02-14
C1045	2026-03-05

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: CRM or Sales wants to report how many customers placed their first order in the selected period. Example request: 'Can you identify customers related to new customers so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH first_orders AS (
SELECT s.customer_id, c.customer_name, MIN(s.order_date) AS first_order_date
FROM sales s LEFT JOIN customers c ON s.customer_id = c.customer_id
GROUP BY s.customer_id, c.customer_name
)
SELECT * FROM first_orders
WHERE first_order_date >= DATE '2026-01-01' AND first_order_date < DATE '2026-02-01';
```

Variant sample result:

customer_id	customer_name	first_order_date
C1001	ACME Italy Srl	2026-02-14
C1045	Nordic Retail AB	2026-03-05

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 62 - Lost Customers

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for lost customers, without rebuilding the logic from scratch each time.

When to use it: Use this when Sales or CRM teams need to identify customer lifecycle status: new, active, returning, inactive, or lost. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, MAX(order_date) AS last_order_date
FROM sales
GROUP BY customer_id
HAVING MAX(order_date) < CURRENT_DATE - INTERVAL '90 days';
```

Sample result:

customer_id	last_order_date
C1880	2026-02-14
C1915	2026-03-05

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Sales Operations wants to identify customers that have not ordered recently so account owners can follow up. Example request: 'Can you identify customers related to lost customers so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT c.customer_id, c.customer_name, MAX(s.order_date) AS last_order_date
FROM customers c LEFT JOIN sales s ON c.customer_id = s.customer_id
GROUP BY c.customer_id, c.customer_name
HAVING MAX(s.order_date) < CURRENT_DATE - INTERVAL '90 days' OR MAX(s.order_date) IS NULL;
```

Variant sample result:

customer_id	customer_name	last_order_date
C1880	Westline GmbH	2026-02-14
C1915	Blue Ocean Ltd	2026-03-05

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 63 - Returning Customers

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for returning customers, without rebuilding the logic from scratch each time.

When to use it: Use this when Sales or CRM teams need to identify customer lifecycle status: new, active, returning, inactive, or lost. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, COUNT(DISTINCT order_id) AS order_count
FROM sales
GROUP BY customer_id
HAVING COUNT(DISTINCT order_id) > 1
ORDER BY order_count DESC;
```

Sample result:

customer_id	order_count
C1001	42
C1045	36

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to returning customers so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, COUNT(DISTINCT DATE_TRUNC('month', order_date)) AS active_months, COUNT(DISTINCT order_id) AS
order_count
FROM sales
GROUP BY customer_id
ORDER BY order_count DESC;
```

Variant sample result:

customer_id	active_months	order_count
C1001	2026-02-01	42
C1045	2026-03-01	36

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 64 - Customer Purchase Frequency

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for customer purchase frequency, without rebuilding the logic from scratch each time.

When to use it: Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, COUNT(DISTINCT order_id) AS order_count
FROM sales
GROUP BY customer_id
HAVING COUNT(DISTINCT order_id) > 1
```

```
ORDER BY order_count DESC;
```

Sample result:

customer_id	order_count
C1001	42
C1045	36

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to customer purchase frequency so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, COUNT(DISTINCT DATE_TRUNC('month', order_date)) AS active_months, COUNT(DISTINCT order_id) AS
order_count
FROM sales
GROUP BY customer_id
ORDER BY order_count DESC;
```

Variant sample result:

customer_id	active_months	order_count
C1001	2026-02-01	42
C1045	2026-03-01	36

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 65 - Average Order Value by Customer

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for average order value by customer, without rebuilding the logic from scratch each time.

When to use it: Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT COUNT(DISTINCT order_id) AS order_count, SUM(sales_amount) AS total_sales, SUM(sales_amount) /
NULLIF(COUNT(DISTINCT order_id),0) AS average_order_value
FROM sales;
```

Sample result:

order_count	total_sales	average_order_value
42	268410.00	268410.00
36	302950.00	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to average order value by customer so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, COUNT(DISTINCT order_id) AS order_count, SUM(sales_amount) AS total_sales, SUM(sales_amount)
/ NULLIF(COUNT(DISTINCT order_id),0) AS average_order_value

FROM sales

GROUP BY customer_id

ORDER BY average_order_value DESC;
```

Variant sample result:

customer_id	order_count	total_sales	average_order_value
C1001	42	268410.00	268410.00
C1045	36	302950.00	302950.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 66 - Top Customers by Revenue

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for top customers by revenue, without rebuilding the logic from scratch each time.

When to use it: Use this when stakeholders ask for the highest-performing customers, products, categories, or business units in a clear ranked list. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT * FROM sales;
```

Sample result:

result
Example A
Example B

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: account managers want a list of the most valuable customers before a quarterly business review. Example request: 'Can you identify customers related to top customers by revenue so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT * FROM sales WHERE order_date >= DATE '2026-01-01';
```

Variant sample result:

result
Example A
Example B

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 67 - Customers with Declining Sales

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for customers with declining sales, without rebuilding the logic from scratch each time.

When to use it: Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH customer_year_sales AS (  
  SELECT customer_id, EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales  
  FROM sales GROUP BY customer_id, EXTRACT(YEAR FROM order_date)  
) , compared AS (  
  SELECT customer_id, sales_year, total_sales, LAG(total_sales) OVER (PARTITION BY customer_id ORDER BY sales_year)  
  AS previous_year_sales  
  FROM customer_year_sales  
)  
  
SELECT * FROM compared  
  
WHERE sales_year = 2026 AND total_sales < previous_year_sales;
```

Sample result:

customer_id	sales_year	total_sales	previous_year_sales
C1001	2025	268410.00	2025
C1880	2026	302950.00	2026

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to customers with declining sales so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH customer_year_sales AS (  
  SELECT customer_id, EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales
```

```

FROM sales GROUP BY customer_id, EXTRACT(YEAR FROM order_date)
), compared AS (
SELECT customer_id, sales_year, total_sales, LAG(total_sales) OVER (PARTITION BY customer_id ORDER BY sales_year)
AS previous_year_sales
FROM customer_year_sales
)
SELECT customer_id, total_sales, previous_year_sales,
ROUND((total_sales-previous_year_sales)/NULLIF(previous_year_sales,0)*100,2) AS growth_pct
FROM compared WHERE sales_year = 2026 AND total_sales < previous_year_sales;

```

Variant sample result:

customer_id	total_sales	previous_year_sales	growth_pct
C1001	268410.00	2025	10.44
C1880	302950.00	2026	-37.81

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 68 - Customers with Growing Sales

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for customers with growing sales, without rebuilding the logic from scratch each time.

When to use it: Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```

WITH customer_year_sales AS (
SELECT customer_id, EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales
FROM sales GROUP BY customer_id, EXTRACT(YEAR FROM order_date)
), compared AS (
SELECT customer_id, sales_year, total_sales, LAG(total_sales) OVER (PARTITION BY customer_id ORDER BY sales_year)
AS previous_year_sales
FROM customer_year_sales
)
SELECT * FROM compared
WHERE sales_year = 2026 AND total_sales > previous_year_sales;

```

Sample result:

customer_id	sales_year	total_sales	previous_year_sales
C1001	2025	268410.00	2025
C1880	2026	302950.00	2026

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to customers with growing sales so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH customer_year_sales AS (  
    SELECT customer_id, EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales  
    FROM sales GROUP BY customer_id, EXTRACT(YEAR FROM order_date)  
), compared AS (  
    SELECT customer_id, sales_year, total_sales, LAG(total_sales) OVER (PARTITION BY customer_id ORDER BY sales_year)  
    AS previous_year_sales  
    FROM customer_year_sales  
)  
  
SELECT customer_id, total_sales, previous_year_sales,  
ROUND((total_sales-previous_year_sales)/NULLIF(previous_year_sales,0)*100,2) AS growth_pct  
FROM compared WHERE sales_year = 2026 AND total_sales > previous_year_sales;
```

Variant sample result:

customer_id	total_sales	previous_year_sales	growth_pct
C1001	268410.00	2025	10.44
C1880	302950.00	2026	-37.81

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 69 - Customers by Number of Orders

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for customers by number of orders, without rebuilding the logic from scratch each time.

When to use it: Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, COUNT(DISTINCT order_id) AS order_count  
FROM sales  
GROUP BY customer_id  
HAVING COUNT(DISTINCT order_id) > 1  
ORDER BY order_count DESC;
```

Sample result:

customer_id	order_count
C1001	42
C1045	36

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to customers by number of orders so

Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, COUNT(DISTINCT DATE_TRUNC('month', order_date)) AS active_months, COUNT(DISTINCT order_id) AS
order_count

FROM sales

GROUP BY customer_id

ORDER BY order_count DESC;
```

Variant sample result:

customer_id	active_months	order_count
C1001	2026-02-01	42
C1045	2026-03-01	36

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 70 - Customer Segmentation by Revenue

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for customer segmentation by revenue, without rebuilding the logic from scratch each time.

When to use it: Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, SUM(sales_amount) AS total_sales, CASE WHEN SUM(sales_amount) >= 100000 THEN 'High value'
WHEN SUM(sales_amount) >= 25000 THEN 'Medium value' ELSE 'Low value' END AS revenue_segment

FROM sales

GROUP BY customer_id;
```

Sample result:

customer_id	total_sales	revenue_segment
C1001	268410.00	268410.00
C1045	302950.00	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to customer segmentation by revenue so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT revenue_segment, COUNT(*) AS customer_count
```

```
FROM (SELECT customer_id, CASE WHEN SUM(sales_amount) >= 100000 THEN 'High value' WHEN SUM(sales_amount) >= 25000
THEN 'Medium value' ELSE 'Low value' END AS revenue_segment FROM sales GROUP BY customer_id) x

GROUP BY revenue_segment;
```

Variant sample result:

customer_id	revenue_segment
C1001	268410.00
C1045	302950.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 71 - Customer Segmentation by Frequency

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for customer segmentation by frequency, without rebuilding the logic from scratch each time.

When to use it: Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, COUNT(DISTINCT order_id) AS order_count
FROM sales
GROUP BY customer_id
HAVING COUNT(DISTINCT order_id) > 1
ORDER BY order_count DESC;
```

Sample result:

customer_id	order_count
C1001	42
C1045	36

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to customer segmentation by frequency so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, COUNT(DISTINCT DATE_TRUNC('month', order_date)) AS active_months, COUNT(DISTINCT order_id) AS
order_count
FROM sales
GROUP BY customer_id
ORDER BY order_count DESC;
```

Variant sample result:

customer_id	active_months	order_count
C1001	2026-02-01	42
C1045	2026-03-01	36

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 72 - Customers Inactive for 90 Days

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for customers inactive for 90 days, without rebuilding the logic from scratch each time.

When to use it: Use this when Sales or CRM teams need to identify customer lifecycle status: new, active, returning, inactive, or lost. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, MAX(order_date) AS last_order_date
FROM sales
GROUP BY customer_id
HAVING MAX(order_date) < CURRENT_DATE - INTERVAL '90 days';
```

Sample result:

customer_id	last_order_date
C1880	2026-02-14
C1915	2026-03-05

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Sales Operations wants to identify customers that have not ordered recently so account owners can follow up. Example request: 'Can you identify customers related to customers inactive for 90 days so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT c.customer_id, c.customer_name, MAX(s.order_date) AS last_order_date
FROM customers c LEFT JOIN sales s ON c.customer_id = s.customer_id
GROUP BY c.customer_id, c.customer_name
HAVING MAX(s.order_date) < CURRENT_DATE - INTERVAL '90 days' OR MAX(s.order_date) IS NULL;
```

Variant sample result:

customer_id	customer_name	last_order_date
C1880	Westline GmbH	2026-02-14
C1915	Blue Ocean Ltd	2026-03-05

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 73 - Customers Active This Month

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for customers active this month, without rebuilding the logic from scratch each time.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT DISTINCT customer_id
FROM sales
WHERE order_date >= DATE_TRUNC('month', CURRENT_DATE)
AND order_date < DATE_TRUNC('month', CURRENT_DATE) + INTERVAL '1 month';
```

Sample result:

DISTINCT_customer_id
C1001
C1045

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to customers active this month so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, COUNT(DISTINCT order_id) AS orders_this_month, SUM(sales_amount) AS sales_this_month
FROM sales
WHERE order_date >= DATE_TRUNC('month', CURRENT_DATE)
AND order_date < DATE_TRUNC('month', CURRENT_DATE) + INTERVAL '1 month'
GROUP BY customer_id
ORDER BY sales_this_month DESC;
```

Variant sample result:

customer_id	orders_this_month	sales_this_month
C1001	2026-02-01	2026-02-01
C1045	2026-03-01	2026-03-01

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 74 - First Purchase Date by Customer

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for first purchase date by customer, without rebuilding the logic from scratch each time.

When to use it: Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH first_orders AS (SELECT customer_id, MIN(order_date) AS first_order_date FROM sales GROUP BY customer_id)
SELECT customer_id, first_order_date
FROM first_orders
WHERE first_order_date >= DATE '2026-01-01' AND first_order_date < DATE '2026-02-01';
```

Sample result:

customer_id	first_order_date
C1001	2026-02-14
C1045	2026-03-05

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to first purchase date by customer so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH first_orders AS (
  SELECT s.customer_id, c.customer_name, MIN(s.order_date) AS first_order_date
  FROM sales s LEFT JOIN customers c ON s.customer_id = c.customer_id
  GROUP BY s.customer_id, c.customer_name
)
SELECT * FROM first_orders
WHERE first_order_date >= DATE '2026-01-01' AND first_order_date < DATE '2026-02-01';
```

Variant sample result:

customer_id	customer_name	first_order_date
C1045	ACME Italy Srl	2026-02-14
C1188	Nordic Retail AB	2026-03-05

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 75 - Last Purchase Date by Customer

Difficulty: Intermediate

Best for: CRM Analyst, Business Analyst, Sales Operations, Marketing Analyst

Problem: You need a repeatable customer analysis query for last purchase date by customer, without rebuilding the logic from scratch each time.

When to use it: Use this when analyzing customer behavior, retention, inactivity, purchase frequency, commercial prioritization, or CRM/reporting segmentation. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, MAX(order_date) AS last_order_date
FROM sales
GROUP BY customer_id
HAVING MAX(order_date) < CURRENT_DATE - INTERVAL '90 days';
```

Sample result:

customer_id	last_order_date
C1880	2026-02-14
C1915	2026-03-05

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a CRM analyst uses this query to answer a recurring customer question without rebuilding the logic from scratch. Example request: 'Can you identify customers related to last purchase date by customer so Sales can prioritize follow-up actions?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT c.customer_id, c.customer_name, MAX(s.order_date) AS last_order_date
FROM customers c LEFT JOIN sales s ON c.customer_id = s.customer_id
GROUP BY c.customer_id, c.customer_name
HAVING MAX(s.order_date) < CURRENT_DATE - INTERVAL '90 days' OR MAX(s.order_date) IS NULL;
```

Variant sample result:

customer_id	customer_name	last_order_date
C1880	Westline GmbH	2026-02-14
C1915	Blue Ocean Ltd	2026-03-05

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

11. Data Quality Control Queries

Use this when validating a dataset before trusting it in a report, dashboard, migration, ERP extract, Power BI model, or automated refresh.

QUERY N. 76 - Find Duplicated Primary Keys

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks find duplicated primary keys before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when you suspect repeated keys or transactions and need to prove whether duplicated records are inflating totals or breaking joins. Adapt the table and field names to your database, then review the returned columns before

using the result.

SQL:

```
SELECT id, COUNT(*) AS record_count
FROM your_table
GROUP BY id
HAVING COUNT(*) > 1;
```

Sample result:

id	record_count
ID-1001	2
ID-1002	3

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before joining a master table to a transaction table, a data analyst checks whether the supposedly unique key appears more than once. Example request: 'Can you validate the dataset for find duplicated primary keys before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT id, COUNT(*) AS record_count, MIN(created_at) AS first_created_at, MAX(created_at) AS last_created_at
FROM your_table
GROUP BY id
HAVING COUNT(*) > 1;
```

Variant sample result:

id	record_count	first_created_at	last_created_at
ID-1001	2	Example A	Example A
ID-1002	3	Example B	Example B

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 77 - Find Missing Foreign Keys

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks find missing foreign keys before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when a report, upload, join, or calculation depends on mandatory fields being populated and you need to identify incomplete records quickly. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT s.order_id, s.customer_id
FROM sales s LEFT JOIN customers c ON s.customer_id = c.customer_id
WHERE c.customer_id IS NULL;
```

Sample result:

order_id	customer_id
SO-100451	C1001
SO-100782	C1045

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before refreshing a Power BI dataset, the reporting owner checks which records are missing required fields that would create blank visuals or failed joins. Example request: 'Can you validate the dataset for find missing foreign keys before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT s.product_id, COUNT(*) AS affected_rows
FROM sales s LEFT JOIN products p ON s.product_id = p.product_id
WHERE p.product_id IS NULL
GROUP BY s.product_id;
```

Variant sample result:

product_id	affected_rows
P-99110	Example A
P-22010	Example B

Common mistakes to avoid: Do not assume blanks and NULLs are the same. Check both conditions when validating imported data.

QUERY N. 78 - Records in Table A but Not in Table B

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks records in table a but not in table b before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when a join, reconciliation, migration, or report refresh gives unexpected totals and you need to compare keys across tables. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT a.product_id
FROM table_a a LEFT JOIN table_b b ON a.product_id = b.product_id
WHERE b.product_id IS NULL;
```

Sample result:

product_id
P-10045
P-22010

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for records in table a but not in table b before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT a.product_id, a.product_name, a.status
FROM products a LEFT JOIN prices b ON a.product_id = b.product_id
WHERE b.product_id IS NULL;
```

Variant sample result:

product_id	product_name	status
P-99110	Old Connector	Active
P-22010	Safety Relay	Discontinued

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 79 - Records in Table B but Not in Table A

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks records in table b but not in table a before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when a join, reconciliation, migration, or report refresh gives unexpected totals and you need to compare keys across tables. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT a.product_id
FROM table_a a LEFT JOIN table_b b ON a.product_id = b.product_id
WHERE b.product_id IS NULL;
```

Sample result:

product_id
P-10045
P-22010

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for records in table b but not in table a before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT a.product_id, a.product_name, a.status
FROM products a LEFT JOIN prices b ON a.product_id = b.product_id
WHERE b.product_id IS NULL;
```

Variant sample result:

product_id	product_name	status
P-99110	Old Connector	Active
P-22010	Safety Relay	Discontinued

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 80 - Check Join Multiplication

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks check join multiplication before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when a join, reconciliation, migration, or report refresh gives unexpected totals and you need to compare keys across tables. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT s.order_id, COUNT(*) AS rows_after_join
FROM sales s LEFT JOIN products p ON s.product_id = p.product_id
GROUP BY s.order_id
HAVING COUNT(*) > 1;
```

Sample result:

order_id	rows_after_join
SO-100451	Example A
SO-100782	Example B

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: after a report total becomes too high, the analyst checks whether a join multiplied rows because of duplicated keys. Example request: 'Can you validate the dataset for check join multiplication before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, COUNT(*) AS product_master_rows
FROM products
GROUP BY product_id
HAVING COUNT(*) > 1;
```

Variant sample result:

product_id	product_master_rows
P-10045	Example A
P-22010	Example B

Common mistakes to avoid: Always check key uniqueness before and after the join. Duplicated keys can multiply rows and inflate totals.

QUERY N. 81 - Count Rows Before and After Join

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks count rows before and after join before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when a join, reconciliation, migration, or report refresh gives unexpected totals and you need to compare keys across tables. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT s.order_id, COUNT(*) AS rows_after_join
FROM sales s LEFT JOIN products p ON s.product_id = p.product_id
GROUP BY s.order_id
HAVING COUNT(*) > 1;
```

Sample result:

order_id	rows_after_join
SO-100451	Example A
SO-100782	Example B

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: after a report total becomes too high, the analyst checks whether a join multiplied rows because of duplicated keys. Example request: 'Can you validate the dataset for count rows before and after join before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, COUNT(*) AS product_master_rows
FROM products
GROUP BY product_id
HAVING COUNT(*) > 1;
```

Variant sample result:

product_id	product_master_rows
P-10045	Example A
P-22010	Example B

Common mistakes to avoid: Always check key uniqueness before and after the join. Duplicated keys can multiply rows and inflate totals.

QUERY N. 82 - Find Mismatched Customer Codes

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks find mismatched customer codes before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when a join, reconciliation, migration, or report refresh gives unexpected totals and you need to compare keys across tables. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT s.order_id, s.customer_id
FROM sales s LEFT JOIN customers c ON s.customer_id = c.customer_id
WHERE c.customer_id IS NULL;
```

Sample result:

order_id	customer_id
SO-100451	C1001
SO-100782	C1045

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for find mismatched customer codes before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT s.product_id, COUNT(*) AS affected_rows
FROM sales s LEFT JOIN products p ON s.product_id = p.product_id
WHERE p.product_id IS NULL
GROUP BY s.product_id;
```

Variant sample result:

product_id	affected_rows
P-10045	Example A
P-22010	Example B

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 83 - Find Mismatched Product Codes

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks find mismatched product codes before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when a join, reconciliation, migration, or report refresh gives unexpected totals and you need to compare keys across tables. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT s.product_id, COUNT(*) AS affected_rows
FROM sales s LEFT JOIN products p ON s.product_id = p.product_id
WHERE p.product_id IS NULL
GROUP BY s.product_id;
```

Sample result:

product_id	affected_rows
P-10045	Example A
P-22010	Example B

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for find mismatched product codes before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT TRIM(UPPER(s.product_id)) AS product_id_clean, COUNT(*) AS affected_rows
FROM sales s LEFT JOIN products p ON TRIM(UPPER(s.product_id)) = TRIM(UPPER(p.product_id))
WHERE p.product_id IS NULL
GROUP BY TRIM(UPPER(s.product_id));
```

Variant sample result:

product_id_clean	affected_rows
P-10045	Example A
P-22010	Example B

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 84 - Find Invalid Negative Quantities

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks find invalid negative quantities before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when validating a dataset before trusting it in a report, dashboard, migration, ERP extract, Power BI model, or automated refresh. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_id, product_id, quantity
FROM order_lines
```

```
WHERE quantity < 0;
```

Sample result:

order_id	product_id	quantity
SO-100451	P-10045	12
SO-100782	P-22010	4

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for find invalid negative quantities before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT order_id, product_id, quantity, 'Negative quantity' AS quantity_check
FROM order_lines
WHERE quantity < 0;
```

Variant sample result:

order_id	product_id	quantity	quantity_check
SO-100451	P-10045	12	Example A
SO-100782	P-22010	4	Example B

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 85 - Find Invalid Negative Prices

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks find invalid negative prices before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when validating a dataset before trusting it in a report, dashboard, migration, ERP extract, Power BI model, or automated refresh. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, unit_price
FROM prices
WHERE unit_price < 0;
```

Sample result:

product_id	unit_price
P-10045	125.50
P-22010	89.90

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Product Data or Finance needs to find items with missing or suspicious commercial values before a price list update. Example request: 'Can you validate the dataset for find invalid negative prices before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT product_id, price_list, unit_price
FROM prices
WHERE unit_price < 0
ORDER BY price_list, product_id;
```

Variant sample result:

product_id	price_list	unit_price
P-10045	Example A	125.50
P-22010	Example B	89.90

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 86 - Find Dates in the Future

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks find dates in the future before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when validating a dataset before trusting it in a report, dashboard, migration, ERP extract, Power BI model, or automated refresh. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_id, order_date
FROM orders
WHERE order_date > CURRENT_DATE;
```

Sample result:

order_id	order_date
SO-100451	2026-02-14
SO-100782	2026-03-05

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for find dates in the future before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT order_id, order_date, 'Future date' AS date_check
FROM orders
WHERE order_date > CURRENT_DATE;
```

Variant sample result:

order_id	order_date	date_check
SO-100451	2026-02-14	2026-02-14
SO-100782	2026-03-05	2026-03-05

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 87 - Find Dates Before Allowed Minimum Date

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks find dates before allowed minimum date before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when validating a dataset before trusting it in a report, dashboard, migration, ERP extract, Power BI model, or automated refresh. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_id, order_date
FROM orders
WHERE order_date < DATE '2000-01-01';
```

Sample result:

order_id	order_date
SO-100451	2026-02-14
SO-100782	2026-03-05

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for find dates before allowed minimum date before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT order_id, order_date, 'Too old' AS date_check
FROM orders
WHERE order_date < DATE '2000-01-01';
```

Variant sample result:

order_id	order_date	date_check
SO-100451	2026-02-14	2026-02-14
SO-100782	2026-03-05	2026-03-05

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 88 - Find Orphan Records

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks find orphan records before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when validating a dataset before trusting it in a report, dashboard, migration, ERP extract, Power BI model, or automated refresh. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT ol.order_id, ol.product_id
FROM order_lines ol LEFT JOIN orders o ON ol.order_id = o.order_id
WHERE o.order_id IS NULL;
```

Sample result:

order_id	product_id
SO-100451	P-10045
SO-100782	P-22010

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for find orphan records before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT ol.order_id, COUNT(*) AS orphan_line_count
FROM order_lines ol LEFT JOIN orders o ON ol.order_id = o.order_id
WHERE o.order_id IS NULL
GROUP BY ol.order_id;
```

Variant sample result:

order_id	orphan_line_count
SO-100451	42
SO-100782	36

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 89 - Validate Mandatory Fields

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks validate mandatory fields before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when validating a dataset before trusting it in a report, dashboard, migration, ERP extract, Power BI model, or automated refresh. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_id, order_date, customer_id, product_id, quantity, sales_amount
FROM sales
WHERE order_id IS NULL OR order_date IS NULL OR customer_id IS NULL OR product_id IS NULL OR quantity IS NULL OR
sales_amount IS NULL;
```

Sample result:

order_id	order_date	customer_id	product_id	quantity	sales_amount
SO-100451	2026-02-14	C1001	P-10045	12	268410.00
SO-100782	2026-03-05	C1045	P-22010	4	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for validate mandatory fields before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT order_id, CASE
WHEN order_date IS NULL THEN 'Missing order date'
WHEN customer_id IS NULL THEN 'Missing customer ID'
WHEN product_id IS NULL THEN 'Missing product ID'
ELSE 'Other missing field' END AS data_quality_issue
FROM sales
WHERE order_date IS NULL OR customer_id IS NULL OR product_id IS NULL;
```

Variant sample result:

order_id	data_quality_issue
SO-100451	Missing customer ID
SO-100782	Missing product ID

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 90 - Compare Totals Between Two Tables

Difficulty: Intermediate

Best for: Data Manager, BI Developer, ERP Key User, Reporting Owner

Problem: You need a control query that checks compare totals between two tables before the data is trusted in a report, integration, migration, or Power BI model.

When to use it: Use this when validating a dataset before trusting it in a report, dashboard, migration, ERP extract, Power BI model, or automated refresh. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT 'source_table' AS table_name, SUM(amount) AS total_amount FROM source_table

UNION ALL

SELECT 'target_table' AS table_name, SUM(amount) AS total_amount FROM target_table;
```

Sample result:

table_name	total_amount
ACME Italy Srl	3,120,000.00
Nordic Retail AB	3,118,450.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a data manager needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you validate the dataset for compare totals between two tables before we trust the dashboard totals?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT s.period, s.source_total, t.target_total, s.source_total - t.target_total AS difference
FROM (SELECT period, SUM(amount) AS source_total FROM source_table GROUP BY period) s
JOIN (SELECT period, SUM(amount) AS target_total FROM target_table GROUP BY period) t ON s.period = t.period;
```

Variant sample result:

period	target_total
2026-02	Example A
2026-02	Example B

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

12. Advanced Practical SQL Queries

Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic.

QUERY N. 91 - ROW_NUMBER for Ranking Records

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for row_number for ranking records that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when stakeholders ask for the highest-performing customers, products, categories, or business units in a clear ranked list. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH ranked AS (  
    SELECT customer_id, order_id, sales_amount, order_date, ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY  
        order_date DESC) AS row_num  
    FROM sales  
)  
  
SELECT * FROM ranked WHERE row_num = 1;
```

Sample result:

customer_id	order_id	sales_amount	order_date	row_num
C1001	SO-100451	268410.00	2026-02-14	1
C1045	SO-100782	302950.00	2026-03-05	2

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a BI developer needs to keep only the latest valid record per customer, product, or price condition. Example request: 'Can you use a practical SQL pattern for row_number for ranking records without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH ranked AS (  
    SELECT customer_id, order_id, sales_amount, order_date, ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY  
        order_date DESC) AS row_num  
    FROM sales  
    WHERE order_date <= CURRENT_DATE  
)  
  
SELECT * FROM ranked WHERE row_num = 1;
```

Variant sample result:

customer_id	order_id	sales_amount	order_date	row_num
C1001	SO-100451	268410.00	2026-02-14	1
C1045	SO-100782	302950.00	2026-03-05	2

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 92 - RANK for Ranking with Ties

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for rank for ranking with ties that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when stakeholders ask for the highest-performing customers, products, categories, or business units in a clear ranked list. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, SUM(sales_amount) AS total_sales, RANK() OVER (ORDER BY SUM(sales_amount) DESC) AS sales_rank  
FROM sales
```

```
GROUP BY customer_id;
```

Sample result:

customer_id	total_sales	sales_rank
C1001	182,450.00	1
C1045	145,220.00	2

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a intermediate analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you use a practical SQL pattern for rank for ranking with ties without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT region, customer_id, SUM(sales_amount) AS total_sales, RANK() OVER (PARTITION BY region ORDER BY
SUM(sales_amount) DESC) AS regional_rank
FROM sales
GROUP BY region, customer_id;
```

Variant sample result:

region	customer_id	total_sales	regional_rank
South Europe	C1001	182,450.00	1
Nordics	C1045	145,220.00	1

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 93 - DENSE_RANK for Compact Ranking

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for dense_rank for compact ranking that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when stakeholders ask for the highest-performing customers, products, categories, or business units in a clear ranked list. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, SUM(sales_amount) AS total_sales, DENSE_RANK() OVER (ORDER BY SUM(sales_amount) DESC) AS
dense_sales_rank
FROM sales
GROUP BY product_id;
```

Sample result:

product_id	total_sales	dense_sales_rank
P-10045	182,450.00	1
P-22010	145,220.00	2

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a intermediate analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you use a practical SQL pattern for dense_rank for compact ranking without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT category, product_id, SUM(sales_amount) AS total_sales, DENSE_RANK() OVER (PARTITION BY category ORDER BY
SUM(sales_amount) DESC) AS category_rank

FROM sales

GROUP BY category, product_id;
```

Variant sample result:

category	product_id	total_sales	category_rank
Sensors	P-10045	182,450.00	1
Relays	P-22010	145,220.00	2

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 94 - LAG to Compare with Previous Row

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for lag to compare with previous row that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, order_date, sales_amount, LAG(sales_amount) OVER (PARTITION BY customer_id ORDER BY
order_date) AS previous_sales_amount

FROM sales;
```

Sample result:

customer_id	order_date	sales_amount	previous_sales_amount
C1001	2026-02-14	268410.00	268410.00
C1045	2026-03-05	302950.00	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: an analyst needs to compare each month, order, or status change with the previous or next record. Example request: 'Can you use a practical SQL pattern for lag to compare with previous row without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, order_date, sales_amount, sales_amount - LAG(sales_amount) OVER (PARTITION BY customer_id
ORDER BY order_date) AS difference_vs_previous_order

FROM sales;
```

Variant sample result:

customer_id	order_date	sales_amount	difference_vs_previous_order
C1001	2026-02-14	268410.00	22530.00
C1045	2026-03-05	302950.00	34540.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 95 - LEAD to Compare with Next Row

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for lead to compare with next row that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT customer_id, order_date, sales_amount, LEAD(sales_amount) OVER (PARTITION BY customer_id ORDER BY
order_date) AS next_sales_amount

FROM sales;
```

Sample result:

customer_id	order_date	sales_amount	next_sales_amount
C1001	2026-02-14	268410.00	268410.00
C1045	2026-03-05	302950.00	302950.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: an analyst needs to compare each month, order, or status change with the previous or next record. Example request: 'Can you use a practical SQL pattern for lead to compare with next row without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, order_date, LEAD(order_date) OVER (PARTITION BY customer_id ORDER BY order_date) AS
next_order_date

FROM sales;
```

Variant sample result:

customer_id	order_date	next_order_date
C1001	2026-02-14	2026-02-14

customer_id	order_date	next_order_date
C1045	2026-03-05	2026-03-05

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 96 - Running Total

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for running total that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT order_date, sales_amount, SUM(sales_amount) OVER (ORDER BY order_date ROWS BETWEEN UNBOUNDED PRECEDING AND
CURRENT ROW) AS running_total
FROM sales
ORDER BY order_date;
```

Sample result:

order_date	sales_amount	running_total
2026-02-14	268410.00	245,880.00
2026-03-05	302950.00	514,290.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Finance or Sales wants cumulative performance over time instead of isolated monthly values. Example request: 'Can you use a practical SQL pattern for running total without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, order_date, sales_amount, SUM(sales_amount) OVER (PARTITION BY customer_id ORDER BY
order_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS customer_running_total
FROM sales
ORDER BY customer_id, order_date;
```

Variant sample result:

customer_id	order_date	sales_amount	customer_running_total
C1001	2026-02-14	268410.00	Example A
C1045	2026-03-05	302950.00	Example B

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 97 - Moving Average

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for moving average that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT sales_month, monthly_sales, AVG(monthly_sales) OVER (ORDER BY sales_month ROWS BETWEEN 2 PRECEDING AND
CURRENT ROW) AS moving_avg_3_months
FROM monthly_sales;
```

Sample result:

sales_month	monthly_sales	moving_avg_3_months
2026-02-01	2026-02-01	2026-02-01
2026-03-01	2026-03-01	2026-03-01

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: management wants to smooth monthly fluctuations and understand the underlying trend. Example request: 'Can you use a practical SQL pattern for moving average without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH monthly_sales AS (SELECT DATE_TRUNC('month', order_date) AS sales_month, SUM(sales_amount) AS monthly_sales
FROM sales GROUP BY DATE_TRUNC('month', order_date))

SELECT sales_month, monthly_sales, AVG(monthly_sales) OVER (ORDER BY sales_month ROWS BETWEEN 5 PRECEDING AND
CURRENT ROW) AS moving_avg_6_months
FROM monthly_sales;
```

Variant sample result:

sales_month	monthly_sales	moving_avg_6_months
2026-02-01	2026-02-01	2026-02-01
2026-03-01	2026-03-01	2026-03-01

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 98 - Deduplicate Using ROW_NUMBER

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for deduplicate using row_number that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when you suspect repeated keys or transactions and need to prove whether duplicated records are inflating totals or breaking joins. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH ranked AS (
```

```

SELECT product_id, product_name, updated_at, ROW_NUMBER() OVER (PARTITION BY product_id ORDER BY updated_at DESC)
AS row_num

FROM products

)

SELECT * FROM ranked WHERE row_num = 1;

```

Sample result:

product_id	product_name	updated_at	row_num
P-10045	Photoelectric Sensor	2026-02-14	1
P-22010	Safety Relay	2026-03-05	2

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: before joining a master table to a transaction table, a data analyst checks whether the supposedly unique key appears more than once. Example request: 'Can you use a practical SQL pattern for deduplicate using row_number without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```

WITH ranked AS (

SELECT product_id, product_name, updated_at, ROW_NUMBER() OVER (PARTITION BY product_id ORDER BY updated_at DESC)
AS row_num

FROM products

WHERE updated_at <= CURRENT_DATE

)

SELECT * FROM ranked WHERE row_num = 1;

```

Variant sample result:

product_id	product_name	updated_at	row_num
P-10045	Photoelectric Sensor	2026-02-14	1
P-22010	Safety Relay	2026-03-05	2

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 99 - Get Latest Record per Customer

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for get latest record per customer that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```

WITH ranked AS (

SELECT customer_id, order_id, sales_amount, order_date, ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY
order_date DESC) AS row_num

FROM sales

```

```
)
SELECT * FROM ranked WHERE row_num = 1;
```

Sample result:

customer_id	order_id	sales_amount	order_date	row_num
C1001	SO-100451	268410.00	2026-02-14	1
C1045	SO-100782	302950.00	2026-03-05	2

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a BI developer needs to keep only the latest valid record per customer, product, or price condition. Example request: 'Can you use a practical SQL pattern for get latest record per customer without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH ranked AS (
  SELECT customer_id, order_id, sales_amount, order_date, ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY
    order_date DESC) AS row_num
  FROM sales
  WHERE order_date <= CURRENT_DATE
)
SELECT * FROM ranked WHERE row_num = 1;
```

Variant sample result:

customer_id	order_id	sales_amount	order_date	row_num
C1001	SO-100451	268410.00	2026-02-14	1
C1045	SO-100782	302950.00	2026-03-05	2

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 100 - Get Latest Price per Product

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for get latest price per product that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH ranked AS (
  SELECT product_id, unit_price, valid_from, ROW_NUMBER() OVER (PARTITION BY product_id ORDER BY valid_from DESC)
    AS row_num
  FROM prices
)
SELECT * FROM ranked WHERE row_num = 1;
```

Sample result:

product_id	unit_price	valid_from	row_num
P-10045	125.50	2026-01-01	1
P-22010	89.90	2026-03-01	2

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: Product Data or Finance needs to find items with missing or suspicious commercial values before a price list update. Example request: 'Can you use a practical SQL pattern for get latest price per product without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH ranked AS (
    SELECT product_id, unit_price, valid_from, ROW_NUMBER() OVER (PARTITION BY product_id ORDER BY valid_from DESC)
    AS row_num
    FROM prices
    WHERE valid_from <= CURRENT_DATE
)
SELECT * FROM ranked WHERE row_num = 1;
```

Variant sample result:

product_id	unit_price	valid_from	row_num
P-10045	125.50	2026-01-01	1
P-22010	89.90	2026-03-01	2

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 101 - Pivot with CASE WHEN

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for pivot with case when that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT product_id, SUM(CASE WHEN country = 'Italy' THEN sales_amount ELSE 0 END) AS sales_italy, SUM(CASE WHEN
country = 'Germany' THEN sales_amount ELSE 0 END) AS sales_germany, SUM(CASE WHEN country = 'France' THEN
sales_amount ELSE 0 END) AS sales_france
FROM sales
GROUP BY product_id;
```

Sample result:

product_id	sales_italy	sales_germany	sales_france
P-10045	45100.00	38200.00	29900.00

product_id	sales_italy	sales_germany	sales_france
P-22010	32500.00	27800.00	21400.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: an Excel or Power BI user needs a flat report table with countries, years, or categories displayed as columns. Example request: 'Can you use a practical SQL pattern for pivot with case when without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
SELECT customer_id, SUM(CASE WHEN EXTRACT(YEAR FROM order_date) = 2025 THEN sales_amount ELSE 0 END) AS
sales_2025, SUM(CASE WHEN EXTRACT(YEAR FROM order_date) = 2026 THEN sales_amount ELSE 0 END) AS sales_2026
FROM sales
GROUP BY customer_id;
```

Variant sample result:

customer_id	sales_2025	sales_2026
C1001	165200.00	182450.00
C1045	130400.00	145220.00

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 102 - Current Month vs Previous Month

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for current month vs previous month that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH monthly_sales AS (SELECT DATE_TRUNC('month', order_date) AS sales_month, SUM(sales_amount) AS total_sales
FROM sales GROUP BY DATE_TRUNC('month', order_date))

SELECT sales_month, total_sales, LAG(total_sales) OVER (ORDER BY sales_month) AS previous_month_sales,
total_sales - LAG(total_sales) OVER (ORDER BY sales_month) AS difference
FROM monthly_sales;
```

Sample result:

sales_month	total_sales	previous_month_sales	difference
2026-02-01	268410.00	2026-02-01	22530.00
2026-03-01	302950.00	2026-03-01	34540.00

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: an analyst needs to compare each month, order, or status change with the previous or next record. Example request: 'Can you use a practical SQL pattern for current month vs previous month without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH monthly_sales AS (SELECT DATE_TRUNC('month', order_date) AS sales_month, SUM(sales_amount) AS total_sales
FROM sales GROUP BY DATE_TRUNC('month', order_date))

SELECT sales_month, total_sales, ROUND((total_sales - LAG(total_sales) OVER (ORDER BY sales_month)) /
NULLIF(LAG(total_sales) OVER (ORDER BY sales_month),0) * 100, 2) AS difference_pct

FROM monthly_sales;
```

Variant sample result:

sales_month	total_sales	difference_pct
2026-02-01	268410.00	12.45
2026-03-01	302950.00	8.72

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 103 - Current Year vs Previous Year

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for current year vs previous year that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when building a period-based report where users need to compare performance by month, year, trend, YTD, or rolling period. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
WITH yearly_sales AS (

SELECT EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales
FROM sales GROUP BY EXTRACT(YEAR FROM order_date)

)

SELECT sales_year, total_sales, LAG(total_sales) OVER (ORDER BY sales_year) AS previous_year_sales
FROM yearly_sales;
```

Sample result:

sales_year	total_sales	previous_year_sales
2025	268410.00	2025
2026	302950.00	2026

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: management wants to compare current year performance with last year using the same business logic. Example request: 'Can you use a practical SQL pattern for current year vs previous year without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```
WITH yearly_sales AS (  
  SELECT EXTRACT(YEAR FROM order_date) AS sales_year, SUM(sales_amount) AS total_sales  
  FROM sales GROUP BY EXTRACT(YEAR FROM order_date)  
)  
  
SELECT sales_year, total_sales, ROUND((total_sales - LAG(total_sales) OVER (ORDER BY sales_year)) /  
  NULLIF(LAG(total_sales) OVER (ORDER BY sales_year),0) * 100, 2) AS growth_pct  
FROM yearly_sales;
```

Variant sample result:

sales_year	total_sales	growth_pct
2025	268410.00	12.45
2026	302950.00	8.72

Common mistakes to avoid: Check date formats and period boundaries carefully. Time components and fiscal calendars can change results.

QUERY N. 104 - Find Gaps in Numeric Sequences

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for find gaps in numeric sequences that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```
SELECT invoice_number, LEAD(invoice_number) OVER (ORDER BY invoice_number) AS next_invoice_number  
FROM invoices;
```

Sample result:

invoice_number	next_invoice_number
INV-100451	INV-100454
INV-100454	INV-100903

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a intermediate analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you use a practical SQL pattern for find gaps in numeric sequences without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.


```

WITH numbered AS (SELECT invoice_number, LEAD(invoice_number) OVER (ORDER BY invoice_number) AS
next_invoice_number FROM invoices)

SELECT invoice_number, next_invoice_number, next_invoice_number - invoice_number AS gap_size

FROM numbered

WHERE next_invoice_number - invoice_number > 1;

```

Variant sample result:

invoice_number	next_invoice_number	gap_size
INV-100451	INV-100454	3
INV-100454	INV-100903	4

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

QUERY N. 105 - Categorize Records Using CASE WHEN

Difficulty: Advanced Practical

Best for: Intermediate Analyst, BI Developer, Data Analyst, Power BI Model Builder

Problem: You need a practical advanced SQL pattern for categorize records using case when that can be reviewed, adapted, and reused in reporting work.

When to use it: Use this when a normal GROUP BY is not enough and you need ranking, latest records, period comparisons, cumulative totals, deduplication, or row-by-row logic. Adapt the table and field names to your database, then review the returned columns before using the result.

SQL:

```

SELECT order_id, sales_amount, CASE WHEN sales_amount >= 10000 THEN 'High' WHEN sales_amount >= 1000 THEN
'Medium' ELSE 'Low' END AS order_value_segment

FROM sales;

```

Sample result:

order_id	sales_amount	order_value_segment
SO-100451	182,450.00	High
SO-100782	52,300.00	Medium

Customize:

Replace the table names with your real company tables.

Replace the output fields and join keys with the equivalent fields in your database.

Adjust dates, statuses, currencies, thresholds, and grouping levels according to your business rules.

Explanation: The query returns a focused result using clear output column names. The sample result below uses the same column names as the SQL output, so users can immediately understand what the query produces.

Practical example: Common use case: a intermediate analyst needs this query to answer a recurring business question without rewriting the logic from scratch. Example request: 'Can you use a practical SQL pattern for categorize records using case when without rewriting the logic from scratch?'

Useful variant:

This variant adds a practical filter, extra business detail, comparison, or grouping level while keeping the same business purpose.

```

SELECT order_value_segment, COUNT(*) AS order_count, SUM(sales_amount) AS total_sales

FROM (SELECT order_id, sales_amount, CASE WHEN sales_amount >= 10000 THEN 'High' WHEN sales_amount >= 1000 THEN
'Medium' ELSE 'Low' END AS order_value_segment FROM sales) x

GROUP BY order_value_segment;

```

Variant sample result:

order_id	sales_amount	order_value_segment
SO-100451	268410.00	High

order_id	sales_amount	order_value_segment
SO-100782	302950.00	Medium

Common mistakes to avoid: Do not copy the query blindly. Replace table names, field names, date filters, and business thresholds before using it.

13. SQL Debugging Guide

Why a JOIN increases rows

The joined table probably has multiple matching rows for the same key. Check duplicates before joining.

LEFT JOIN vs INNER JOIN

INNER JOIN keeps only matching records. LEFT JOIN keeps all records from the left table and is better for missing-data checks.

GROUP BY errors

Every selected non-aggregated field normally needs to appear in GROUP BY.

NULL values in calculations

Any calculation involving NULL can return NULL. Use COALESCE only when the fallback value is business-correct.

Date conversion failures

The text format may not match the expected date format. Use safe conversion functions where available.

Testing step by step

Start with row counts, add filters one by one, add joins, then aggregate.

14. Final SQL Cheat Sheet

Filter dates safely

```
WHERE order_date >= DATE '2026-01-01' AND order_date < DATE '2026-02-01'
```

Count rows

```
SELECT COUNT(*) AS row_count FROM your_table;
```

Find duplicates

```
GROUP BY key_field HAVING COUNT(*) > 1
```

Replace NULL

```
COALESCE(value, 0)
```

Conditional label

```
CASE WHEN amount >= 10000 THEN 'High' ELSE 'Low' END
```

Running total

```
SUM(amount) OVER (ORDER BY date_field)
```

Latest record

```
ROW_NUMBER() OVER (PARTITION BY key ORDER BY date_field DESC)
```