SQL

Advanced Querying

★★★★★ *With SQL Pocket Guide Author Alice Zhao*

MAVEN ANALYTICS

COURSE STRUCTURE

This is a **hands-on, project-based** course for students looking for a practical approach to learning advanced SQL querying techniques

Additional resources include:

- **Downloadable PDF** to serve as a helpful reference when you're offline or on the go
- **Quizzes** & **Assignments** to test and reinforce key concepts, with step-by-step solutions
- **Interactive demos** to keep you engaged and apply your skills throughout the course

COURSE OUTLINE

1	**SQL Basics Review**	*Review the big 6 clauses of a SQL query along with other commonly used keywords like LIMIT, DISTINCT, and more*
2	**Multi-Table Analysis**	*Review JOIN basics (INNER, LEFT, RIGHT, OUTER) and introduce variations like self joins, CROSS JOINs, and more*
3	**Subqueries & CTEs**	*Learn how to write subqueries and Common Table Expressions and understand the best situations for using certain techniques*
4	**Window Functions**	*Introduce window functions to perform calculations across a set of rows and discuss various function options and applications*
5	**Functions by Data Type**	*Discover the many SQL functions that can be applied to fields of numeric, datetime, string, and NULL data types*
6	**Data Analysis Applications**	*Apply advanced querying techniques to common data analysis scenarios, including pivoting data, rolling calculations, and more*

PREVIEW: FINAL PROJECT

THE SITUATION

You've just been hired as a Data Analyst Intern for **Major League Baseball** (MLB), who has recently gotten access to a large amount of historical player data

THE ASSIGNMENT

You have access to decades worth of data including player statistics like schools attended, salaries, teams played for, height and weight, and more

Your task is to use **advanced SQL querying techniques** to track how player statistics have changed over time and across different teams in the league

THE OBJECTIVES

1. What **schools** do MLB players attend?
2. How much do teams spend on player **salaries**?
3. What does each player's **career** look like?
4. How do player **attributes** compare?

Data source: *http://seanlahman.com*

SETTING EXPECTATIONS

This course focuses on **querying**, not database management

- *We'll cover concepts commonly used by analysts & data scientists to write complex SELECT statements*
- *We will NOT cover concepts more often used for database management such as stored procedures, triggers, user-defined functions (UDFs), etc. – we cover most of those in our Advanced MySQL DBA course!*

This is an **advanced** course, so we'll only quickly review the basics

- *It is strongly recommended that you complete our MySQL Data Analysis course before taking this one*

You'll be learning common **data analysis applications** within SQL

- *We'll apply SQL code to use cases that can be done within other tools like Excel or Python, including summarizing and pivoting, dealing with duplicate data, performing rolling calculations, and more*

We'll cover **general SQL syntax**, but the demos will be in **MySQL**

- *The SQL concepts taught in this course will apply to any relational database management system (Oracle, PostgreSQL, SQL Server, SQLite, etc.), so you are welcome to use the SQL editor of your choice*

INSTALLATION & SETUP

INSTALLATION & SETUP

In this section we'll discuss where you can write SQL code, walk through the **MySQL & MySQL Workbench** installation process and help you load the data for this course

TOPICS WE'LL COVER:

- Where to Write SQL Code
- Installing MySQL & MySQL Workbench
- Getting Started with MySQL Workbench
- Loading Data For This Course

GOALS FOR THIS SECTION:

- Pick a place to write SQL code
- Install MySQL and MySQL Workbench *(optional)*
- Create a MySQL Connection to be able to start writing SQL code *(optional)*
- Load the tables for this course into your SQL editor

WHERE CAN I WRITE SQL CODE?

There are **many options** when it comes to picking a place to write SQL code:

Where to Write SQL Code

Installing MySQL & MySQL Workbench

Getting Started with MySQL

Loading Data for This Course

1 RDBMS (Relational Database Management System)

- Open source: MySQL, PostgreSQL, SQLite, etc.
- Proprietary: Oracle Database, Microsoft SQL Server, etc.

2 SQL Editor

- Open source: MySQL Workbench, pgAdmin, DBeaver, etc.
- Proprietary: SQL Server Management Studio, Toad, etc.

3 Within Another Language

- Python, R, Java, etc.

4 Online SQL Editors

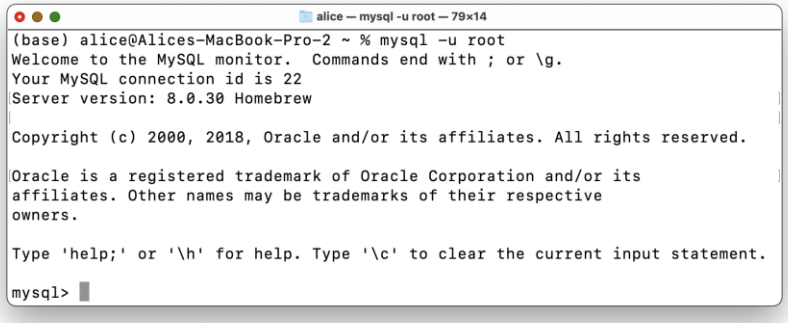
PRO TIP: To quickly start writing SQL code with no installations, you can also write simple SQL queries within some websites – just do a search for "online SQL editor"

WHERE CAN I WRITE SQL CODE?

- Where to Write SQL Code
- Installing MySQL & MySQL Workbench
- Getting Started with MySQL
- Loading Data for This Course

If you'd like to see the exact same interface as the demos in this course, you'll need to **download & install** two programs:

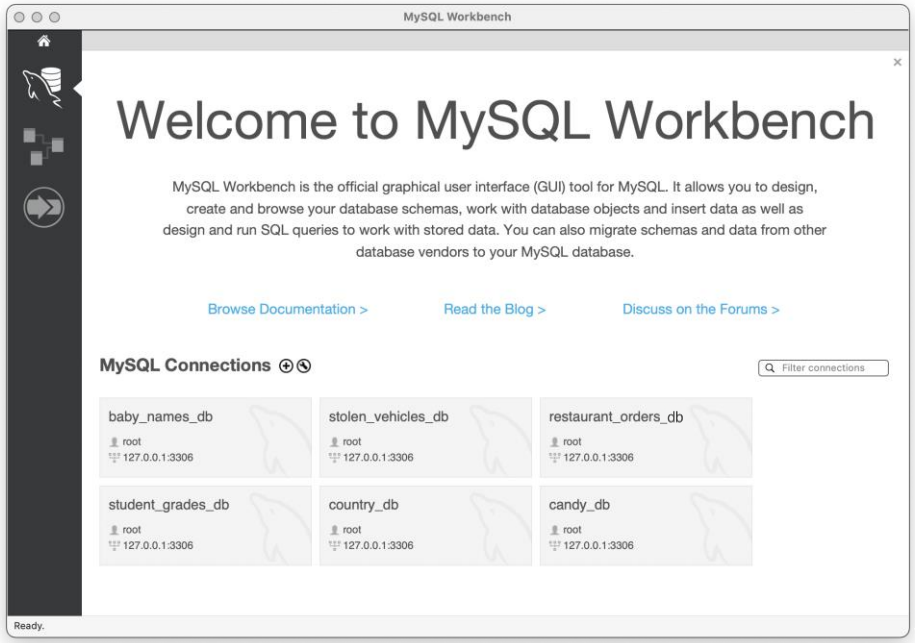
1 RDBMS: **MySQL**



These installs will take ~10 min to complete

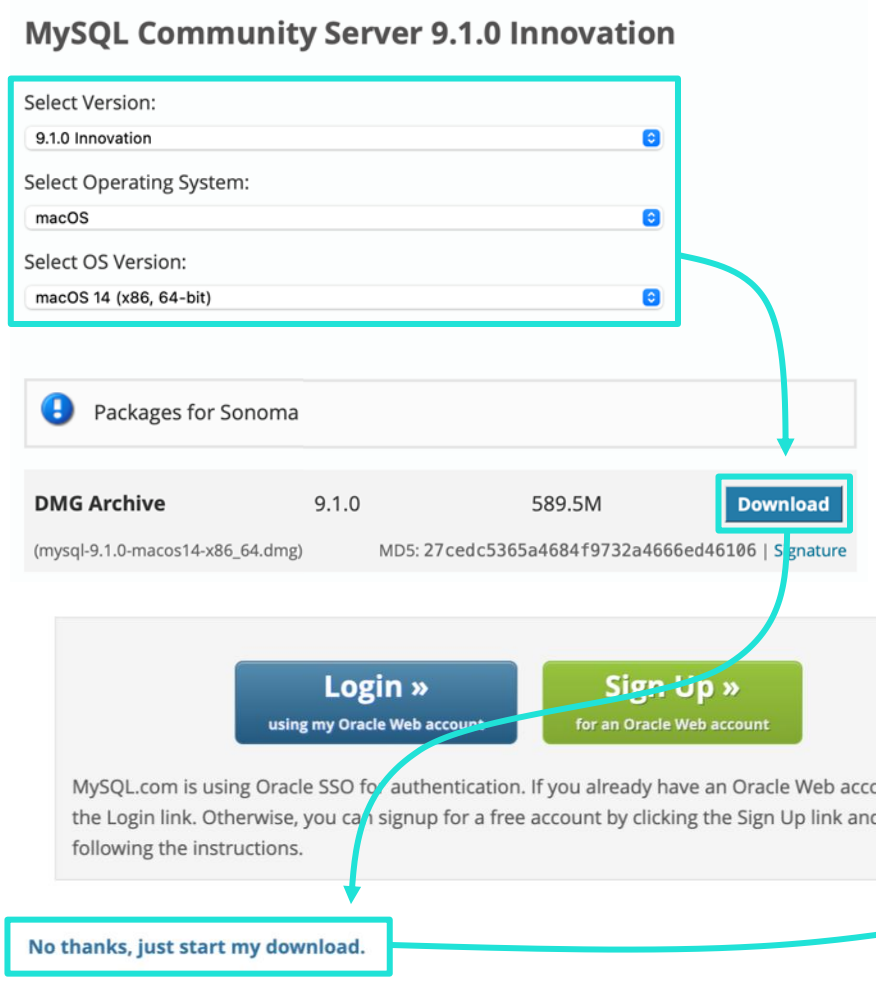
If you're already working with another SQL editor, feel free to use that one for this course

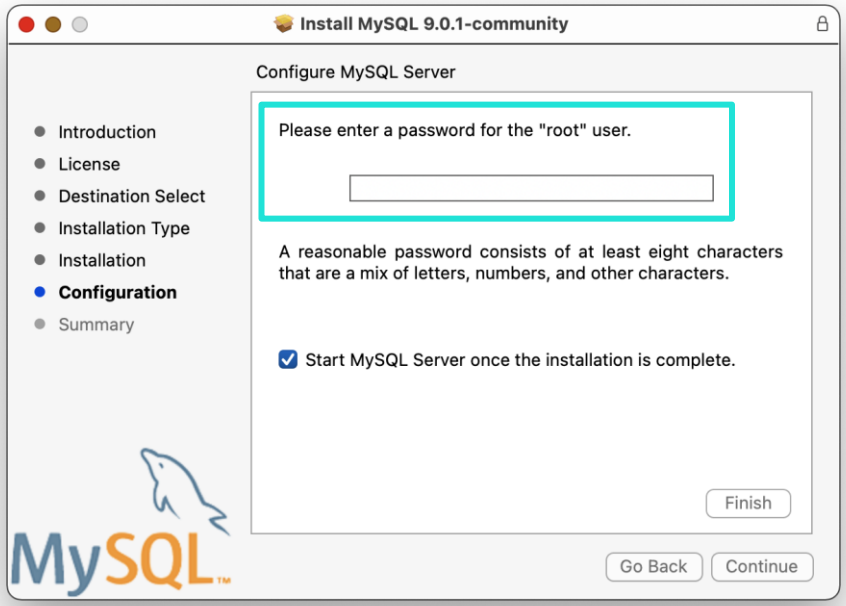
2 SQL Editor: **MySQL Workbench**

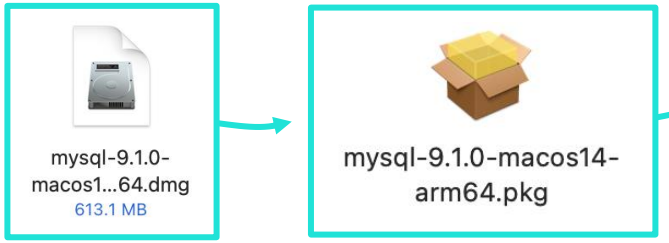


INSTALLING MYSQL (MAC)

- Where to Write SQL Code
- Installing MySQL & MySQL Workbench
- Getting Started with MySQL
- Loading Data for This Course







INSTALLING MYSQL (MAC)

Where to Write SQL Code

Installing MySQL & MySQL Workbench

Getting Started with MySQL

Loading Data for This Course

1. Go to **dev.mysql.com/downloads/mysql/** to download MySQL Community Server

2. Select the following menu options and download the **DMG Archive** version
   - **Version:** 9.1.0 Innovation *(latest version)*
   - **Operating System:** macOS
   - **OS Version:** x86 or ARM *(if you're on an M1 / M2 / M3 Mac)*

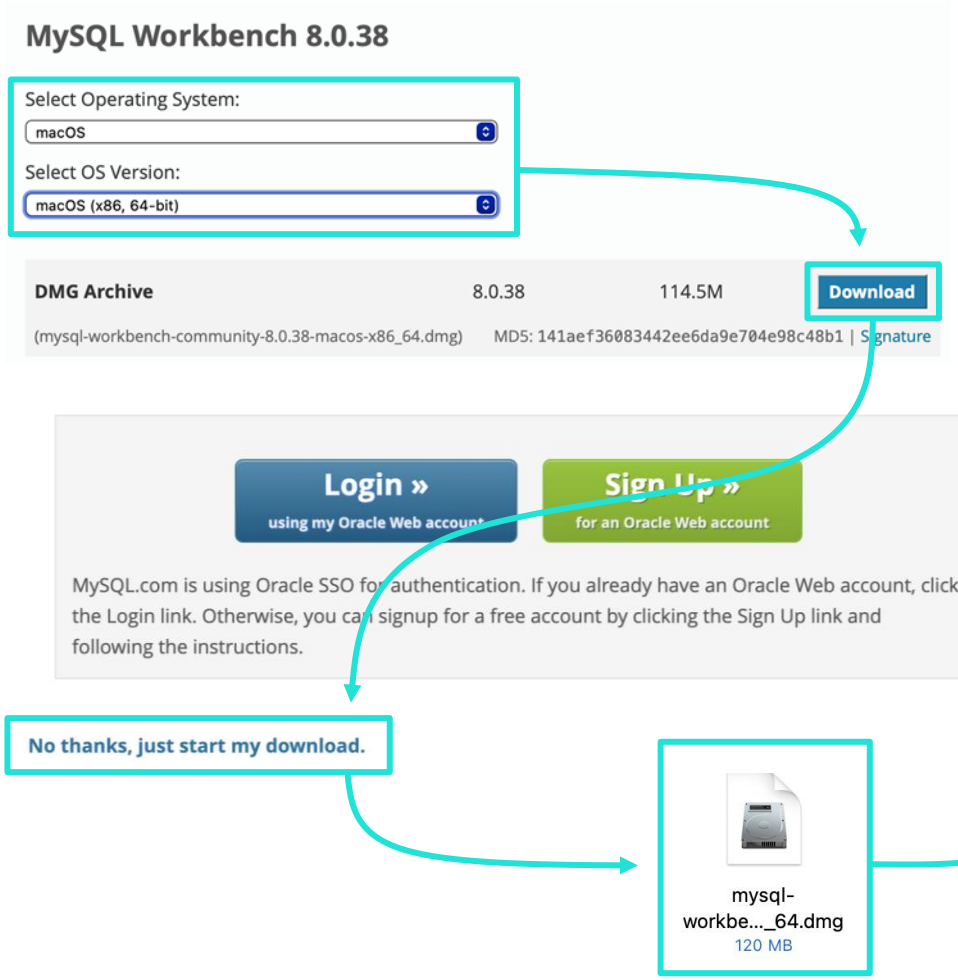
3. No need to Login or Sign Up, just click **"No thanks, just start my download"**

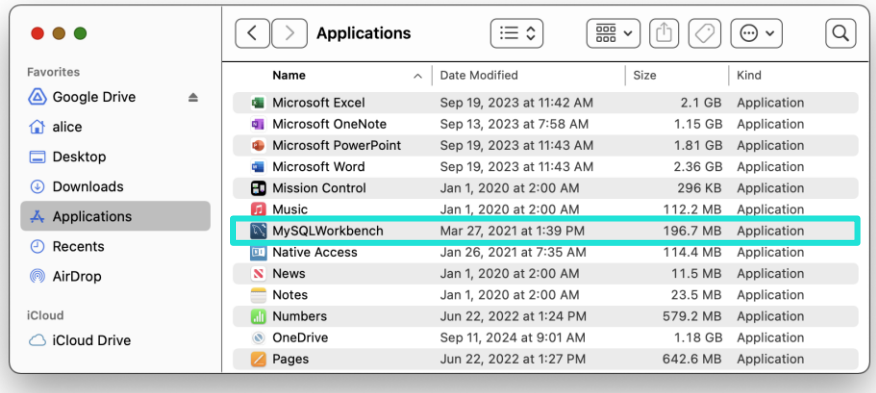
4. Find the install file in your downloads, then double click to run the installer package

5. Click through each install step, leaving defaults unless you need customized settings
   - **NOTE:** Make sure you store your **root password** somewhere, you'll need this later!

INSTALLING MYSQL WORKBENCH (MAC)

- Where to Write SQL Code
- Installing MySQL & MySQL Workbench
- Getting Started with MySQL
- Loading Data for This Course





INSTALLING MYSQL WORKBENCH (MAC)

Where to Write SQL Code

Installing MySQL & MySQL Workbench

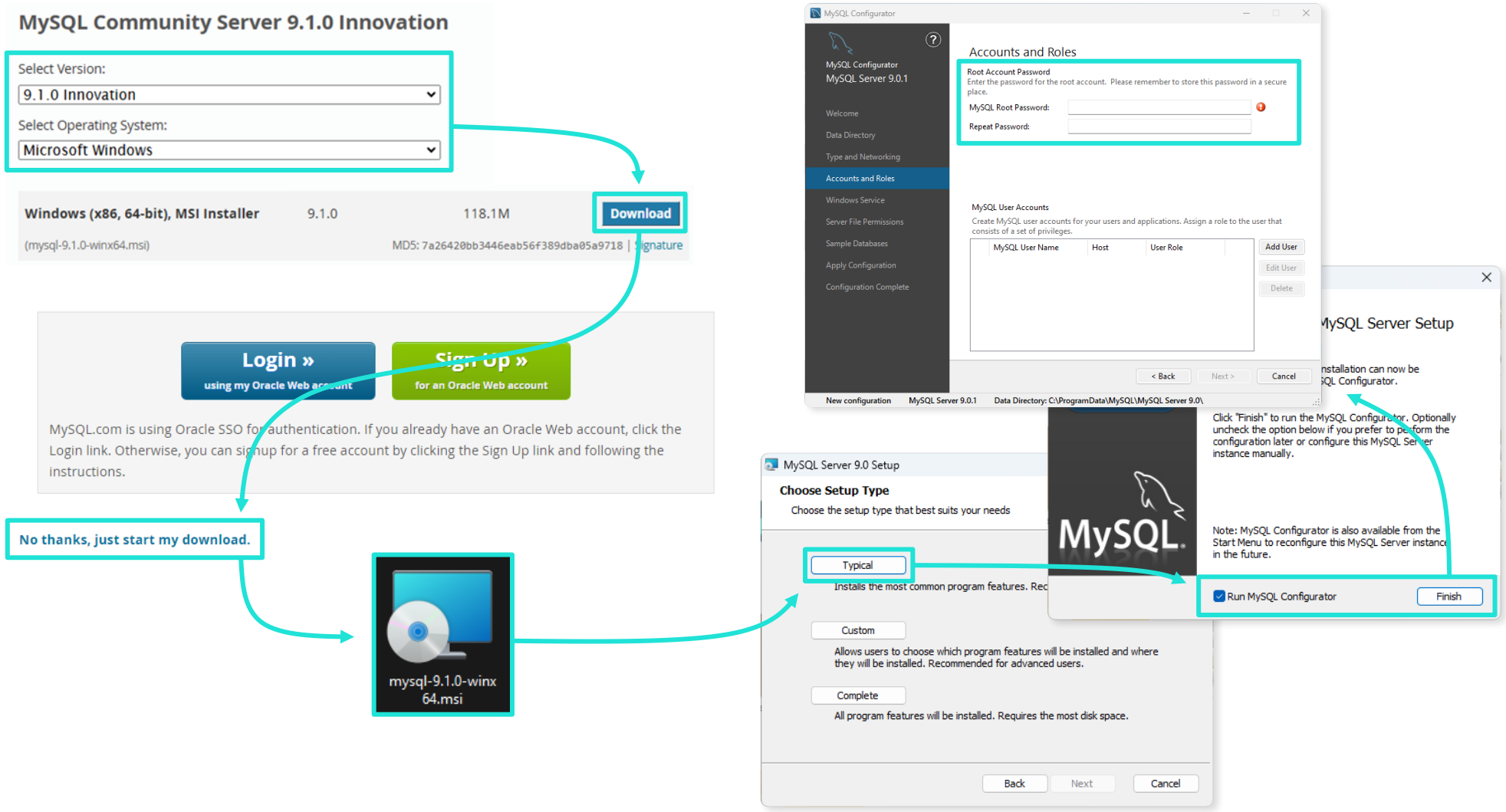
Getting Started with MySQL

Loading Data for This Course

1. Go to **dev.mysql.com/downloads/workbench/** to download MySQL Workbench
2. Select the following menu options and download the **DMG Archive** version
   - **Operating System:** macOS
   - **OS Version:** x86 or ARM *(if you're on an M1 / M2 / M3 Mac)*
3. No need to Login or Sign Up, just click **"No thanks, just start my download"**
4. Find the install file in your downloads, then double click to open the installer package
5. Drag the MySQL Workbench icon into the **Applications** folder icon
6. Open the **MySQL Workbench** app from your Applications folder

INSTALLING MYSQL (PC)

- Where to Write SQL Code
- Installing MySQL & MySQL Workbench
- Getting Started with MySQL
- Loading Data for This Course



INSTALLING MYSQL (PC)

Where to Write SQL Code

Installing MySQL & MySQL Workbench

Getting Started with MySQL

Loading Data for This Course

1. Go to **dev.mysql.com/downloads/mysql/** to download MySQL Community Server

2. Select the following menu options and download the **MSI Installer** version
   - **Version:** 9.1.0 Innovation *(latest version)*
   - **Operating System:** Microsoft Windows

3. No need to Login or Sign Up, just click **"No thanks, just start my download"**

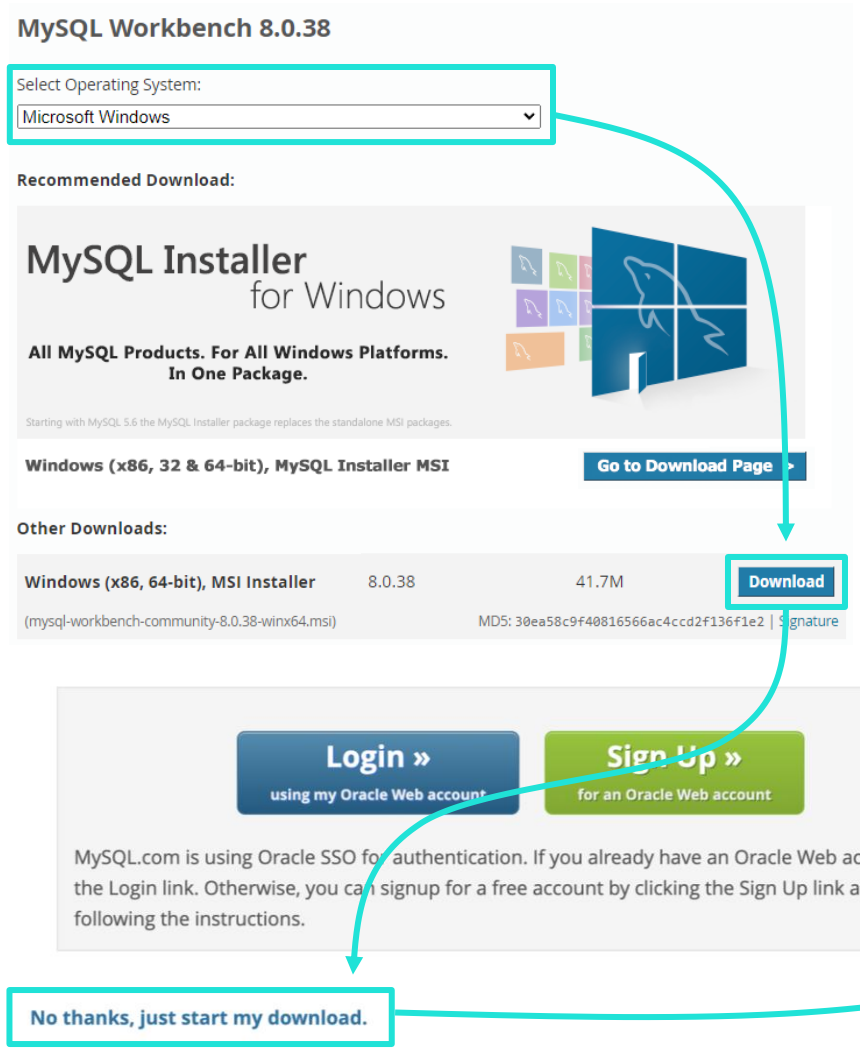
4. Find the install file in your downloads, then double click to open the installer package

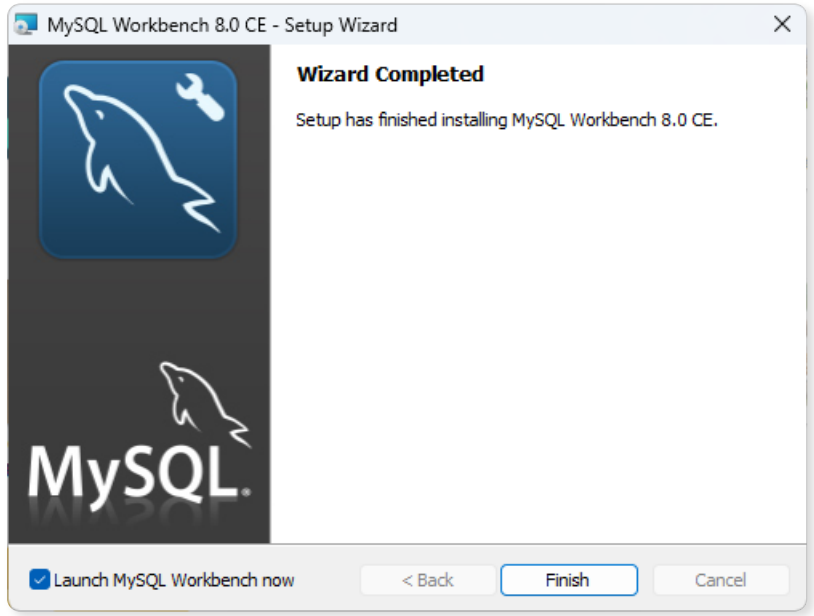
5. Click through each step, leaving defaults unless you need customized settings
   - Choose Setup Type: **Typical**
   - On the last step, make sure **Run MySQL Configurator** is checked and click **Finish**

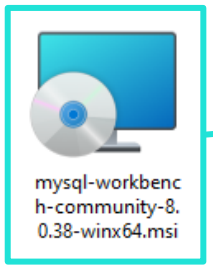
6. In the MySQL Configurator pop up window, click through each step, leaving defaults unless you need customized settings
   - **NOTE:** Make sure you store your **root password** somewhere, you'll need this later!

INSTALLING MYSQL WORKBENCH (PC)

- Where to Write SQL Code
- Installing MySQL & MySQL Workbench
- Getting Started with MySQL
- Loading Data for This Course







INSTALLING MYSQL WORKBENCH (PC)

Where to Write SQL Code

Installing MySQL & MySQL Workbench

Getting Started with MySQL

Loading Data for This Course

1. Go to **dev.mysql.com/downloads/workbench/** to download MySQL Workbench

2. Select the following menu options and download the **MSI Installer** version
   - **Operating System:** Microsoft Windows

3. No need to Login or Sign Up, just click **"No thanks, just start my download"**

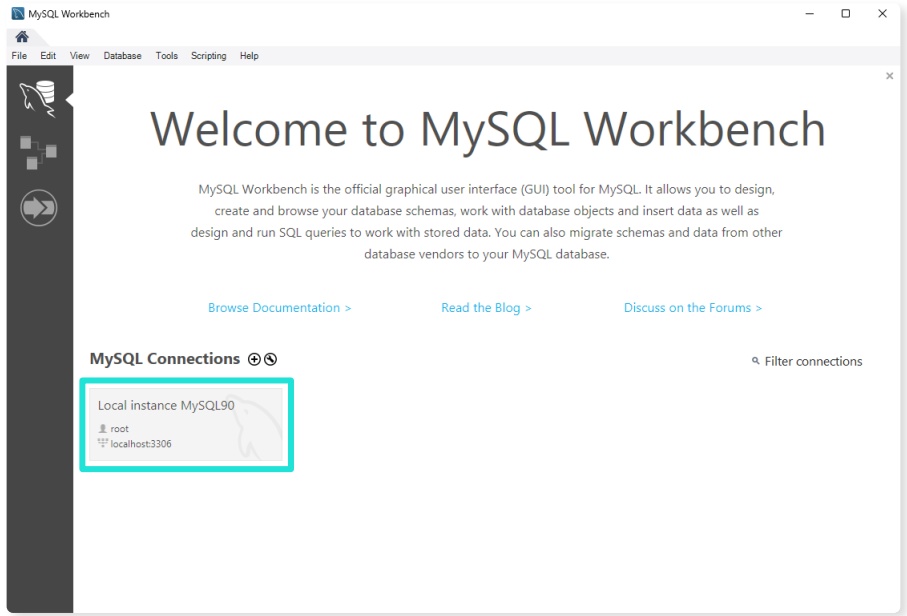
4. Find the install file in your downloads, then double click to open the installer package

5. Click through each step, leaving defaults unless you need customized settings

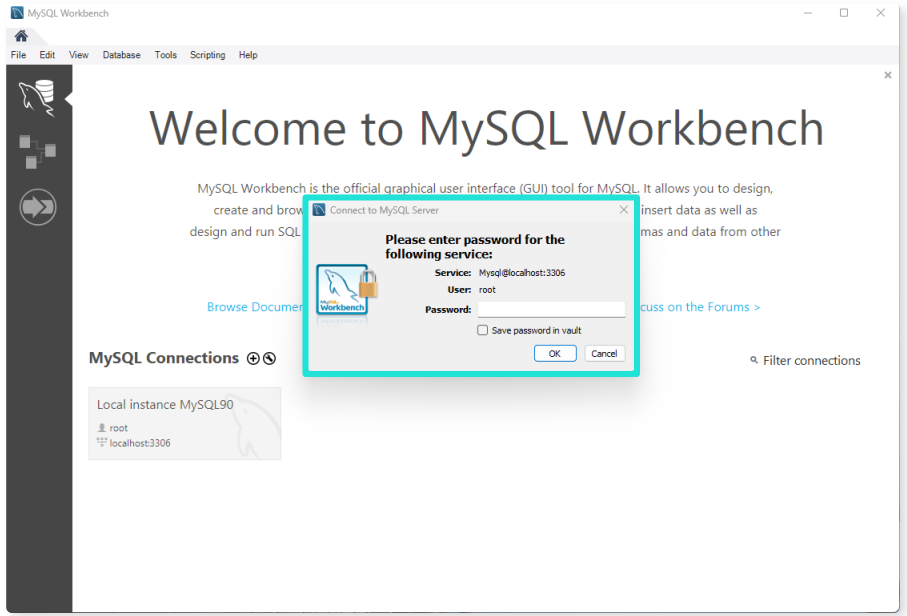
GETTING STARTED WITH MYSQL WORKBENCH

- Where to Write SQL Code
- Installing MySQL & MySQL Workbench
- Getting Started with MySQL
- Loading Data for This Course

Double click on the tile below "MySQL Connections" to **connect MySQL Workbench to MySQL**



Enter the **password** you chose during the installation



🎉 **You're ready to write SQL code!**

MYSQL WORKBENCH INTERFACE (MAC VS. PC)

Where to Write SQL Code

Installing MySQL & MySQL Workbench

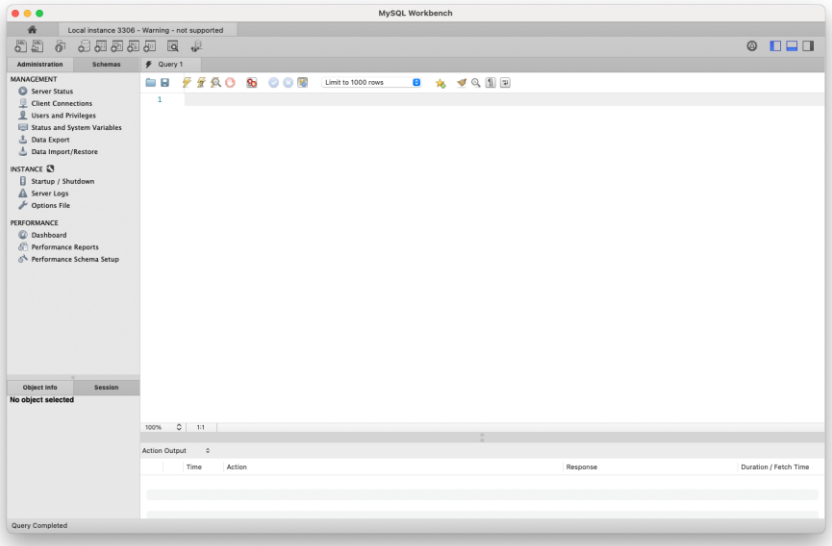
Getting Started with MySQL

Loading Data for This Course

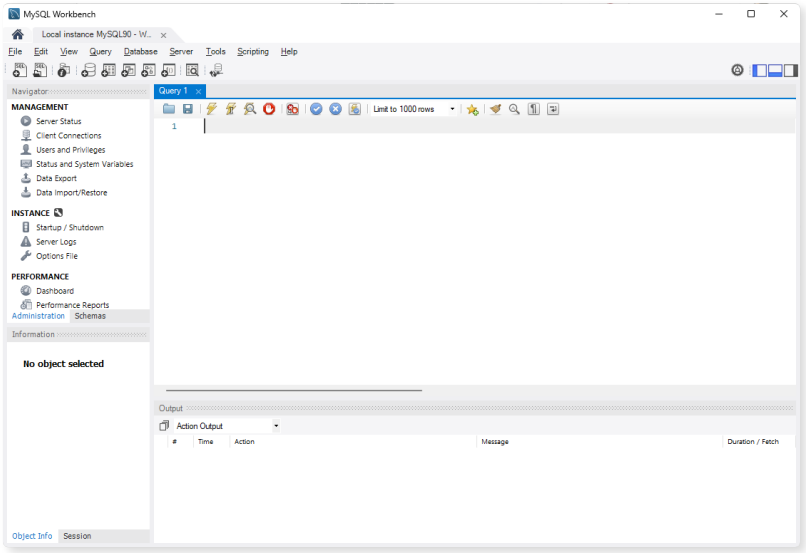
MySQL Workbench **looks slightly different** on Mac vs. PC, but everything you need is found in the same place

- While the course is recorded on a Mac, you should have no problem keeping up on a PC

Mac Interface



PC Interface



LOADING DATA FOR THIS COURSE

- Where to Write SQL Code
- Installing MySQL & MySQL Workbench
- Getting Started with MySQL
- Loading Data for This Course

You have two options when it comes to **loading data** for this course:

create_statements_mysql.sql
create_statements.sql

SQL Scripts

This code is MySQL specific

This code will work in any RDBMS

```
-- Specify the schema
DROP SCHEMA IF EXISTS maven_advanced_sql;
CREATE SCHEMA maven_advanced_sql;
USE maven_advanced_sql;

--
-- Table structure for table `students`
--

CREATE TABLE students (
    id INT PRIMARY KEY,
    student_name VARCHAR(50),
    grade_level INT,
    gpa DECIMAL(2, 1),
    school_lunch VARCHAR(3),
    birthday DATE,
    email VARCHAR(100)
);
```

If you need to edit a SQL script, you can do so within your SQL editor or within a text editor, like Visual Studio Code

CSV Files

- country_stats.csv
- customers.csv
- happiness_scores_current.csv
- happiness_scores.csv
- inflation_rates.csv
- orders.csv
- players.csv
- products.csv
- salaries.csv
- school_details.csv
- schools.csv
- student_grades.csv
- students.csv

You can load these into the SQL editor of your choice

SQL BASICS REVIEW

SQL BASICS REVIEW

In this section we'll quickly **review the basics of SELECT statements** so we're on the same page going into advanced querying concepts

TOPICS WE'LL COVER:

The Big 6

Common SQL Keywords

GOALS FOR THIS SECTION:

- Review the "Big 6" SQL clauses
- Review common keywords used in SQL queries

THE BIG 6

The **Big 6** are the foundational clauses used in SQL queries:

The Big 6

Common SQL Keywords

```
SELECT    grade_level,
          AVG(gpa) AS avg_gpa
FROM      students
WHERE     school_lunch = 'Yes'
GROUP BY  grade_level
HAVING    avg_gpa < 3.3
ORDER BY  grade_level;
```

← Column(s) to display

← Table(s) to pull data from

← Criteria to filter the rows by

← Column to group the rows by

← Criteria to filter the grouped rows by

← Column to sort values by

*The clauses must **always be written in this order***
(one way to remember the order is the pneumonic:
***S**tart **F**ridays **W**ith **G**randma's **H**omemade **O**atmeal)*

The only required clause in a SQL query is the SELECT clause

COMMON SQL KEYWORDS

The Big 6

Common SQL Keywords

In addition to the Big 6, there are **common SQL keywords** used in queries

These are popular keywords found in the SELECT clause:

```
SELECT   DISTINCT grade_level
FROM     students;
```

DISTINCT returns unique values

```
SELECT   COUNT(DISTINCT grade_level)
FROM     students;
```

Aggregate functions like COUNT, SUM, AVG, MIN, MAX are used to make calculations

```
SELECT   MAX(gpa) - MIN(gpa) AS gpa_range
FROM     students;
```

AS renames a column or table to an alias

Math operators include +, -, x, /, %

COMMON SQL KEYWORDS

The Big 6

Common SQL Keywords

In addition to the Big 6, there are **common SQL keywords** used in queries

These are popular keywords found in the WHERE clause:

```
SELECT  *
FROM    students
WHERE   grade_level < 12 AND school_lunch = 'Yes';
```

Comparison operators include =, !=, <>, <, <=, >, >=

Logical operators include AND, OR, and NOT

```
SELECT  *
FROM    students
WHERE   grade_level IN (10, 11, 12);
```

```
SELECT  *
FROM    students
WHERE   email LIKE '%.com';
```

Comparison keywords include, IN, LIKE, BETWEEN ... AND, IS NULL and more

COMMON SQL KEYWORDS

In addition to the Big 6, there are **common SQL keywords** used in queries

These are other popular keywords :

The Big 6

Common SQL Keywords

```
SELECT    student_name, gpa
FROM      students
ORDER BY  gpa DESC;
```

DESC stands for "descending", while the default order is ASC (ascending)

```
SELECT  *
FROM    students
LIMIT   10;
```

LIMIT specifies the number of rows in the output (TOP in SQL Server and FETCH FIRST in Oracle)

COMMON SQL KEYWORDS

The Big 6

Common SQL Keywords

In addition to the Big 6, there are **common SQL keywords** used in queries

These are other popular keywords :

```
SELECT   student_name, grade_level,
         CASE WHEN grade_level = 9 THEN 'Freshman'
              WHEN grade_level = 10 THEN 'Sophomore'
              WHEN grade_level = 11 THEN 'Junior'
              ELSE 'Senior' END AS student_class
FROM     students;
```

This creates a new "student_class" column based on the grade level for each student

Case statements use the following syntax to do IF-ELSE logic within SQL:
CASE WHEN … THEN … WHEN … THEN … ELSE … END

student_name	grade_level	student_class
Abby Johnson	10	Sophomore
Bob Smith	11	Junior
Catherine Davis	12	Senior
Daniel Brown	9	Freshman
Eva Martinez	10	Sophomore

MULTI-TABLE ANALYSIS

MULTI-TABLE ANALYSIS

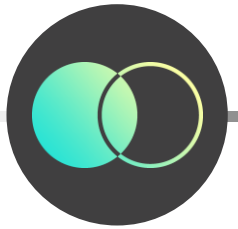
In this section we'll talk about **combining multiple tables** in a single SQL query, such as using JOINs to add new columns from related tables, and UNIONs to add new rows

TOPICS WE'LL COVER:

- Multi-Table Analysis
- JOIN Basics
- JOIN Variations
- UNION Basics

GOALS FOR THIS SECTION:

- Understand the difference between combining tables using a JOIN and a UNION
- Review the different types of JOINs
- Learn how and when to apply JOINs and UNIONs



WORKING WITH MULTIPLE TABLES

- Multi-Table Analysis
- JOIN Basics
- JOIN Variations
- UNION Basics

Simple queries will return data from a single table, but in practice it's helpful to **combine multiple tables** to analyze data properly

```
SELECT *
FROM happiness_scores;
```

year	country	region	happiness_score
2015	Afghanistan	South Asia	3.575
2015	Albania	Central and Eastern Europe	4.959
2015	Algeria	Middle East and North Africa	5.605
2015	Angola	Sub-Saharan Africa	4.033
2015	Argentina	Latin America and Caribbean	6.574
2015	Armenia	Central and Eastern Europe	4.350
2015	Australia	North America and ANZ	7.284
2015	Austria	Western Europe	7.200
2015	Azerbaijan	Central and Eastern Europe	5.212
2015	Bahrain	Middle East and North Africa	5.960
2015	Bangladesh	South Asia	4.694

```
SELECT *
FROM country_stats;
```

country	continent	population	urban_population
Afghanistan	Asia	38041754	9797273
Albania	Europe	2854191	1747593
Algeria	Africa	43053054	31510100
Angola	Africa	31825295	21061025
Argentina	South America	44938712	41339571
Armenia	Asia	2957731	1869848
Australia	Australia/Oceania	25766605	21844756
Austria	Europe	8877067	5194416
Azerbaijan	Asia	10023318	5616165
Bahrain	Asia	1501635	1467109
Bangladesh	Asia	1673108…	60987417

WORKING WITH MULTIPLE TABLES

Multi-Table Analysis

JOIN Basics

JOIN Variations

UNION Basics

There are two ways to combine multiple tables into a single table for analysis:

- **JOIN** adds related columns from one table to another, based on common columns
- **UNION** stacks the rows from multiple tables with the same column structure

year	country	happiness_score
2015	Afghanistan	3.575
2015	Albania	4.959
2015	Algeria	5.605
2015	Angola	4.033
2015	Argentina	6.574
2015	Armenia	4.350
2015	Australia	7.284
2015	Austria	7.200
2015	Azerbaijan	5.212

country	continent	population
Afghanistan	Asia	38041754
Albania	Europe	2854191
Algeria	Africa	43053054
Angola	Africa	31825295
Argentina	South America	44938712
Armenia	Asia	2957731
Australia	Australia/Oceania	25766605
Austria	Europe	8877067
Azerbaijan	Asia	10023318

year	country	happiness_score	continent	population
2015	Afghanistan	3.575	Asia	38041754
2015	Albania	4.959	Europe	2854191
2015	Algeria	5.605	Africa	43053054
2015	Angola	4.033	Africa	31825295
2015	Argentina	6.574	South America	44938712
2015	Armenia	4.350	Asia	2957731
2015	Australia	7.284	Australia/Oceania	25766605
2015	Austria	7.200	Europe	8877067
2015	Azerbaijan	5.212	Asia	10023318

*The continent and population columns were added using a **JOIN**, based on the matching country column*

country	year	happiness_score
Afghanistan	2015	3.575
Albania	2015	4.959
Algeria	2015	5.605
Angola	2015	4.033
Argentina	2015	6.574
Armenia	2015	4.350

country	year	happiness_score
Afghanistan	2016	3.360
Albania	2016	4.655
Algeria	2016	6.355
Angola	2016	3.866
Argentina	2016	6.650
Armenia	2016	4.360

country	year	happiness_score
Afghanistan	2015	3.575
Albania	2015	4.959
Algeria	2015	5.605
Angola	2015	4.033
Argentina	2015	6.574
Armenia	2015	4.350
Afghanistan	2016	3.360
Albania	2016	4.655
Algeria	2016	6.355
Angola	2016	3.866
Argentina	2016	6.650
Armenia	2016	4.360

*The rows from the two tables with the same columns were stacked using a **UNION***

BASIC JOINS

Multi-Table Analysis

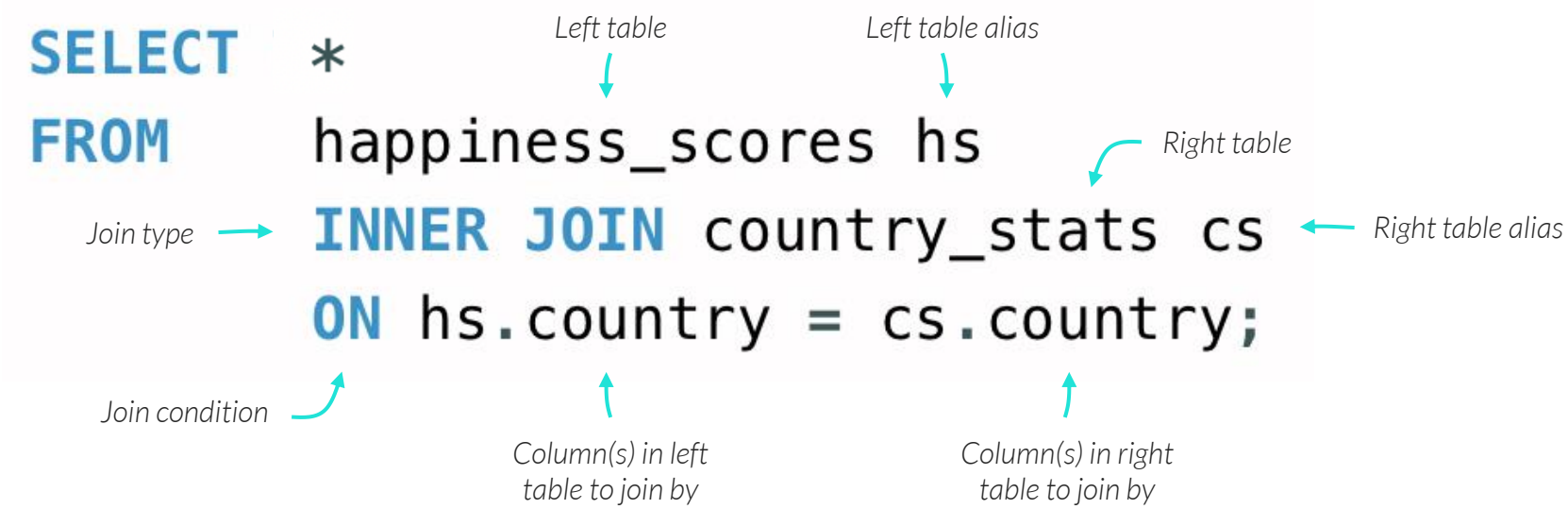
JOIN Basics

JOIN Variations

UNION Basics

Use **JOIN** to combine two tables based on common values in a column(s)

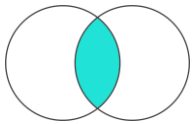
- The tables must have at least one column with matching values
- Basic JOIN options include INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN



BASIC JOIN TYPES

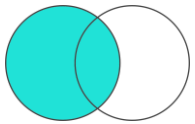
- Multi-Table Analysis
- JOIN Basics
- JOIN Variations
- UNION Basics

These are the four **basic JOIN types** in SQL:



INNER

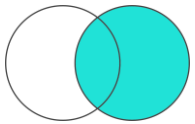
Returns records that exist in BOTH tables, and excludes unmatched records from either table



LEFT

Returns ALL records from the LEFT table, and any matching records from the RIGHT table

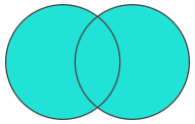
These are the most common



RIGHT

Returns ALL records from the RIGHT table, and any matching records from the LEFT table

This is less often used in practice; switch the tables and use a LEFT JOIN instead



FULL OUTER

Returns ALL records from BOTH tables, including non-matching records

While INNER and LEFT JOINs are supported in all RDBMS's, **RIGHT and FULL OUTER are not** – for example, SQLite does not support RIGHT JOINs, and MySQL and SQLite do not support FULL OUTER JOINs

BASIC JOIN TYPES

- Multi-Table Analysis
- JOIN Basics
- JOIN Variations
- UNION Basics

Left Table *n=5*

country	happiness_score
Canada	6.961
China	5.818
India	4.036
Mexico	6.330
United States	6.894

Right Table *n=3*

country	continent
Australia	Australia/Oceania
Brazil	South America
United States	North America

INNER *n=1*

country	happiness_score	country	continent
United States	6.894	United States	North America

FULL OUTER *n=7*

country	happiness_score	country	continent
Canada	6.961	NULL	NULL
China	5.818	NULL	NULL
India	4.036	NULL	NULL
Mexico	6.330	NULL	NULL
NULL	NULL	Australia	Australia/Oceania
NULL	NULL	Brazil	South America
United States	6.894	United States	North America

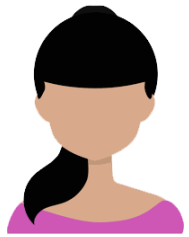
LEFT *n=5*

country	happiness_score	country	continent
Canada	6.961	NULL	NULL
China	5.818	NULL	NULL
India	4.036	NULL	NULL
Mexico	6.330	NULL	NULL
United States	6.894	United States	North America

RIGHT *n=3*

country	happiness_score	country	continent
NULL	NULL	Australia	Australia/Oceania
NULL	NULL	Brazil	South America
United States	6.894	United States	North America

ASSIGNMENT: BASIC JOINS



NEW MESSAGE

October 31, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Compare rows in tables**

Hi there,

We've learned that there's a discrepancy between our orders and products tables in the candy database.

Could you use your JOIN knowledge to figure out which products exist in one table, but not the other?

Thanks, and Happy Halloween!
Mandy

Reply Forward

Results Preview

product_id	product_name	product_id_in_orders
SUG-LOO-45000	Loopy Lollipops	NULL
SUG-NER-92001	Tropical Nerds	NULL
SUG-PIX-62000	Pixy Stix	NULL

SOLUTION: BASIC JOINS

NEW MESSAGE

October 31, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Compare rows in tables**

Hi there,

We've learned that there's a discrepancy between our orders and products tables in the candy database.

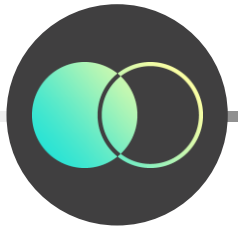
Could you use your JOIN knowledge to figure out which products exist in one table, but not the other?

Thanks, and Happy Halloween!
Mandy

Reply Forward

Solution Code

```
-- Use a LEFT JOIN to join products and orders
SELECT  p.product_id, p.product_name,
        o.product_id AS product_id_in_orders
FROM    products p LEFT JOIN orders o
        ON p.product_id = o.product_id
WHERE   o.product_id IS NULL;
```



JOINING ON MULTIPLE COLUMNS

- Multi-Table Analysis
- JOIN Basics
- JOIN Variations
- UNION Basics

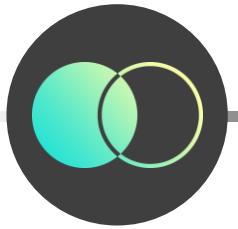
You can **join tables on multiple columns** by using "AND" in the join condition

```
SELECT  year, country,
        happiness_score
FROM    happiness_scores;
```

year	country	happiness_score
2015	Afghanistan	3.575
2015	Albania	4.959
2015	Algeria	5.605
2015	Angola	4.033
2015	Argentina	6.574

```
SELECT  year, country_name,
        inflation_rate
FROM    inflation_rates;
```

year	country_name	inflation_rate
2015	Bangladesh	6.1
2015	Brazil	9.0
2015	China	1.4
2015	DR Congo	1.6
2015	Ethiopia	6.1



JOINING ON MULTIPLE COLUMNS

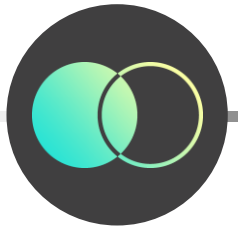
You can **join tables on multiple columns** by using "AND" in the join condition

```
SELECT   hs.year, hs.country,
         hs.happiness_score, ir.inflation_rate
FROM     happiness_scores hs
         INNER JOIN inflation_rates ir
         ON hs.year = ir.year AND hs.country = ir.country_name;
```

Use table aliases as a best practice when working with multiple tables

These column names are different but we're still able to join them!

year	country	happiness_score	inflation_rate
2015	Bangladesh	4.694	6.1
2015	Brazil	6.983	9.0
2015	China	5.140	1.4
2015	Ethiopia	4.512	6.1
2015	France	6.575	0.1



JOINING MULTIPLE TABLES

Multi-Table Analysis

JOIN Basics

JOIN Variations

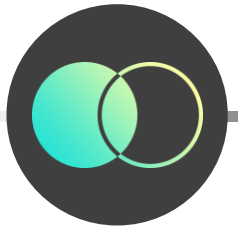
UNION Basics

You can **join more than two tables** as long as you specify the columns that link the tables together

```
SELECT  hs.year, hs.country, hs.happiness_score,
        cs.continent, ir.inflation_rate
FROM    happiness_scores hs
        LEFT JOIN country_stats cs
            ON hs.country = cs.country
        LEFT JOIN inflation_rates ir
            ON hs.year = ir.year AND hs.country = ir.country_name;
```

year	country	happiness_score	continent	inflation_rate
2015	India	4.565	Asia	4.9
2015	Indonesia	5.399	Asia	6.4
2015	Iran	4.686	Asia	14.6
2015	Iraq	4.677	Asia	NULL
2015	Israel	7.278	Asia	NULL
2015	Japan	5.987	Asia	0.8

PRO TIP: The code for joining multiple tables gets messy very quickly, so it's important to include ample spacing and consistent formatting for readability



SELF JOINS

Multi-Table Analysis

JOIN Basics

JOIN Variations

UNION Basics

A **self join** lets you join a table with itself, and typically involves two steps:

1. Combine a table with itself based on a matching column
2. Filter on the resulting rows based on some criteria

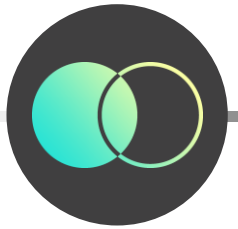
EXAMPLE *Identifying matching rows within a table*

```
SELECT * FROM employees;
```

employee_id	name	salary
1	Ava	85000
2	Bob	72000
3	Cat	59000
4	Dan	85000

```
-- Employees with the same salary
SELECT    e1.name, e1.salary,
          e2.name, e2.salary
FROM      employees e1 INNER JOIN employees e2
          ON e1.salary = e2.salary
WHERE     e1.name <> e2.name
ORDER BY  e1.name;
```

name	salary	name	salary
Ava	85000	Dan	85000
Dan	85000	Ava	85000



SELF JOINS

Multi-Table Analysis

JOIN Basics

JOIN Variations

UNION Basics

A **self join** lets you join a table with itself, and typically involves two steps:

1. Combine a table with itself based on a matching column
2. Filter on the resulting rows based on some criteria

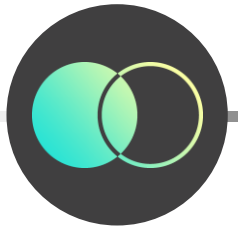
EXAMPLE *Comparing values within rows in a table*

```
SELECT * FROM employees;
```

employee_id	name	salary
1	Ava	85000
2	Bob	72000
3	Cat	59000
4	Dan	85000

```
-- Employees that have a greater salary
SELECT    e1.name, e1.salary,
          e2.name, e2.salary
FROM      employees e1 INNER JOIN employees e2
          ON e1.salary > e2.salary
ORDER BY  e1.name;
```

name	salary	name	salary
Ava	85000	Bob	72000
Ava	85000	Cat	59000
Bob	72000	Cat	59000
Dan	85000	Bob	72000
Dan	85000	Cat	59000



SELF JOINS

Multi-Table Analysis

JOIN Basics

JOIN Variations

UNION Basics

A **self join** lets you join a table with itself, and typically involves two steps:

1. Combine a table with itself based on a matching column
2. Filter on the resulting rows based on some criteria

EXAMPLE *Displaying relationships within a table*

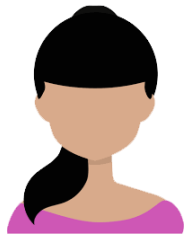
```
SELECT * FROM managers;
```

employee_id	employee_name	manager_id
1	Ava	NULL
2	Bob	1
3	Cat	1
4	Dan	2

```
-- Employees and their managers
SELECT   m1.employee_id, m1.employee_name,
         m1.manager_id,
         m2.employee_name AS manager_name
FROM     managers m1 LEFT JOIN managers m2
         ON m1.manager_id = m2.employee_id;
```

employee_id	employee_name	manager_id	manager_name
1	Ava	NULL	NULL
2	Bob	1	Ava
3	Cat	1	Ava
4	Dan	2	Bob

ASSIGNMENT: SELF JOINS



NEW MESSAGE

November 1, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Compare product prices**

Hi again,

Thanks for your help earlier!

Our marketing team wants to do some analysis to identify which of our products are similar in terms of price.

Could you write a query to determine which products are within 25 cents of each other in terms of unit price and return a list of all the candy pairs?

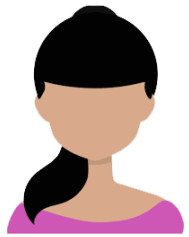
Thanks!
Mandy

Reply Forward

Results Preview

product_name	unit_price	product_name	unit_price	price_diff
Fizzy Lifting Drinks	3.75	Wonka Bar - Fudge Mallows	3.60	0.15
Fizzy Lifting Drinks	3.75	Wonka Bar - Scrumdiddlyumptious	3.60	0.15
Wonka Bar - Fudge Mallows	3.60	Wonka Bar - Nutty Crunch Surprise	3.49	0.11
Kazookles	3.25	Wonka Bar - Milk Chocolate	3.25	0.00
Wonka Bar - Fudge Mallows	3.60	Wonka Bar - Scrumdiddlyumptious	3.60	0.00
Fizzy Lifting Drinks	3.75	Wonka Bar - Triple Dazzle Caramel	3.75	0.00
Pixy Stix	1.25	Wonka Gum	1.25	0.00
Fun Dip	1.50	Nerds	1.50	0.00
Nerds	1.50	SweeTARTS	1.50	0.00
Fun Dip	1.50	SweeTARTS	1.50	0.00
Wonka Bar - Nutty Crunch Surprise	3.49	Wonka Bar - Scrumdiddlyumptious	3.60	-0.11
Wonka Bar - Scrumdiddlyumptious	3.60	Wonka Bar - Triple Dazzle Caramel	3.75	-0.15
Wonka Bar - Fudge Mallows	3.60	Wonka Bar - Triple Dazzle Caramel	3.75	-0.15
Kazookles	3.25	Wonka Bar - Nutty Crunch Surprise	3.49	-0.24
Wonka Bar - Milk Chocolate	3.25	Wonka Bar - Nutty Crunch Surprise	3.49	-0.24

SOLUTION: SELF JOINS



NEW MESSAGE

November 1, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Compare product prices**

Hi again,

Thanks for your help earlier!

Our marketing team wants to do some analysis to identify which of our products are similar in terms of price.

Could you write a query to determine which products are within 25 cents of each other in terms of unit price and return a list of all the candy pairs?

Thanks!
Mandy

Reply Forward

Solution Code

```sql
-- Calculate the price difference, do a self join,
-- and then return only price differences under 25c
SELECT  p1.product_name, p1.unit_price,
        p2.product_name, p2.unit_price,
        p1.unit_price - p2.unit_price AS price_diff
FROM    products p1 INNER JOIN products p2
        ON p1.product_id <> p2.product_id
WHERE   ABS(p1.unit_price - p2.unit_price) < 0.25
        AND p1.product_name < p2.product_name
ORDER BY price_diff DESC;
```

CROSS JOIN

A **cross join** returns all combinations of rows within two or more tables

```
SELECT * FROM tops;
```

id	type
1	T-Shirt
2	Hoodie

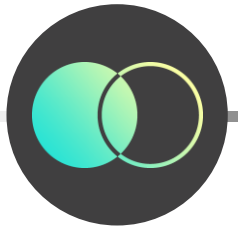
```
SELECT * FROM sizes;
```

id	size
1	Small
2	Medium
3	Large

```
SELECT  t.type, s.size
FROM    tops t CROSS JOIN sizes s;
```

type	size
Hoodie	Small
T-Shirt	Small
Hoodie	Medium
T-Shirt	Medium
Hoodie	Large
T-Shirt	Large

PRO TIP: Cross joins can produce very large outputs, so be careful using this on larger tables to avoid performance issues (in general, they are less common)



UNION & UNION ALL

Use a **UNION** to stack multiple tables or queries on top of one another

- UNION removes duplicate values, while UNION ALL retains them

```
SELECT * FROM tops;
```

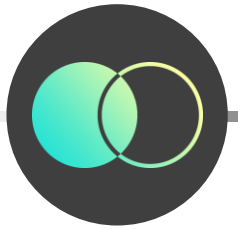
id	type
1	T-Shirt
2	Hoodie

```
SELECT * FROM outerwear;
```

id	type
2	Hoodie
3	Jacket
4	Coat

```
SELECT * FROM tops
UNION
SELECT * FROM outerwear;
```

id	type
1	T-Shirt
2	Hoodie
3	Jacket
4	Coat



UNION & UNION ALL

- Multi-Table Analysis
- JOIN Basics
- JOIN Variations
- UNION Basics

Use a **UNION** to stack multiple tables or queries on top of one another

- UNION removes duplicate values, while UNION ALL retains them

```
SELECT * FROM tops;
```

id	type
1	T-Shirt
2	Hoodie

```
SELECT * FROM outerwear;
```

id	type
2	Hoodie
3	Jacket
4	Coat

```
SELECT * FROM tops
UNION ALL
SELECT * FROM outerwear;
```

id	type
1	T-Shirt
2	Hoodie
2	Hoodie
3	Jacket
4	Coat

PRO TIP: If you know there are no duplicate values in the two tables you're combining, a UNION ALL will run much faster than a UNION

KEY TAKEAWAYS

- A **JOIN** combines data from two or more tables based on related column(s)
  - *Multiple JOINs can be written within the FROM clause of a single query aka SELECT statement*
- The main JOIN types are **INNER**, **LEFT**, **RIGHT**, and **FULL OUTER**
  - *INNER returns matches from both tables, LEFT includes everything from the left table, RIGHT includes everything from the right table, and FULL OUTER returns all rows from both tables*
- **Self joins** and **cross joins** are additional JOIN options you can use
  - *Self joins are useful for side-by-side comparisons of rows within the same table*
  - *CROSS JOINs return all combinations of rows within two or more tables, but are less commonly used*
- **UNION** and **UNION ALL** stack the results of two or more queries
  - *UNION removes duplicate rows and UNION ALL keeps them, making it the faster option of the two*

SUBQUERIES & CTES

SUBQUERIES & CTEs

In this section we'll cover **subqueries** and **common table expressions (CTEs)**, which are different ways of working with nested queries

TOPICS WE'LL COVER:

- Subqueries
- CTEs
- Technique Comparison

GOALS FOR THIS SECTION:

- Learn the syntax for writing subqueries and CTEs and identify the appropriate use cases for each
- Understand the difference between subqueries, CTEs, temporary tables, and views

SUBQUERY BASICS

Subqueries

CTEs

Technique Comparison

A **subquery** is a query nested within a main query, and is typically used for solving a problem in multiple steps

EXAMPLE

Return all countries that have an above average happiness score

Step 1: *Calculate the average happiness score*

```
SELECT  AVG(happiness_score)
FROM    happiness_scores;
```

AVG(happiness_score)
5.4400624

This is the main, or outer, query

Step 2: *Return all rows with a happiness score greater than the first query result*

```
SELECT  *
FROM    happiness_scores
WHERE   happiness_score > (SELECT AVG(happiness_score)
                           FROM   happiness_scores);
```

*This is a **subquery**!*

year	country	region	happiness_score
2015	Algeria	Middle East and North Africa	5.605
2015	Argentina	Latin America and Caribbean	6.574
2015	Australia	North America and ANZ	7.284
2015	Austria	Western Europe	7.200

The **subquery is run first**, before the main query

SUBQUERY BASICS

Subqueries

CTEs

Technique Comparison

Subqueries can occur in **multiple places** within a query:

- Calculations in the SELECT clause
- As part of a JOIN in the FROM clause
- Filtering in the WHERE and HAVING clauses

```
SELECT  *
FROM    happiness_scores
WHERE   happiness_score > (SELECT AVG(happiness_score)
                           FROM   happiness_scores);
```

This is an example of a subquery used as a filter in the WHERE clause

year	country	region	happiness_score
2015	Algeria	Middle East and North Africa	5.605
2015	Argentina	Latin America and Caribbean	6.574
2015	Australia	North America and ANZ	7.284
2015	Austria	Western Europe	7.200

SUBQUERIES IN THE SELECT CLAUSE

Subqueries

CTEs

Technique Comparison

EXAMPLE

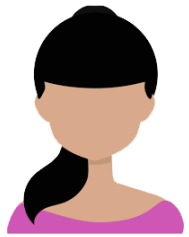
Return the difference between each country's happiness score and the average

```
SELECT   country,
         happiness_score - (SELECT AVG(happiness_score) FROM happiness_scores) AS diff_from_avg
FROM     happiness_scores;
```

This subquery lets you subtract the average happiness score from each row

country	diff_from_avg
Afghanistan	-1.8650624
Albania	-0.4810624
Algeria	0.1649376
Angola	-1.4070624
Argentina	1.1339376
Armenia	-1.0900624
Australia	1.8439376
Austria	1.7599376
Azerbaijan	-0.2280624

ASSIGNMENT: SUBQUERIES IN THE SELECT CLAUSE



NEW MESSAGE

November 4, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Most expensive products**

Hello,

Our product team plans on evaluating our product prices later this week to see if any adjustments need to be made for next year.

Can you give me a list of our products from most to least expensive, along with how much each product differs from the average unit price?

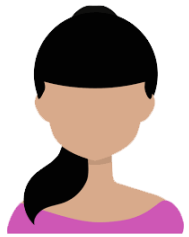
Thanks!
Mandy

Reply Forward

Results Preview

product_id	product_name	unit_price	avg_unit_price	diff_price
OTH-LIC-15000	Lickable Wallpaper	20.00	4.172353	15.827647
SUG-EVE-47000	Everlasting Gobstopper	10.00	4.172353	5.827647
SUG-HAI-55000	Hair Toffee	4.50	4.172353	0.327647
CHO-TRI-54000	Wonka Bar - Triple Dazzle Caramel	3.75	4.172353	-0.422353
OTH-FIZ-56000	Fizzy Lifting Drinks	3.75	4.172353	-0.422353
CHO-FUD-51000	Wonka Bar - Fudge Mallows	3.60	4.172353	-0.572353
CHO-SCR-58000	Wonka Bar - Scrumdiddlyumptious	3.60	4.172353	-0.572353
CHO-NUT-13000	Wonka Bar - Nutty Crunch Surprise	3.49	4.172353	-0.682353
CHO-MIL-31000	Wonka Bar - Milk Chocolate	3.25	4.172353	-0.922353
OTH-KAZ-38000	Kazookles	3.25	4.172353	-0.922353
SUG-LOO-45000	Loopy Lollipops	2.75	4.172353	-1.422353

SOLUTION: SUBQUERIES IN THE SELECT CLAUSE



1 **NEW MESSAGE**

November 4, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Most expensive products**

Hello,

Our product team plans on evaluating our product prices later this week to see if any adjustments need to be made for next year.

Can you give me a list of our products from most to least expensive, along with how much each product differs from the average unit price?

Thanks!
Mandy

Reply Forward

Solution Code

```sql
-- Return the product details and difference
-- between each unit price and the average unit price
SELECT  product_id, product_name, unit_price,
        (SELECT AVG(unit_price) FROM products)
            AS avg_unit_price,
        unit_price - (SELECT AVG(unit_price) FROM products)
            AS diff_price
FROM    products
ORDER BY unit_price DESC;
```

SUBQUERIES IN THE FROM CLAUSE

Subqueries

CTEs

Technique Comparison

EXAMPLE

Return each country's happiness score for the year alongside the country's average happiness score

```
SELECT  hs.year, hs.country, hs.happiness_score,
        country_hs.avg_hs_by_country
FROM    happiness_scores hs LEFT JOIN
            (SELECT country, AVG(happiness_score) AS avg_hs_by_country
            FROM    happiness_scores
            GROUP BY country) AS country_hs
            ON hs.country = country_hs.country;
```

This subquery calculates the average happiness score by country, and is then joined with the main query

*Subqueries in the FROM clause need to have an **alias***

year	country	happiness_score	avg_hs_by_country
2015	Afghanistan	3.575	2.9907778
2015	Albania	4.959	4.8932222
2015	Algeria	5.605	5.4090000
2016	Afghanistan	3.360	2.9907778
2016	Albania	4.655	4.8932222
2016	Algeria	6.355	5.4090000
2017	Afghanistan	3.794	2.9907778
2017	Albania	4.644	4.8932222
2017	Algeria	5.872	5.4090000

PRO TIP: Using subqueries within the JOIN clause is great for speeding up queries, since it allows you to join smaller tables

PRO TIP: MULTIPLE SUBQUERIES

Queries can contain **multiple subqueries** as long as each one has a different alias

Subqueries

CTEs

Technique Comparison

```sql
SELECT  hs.year, hs.country, hs.happiness_score,
        country_hs.avg_hs_by_country
FROM    (SELECT year, country, happiness_score FROM happiness_scores
         UNION ALL
         SELECT 2024, country, ladder_score FROM happiness_scores_current) AS hs
        LEFT JOIN
        (SELECT country, AVG(happiness_score) AS avg_hs_by_country
         FROM   happiness_scores
         GROUP BY country) AS country_hs
        ON hs.country = country_hs.country;
```

This subquery stacks the historical and current happiness scores for each country

This subquery calculates the average happiness score by country, and is then joined with the first subquery

year	country	happiness_score	avg_hs_by_country
2024	Venezuela	5.607	5.3042222
2024	Vietnam	6.043	5.3094444
2024	Yemen	3.561	3.6888750
2024	Zambia	3.502	4.2773333
2024	Zimbabwe	3.341	3.6306667

PRO TIP: MULTIPLE SUBQUERIES

Subqueries

CTEs

Technique Comparison

Queries can contain **multiple subqueries** as long as each one has a different alias

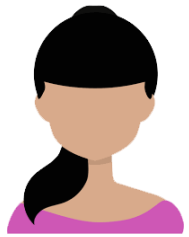
```
SELECT * FROM

(SELECT hs.year, hs.country, hs.happiness_score,
        country_hs.avg_hs_by_country
 FROM   (SELECT year, country, happiness_score FROM happiness_scores
         UNION ALL
         SELECT 2024, country, ladder_score FROM happiness_scores_current) AS hs
        LEFT JOIN
        (SELECT country, AVG(happiness_score) AS avg_hs_by_country
         FROM   happiness_scores
         GROUP BY country) AS country_hs
        ON hs.country = country_hs.country) AS hs_country_hs

WHERE   happiness_score > avg_hs_by_country + 1;
```

*This is a **nested subquery** now, used to return years where the happiness is a whole point greater that the country's average score*

year	country	happiness_score	avg_hs_by_country
2015	Lesotho	4.898	3.8561429
2015	Venezuela	6.810	5.3042222

PRO TIP: Interpreting nested subqueries can be overwhelming, so start from the inner subqueries and work your way out – or use CTEs instead *(coming up shortly)*

ASSIGNMENT: SUBQUERIES IN THE FROM CLAUSE



NEW MESSAGE

November 5, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Products in each factory**

Hello,

Our inventory management team would like to review the products produced by each factory.

Can you give me a list of our factories, along with the names of the products they produce and the number of products they produce?

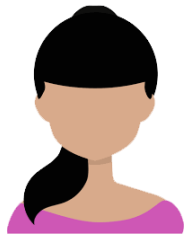
Thanks!
Mandy

Reply Forward

Results Preview

factory	product_name	num_products
Lot's O' Nuts	Wonka Bar - Fudge Mallows	3
Lot's O' Nuts	Wonka Bar - Nutty Crunch Surprise	3
Lot's O' Nuts	Wonka Bar - Scrumdiddlyumptious	3
Secret Factory	Everlasting Gobstopper	3
Secret Factory	Lickable Wallpaper	3
Secret Factory	Wonka Gum	3
Sugar Shack	Fizzy Lifting Drinks	8
Sugar Shack	Fun Dip	8
Sugar Shack	Laffy Taffy	8
Sugar Shack	Loopy Lollipops	8
Sugar Shack	Nerds	8
Sugar Shack	Pixy Stix	8
Sugar Shack	SweeTARTS	8
Sugar Shack	Tropical Nerds	8
The Other Factory	Hair Toffee	2
The Other Factory	Kazookles	2
Wicked Choccy's	Wonka Bar - Milk Chocolate	2
Wicked Choccy's	Wonka Bar - Triple Dazzle Caramel	2

SOLUTION: SUBQUERIES IN THE FROM CLAUSE



1 **NEW MESSAGE**

November 5, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Products in each factory**

Hello,

Our inventory management team would like to review the products produced by each factory.

Can you give me a list of our factories, along with the names of the products they produce and the number of products they produce?

Thanks!
Mandy

Reply Forward

Solution Code

```
-- All factories, products and total num of products
SELECT  fp.factory, fp.product_name, fn.num_products
FROM

(SELECT factory, product_name
FROM    products) fp

LEFT JOIN

(SELECT  factory, COUNT(product_id) AS num_products
FROM     products
GROUP BY factory) fn

ON fp.factory = fn.factory
ORDER BY fp.factory, fp.product_name;
```

SUBQUERIES IN THE WHERE & HAVING CLAUSES

Subqueries

CTEs

Technique Comparison

EXAMPLE

Return regions with above average happiness scores

```
SELECT region, AVG(happiness_score) AS avg_hs
FROM happiness_scores
GROUP BY region
HAVING avg_hs > (SELECT AVG(happiness_score) FROM happiness_scores);
```

region	avg_hs
Central and Eastern Europe	5.5857617
Latin America and Caribbean	5.9938526
North America and ANZ	7.1760833
Western Europe	6.8229946
East Asia	5.7321296
Commonwealth of Independent States	5.6390682

This subquery filters the grouped regional data

ANY VS ALL

Keywords like **ANY**, **ALL**, and EXISTS can provide more specific filtering logic

Subqueries

CTEs

Technique Comparison

EXAMPLE

Return happiness scores that are greater than ANY / ALL of the current happiness scores

```sql
-- Scores that are greater than ANY 2024 scores
SELECT  *
FROM    happiness_scores
WHERE   happiness_score >
        ANY(SELECT  ladder_score
            FROM    happiness_scores_current);
```

```sql
-- Scores that are greater than ALL 2024 scores
SELECT  *
FROM    happiness_scores
WHERE   happiness_score >
        ALL(SELECT  ladder_score
            FROM    happiness_scores_current);
```

year	country	region
2015	Afghanistan	South Asia
2015	Albania	Central and Eastern Europe
2015	Algeria	Middle East and North Africa
2015	Angola	Sub-Saharan Africa
2015	Argentina	Latin America and Caribbean
2015	Armenia	Central and Eastern Europe

All rows are returned

year	country	region
2019	Finland	Western Europe
2020	Finland	Western Europe
2021	Finland	Western Europe
2022	Finland	Western Europe
2023	Finland	Western Europe

Only 5 rows are returned

EXISTS

Keywords like ANY, ALL, and **EXISTS** can provide more specific filtering logic

Subqueries

CTEs

Technique Comparison

EXAMPLE

Only return happiness scores for countries that EXIST in the inflation rates table

```
SELECT *
FROM happiness_scores h
WHERE EXISTS (
    SELECT i.country_name
    FROM inflation_rates i
    WHERE i.country_name = h.country);
```

*This is known as a **correlated subquery**, since it references an outside table and cannot stand alone*

year	country	region	happiness_score
2015	Bangladesh	South Asia	4.694
2015	Brazil	Latin America and Caribbean	6.983
2015	China	East Asia	5.140
2015	Ethiopia	Sub-Saharan Africa	4.512
2015	France	Western Europe	6.575
2015	Germany	Western Europe	6.750

Correlated subqueries are **known to be slow**, but some RDBMSs automatically optimize for this

PRO TIP: CORRELATED SUBQUERIES

Subqueries

CTEs

Technique Comparison

Correlated subqueries can be **rewritten as INNER JOINs** to run faster

EXAMPLE

Only return happiness scores for countries that EXIST in the inflation rates table

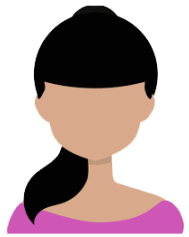
```sql
SELECT *
FROM happiness_scores h
WHERE EXISTS (
    SELECT i.country_name
    FROM inflation_rates i
    WHERE i.country_name = h.country);
```

```sql
SELECT *
FROM happiness_scores h
    INNER JOIN inflation_rates i
    ON h.country = i.country_name
    AND h.year = i.year;
```

This is arguably less readable, so it's important to consider readability and run time when selecting an approach

year	country	region	happiness_score	year	country_name	inflation_rate
2015	Bangladesh	South Asia	4.694	2015	Bangladesh	6.1
2015	Brazil	Latin America and Caribbean	6.983	2015	Brazil	9.0
2015	China	East Asia	5.140	2015	China	1.4
2015	Ethiopia	Sub-Saharan Africa	4.512	2015	Ethiopia	6.1
2015	France	Western Europe	6.575	2015	France	0.1
2015	Germany	Western Europe	6.750	2015	Germany	0.3

ASSIGNMENT: SUBQUERIES IN THE WHERE CLAUSE



NEW MESSAGE

November 6, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Low unit price products**

Hello,

Our Wicked Choccy's factory has some extra bandwidth, and we'd like to see if there are any lower priced products that they can help produce going forward.

Can you help us identify products that have a unit price less than the unit price of all products from Wicked Choccy's? Please include which factory is currently producing them as well.

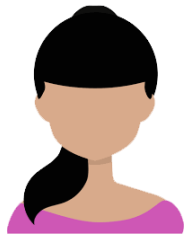
Thanks!
Mandy

Reply Forward

Results Preview

product_id	product_name	factory	division	unit_price
OTH-GUM-21000	Wonka Gum	Secret Factory	Other	1.25
SUG-FUN-75000	Fun Dip	Sugar Shack	Sugar	1.50
SUG-LAF-25000	Laffy Taffy	Sugar Shack	Sugar	1.99
SUG-LOO-45000	Loopy Lollipops	Sugar Shack	Sugar	2.75
SUG-NER-92000	Nerds	Sugar Shack	Sugar	1.50
SUG-PIX-62000	Pixy Stix	Sugar Shack	Sugar	1.25
SUG-SWE-91000	SweeTARTS	Sugar Shack	Sugar	1.50

SOLUTION: SUBQUERIES IN THE WHERE CLAUSE



1 **NEW MESSAGE**

November 6, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Low unit price products**

Hello,

Our Wicked Choccy's factory has some extra bandwidth, and we'd like to see if there are any lower priced products that they can help produce going forward.

Can you help us identify products that have a unit price less than the unit price of all products from Wicked Choccy's? Please include which factory is currently producing them as well.

Thanks!
Mandy

Reply Forward

Solution Code

```
-- Products where the unit price is less than the
-- unit price of all products from Wicked Choccy's
SELECT  *
FROM    products
WHERE   unit_price <
        ALL (SELECT unit_price
             FROM products
             WHERE factory = "Wicked Choccy's");
```

COMMON TABLE EXPRESSIONS

Subqueries

CTEs

Technique Comparison

A **common table expression (CTE)** creates a named, temporary output that can be referenced within another query

EXAMPLE *Return each country's happiness score for the year alongside the country's average happiness score*

```
WITH country_hs AS (SELECT  country,
                    AVG(happiness_score) AS avg_hs_by_country
                    FROM    happiness_scores
                    GROUP BY country)
```

← *This is a **CTE** named "country_hs"*

```
SELECT  hs.year, hs.country, hs.happiness_score,
        country_hs.avg_hs_by_country
FROM    happiness_scores hs
        LEFT JOIN country_hs
        ON hs.country = country_hs.country;
```

← *It's being referenced here!*

year	country	happiness_score	avg_hs_by_country
2015	Afghanistan	3.575	2.9907778
2015	Albania	4.959	4.8932222
2015	Algeria	5.605	5.4090000
2016	Afghanistan	3.360	2.9907778

COMMON TABLE EXPRESSIONS

Subqueries

CTEs

Technique Comparison

A **common table expression (CTE)** creates a named, temporary output that can be referenced within another query

EXAMPLE *Return each country's happiness score for the year alongside the country's average happiness score*

```
WITH country_hs AS (SELECT  country,
                    AVG(happiness_score) AS avg_hs_by_country
                    FROM    happiness_scores
                    GROUP BY country)

SELECT  hs.year, hs.country, hs.happiness_score,
        country_hs.avg_hs_by_country
FROM    happiness_scores hs
        LEFT JOIN country_hs
        ON hs.country = country_hs.country;
```

*CTEs need to start with the **WITH** keyword, followed by the alias, the **AS** keyword, and the query between parentheses*

year	country	happiness_score	avg_hs_by_country
2015	Afghanistan	3.575	2.9907778
2015	Albania	4.959	4.8932222
2015	Algeria	5.605	5.4090000
2016	Afghanistan	3.360	2.9907778

COMMON TABLE EXPRESSIONS

Subqueries

CTEs

Technique Comparison

A **common table expression (CTE)** creates a named, temporary output that can be referenced within another query

Why use CTEs instead of subqueries?

- **Readability**: Complex queries with CTEs are much easier to read
- **Reusability**: CTEs can be referenced multiple times within a query
- **Recursiveness**: CTEs can handle recursive queries

Despite all this, you **shouldn't forget about subqueries**:

- Most modern RDBMSs support CTEs, but not all of them do
- For simple queries, sometimes a subquery is readable enough and works just fine

READABILITY

Subqueries

CTEs

Technique Comparison

EXAMPLE *Return each country's happiness score for the year alongside the country's average happiness score*

Subquery

```
SELECT  hs.year, hs.country, hs.happiness_score,
        country_hs.avg_hs
FROM    happiness_scores hs LEFT JOIN
            (SELECT country,
                    AVG(happiness_score) AS avg_hs
            FROM    happiness_scores
            GROUP BY country) AS country_hs
            ON hs.country = country_hs.country;
```

year	country	happiness_score	avg_hs
2015	Afghanistan	3.575	2.9907778
2015	Albania	4.959	4.8932222
2015	Algeria	5.605	5.4090000
2016	Afghanistan	3.360	2.9907778

Same outputs!

CTE

```
WITH country_hs AS (SELECT  country,
                            AVG(happiness_score) AS avg_hs
                    FROM    happiness_scores
                    GROUP BY country)

SELECT  hs.year, hs.country, hs.happiness_score,
        country_hs.avg_hs
FROM    happiness_scores hs
        LEFT JOIN country_hs
        ON hs.country = country_hs.country;
```

year	country	happiness_score	avg_hs
2015	Afghanistan	3.575	2.9907778
2015	Albania	4.959	4.8932222
2015	Algeria	5.605	5.4090000
2016	Afghanistan	3.360	2.9907778

REUSABILITY

Subqueries

CTEs

Technique Comparison

Unlike subqueries, CTEs can be **referenced multiple times** within a query, which helps avoid repeating code

EXAMPLE

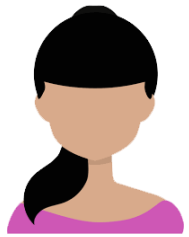
For each country, return countries from the same region with a lower happiness score in 2023

```
WITH hs AS (SELECT *
            FROM happiness_scores
            WHERE year = 2023)

SELECT  hs1.region, hs1.country, hs1.happiness_score,
        hs2.country, hs2.happiness_score
FROM    hs hs1 INNER JOIN hs hs2
        ON hs1.region = hs2.region
WHERE   hs1.happiness_score > hs2.happiness_score;
```

We're performing a self join using the CTE!

region	country	happiness_score	country	happiness_score
Central and Eastern Europe	Albania	5.277	Georgia	5.109
Central and Eastern Europe	Albania	5.277	North Macedonia	5.254
Central and Eastern Europe	Albania	5.277	Ukraine	5.071
Middle East and North Africa	Algeria	5.329	Egypt	4.170
Middle East and North Africa	Algeria	5.329	Iran	4.876

ASSIGNMENT: CTES



NEW MESSAGE

November 7, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Largest orders**

Hello,

The sales director wants a list of our biggest orders. In addition to sending over a list of all the orders over $200, could you also tell him the number of orders over $200?

Thanks!
Mandy

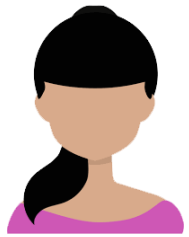
Reply Forward

Results Preview

order_id	total_amount_spent
US-2021-122336-44299	352.20
US-2024-164756-45553	304.14
US-2023-124163-45194	247.35
US-2023-164770-45262	232.40
US-2024-163790-45598	208.95
US-2024-164147-45324	204.65
US-2024-140326-45539	202.05

COUNT(*)
7

SOLUTION: CTES



NEW MESSAGE

November 7, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Largest orders**

Hello,

The sales director wants a list of our biggest orders. In addition to sending over a list of all the orders over $200, could you also tell him the number of orders over $200?

Thanks!
Mandy

Reply Forward

Solution Code

```sql
-- Return all orders over $200
SELECT   o.order_id,
         SUM(o.units * p.unit_price) AS total_amount_spent
FROM     orders o LEFT JOIN products p
         ON o.product_id = p.product_id
GROUP BY o.order_id
HAVING   total_amount_spent > 200
ORDER BY total_amount_spent DESC;

-- Return the number of orders over $200
WITH tas AS (SELECT  o.order_id,
                     SUM(o.units * p.unit_price)
                        AS total_amount_spent
             FROM    orders o LEFT JOIN products p
                     ON o.product_id = p.product_id
             GROUP BY o.order_id
             HAVING   total_amount_spent > 200
             ORDER BY total_amount_spent DESC)

SELECT  COUNT(*) FROM tas;
```

This ORDER BY clause in the CTE doesn't affect the final output and can be removed to make the code run more efficiently

MULTIPLE CTES

You can use **multiple CTEs** in a query, and even **combine them with subqueries**

Subqueries

CTEs

Technique Comparison

```sql
-- Compare 2023 vs 2024 happiness scores side by side
WITH hs23 AS (SELECT *
              FROM happiness_scores
              WHERE year = 2023),
     hs24 AS (SELECT *
              FROM happiness_scores_current)

SELECT  hs23.country,
        hs23.happiness_score AS hs_2023,
        hs24.ladder_score AS hs_2024
FROM    hs23 INNER JOIN hs24
        ON hs23.country = hs24.country;
```

*Note that we're using a single **WITH** keyword to create two CTEs, and they are separated by a comma*

country	hs_2023	country	hs_2024
Afghanistan	1.859	Afghanistan	1.721
Albania	5.277	Albania	5.304
Algeria	5.329	Algeria	5.364
Argentina	6.024	Argentina	6.188
Armenia	5.342	Armenia	5.455

MULTIPLE CTES

Subqueries

CTEs

Technique Comparison

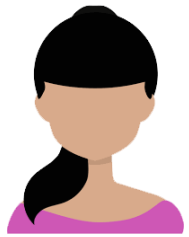
You can use **multiple CTEs** in a query, and even **combine them with subqueries**

```sql
-- Return the countries where the score increased
SELECT * FROM

(WITH hs23 AS (SELECT *
               FROM happiness_scores
               WHERE year = 2023),
     hs24 AS (SELECT *
              FROM happiness_scores_current)

SELECT  hs23.country,
        hs23.happiness_score AS hs_2023,
        hs24.ladder_score AS hs_2024
FROM    hs23 INNER JOIN hs24
        ON hs23.country = hs24.country) AS hs_23_24

WHERE hs_2023 < hs_2024;
```

Our previous query with two CTEs is now a subquery!

country	hs_2023	hs_2024
Albania	5.277	5.304
Algeria	5.329	5.364
Argentina	6.024	6.188
Armenia	5.342	5.455

ASSIGNMENT: MULTIPLE CTES



NEW MESSAGE

November 8, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **RE: Products in each factory**

Hi again –

Regarding my earlier message, could you rewrite your code using CTEs instead of subqueries? Thanks!

Our inventory management team would like to review the products produced by each factory.

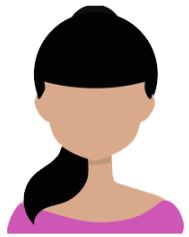
Can you give me a list of our factories, along with the names of the products they produce and the number of products they produce?

Reply Forward

Results Preview

factory	product_name	num_products
Lot's O' Nuts	Wonka Bar - Fudge Mallows	3
Lot's O' Nuts	Wonka Bar - Nutty Crunch Surprise	3
Lot's O' Nuts	Wonka Bar - Scrumdiddlyumptious	3
Secret Factory	Everlasting Gobstopper	3
Secret Factory	Lickable Wallpaper	3
Secret Factory	Wonka Gum	3
Sugar Shack	Fizzy Lifting Drinks	8
Sugar Shack	Fun Dip	8
Sugar Shack	Laffy Taffy	8
Sugar Shack	Loopy Lollipops	8
Sugar Shack	Nerds	8
Sugar Shack	Pixy Stix	8
Sugar Shack	SweeTARTS	8
Sugar Shack	Tropical Nerds	8
The Other Factory	Hair Toffee	2
The Other Factory	Kazookles	2
Wicked Choccy's	Wonka Bar - Milk Chocolate	2
Wicked Choccy's	Wonka Bar - Triple Dazzle Caramel	2

SOLUTION: MULTIPLE CTES



NEW MESSAGE

November 8, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **RE: Products in each factory**

Hi again –

Regarding my earlier message, could you rewrite your code using CTEs instead of subqueries? Thanks!

Our inventory management team would like to review the products produced by each factory.

Can you give me a list of our factories, along with the names of the products they produce and the number of products they produce?

Reply Forward

Solution Code

```sql
-- Subqueries rewritten as CTEs
WITH fp AS (SELECT factory,
                   product_name
            FROM products),
     fn AS (SELECT   factory,
                     COUNT(product_id) AS num_products
            FROM     products
            GROUP BY factory)

SELECT  fp.factory, fp.product_name, fn.num_products
FROM    fp LEFT JOIN fn
        ON fp.factory = fn.factory
ORDER BY fp.factory, fp.product_name;
```

RECURSIVE CTEs

A **recursive CTE** is a query that references itself, which is useful for generating sequences and working with hierarchical data

- Subqueries
- CTEs
- Technique Comparison

*Recursive CTEs use the **RECURSIVE** keyword*

```
WITH RECURSIVE cte_name AS (
    SELECT ...
    UNION ALL
    SELECT ...
    FROM cte_name
)
SELECT * FROM cte_name;
```

*They have an **anchor** member*

*And a **recursive** member that references the CTE*

***UNION** or **UNION ALL** are used as connectors*

The **syntax is slightly different** in each RDBMS

RECURSIVE CTEs

Subqueries

CTEs

Technique Comparison

EXAMPLE

Return daily stock prices, including dates with missing prices

```sql
SELECT *
FROM stock_prices;
```

date	price
2024-11-01	678.27
2024-11-03	688.83
2024-11-04	645.40
2024-11-06	591.01

Notice the missing dates

Step 1: *Generate a column of dates*

```sql
WITH RECURSIVE my_dates(dt) AS
    (SELECT '2024-11-01'
     UNION ALL
     SELECT dt + INTERVAL 1 DAY
     FROM my_dates
     WHERE dt < '2024-11-06')

SELECT * FROM my_dates;
```

This generates the next date after the anchor

This stops the recursion after reaching this date

dt
2024-11-01
2024-11-02
2024-11-03
2024-11-04
2024-11-05
2024-11-06

RECURSIVE CTEs

Subqueries

CTEs

Technique Comparison

EXAMPLE

Return daily stock prices, including dates with missing prices

```
SELECT *
FROM stock_prices;
```

date	price
2024-11-01	678.27
2024-11-03	688.83
2024-11-04	645.40
2024-11-06	591.01

Notice the missing dates

Step 1: *Generate a column of dates*

```
WITH RECURSIVE my_dates(dt) AS
    (SELECT '2024-11-01'
     UNION ALL
     SELECT dt + INTERVAL 1 DAY
     FROM my_dates
     WHERE dt < '2024-11-06')

SELECT * FROM my_dates;
```

dt
2024-11-01
2024-11-02
2024-11-03
2024-11-04
2024-11-05
2024-11-06

Step 2: *Join with the stock prices table*

```
WITH RECURSIVE my_dates(dt) AS
    (SELECT '2024-11-01'
     UNION ALL
     SELECT dt + INTERVAL 1 DAY
     FROM my_dates
     WHERE dt < '2024-11-06')

SELECT   md.dt, sp.price
FROM     my_dates md
         LEFT JOIN stock_prices sp
         ON md.dt = sp.date;
```

dt	price
2024-11-01	678.27
2024-11-02	NULL
2024-11-03	688.83
2024-11-04	645.40
2024-11-05	NULL
2024-11-06	591.01

RECURSIVE CTEs

Subqueries

CTEs

Technique Comparison

EXAMPLE

Return the reporting chain for each employee

```
SELECT  *
FROM    employees;
```

employee_id	employee_name	manager_id
1	Ava	NULL
2	Bob	1
3	Cat	1
4	Dan	2

```
WITH RECURSIVE employee_hierarchy AS (
    SELECT  employee_id, employee_name, manager_id,
            employee_name AS hierarchy
    FROM    employees
    WHERE   manager_id IS NULL

    UNION ALL

    SELECT  e.employee_id, e.employee_name, e.manager_id,
            CONCAT(eh.hierarchy, ' > ', e.employee_name) AS hierarchy
    FROM    employees e INNER JOIN employee_hierarchy eh
            ON e.manager_id = eh.employee_id)

SELECT  employee_id, employee_name, manager_id, hierarchy
FROM    employee_hierarchy
ORDER BY employee_id;
```

This sets the highest-ranking employee as the anchor

This stops the recursion when there is nothing left to join

employee_id	employee_name	manager_id	hierarchy
1	Ava	NULL	Ava
2	Bob	1	Ava > Bob
3	Cat	1	Ava > Cat
4	Dan	2	Ava > Bob > Dan

RECURSIVE CTEs

Subqueries

CTEs

Technique Comparison

EXAMPLE

Return the reporting chain for each employee

```
SELECT  *
FROM    employees;
```

employee_id	employee_name	manager_id
1	Ava	NULL
2	Bob	1
3	Cat	1
4	Dan	2

```
WITH RECURSIVE employee_hierarchy AS (
    SELECT  employee_id, employee_name, manager_id,
            employee_name AS hierarchy
    FROM    employees
    WHERE   manager_id IS NULL

    UNION ALL

    SELECT  e.employee_id, e.employee_name, e.manager_id,
            CONCAT(eh.hierarchy, ' > ', e.employee_name) AS hierarchy
    FROM    employees e INNER JOIN employee_hierarchy eh
            ON e.manager_id = eh.employee_id)

SELECT  employee_id, employee_name, manager_id, hierarchy
FROM    employee_hierarchy
ORDER BY employee_id;
```

employee_id	employee_name	manager_id	hierarchy
1	Ava	NULL	Ava
2	Bob	1	Ava > Bob
3	Cat	1	Ava > Cat

RECURSIVE CTEs

Subqueries

CTEs

Technique Comparison

EXAMPLE

Return the reporting chain for each employee

```
SELECT  *
FROM    employees;
```

employee_id	employee_name	manager_id
1	Ava	NULL
2	Bob	1
3	Cat	1
4	Dan	2

```
WITH RECURSIVE employee_hierarchy AS (
    SELECT  employee_id, employee_name, manager_id,
            employee_name AS hierarchy
    FROM    employees
    WHERE   manager_id IS NULL

    UNION ALL

    SELECT  e.employee_id, e.employee_name, e.manager_id,
            CONCAT(eh.hierarchy, ' > ', e.employee_name) AS hierarchy
    FROM    employees e INNER JOIN employee_hierarchy eh
            ON e.manager_id = eh.employee_id)

SELECT  employee_id, employee_name, manager_id, hierarchy
FROM    employee_hierarchy
ORDER BY employee_id;
```

employee_id	employee_name	manager_id	hierarchy
1	Ava	NULL	Ava
2	Bob	1	Ava > Bob
3	Cat	1	Ava > Cat
4	Dan	2	Ava > Bob > Dan

RECURSIVE CTEs

Subqueries

CTEs

Technique Comparison

EXAMPLE

Return the reporting chain for each employee

```
SELECT  *
FROM    employees;
```

employee_id	employee_name	manager_id
1	Ava	NULL
2	Bob	1
3	Cat	1
4	Dan	2

PRO TIP: Recursive CTEs aren't very common, so instead of memorizing syntax, keep in mind the general concepts of generating sequences and returning hierarchies

```
WITH RECURSIVE employee_hierarchy AS (
    SELECT  employee_id, employee_name, manager_id,
            employee_name AS hierarchy
    FROM    employees
    WHERE   manager_id IS NULL

    UNION ALL

    SELECT  e.employee_id, e.employee_name, e.manager_id,
            CONCAT(eh.hierarchy, ' > ', e.employee_name) AS hierarchy
    FROM    employees e INNER JOIN employee_hierarchy eh
            ON e.manager_id = eh.employee_id)

SELECT  employee_id, employee_name, manager_id, hierarchy
FROM    employee_hierarchy
ORDER BY employee_id;
```

employee_id	employee_name	manager_id	hierarchy
1	Ava	NULL	Ava
2	Bob	1	Ava > Bob
3	Cat	1	Ava > Cat
4	Dan	2	Ava > Bob > Dan

TEMPORARY TABLES & VIEWS

Subqueries

CTEs

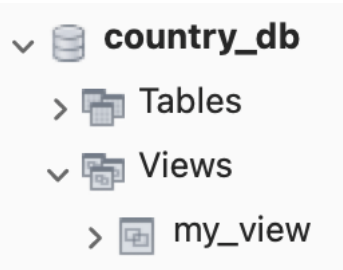
Technique Comparison

Temporary tables and **views** are other options for querying the results of a query

- Both subqueries and CTEs only exist for the duration of the query
- Temporary tables exist for a session and views continue to exist until modified or dropped

```sql
-- Temporary table
CREATE TEMPORARY TABLE my_temp_table AS
SELECT  year, country, happiness_score FROM happiness_scores
UNION ALL
SELECT  2024, country, ladder_score FROM happiness_scores_current;

SELECT * FROM my_temp_table;
```

```sql
-- View
CREATE VIEW my_view AS
SELECT  year, country, happiness_score FROM happiness_scores
UNION ALL
SELECT  2024, country, ladder_score FROM happiness_scores_current;

SELECT * FROM my_view;
```



TECHNIQUE COMPARISON

Here's how these **techniques compare** with each other:

Subqueries

CTEs

Technique Comparison

Technique	Description	Persistence	Permissions	Summary	In Practice
Subqueries	A query nested within a query	Exists only during execution	Fewer permissions	**Simple queries** Great for simple queries that require multiple steps	Start here if you need to query the results of a query
CTEs	Named temporary output	Exists only during execution	Fewer permissions	**Complex queries** Great for organizing complex queries	To improve readability and reusability, or use recursion, consider rewriting complex subqueries as CTEs
Temporary Tables	Temporary data storage within a session	Exists only during a session	More permissions	**Multiple uses in a session** Great for referencing multiple times in a session	If you find yourself referencing the same CTE, consider a temporary table
Views	Virtual table based on a query	Always available until modified or dropped	More permissions	**Use in multiple sessions** Great for accessing the results of complex queries	For a more permanent solution, consider a view

KEY TAKEAWAYS

A **subquery** is a query nested within a main query

- *In the SELECT clause, they can be used to make calculations*
- *In the FROM clause, they can be used to join query results and require an alias using AS*
- *In the WHERE or HAVING clause, they can be used to filter results (can use keywords like ANY, ALL and EXISTS)*
- *Avoid correlated subqueries by using INNER JOINs if possible*

A **common table expression (CTE)** creates a named temporary output

- *CTEs are more readable, can be referenced multiple times, and you can create multiple CTEs within a query*
- *Recursive CTEs are useful for generating values and working with hierarchical data, but are less common*

Temporary tables & **views** are other options for using the results of a query

- *Subqueries and CTEs exists only for the duration of a query, and require minimal permissions to create*
- *Temporary tables exist for the duration of a session and views exist indefinitely, but they often require additional permissions to create and maintain*

WINDOW FUNCTIONS

WINDOW FUNCTIONS

In this section, we'll break down each component of a **window function**, introduce common window functions, and preview some of their applications

TOPICS WE'LL COVER:

- Window Function Basics
- Window Functions
- Applications Preview

GOALS FOR THIS SECTION:

- Review the similarities and differences between aggregate and window functions
- Identify the individual components of a window function and understand what each of them do
- Learn about the numerous functions that are available when working with window functions
- Get a taste of the practical applications of window functions *(more on this in the last section!)*

WINDOW FUNCTION BASICS

Window Function Basics

Window Functions

Applications Preview

Window functions are used to apply a function to a "window" of data

- Windows are essentially **groups of rows** of data

This is a window

This is a window

This is a window

country	year	happiness_score	row_num
Afghanistan	2020	2.567	1
Afghanistan	2021	2.523	2
Afghanistan	2022	2.404	3
Afghanistan	2023	1.859	4
Albania	2020	4.883	1
Albania	2021	5.117	2
Albania	2022	5.199	3
Albania	2023	5.277	4
Algeria	2020	5.005	1
Algeria	2021	4.887	2
Algeria	2022	5.122	3
Algeria	2023	5.329	4

The window function applied here is the ROW_NUMBER() function, which returns a series of numbers for each window

AGGREGATE VS WINDOW FUNCTIONS

How are window functions different than a GROUP BY?

- **Aggregate functions** collapse the rows in each group and apply a calculation
- **Window functions** leave the rows as they are and apply calculations by window

```sql
-- Aggregate function
SELECT    country,
          AVG(happiness_score) AS avg_hs
FROM      hscores
GROUP BY  country;
```

country	avg_hs
Afghanistan	2.3382500
Albania	5.1190000
Algeria	5.0857500
Argentina	5.9737500
Armenia	5.1752500

A calculation is made for each group

There is one row per country

```sql
-- Window function
SELECT    country, year, happiness_score,
          ROW_NUMBER() OVER
          (PARTITION BY country
          ORDER BY year) AS row_num
FROM      hscores;
```

country	year	happiness_score	row_num
Afghanistan	2020	2.567	1
Afghanistan	2021	2.523	2
Afghanistan	2022	2.404	3
Afghanistan	2023	1.859	4
Albania	2020	4.883	1
Albania	2021	5.117	2
Albania	2022	5.199	3
Albania	2023	5.277	4

The original row granularity is kept

A calculation is applied to each window

WINDOW FUNCTION COMPONENTS

Window Function Basics

Window Functions

Applications Preview

Window functions consist of **four main components**:

States that this is a window function
(required)

How each window should be sorted before the function is applied
(optional in MySQL, PostgreSQL, SQLite)
(required in Oracle, SQL Server)

```
ROW_NUMBER() OVER (PARTITION BY country ORDER BY happiness_score)
```

The function to apply to each window (required)

Examples:

- *ROW_NUMBER*
- *FIRST_VALUE*
- *LAG*

How you're splitting the rows into windows (optional)

Examples:

- *One column*
- *Multiple columns*
- *The entire table (if left blank)*

ROW_NUMBER() OVER

```
ROW_NUMBER() OVER (PARTITION BY country ORDER BY happiness_score)
```


Window Function Basics

Window Functions

Applications Preview

This is the most basic window function:

- **ROW_NUMBER()** is a commonly used window function that numbers each row in a window
- **OVER** states that we are using a window function

```
SELECT *
FROM hscores;
```

country	year	happiness_score
Afghanistan	2020	2.567
Afghanistan	2021	2.523
Afghanistan	2022	2.404
Afghanistan	2023	1.859
Albania	2020	4.883
Albania	2021	5.117
Albania	2022	5.199
Albania	2023	5.277

```
-- Return all row numbers
SELECT   country, year, happiness_score,
         ROW_NUMBER() OVER() AS row_num
FROM     hscores;
```

country	year	happiness_score	row_num
Afghanistan	2020	2.567	1
Afghanistan	2021	2.523	2
Afghanistan	2022	2.404	3
Afghanistan	2023	1.859	4
Albania	2020	4.883	5
Albania	2021	5.117	6
Albania	2022	5.199	7
Albania	2023	5.277	8

Because we didn't specify a window with PARTITION BY, the function is applied to the entire table

PARTITION BY

```
ROW_NUMBER() OVER (PARTITION BY country ORDER BY happiness_score)
```

Window Function Basics

Window Functions

Applications Preview

A **PARTITION BY** allows you to define your windows

- You can partition by one or more columns

```
SELECT *
FROM hscores;
```

country	year	happiness_score
Afghanistan	2020	2.567
Afghanistan	2021	2.523
Afghanistan	2022	2.404
Afghanistan	2023	1.859
Albania	2020	4.883
Albania	2021	5.117
Albania	2022	5.199
Albania	2023	5.277

```
-- Return all row numbers within each window
SELECT   country, year, happiness_score,
         ROW_NUMBER() OVER(PARTITION BY country) AS row_num
FROM     hscores;
```

country	year	happiness_score	row_num
Afghanistan	2020	2.567	1
Afghanistan	2021	2.523	4
Afghanistan	2022	2.404	2
Afghanistan	2023	1.859	3
Albania	2020	4.883	2
Albania	2021	5.117	1
Albania	2022	5.199	3
Albania	2023	5.277	4

Note that the numbers go from 1 to 4 for each country, but they are in a seemingly random order (more on this up next!)

ORDER BY

```
ROW_NUMBER() OVER (PARTITION BY country ORDER BY happiness_score)
```


Window Function Basics

Window Functions

Applications Preview

An **ORDER BY** allows you to specify the order of the rows within your windows

- You can order in ASC *(default)* or DESC order by one or more columns

```
SELECT *
FROM hscores;
```

country	year	happiness_score
Afghanistan	2020	2.567
Afghanistan	2021	2.523
Afghanistan	2022	2.404
Afghanistan	2023	1.859
Albania	2020	4.883
Albania	2021	5.117
Albania	2022	5.199
Albania	2023	5.277

```
-- Return all row numbers within each window
-- where the rows are ordered by happiness score
SELECT   country, year, happiness_score,
         ROW_NUMBER() OVER(PARTITION BY country
         ORDER BY happiness_score) AS row_num
FROM     hscores
ORDER BY country, row_num;
```

country	year	happiness_score	row_num
Afghanistan	2023	1.859	1
Afghanistan	2022	2.404	2
Afghanistan	2021	2.523	3
Afghanistan	2020	2.567	4
Albania	2020	4.883	1
Albania	2021	5.117	2
Albania	2022	5.199	3
Albania	2023	5.277	4

These are now ordered by the happiness score for each country, from lowest to highest

ORDER BY

```
ROW_NUMBER() OVER (PARTITION BY country ORDER BY happiness_score)
```

Window Function Basics

Window Functions

Applications Preview

An **ORDER BY** allows you to specify the order of the rows within your windows

- You can order in ASC *(default)* or DESC order by one or more columns

```
SELECT *
FROM hscores;
```

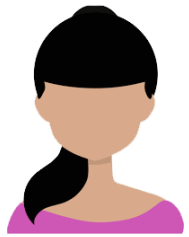
country	year	happiness_score
Afghanistan	2020	2.567
Afghanistan	2021	2.523
Afghanistan	2022	2.404
Afghanistan	2023	1.859
Albania	2020	4.883
Albania	2021	5.117
Albania	2022	5.199
Albania	2023	5.277

```
-- Return all row numbers within each window
-- where the rows are ordered by score descending
SELECT   country, year, happiness_score,
         ROW_NUMBER() OVER(PARTITION BY country
         ORDER BY happiness_score DESC) AS row_num
FROM     hscores
ORDER BY country, row_num;
```

country	year	happiness_score	row_num
Afghanistan	2020	2.567	1
Afghanistan	2021	2.523	2
Afghanistan	2022	2.404	3
Afghanistan	2023	1.859	4
Albania	2023	5.277	1
Albania	2022	5.199	2
Albania	2021	5.117	3
Albania	2020	4.883	4

And now from highest to lowest

ASSIGNMENT: WINDOW FUNCTION BASICS



NEW MESSAGE

November 12, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **New transaction number column**

Hello,

We currently have an orders report with customer, order and transaction IDs, and we would like to add an additional column that contains the transaction number for each customer as well.

Could you help us do this using window functions?

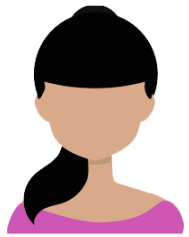
Thanks!
Mandy

Reply Forward

Results Preview

customer_id	order_id	order_date	transaction_id	transaction_number
10971	US-2024-133235-45505	2024-08-01	T6939	1
10973	US-2022-129525-44880	2022-11-15	T3064	1
10973	US-2022-130855-44924	2022-12-29	T3469	2
10978	US-2021-140165-44329	2021-05-13	T0395	1
10978	US-2021-140165-44329	2021-05-13	T0396	2
10978	US-2022-127509-44874	2022-11-09	T3009	3
10978	US-2022-127509-44874	2022-11-09	T3010	4
10978	US-2022-127509-44874	2022-11-09	T3011	5
10978	US-2023-103891-45119	2023-07-12	T4340	6
10985	US-2021-151953-44459	2021-09-20	T0977	1
10985	US-2023-105081-45285	2023-12-25	T5649	2
10985	US-2023-105081-45285	2023-12-25	T5650	3
10994	US-2022-153500-44745	2022-07-03	T2328	1
10994	US-2022-153500-44745	2022-07-03	T2329	2
10994	US-2022-153500-44745	2022-07-03	T2330	3

SOLUTION: WINDOW FUNCTION BASICS



NEW MESSAGE

November 12, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **New transaction number column**

Hello,

We currently have an orders report with customer, order and transaction IDs, and we would like to add an additional column that contains the transaction number for each customer as well.

Could you help us do this using window functions?

Thanks!
Mandy

Reply Forward

Solution Code

```sql
-- For each customer, add a column for transaction number
SELECT  customer_id, order_id, order_date, transaction_id,
        ROW_NUMBER() OVER(PARTITION BY customer_id
                          ORDER BY transaction_id) AS transaction_num
FROM    orders
ORDER BY customer_id, transaction_id;
```

WINDOW FUNCTIONS

Window Function Basics

Window Functions

Applications Preview

There are many **functions** to choose from when writing window functions:

Category	Functions
Row Numbering	• ROW_NUMBER • RANK • DENSE_RANK
Value Within a Window	• FIRST_VALUE • LAST_VALUE • NTH_VALUE
Value Relative to a Row	• LEAD • LAG
Aggregate Functions	• SUM, AVG, COUNT • MIN, MAX
Statistical Functions	• NTILE • CUME_DIST • PERCENT_RANK

These top three function categories are the most common, and we'll be covering these in detail in this section of the course

ROW NUMBERING

There are three different ways of **numbering rows** within a window:

- **ROW_NUMBER()** gives every row a unique number
- **RANK()** accounts for ties
- **DENSE_RANK()** accounts for ties and leaves no missing numbers in between

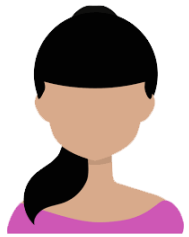
```
SELECT *
FROM baby_girl_names;
```

name	babies
Olivia	99
Emma	80
Charlotte	80
Amelia	75
Sophia	72
Isabella	70
Ava	70
Mia	64

```
SELECT   name, babies,
         ROW_NUMBER() OVER(ORDER BY babies DESC) AS b_row_number,
         RANK() OVER(ORDER BY babies DESC) AS b_rank,
         DENSE_RANK() OVER(ORDER BY babies DESC) AS b_dense_rank
FROM     baby_girl_names;
```

name	babies	b_row_number	b_rank	b_dense_rank
Olivia	99	1	1	1
Emma	80	2	2	2
Charlotte	80	3	2	2
Amelia	75	4	4	3
Sophia	72	5	5	4
Isabella	70	6	6	5
Ava	70	7	6	5
Mia	64	8	8	6

ASSIGNMENT: ROW NUMBERING



NEW MESSAGE

November 13, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Product rank**

Hello,

Our product team would like to know which products are most popular within each order.

Could you create a product rank field that returns a 1 for the most popular product in an order, 2 for second most, and so on? Please take a look at the results preview to get an idea of what they'd like the ranking to look like.

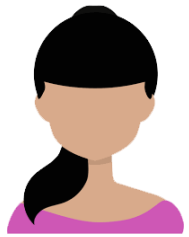
Thanks!
Mandy

Reply Forward

Results Preview

order_id	product_id	units	product_rank
US-2021-104563-44262	OTH-KAZ-38000	6	1
US-2021-104563-44262	CHO-SCR-58000	5	2
US-2021-104563-44262	CHO-FUD-51000	5	2
US-2021-104563-44262	CHO-NUT-13000	2	3
US-2021-104738-44560	CHO-SCR-58000	5	1
US-2021-104738-44560	CHO-FUD-51000	5	1
US-2021-104738-44560	CHO-NUT-13000	3	2
US-2021-104759-44286	CHO-MIL-31000	7	1
US-2021-104759-44286	CHO-NUT-13000	2	2
US-2021-104773-44538	CHO-FUD-51000	2	1
US-2021-104780-44337	CHO-MIL-31000	8	1

SOLUTION: ROW NUMBERING



NEW MESSAGE

November 13, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Product rank**

Hello,

Our product team would like to know which products are most popular within each order.

Could you create a product rank field that returns a 1 for the most popular product in an order, 2 for second most, and so on? Please take a look at the results preview to get an idea of what they'd like the ranking to look like.

Thanks!
Mandy

Reply Forward

Solution Code

```
-- For each order, rank the products from
-- most units to fewest units, while keeping ties
SELECT  order_id, product_id, units,
        DENSE_RANK() OVER(PARTITION BY order_id
                          ORDER BY units DESC) AS product_rank
FROM    orders
ORDER BY order_id, product_rank;
```

VALUE WITHIN A WINDOW

Window Function Basics
Window Functions
Applications Preview

There are three different ways of **extracting a particular value** within a window:

- **FIRST_VALUE()** extracts the first value in a window, in sequential row order
- **LAST_VALUE()** extracts the last value
- **NTH_VALUE()** extracts the value at a specified position

```
SELECT *
FROM baby_names;
```

gender	name	babies
Female	Charlotte	80
Female	Emma	82
Female	Olivia	99
Male	James	85
Male	Liam	110
Male	Noah	95

```
-- Return the first name in each window
SELECT   gender, name, babies,
         FIRST_VALUE(name) OVER (PARTITION BY gender
         ORDER BY babies DESC) AS top_name
FROM     baby_names;
```

gender	name	babies	top_name
Female	Olivia	99	Olivia
Female	Emma	82	Olivia
Female	Charlotte	80	Olivia
Male	Liam	110	Liam
Male	Noah	95	Liam
Male	James	85	Liam

Note that the first value is repeated across all rows in a window

VALUE WITHIN A WINDOW

There are three different ways of **extracting a particular value** within a window:

- **FIRST_VALUE()** extracts the first value in a window, in sequential row order
- **LAST_VALUE()** extracts the last value
- **NTH_VALUE()** extracts the value at a specified position

```
SELECT *
FROM baby_names;
```

gender	name	babies
Female	Charlotte	80
Female	Emma	82
Female	Olivia	99
Male	James	85
Male	Liam	110
Male	Noah	95

```
-- Return the second name in each window
SELECT   gender, name, babies,
         NTH_VALUE(name, 2) OVER (PARTITION BY gender
         ORDER BY babies DESC) AS second_name
FROM     baby_names;
```

gender	name	babies	second_name
Female	Olivia	99	NULL
Female	Emma	82	Emma
Female	Charlotte	80	Emma
Male	Liam	110	NULL
Male	Noah	95	Noah
Male	James	85	Noah

These values are NULL because the second value hasn't been encountered yet when scanning through the rows in the window

SQL Server doesn't support NTH_VALUE (you can use ROW_NUMBER instead)

VALUE WITHIN A WINDOW

Window Function Basics

Window Functions

Applications Preview

EXAMPLE

Return the top name for each gender

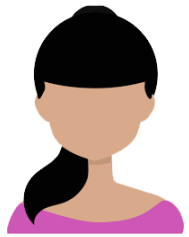
```
SELECT * FROM

(SELECT gender, name, babies,
        FIRST_VALUE(name) OVER (PARTITION BY gender
        ORDER BY babies DESC) AS top_name
FROM    baby_names) AS tn

WHERE name = top_name;
```

*Extract the first value for each gender ordered by number of babies in a **subquery***

Then simply keep the top names!

gender	name	babies	top_name
Female	Olivia	99	Olivia
Male	Liam	110	Liam

ASSIGNMENT: VALUE WITHIN A WINDOW



1 **NEW MESSAGE**

November 14, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Second most popular product**

Hello,

Could you specifically give me a list of the 2nd most popular product within each order?

The sales team is going to try to see if they can bundle them with some other products to increase units sold within each order.

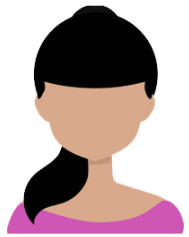
Thanks!
Mandy

Reply Forward

Results Preview

order_id	product_id	units
CA-2021-115238-44217	CHO-FUD-51000	6
CA-2021-117964-44532	CHO-TRI-54000	5
CA-2021-119508-44534	CHO-FUD-51000	5
CA-2021-125388-44488	OTH-LIC-15000	3
CA-2021-129322-44416	CHO-SCR-58000	2
CA-2021-131807-44474	OTH-LIC-15000	5
CA-2021-139675-44268	CHO-SCR-58000	4
CA-2021-151799-44544	CHO-SCR-58000	6
CA-2021-152443-44337	CHO-FUD-51000	6

SOLUTION: VALUE WITHIN A WINDOW



1 **NEW MESSAGE**

November 14, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Second most popular product**

Hello,

Could you specifically give me a list of the 2nd most popular product within each order?

The sales team is going to try to see if they can bundle them with some other products to increase units sold within each order.

Thanks!
Mandy

Reply Forward

Solution Code

```
-- Return the 2nd most popular product for each order
SELECT * FROM

(SELECT order_id, product_id, units,
        NTH_VALUE(product_id, 2)
        OVER(PARTITION BY order_id
             ORDER BY units DESC) AS second_product
FROM    orders
ORDER BY order_id, second_product) AS sp

WHERE product_id = second_product;
```

ORDER BY in subquery is not needed and can be omitted

VALUE RELATIVE TO A ROW

Window Function Basics

Window Functions

Applications Preview

LEAD() and **LAG()** allow you to return the value from the next and previous row, respectively, within each window

```
-- Return the prior year's happiness score
SELECT    country, year, happiness_score,
          LAG(happiness_score) OVER(PARTITION BY country
          ORDER BY year) AS prior_happiness_score
FROM      hscores;
```

country	year	happiness_score	prior_happiness_score
Afghanistan	2020	2.567	NULL
Afghanistan	2021	2.523	2.567
Afghanistan	2022	2.404	2.523
Afghanistan	2023	1.859	2.404
Albania	2020	4.883	NULL
Albania	2021	5.117	4.883
Albania	2022	5.199	5.117
Albania	2023	5.277	5.199

The first value is NULL because there is no previous value in the window

VALUE RELATIVE TO A ROW

Window Function Basics

Window Functions

Applications Preview

EXAMPLE

Calculate the difference in happiness scores over time, by country

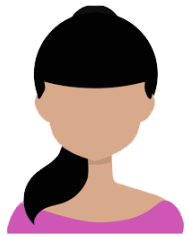
```
WITH hs_prior AS (SELECT country, year, happiness_score,
                         LAG(happiness_score) OVER(PARTITION BY country
                         ORDER BY year) AS prior_happiness_score
                  FROM   hscores)

SELECT   country, year, happiness_score, prior_happiness_score,
         happiness_score - prior_happiness_score AS hs_change
FROM     hs_prior;
```

← *Return the prior year's happiness score for each country in a **CTE***

← *Then simply subtract them!*

country	year	happiness_score	prior_happiness_score	hs_change
Afghanistan	2020	2.567	NULL	NULL
Afghanistan	2021	2.523	2.567	-0.044
Afghanistan	2022	2.404	2.523	-0.119
Afghanistan	2023	1.859	2.404	-0.545
Albania	2020	4.883	NULL	NULL
Albania	2021	5.117	4.883	0.234
Albania	2022	5.199	5.117	0.082
Albania	2023	5.277	5.199	0.078

ASSIGNMENT: VALUE RELATIVE TO A ROW



NEW MESSAGE

November 15, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Change in orders over time**

Hello,

We'd like to look into how orders have changed over time for each customer.

Could you produce a table that contains info about each customer and their orders, the number of units in each order, and the change in units from order to order?

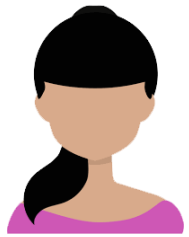
Thanks!
Mandy

Reply Forward

Results Preview

customer_id	order_id	total_units	prior_units	diff_units
11148	US-2021-124394-44486	8	NULL	NULL
11148	US-2022-146290-44685	7	8	-1
11148	US-2023-102792-45273	3	7	-4
11148	US-2024-167570-45634	4	3	1
11155	US-2024-128447-45606	6	NULL	NULL
11162	US-2022-112711-44754	4	NULL	NULL
11162	US-2022-103723-44896	5	4	1
11162	US-2024-128076-45324	6	5	1

SOLUTION: VALUE RELATIVE TO A ROW



1 **NEW MESSAGE**

November 15, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Change in orders over time**

Hello,

We'd like to look into how orders have changed over time for each customer.

Could you produce a table that contains info about each customer and their orders, the number of units in each orders, and the change in units from order to order?

Thanks!
Mandy

Reply Forward

Solution Code

```sql
-- For each customer, find the change in units per order over time

-- APPROACH 1: One CTE - more concise approach
WITH my_cte AS (SELECT   customer_id, order_id,
                         MIN(transaction_id) AS min_tid, SUM(units) AS total_units
                FROM     orders
                GROUP BY customer_id, order_id)

SELECT  customer_id, order_id, total_units,
        LAG(total_units) OVER(PARTITION BY customer_id ORDER BY min_tid) AS prior_units,
        total_units - LAG(total_units) OVER(PARTITION BY customer_id ORDER BY min_tid)
FROM    my_cte;

-- APPROACH 2: Multiple CTEs - step-by-step approach
WITH my_cte AS (SELECT   customer_id, order_id,
                         MIN(transaction_id) min_tid, SUM(units) AS total_units
                FROM     orders
                GROUP BY customer_id, order_id),

     prior_cte AS (SELECT   customer_id, order_id, total_units,
                            LAG(total_units) OVER(
                                PARTITION BY customer_id ORDER BY min_tid) AS prior_units
                   FROM     my_cte)

SELECT  customer_id, order_id, total_units,
        prior_units, total_units - prior_units AS diff_units
FROM    prior_cte;
```

STATISTICAL FUNCTIONS

Window Function Basics

Window Functions

Applications Preview

NTILE() divides the rows in a window into a specified number of percentiles

EXAMPLE

View the top 25% of happiness scores for each region

```
SELECT  region, country, happiness_score,
        NTILE(4) OVER(PARTITION BY region ORDER BY happiness_score DESC) AS hs_percentile
FROM    happiness_scores
WHERE   year = 2023;
```

region	country	happiness_score	hs_percentile
Central and Eastern Europe	Czechia	6.845	1
Central and Eastern Europe	Lithuania	6.763	1
Central and Eastern Europe	Slovenia	6.650	1
Central and Eastern Europe	Romania	6.589	1
Central and Eastern Europe	Slovakia	6.469	1
Central and Eastern Europe	Estonia	6.455	1
Central and Eastern Europe	Kosovo	6.368	2
Central and Eastern Europe	Poland	6.260	2
Central and Eastern Europe	Latvia	6.213	2

Because we specified NTILE(4), the range of 100% of the rows in each window is divided into 4 groups of 25%, with 1 representing the top percentile group, and 4 the bottom

NTILE is not supported in SQLite, but you can simulate it using other window functions

STATISTICAL FUNCTIONS

Window Function Basics

Window Functions

Applications Preview

NTILE() divides the rows in a window into a specified number of percentiles

EXAMPLE

View the top 25% of happiness scores for each region

```
WITH hs_pct AS (SELECT   region, country, happiness_score,
                         NTILE(4) OVER(PARTITION BY region ORDER BY happiness_score DESC) AS hs_percentile
                FROM     happiness_scores
                WHERE    year = 2023)

SELECT   *
FROM     hs_pct
WHERE    hs_percentile = 1
ORDER BY region, happiness_score DESC;
```

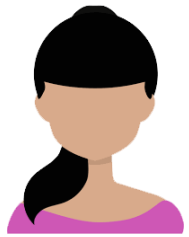
*Return the percentiles in a **CTE***

Then simply filter for the top percentile (25%)

region	country	happiness_score	hs_percentile
North America and ANZ	New Zealand	7.123	1
South Asia	Nepal	5.360	1
South Asia	Pakistan	4.555	1
Southeast Asia	Singapore	6.587	1
Southeast Asia	Malaysia	6.012	1
Southeast Asia	Thailand	5.843	1

The number of results will differ by region because some regions have more countries, so the top 25% calculation can contain more or fewer rows

ASSIGNMENT: STATISTICAL FUNCTIONS



NEW MESSAGE

November 18, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Top 1% of customers**

Hello,

The customer engagement team would like to create a rewards program for our top 1% of customers.

Could you pull a list of the top 1% of customers in terms of how much they've spent with us?

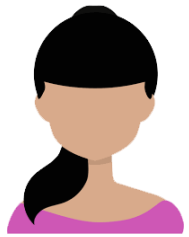
Thanks!
Mandy

Reply Forward

Results Preview

customer_id	total_spend	spend_pct
31031	352.20	1
15640	334.00	1
32571	322.30	1
35926	304.14	1
28417	298.86	1
45958	265.74	1
40090	261.62	1
28224	250.05	1
46434	247.35	1
38615	237.05	1
16076	232.46	1
27307	232.40	1
28527	232.40	1

SOLUTION: STATISTICAL FUNCTIONS



NEW MESSAGE

November 18, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Top 1% of customers**

Hello,

The customer engagement team would like to create a rewards program for our top 1% of customers.

Could you pull a list of the top 1% of customers in terms of how much they've spent with us?

Thanks!
Mandy

Reply Forward

Solution Code

```
-- Return the top 1% of customers in terms of spending
WITH ts AS (SELECT   o.customer_id,
                     SUM(o.units * p.unit_price) AS total_spend
            FROM     orders o LEFT JOIN products p
                     ON o.product_id = p.product_id
            GROUP BY o.customer_id
            ORDER BY total_spend DESC),

     sp AS (SELECT   customer_id, total_spend,
                     NTILE(100)
                     OVER(ORDER BY total_spend DESC) AS spend_pct
            FROM     ts)

SELECT   *
FROM     sp
WHERE    spend_pct = 1;
```

ORDER BY in CTE is not needed and can be omitted

PREVIEW: WINDOW FUNCTION APPLICATIONS

There are many **practical applications** of window functions, including calculating moving averages, running totals, and more

- Window Function Basics
- Window Functions
- Applications Preview

```sql
-- Calculate the three year moving average of happiness scores
SELECT   country, year, happiness_score,
         AVG(happiness_score) OVER (PARTITION BY country
         ORDER BY year ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
         AS three_year_ma
FROM     happiness_scores;
```

Note that this uses an AVG() aggregate function as part of a window function

We've added some additional keywords at the end that specify rows within the window

country	year	happiness_score	three_year_ma
Afghanistan	2015	3.575	3.5750000
Afghanistan	2016	3.360	3.4675000
Afghanistan	2017	3.794	3.5763333
Afghanistan	2018	3.632	3.5953333
Afghanistan	2019	3.203	3.5430000
Afghanistan	2020	2.567	3.1340000
Afghanistan	2021	2.523	2.7643333
Afghanistan	2022	2.404	2.4980000
Afghanistan	2023	1.859	2.2620000

In the **Data Analysis Applications** section of this course, we'll cover the following applications of window functions:

- Removing duplicate rows
- Min / max value filtering
- Rolling calculations

KEY TAKEAWAYS

Window functions are used to apply a function across windows of data

- *Windows refer to groups of rows in a table*
- *Aggregate functions collapse the rows in each group, but window functions leave the rows untouched*

The general syntax is **FUNCTION OVER(PARTITION BY x ORDER BY y)**

- *OVER indicates that we're writing a window function*
- *PARTITION BY states how we'd like to split up the rows into groups*
- *ORDER BY states how the rows within each window should be ordered before applying the function*

The **function** portion of a window function is applied to each window

- *You can number rows with ROW_NUMBER(), RANK(), and DENSE_RANK()*
- *You can identify values within a window with FIRST_VALUE(), LAST_VALUE() and NTH_VALUE()*
- *You can return values from relative rows with LEAD() and LAG()*
- *You can use statistical functions like NTILE() for making percentile calculations*
- *You can use aggregate functions like AVG() for making moving average calculations*

FUNCTIONS BY DATA TYPE

FUNCTIONS BY DATA TYPE

In this section, we'll go over commonly used **functions by data type**, including numeric, datetime, string functions, and more

TOPICS WE'LL COVER:

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

GOALS FOR THIS SECTION:

- Review three categories of SQL functions: aggregate, window, and general functions
- Recognize that most SQL functions will only work on specific data types
- Learn and apply commonly used SQL functions across different data types

FUNCTION BASICS

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

SQL functions take in zero or more inputs, apply a calculation or transformation, and output a value

- You can recognize a function by the parentheses () that follow a keyword

```sql
-- Function that applies a calculation
SELECT  COUNT(*)
FROM    happiness_scores;
```

```sql
-- Function that applies a transformation
SELECT  ROUND(happiness_score)
FROM    happiness_scores;
```

```sql
-- Function with no inputs
SELECT  CURRENT_DATE();
```

```sql
-- DISTINCT is a keyword, not a function
SELECT  COUNT(DISTINCT country)
FROM    happiness_scores;
```

PRO TIP: While SQL is case insensitive, it's a best practice to **capitalize functions** so they stand out, similar to *clauses*. If you're using a SQL editor, they will automatically be highlighted a different color.

FUNCTION CATEGORIES

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

These are three **function categories**: aggregate, window, and general functions

Aggregate

- Applies a calculation to many rows of data and returns a single value
- Often used alongside a GROUP BY

```
SELECT  AVG(happiness_score) AS avg_hs
FROM    happiness_scores;
```

avg_hs
5.4400624

Window

- Performs calculations across a window of rows

```
SELECT   country,
         AVG(happiness_score) AS avg_hs
FROM     happiness_scores
GROUP BY country;
```

country	avg_hs
Afghanistan	2.9907778
Albania	4.8932222
Algeria	5.4090000

General

- Performs calculations on all individual values within a column

FUNCTION CATEGORIES

These are three **function categories**: aggregate, window, and general functions

Aggregate

- Applies a calculation to many rows of data and returns a single value
- Often used alongside a GROUP BY

Window

- Performs calculations across a window of rows

General

- Performs calculations on all individual values within a column

```
-- Happiest countries each year
SELECT   year, country, happiness_score,
         ROW_NUMBER() OVER(PARTITION BY year
         ORDER BY happiness_score DESC)
         AS happiness_score_annual_ranking
FROM     happiness_scores;
```

year	country	happiness_score	happiness_score_annual_ranking
2015	Switzerland	7.587	1
2015	Iceland	7.561	2
2015	Denmark	7.527	3
2015	Norway	7.522	4
2015	Canada	7.427	5
2015	Finland	7.406	6
2015	Netherlands	7.378	7
2015	Sweden	7.364	8

FUNCTION CATEGORIES

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

These are three **function categories**: aggregate, window, and general functions

Aggregate

- Applies a calculation to many rows of data and returns a single value
- Often used alongside a GROUP BY

Window

- Performs calculations across a window of rows

General

- Performs calculations on all individual values within a column

```
-- Apply functions to two columns
SELECT  year, country, happiness_score,
        UPPER(country) AS country_upper,
        ROUND(happiness_score, 1) AS hs_rounded
FROM    happiness_scores;
```

year	country	happiness_score	country_upper	hs_rounded
2015	Afghanistan	3.575	AFGHANISTAN	3.6
2015	Albania	4.959	ALBANIA	5.0
2015	Algeria	5.605	ALGERIA	5.6
2015	Angola	4.033	ANGOLA	4.0
2015	Argentina	6.574	ARGENTINA	6.6
2015	Armenia	4.350	ARMENIA	4.4
2015	Australia	7.284	AUSTRALIA	7.3
2015	Austria	7.200	AUSTRIA	7.2
2015	Azerbaijan	5.212	AZERBAIJAN	5.2

NUMERIC FUNCTIONS

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

Numeric functions can be applied to numeric columns *(integer, decimal, etc.)*

Category	Function	Description
Math	ABS	Absolute value
	SIGN	Returns -1, 0, or 1 depending on the sign of the number
	POWER	x to the power of y
	SQRT	Square root
	LOG	Log of y base x
	MOD *(% in SQL Server)*	Remainder of x / y
Rounding	ROUND	Rounds a number to n decimal places
	FLOOR	Rounds a number down (*not supported in SQLite*)
	CEIL (*CEILING in SQL Server*)	Rounds a number up (*not supported in SQLite*)

NUMERIC FUNCTIONS

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

Numeric functions can be applied to numeric columns *(integer, decimal, etc.)*

EXAMPLE

Applying a log transform to the population of each country

```
SELECT   country, population,
         LOG(population) AS log_pop,
         ROUND(LOG(population), 2) AS log_pop_2
FROM     country_stats;
```

*This is a **nested function***

country	population	log_pop	log_pop_2
Afghanistan	38041754	17.45419490393703	17.45
Albania	2854191	14.864298998395611	14.86
Algeria	43053054	17.57794372708839	17.58
Angola	31825295	17.275771971807046	17.28
Argentina	44938712	17.6208101638746	17.62
Armenia	2957731	14.8999329782934	14.9
Australia	25766605	17.064589831613713	17.06

PRO TIP: LEAST & GREATEST

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

LEAST() and **GREATEST()** are two lesser-known functions that apply a calculation across a row instead of a column

```
SELECT *
FROM miles_run;
```

name	q1	q2	q3	q4
Ali	100	200	150	NULL
Bolt	350	400	380	300
Jordan	200	250	300	320

```
SELECT  GREATEST(q1, q2, q3, q4) AS most_miles
FROM    miles_run;
```

most_miles
NULL
400
320

We'll discuss how to handle NULL values like these later in the section

LEAST() and GREATEST() are **not supported in SQL Server**

PRO TIP: CAST & CONVERT

Sometimes you may need to **cast columns to different data types**, so that you can utilize specific functions


```sql
CREATE TABLE sample_table (
    id INT,
    str_value CHAR(50)
);

INSERT INTO sample_table (id, str_value) VALUES
(1, '100.2'),
(2, '200.4'),
(3, '300.6');
```

```sql
-- Turn the string to a float
SELECT  id, str_value,
        CAST(str_value AS FLOAT)*2 AS float_value
FROM    sample_table;
```

id	str_value	float_value
1	100.2	200.4
2	200.4	400.8
3	300.6	601.2

In some RDBMS's like SQL Server, use CONVERT instead of CAST

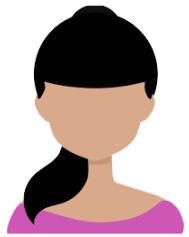
```sql
-- Turn the integer into a float
SELECT  employee_id, name, salary,
        salary / 1.0 AS salary_float
FROM    employees;
```

Dividing by 1.0 turns an integer into a float (includes decimals)

employee_id	name	salary	salary_float
1	Ava	85000	85000.0000
2	Bob	72000	72000.0000
3	Cat	59000	59000.0000
4	Dan	85000	85000.0000

The CAST / CONVERT functions only change the data type **for the duration of the query**, not permanently

ASSIGNMENT: NUMERIC FUNCTIONS



NEW MESSAGE

November 19, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Customer spend bins**

Hello,

Our market research team is interested in seeing how many customers have spent $0-$10 on our products, $10-$20, and so on for every $10 range.

Could you generate this table for them?

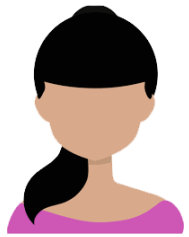
Thanks!
Mandy

Reply Forward

Results Preview

total_spend_bin	num_customers
0	367
10	600
20	484
30	393
40	299
50	270
60	175
70	146
80	100
90	88
100	71

SOLUTION: NUMERIC FUNCTIONS



NEW MESSAGE

November 19, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Customer spend bins**

Hello,

Our market research team is interested in seeing how many customers have spent $0-$10 on our products, $10-$20, and so on for every $10 range.

Could you generate this table for them?

Thanks!
Mandy

Reply Forward

Solution Code

```
-- Number of customers in each spend bin
WITH bin AS (SELECT o.customer_id,
                    SUM(o.units * p.unit_price) AS total_spend,
                    FLOOR(SUM(o.units * p.unit_price) / 10) * 10
                        AS total_spend_bin
             FROM   orders o LEFT JOIN products p
                    ON o.product_id = p.product_id
             GROUP BY o.customer_id)

SELECT   total_spend_bin, COUNT(customer_id) AS num_customers
FROM     bin
GROUP BY total_spend_bin
ORDER BY total_spend_bin;
```

DATETIME FUNCTIONS

Datetime functions can be applied to datetime columns *(date, time, etc.)*

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

Category	Function	Description
Current	CURRENT_DATE	Current date
	CURRENT_TIMESTAMP	Current date and time
Extract	YEAR, MONTH, etc.	Extract a specific portion of the datetime value
	DAYOFWEEK	Extract the day of the week from the datetime value
Difference	DATEDIFF / AGE	Interval between two datetimes
	DATE_ADD / DATE_SUB	Add or subtract an interval to or from a datetime

Datetime and string functions vary widely by RDMBS. The functions in this section are for **MySQL**, but you may need to look up the specific function in your RDBMS

DATETIME FUNCTIONS

Function Basics

Numeric Functions

Datetime Functions

String Functions

NULL Functions

Datetime functions can be applied to datetime columns *(date, time, etc.)*

```sql
-- Extract info about datetime values
SELECT  event_name, event_date,
        YEAR(event_date) AS event_year,
        MONTH(event_date) AS event_month,
        DAYOFWEEK(event_date) AS event_dow
FROM    events;
```

event_name	event_date	event_year	event_month	event_dow
New Year's Day	2025-01-01	2025	1	4
Lunar New Year	2025-01-29	2025	1	4
Persian New Year	2025-03-20	2025	3	5
Birthday	2025-05-13	2025	5	3
Last Day of School	2025-06-12	2025	6	5
Vacation	2025-08-01	2025	8	6
First Day of School	2025-08-18	2025	8	2

← *For day of the week, 1 is Sunday, 2 is Monday, and so on*

```sql
-- Get current date and time
SELECT CURRENT_DATE(), CURRENT_TIMESTAMP();
```

CURRENT_DATE	CURRENT_TIMESTAMP
2024-10-11	2024-10-11 15:45:46

DATETIME FUNCTIONS

Datetime functions can be applied to datetime columns *(date, time, etc.)*

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

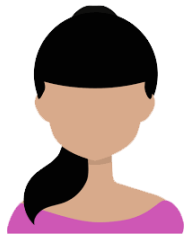
```sql
-- Calculate an interval between datetime values
SELECT  event_name, event_date, CURRENT_DATE,
        DATEDIFF(event_date, CURRENT_DATE) AS days_until
FROM    events;
```

event_name	event_date	CURRENT_DATE	days_until
New Year's Day	2025-01-01	2024-10-11	82
Lunar New Year	2025-01-29	2024-10-11	110
Persian New Year	2025-03-20	2024-10-11	160

```sql
-- Add / subtract an interval from a datetime value
SELECT  event_name, event_datetime,
        DATE_ADD(event_datetime, INTERVAL 1 HOUR) AS plus_one_hour
FROM    events;
```

event_name	event_datetime	plus_one_hour
New Year's Day	2025-01-01 00:00:00	2025-01-01 01:00:00
Lunar New Year	2025-01-29 10:00:00	2025-01-29 11:00:00
Persian New Year	2025-03-20 12:00:00	2025-03-20 13:00:00

ASSIGNMENT: DATETIME FUNCTIONS



NEW MESSAGE

November 20, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Add shipping dates for Q2 orders**

Hello,

The market research team wants to do a deep dive on the Q2 2024 orders data we currently have.

Can you pull that data for them?

In addition, they also requested that we include a ship_date column for them that's 2 days after the order_date.

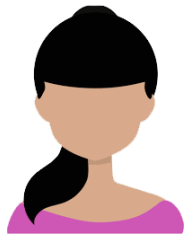
Thanks for your help!
Mandy

Reply Forward

Results Preview

order_id	order_date	ship_date
US-2024-129028-45383	2024-04-01	2024-04-03
US-2024-134495-45383	2024-04-01	2024-04-03
US-2024-134495-45383	2024-04-01	2024-04-03
US-2024-134495-45383	2024-04-01	2024-04-03
US-2024-145877-45383	2024-04-01	2024-04-03
US-2024-145877-45383	2024-04-01	2024-04-03
US-2024-147942-45383	2024-04-01	2024-04-03
US-2024-147942-45383	2024-04-01	2024-04-03
US-2024-149881-45383	2024-04-01	2024-04-03
US-2024-149881-45383	2024-04-01	2024-04-03
US-2024-118360-45383	2024-04-01	2024-04-03
US-2024-121419-45384	2024-04-02	2024-04-04
US-2024-121419-45384	2024-04-02	2024-04-04

SOLUTION: DATETIME FUNCTIONS



NEW MESSAGE

November 20, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Add shipping dates for Q2 orders**

Hello,

The market research team wants to do a deep dive on the Q2 2024 orders data we currently have.

Can you pull that data for them?

In addition, they also requested that we include a ship_date column for them that's 2 days after the order_date.

Thanks for your help!
Mandy

Reply Forward

Solution Code

```sql
-- Q2 2024 orders, order dates and shipping dates
SELECT  order_id, order_date,
        DATE_ADD(order_date, INTERVAL 2 DAY) AS ship_date
FROM    orders
WHERE   YEAR(order_date) = 2024
        AND MONTH(order_date) BETWEEN 4 AND 6;


/*  This function varies by RDBMS:
- MySQL:      DATE_ADD(order_date, INTERVAL 2 DAY)
- Oracle:     order_date + INTERVAL '2' DAY
- PostgreSQL: order_date + INTERVAL '2 days'
- SQL Server: DATEADD(DAY, 2, order_date)
- SQLite:     DATE(order_date, '+2 days')
*/
```

STRING FUNCTIONS

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

String functions can be applied to string columns *(char, varchar, text, etc.)*

Category	Function	Description
Length	LENGTH	Returns the number of characters in a string
Update	TRIM	Removes leading or trailing spaces (or other characters)
	REPLACE	Replaces a substring with another
Case	UPPER / LOWER	Converts all characters to uppercase or lowercase
Combine	CONCAT	Combines multiple strings into one
Find Location	INSTR	Returns the position of a substring within a string
	SUBSTR	Extracts part of a string of a specified length, starting from a given position
Find Pattern	LIKE*	Performs a pattern match within a string, typically using wildcards
	REGEXP	Matches a string against a regular expression pattern

This is a **keyword, not a function*

STRING FUNCTIONS

String functions can be applied to string columns *(char, varchar, text, etc.)*

- Function Basics
- Numeric Functions
- Datetime Functions
- String Functions
- NULL Functions

```sql
-- Cleaned up event type and found the length of the description
SELECT  event_name, event_type,
        TRIM(REPLACE(event_type, '!', '')) AS event_type_clean,
        event_desc,
        LENGTH(event_desc) AS desc_len
FROM    events;
```

event_name	event_type	event_type_clean	event_desc	desc_len
New Year's Day	Holiday	Holiday	A global celebration to mark the beginning of the New Year. Festivities…	214
Lunar New Year	Holiday	Holiday	A significant cultural event in many Asian countries, the Lunar New Ye…	220
Persian New Year	Holiday	Holiday	Known as Nowruz, this celebration marks the first day of spring and th…	228
Birthday	Personal!	Personal	A personal celebration marking the anniversary of one's birth. This spe…	214
Last Day of School	Personal!	Personal	The final day of the academic year, celebrated by students and teache…	223
Vacation	Personal!	Personal	A much-anticipated break from daily routines, this vacation period allo…	212

STRING FUNCTIONS

Function Basics

Numeric Functions

Datetime Functions

String Functions

NULL Functions

String functions can be applied to string columns *(char, varchar, text, etc.)*

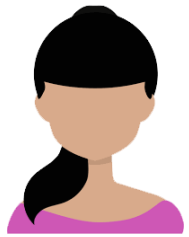
```
-- Change the case
SELECT  event_name, UPPER(event_name), LOWER(event_name)
FROM    events;
```

event_name	UPPER(event_name)	LOWER(event_name)
New Year's Day	NEW YEAR'S DAY	new year's day
Lunar New Year	LUNAR NEW YEAR	lunar new year
Persian New Year	PERSIAN NEW YEAR	persian new year

```
-- Combine the type and description columns
SELECT  event_name, event_type_clean, event_desc,
        CONCAT(event_type_clean, ' | ', event_desc) AS full_desc
FROM    events;
```

event_name	event_type_clean	event_desc	full_desc
New Year's Day	Holiday	A global celebration to mark…	Holiday | A global celebration to mark…
Lunar New Year	Holiday	A significant cultural event i…	Holiday | A significant cultural event i…
Persian New Year	Holiday	Known as Nowruz, this cele…	Holiday | Known as Nowruz, this cele…

ASSIGNMENT: STRING FUNCTIONS



NEW MESSAGE

November 21, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Update product ID**

Hello,

We're updating our product_ids to include the factory name and product name.

Here's what we're thinking it should look like.

Could you write the SQL code to produce this?

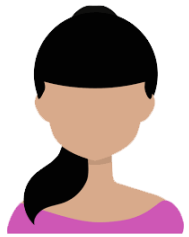
Thank you!
Mandy

Reply Forward

Results Preview

factory	product_id	factory_product_id
Lot's O' Nuts	CHO-FUD-51000	Lots-O-Nuts-CHO-FUD-51000
Lot's O' Nuts	CHO-NUT-13000	Lots-O-Nuts-CHO-NUT-13000
Lot's O' Nuts	CHO-SCR-58000	Lots-O-Nuts-CHO-SCR-58000
Secret Factory	OTH-GUM-21000	Secret-Factory-OTH-GUM-21000
Secret Factory	OTH-LIC-15000	Secret-Factory-OTH-LIC-15000
Secret Factory	SUG-EVE-47000	Secret-Factory-SUG-EVE-47000
Sugar Shack	OTH-FIZ-56000	Sugar-Shack-OTH-FIZ-56000
Sugar Shack	SUG-FUN-75000	Sugar-Shack-SUG-FUN-75000
Sugar Shack	SUG-LAF-25000	Sugar-Shack-SUG-LAF-25000

SOLUTION: STRING FUNCTIONS



NEW MESSAGE

November 21, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Update product ID**

Hello,

We're updating our product_ids to include the factory name and product name.

Here's what we're thinking it should look like.

Could you write the SQL code to produce this?

Thank you!
Mandy

Reply Forward

Solution Code

```
-- Create new ID column called factory_product_id
WITH fp AS (SELECT  factory, product_id,
                   REPLACE(REPLACE(factory, "'", ""), " ", "_")
                       AS factory_clean
            FROM    products
            ORDER BY factory, product_id)

SELECT  factory_clean, product_id,
        CONCAT(factory_clean, "-", product_id) AS factory_product_id
FROM    fp;


/*  The CONCAT function will work in MySQL and SQL Server
    In Oracle, PostgreSQL and SQLite, use the following instead:
    factory_clean || '-' || product_id AS factory_product_id
*/
```

ORDER BY in CTE is not needed and can be omitted

STRING FUNCTIONS: PATTERN MATCHING

String functions can be applied to string columns *(char, varchar, text, etc.)*

```sql
-- Return the first word of each event
SELECT  event_name,
        SUBSTR(event_name, 1, INSTR(event_name, ' ') - 1) AS first_word
FROM    events;
```

event_name	first_word
New Year's Day	New
Lunar New Year	Lunar
Persian New Year	Persian
Birthday	
Last Day of School	Last
Vacation	

Since this extracts the text before the first space, it doesn't work for strings with a single word

```sql
-- Update to handle single word events
SELECT  event_name,
        CASE WHEN INSTR(event_name, ' ') = 0 THEN event_name
             ELSE SUBSTR(event_name, 1, INSTR(event_name, ' ') - 1)
             END AS first_word
FROM    events;
```

event_name	first_word
New Year's Day	New
Lunar New Year	Lunar
Persian New Year	Persian
Birthday	Birthday
Last Day of School	Last
Vacation	Vacation

This CASE statement returns the full event name if it can't find a space in the text string

STRING FUNCTIONS: PATTERN MATCHING

String functions can be applied to string columns (*char*, *varchar*, *text*, *etc.*)

```
-- Return descriptions that contain 'family'
SELECT  *
FROM    events
WHERE   event_desc LIKE '%family%';
```

*This is a **keyword**, not a function*

event_name	event_date	event_datetime	event_type	event_desc
Lunar New Year	2025-01-29	2025-01-29 10:00:00	Holiday	A significant cultural event in many Asia…
Persian New Year	2025-03-20	2025-03-20 12:00:00	Holiday	Known as Nowruz, this celebration mark…
Birthday	2025-05-13	2025-05-13 18:00:00	Personal!	A personal celebration marking the anni…
Thanksgiving	2025-11-27	2025-11-27 12:00:00	Holiday	A holiday rooted in gratitude and family,…

```
-- Return descriptions that start with 'A'
SELECT  *
FROM    events
WHERE   event_desc LIKE 'A %';
```

*"%" and "_" are **wildcard** characters*

event_name	event_date	event_datetime	event_type	event_desc
New Year's Day	2025-01-01	2025-01-01 00:00:00	Holiday	A global celebration to mark the beginni…
Lunar New Year	2025-01-29	2025-01-29 10:00:00	Holiday	A significant cultural event in many Asia…
Birthday	2025-05-13	2025-05-13 18:00:00	Personal!	A personal celebration marking the anni…
Vacation	2025-08-01	2025-08-01 08:00:00	Personal!	A much-anticipated break from daily rout…

STRING FUNCTIONS: PATTERN MATCHING

String functions can be applied to string columns *(char, varchar, text, etc.)*

```
-- Note any celebration word in the sentence
SELECT  event_desc,
        REGEXP_SUBSTR(event_desc, 'celebration|festival|holiday') AS celebration_word
FROM    events;
```

event_desc	celebration_word
A global celebration to mark the beginning of the New Year. Festivities often include firework…	celebration
A significant cultural event in many Asian countries, the Lunar New Year, also known as the…	Festival
Known as Nowruz, this celebration marks the first day of spring and the beginning of the yea…	celebration
A personal celebration marking the anniversary of one's birth. This special day often involves…	celebration
The final day of the academic year, celebrated by students and teachers alike. It often includ…	NULL
A much-anticipated break from daily routines, this vacation period allows individuals and fam…	NULL

STRING FUNCTIONS: PATTERN MATCHING

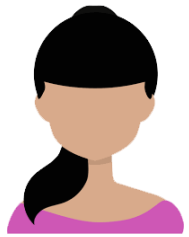
String functions can be applied to string columns *(char, varchar, text, etc.)*

```
-- Return words with hyphens in them
SELECT  event_desc,
        REGEXP_SUBSTR(event_desc, '[A-Z][a-z]+(-[A-Za-z]+)+') AS hyphen_phrase
FROM    events;
```

event_desc	hyphen_phrase
A much-anticipated break from daily routines, this vacation period allows individuals and fam…	much-anticipated
An exciting and sometimes nerve-wracking day for students, marking the beginning of a new…	nerve-wracking
A festive occasion celebrated with costumes, trick-or-treating, and various spooky activities.…	trick-or-treating
A holiday rooted in gratitude and family, Thanksgiving is celebrated with a large feast that ty…	NULL
A major holiday celebrated around the world, Christmas commemorates the birth of Jesus C…	gift-giving

Regular expressions allow you to find patterns in your text – they are language agnostic, which means they work in multiple languages like SQL and Python *(they take extra processing power, so they aren't recommended for large datasets)*

ASSIGNMENT: PATTERN MATCHING



NEW MESSAGE

November 22, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Update product names**

Hi again,

The marketing team has kicked off an initiative to simplify our product names, starting with our Wonka Bars products.

Could you remove "Wonka Bar" from any products that contain the term?

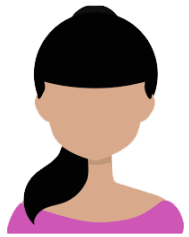
Thank you!
Mandy

Reply Forward

Results Preview

product_name	new_product_name
Nerds	Nerds
Pixy Stix	Pixy Stix
SweeTARTS	SweeTARTS
Tropical Nerds	Tropical Nerds
Wonka Bar - Fudge Mallows	Fudge Mallows
Wonka Bar - Milk Chocolate	Milk Chocolate
Wonka Bar - Nutty Crunch Surprise	Nutty Crunch Surprise
Wonka Bar - Scrumdiddlyumptious	Scrumdiddlyumptious
Wonka Bar - Triple Dazzle Caramel	Triple Dazzle Caramel
Wonka Gum	Wonka Gum

SOLUTION: PATTERN MATCHING



NEW MESSAGE

November 22, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Update product names**

Hi again,

The marketing team has kicked off an initiative to simplify our product names, starting with our Wonka Bars products.

Could you remove "Wonka Bar" from any products that contain the term?

Thank you!
Mandy

Reply Forward

Solution Code

```sql
-- Only extract text after the hyphen for Wonka Bars

-- Replace approach
SELECT   product_name,
         REPLACE(product_name, 'Wonka Bar - ', '')
            AS new_product_name
FROM     products
ORDER BY product_name;

-- Substring approach
SELECT   product_name,
         CASE WHEN INSTR(product_name, '-') = 0
              THEN product_name
              ELSE SUBSTR(product_name,
                          INSTR(product_name, '-') + 2)
              END AS new_product_name
FROM     products
ORDER BY product_name;

/*  INSTR and SUBSTR work in MySQL, Oracle and SQLite
    In PostgreSQL, use POSITION and SUBSTRING
    In SQL Server, use CHARINDEX AND SUBSTRING
*/
```

NULL FUNCTIONS

NULL functions are used to replace NULL values with an alternative value

To do a simple IF-ELSE NULL check:

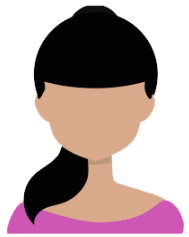
- Use the **IFNULL** function (*NVL in Oracle, not supported in PostgreSQL*)

To do more complex IF-ELSE NULL checks:

- Use the **COALESCE** function (*supported in most modern RDBMS's*)

PRO TIP: COALESCE is a more flexible version of the two, and will allow you to do multiple NULL checks, and returns the first non-NULL value

ASSIGNMENT: NULL FUNCTIONS



NEW MESSAGE

November 25, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Fill in NULL values**

Hello,

Sugar Shack and The Other Factory just added two new products that don't have divisions assigned to them.

For simplicity's sake, could you update those NULL values to have a value of "Other"?

Here's an extra challenge for you – instead of updating them to "Other", could you update them to be the same division as the most common division within their respective factories?

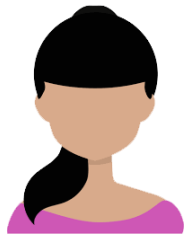
Thanks!

Reply Forward

Results Preview

product_name	factory	division	division_other	division_top
Wonka Bar - Fudge Mallows	Lot's O' Nuts	Chocolate	Chocolate	Chocolate
Wonka Bar - Nutty Crunch Surprise	Lot's O' Nuts	Chocolate	Chocolate	Chocolate
Wonka Bar - Scrumdiddlyumptious	Lot's O' Nuts	Chocolate	Chocolate	Chocolate
Everlasting Gobstopper	Secret Factory	Sugar	Sugar	Sugar
Wonka Gum	Secret Factory	Other	Other	Other
Lickable Wallpaper	Secret Factory	Other	Other	Other
Fun Dip	Sugar Shack	Sugar	Sugar	Sugar
Laffy Taffy	Sugar Shack	Sugar	Sugar	Sugar
Loopy Lollipops	Sugar Shack	Sugar	Sugar	Sugar
Nerds	Sugar Shack	Sugar	Sugar	Sugar
Pixy Stix	Sugar Shack	Sugar	Sugar	Sugar
SweeTARTS	Sugar Shack	Sugar	Sugar	Sugar
Tropical Nerds	Sugar Shack	Other	Other	Other
Fizzy Lifting Drinks	Sugar Shack	NULL	Other	Sugar
Hair Toffee	The Other F…	Sugar	Sugar	Sugar
Kazookles	The Other F…	NULL	Other	Sugar
Wonka Bar - Milk Chocolate	Wicked Choc…	Chocolate	Chocolate	Chocolate
Wonka Bar - Triple Dazzle Caramel	Wicked Choc…	Chocolate	Chocolate	Chocolate

SOLUTION: NULL FUNCTIONS



NEW MESSAGE

November 25, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Fill in NULL values**

Hello,

Sugar Shack and The Other Factory just added two new products that don't have divisions assigned to them.

For simplicity's sake, could you update those NULL values to have a value of "Other"?

Here's an extra challenge for you – instead of updating them to "Other", could you update them to be the same division as the most common division within their respective factories?

Thanks!

Reply Forward

Solution Code

```
-- Replace division with Other value and top division
WITH np AS (SELECT  factory, division, COUNT(product_name) AS num_products
            FROM    products
            WHERE   division IS NOT NULL
            GROUP BY factory, division
            ORDER BY factory, division),

    np_rank AS (SELECT factory, division, num_products,
                       ROW_NUMBER() OVER(PARTITION BY factory
                                         ORDER BY num_products DESC) AS np_rank
                FROM   np),

    top_div AS (SELECT factory, division
                FROM   np_rank
                WHERE  np_rank = 1)

SELECT   p.product_name, p.factory, p.division,
         COALESCE(p.division, 'Other') AS division_other,
         COALESCE(p.division, td.division) AS division_top
FROM     products p LEFT JOIN top_div td
         ON p.factory = td.factory
ORDER BY p.factory, p.division;
```

ORDER BY in CTE is not needed and can be omitted

KEY TAKEAWAYS

A **function** applies a calculation or transformation to rows of data

- *An aggregate function applies a calculation to all rows and returns a single value (COUNT, SUM, etc.)*
- *A window function performs a calculation across a window of rows (OVER, PARTITION BY, etc.)*
- *A general function performs a calculation or transformation on all rows*

Specific functions can be applied to **specific data types**

- *If needed, you can CAST or CONVERT a field into a different data type to apply a particular function*
- *Common numeric functions include LOG, ROUND, etc.*
- *Common datetime functions include YEAR, DATEDIFF, etc.*
- *Common string functions include TRIM, REPLACE, REGEXP, etc.*
- *Common NULL functions include IFNULL, COALESCE, etc.*

DATA ANALYSIS APPLICATIONS

DATA ANALYSIS APPLICATIONS

Now that we've introduced a variety of advanced SQL concepts, we'll use this section to go over common **data analysis applications** that utilize the techniques we've learned

TOPICS WE'LL COVER:

- Duplicate Values
- Min/Max Value Filtering
- Pivoting
- Rolling Calculations
- Imputing NULL Values

GOALS FOR THIS SECTION:

- Learn to identify and handle duplicate values
- Apply min / max value filtering in various ways
- Pivot data using conditional aggregations
- Perform rolling calculations, including subtotals, cumulative sums, and moving averages
- Filling in NULL values with imputed values

DUPLICATE VALUES

- Duplicate Values
- Min / Max Value Filtering
- Pivoting
- Rolling Calculations
- Imputing NULL Values

Duplicate values can be present in various ways:

```
SELECT  *
FROM    employee_details;
```

region	employee_name	salary
East	Ava	85000
East	Ava	85000
East	Bob	72000
East	Cat	59000
West	Cat	63000
West	Dan	85000
West	Eve	72000
West	Eve	75000

These are fully duplicate rows

These could potentially be:
1. Two employees with the same name, in different regions
2. The same employee who transferred regions

This employee seems to have gotten a raise

DUPLICATE VALUES

- Duplicate Values
- Min / Max Value Filtering
- Pivoting
- Rolling Calculations
- Imputing NULL Values

To **view duplicate values**:

- Use a combination of GROUP BY, COUNT, and HAVING

To **exclude duplicate values**:

- Use DISTINCT to exclude fully duplicate rows
- Use window functions to exclude partially duplicate rows

ASSIGNMENT: DUPLICATE VALUES

NEW MESSAGE

December 2, 2024

From: **Stu Dious** *(Admin, Maven High School)*

Subject: **Remove duplicate students**

Good morning!

We've learned that there's a student who's showing up multiple times in our student records.

Can you generate a report of the students and their emails, and exclude the duplicate student record?

Thank you!
Stu

Reply Forward

Results Preview

id	student_name	email
1	Abby Johnson	abby.johnson@mavenhighschool.com
2	Bob Smith	bob.smith@mavenhighschool.com
3	Catherine Davis	catherine.davis@mavenhighschool.com
4	Daniel Brown	daniel.brown@mavenhighschool.edu
5	Eva Martinez	eva.martinez@mavenhighschool.com
6	Frank Wilson	frank.wilson@mavenhighschool.com
7	Grace Lee	grace.lee@mavenhighschool.com
8	Henry Taylor	NULL
9	Isabella Moore	isabella.moore@mavenhighschool.com
10	Jack Thompson	jack.thompson@mavenhighschool.com
11	Karen White	karen.white@mavenhighschool.edu
12	Liam Green	liam.green@mavenhighschool.com
13	Mia Harris	mia.harris@mavenhighschool.com
15	Olivia Adams	olivia.adams@mavenhighschool.edu
16	Peter Park	peter.park@mavenhighschool.com
17	Noah Scott	noah.scott@mavenhighschool.com

SOLUTION: DUPLICATE VALUES

NEW MESSAGE

December 2, 2024

From: **Stu Dious** *(Admin, Maven High School)*

Subject: **Remove duplicate students**

Good morning!

We've learned that there's a student who's showing up multiple times in our student records.

Can you generate a report of the students and their emails, and exclude the duplicate student record?

Thank you!
Stu

Reply Forward

Solution Code

```sql
-- Return student ids, names and emails,
-- excluding duplicates students
WITH sc AS (SELECT  id, student_name, email,
                    ROW_NUMBER() OVER
                    (PARTITION BY student_name
                    ORDER BY id DESC) AS student_count
            FROM    students)

SELECT   id, student_name, email
FROM     sc
WHERE    student_count = 1
ORDER BY id;
```

MIN / MAX VALUE FILTERING

- Duplicate Values
- Min / Max Value Filtering
- Pivoting
- Rolling Calculations
- Imputing NULL Values

Min / max value filtering allows you to filter data based on the lowest or highest values within each group

EXAMPLE *Return the most recent sales amount for each sales rep*

```
SELECT * FROM sales;
```

id	sales_rep	date	sales
1	Emma	2024-08-01	6
2	Emma	2024-08-02	17
3	Jack	2024-08-02	14
4	Emma	2024-08-04	20
5	Jack	2024-08-05	5
6	Emma	2024-08-07	1

```
SELECT    sales_rep,
          MAX(date) AS most_recent_date
FROM      sales
GROUP BY sales_rep;
```

sales_rep	most_recent_date
Emma	2024-08-07
Jack	2024-08-05

← *Only the date is returned, but we want the sales amount as well*

There are two approaches you can take to include the sales amount:

- Use a GROUP BY with a JOIN
- Use a window function

ASSIGNMENT: MIN / MAX VALUE FILTERING

NEW MESSAGE

December 3, 2024

From: **Stu Dious** *(Admin, Maven High School)*

Subject: **Top grade for each student**

Hi again,

Can you create a report of each student with their highest grade for the semester, as well as which class it was in?

Thanks!
Stu

Reply Forward

Results Preview

id	student_name	top_grade	class_name
1	Abby Johnson	94	Chemistry
2	Bob Smith	88	Chemistry
2	Bob Smith	88	Statistics
3	Catherine Davis	98	Calculus
4	Daniel Brown	95	Physical Education
5	Eva Martinez	87	English
6	Frank Wilson	98	Chemistry
7	Grace Lee	94	World History
8	Henry Taylor	90	Physical Education
9	Isabella Moore	90	Physical Education
10	Jack Thompson	85	Physical Education
11	Karen White	98	English
12	Liam Green	99	Statistics
13	Mia Harris	90	Physical Education
15	Olivia Adams	99	English
16	Peter Park	86	World History
17	Noah Scott	89	English

SOLUTION: MIN / MAX VALUE FILTERING

NEW MESSAGE

December 3, 2024

From: **Stu Dious** *(Admin, Maven High School)*

Subject: **Top grade for each student**

Hi again,

Can you create a report of each student with their highest grade for the semester, as well as which class it was in?

Thanks!
Stu

Reply Forward

Solution Code

```sql
-- Return each student's top grade and corresponding class

-- GROUP BY + JOIN approach
WITH tg AS (SELECT   s.id, s.student_name,
                     MAX(sg.final_grade) AS top_grade
            FROM     students s INNER JOIN student_grades sg
                     ON s.id = sg.student_id
            GROUP BY s.id, s.student_name
            ORDER BY s.id)

SELECT  tg.id, tg.student_name, tg.top_grade, sg.class_name
FROM    tg LEFT JOIN student_grades sg
        ON tg.id = sg.student_id AND tg.top_grade = sg.final_grade;

-- Window function approach
SELECT id, student_name, class_name, final_grade FROM

(SELECT s.id, s.student_name, sg.class_name, sg.final_grade,
        DENSE_RANK() OVER (PARTITION BY s.student_name
            ORDER BY sg.final_grade DESC) AS grade_rank
FROM    students s LEFT JOIN student_grades sg
        ON s.id = sg.student_id) AS gr

WHERE grade_rank = 1;
```

ORDER BY in CTE is not needed and can be omitted

PIVOTING

- Duplicate Values
- Min / Max Value Filtering
- Pivoting
- Rolling Calculations
- Imputing NULL Values

Pivoting lets you transform rows into columns to summarize your data

- This can be achieved using CASE statements
- PIVOT is available in some RDBMS's like SQL Server and Oracle

EXAMPLE

Create a summary table by pivoting the crust type column in the pizza table

category	crust_type	pizza_name	price
Chicken	Gluten-Free Crust	California Chicken	21.75
Chicken	Thin Crust	Chicken Pesto	20.75
Classic	Standard Crust	Greek	21.50
Classic	Standard Crust	Hawaiian	19.50
Classic	Standard Crust	Pepperoni	18.75
Supreme	Standard Crust	Spicy Italian	22.75
Veggie	Thin Crust	Five Cheese	18.50
Veggie	Thin Crust	Margherita	19.50
Veggie	Gluten-Free Crust	Garden Delight	21.50

category	standard_crust	thin_crust	gluten_free_crust
Chicken	0	1	1
Classic	3	0	0
Supreme	1	0	0
Veggie	0	2	1

ASSIGNMENT: PIVOTING

NEW MESSAGE

December 4, 2024

From: **Stu Dious** *(Admin, Maven High School)*

Subject: **Summary table**

Hello,

Can you help us create a summary table that shows the average grade for each department and grade level?

Thanks for all your help this week!
Stu

Reply Forward

Results Preview

department	freshman	sophomore	junior	senior
General	94	89	88	NULL
Humanities	88	88	84	90
Math	87	86	86	85
Science	84	85	87	84

SOLUTION: PIVOTING

NEW MESSAGE

December 4, 2024

From: **Stu Dious** *(Admin, Maven High School)*

Subject: **Summary table**

Hello,

Can you help us create a summary table that shows the average grade for each department and grade level?

Thanks for all your help this week!
Stu

Reply Forward

Solution Code

```sql
-- Create a summary table of depts & grade levels
SELECT   sg.department,
         ROUND(AVG(CASE WHEN s.grade_level = 9
             THEN sg.final_grade END)) AS freshman,
         ROUND(AVG(CASE WHEN s.grade_level = 10
             THEN sg.final_grade END)) AS sophomore,
         ROUND(AVG(CASE WHEN s.grade_level = 11
             THEN sg.final_grade END)) AS junior,
         ROUND(AVG(CASE WHEN s.grade_level = 12
             THEN sg.final_grade END)) AS senior
FROM     students s LEFT JOIN student_grades sg
         ON s.id = sg.student_id
WHERE     sg.department IS NOT NULL
GROUP BY sg.department
ORDER BY sg.department;
```

ROLLING CALCULATIONS

- Duplicate Values
- Min / Max Value Filtering
- Pivoting
- **Rolling Calculations**
- Imputing NULL Values

Rolling calculations including subtotals, cumulative sums, and moving averages allow you to perform calculations across rows of data

order_id	customer_name	order_date	pizza_name	price
1	Jack	2024-12-01	Pepperoni	18.75
2	Jack	2024-12-02	Pepperoni	18.75
3	Jack	2024-12-03	Pepperoni	18.75
4	Jack	2024-12-04	Pepperoni	18.75
5	Jack	2024-12-05	Spicy Italian	22.75
6	Jill	2024-12-01	Five Cheese	18.50
7	Jill	2024-12-03	Margherita	19.50
8	Jill	2024-12-05	Garden Delight	21.50
9	Jill	2024-12-05	Greek	21.50
10	Tom	2024-12-02	Hawaiian	19.50
11	Tom	2024-12-04	Chicken Pesto	20.75
12	Tom	2024-12-05	Spicy Italian	22.75
13	Jerry	2024-12-01	California Chi…	21.75
14	Jerry	2024-12-02	Margherita	19.50
15	Jerry	2024-12-04	Greek	21.50

Subtotals

customer_name	order_date	total_sales
Jack	2024-12-01	18.75
Jack	2024-12-02	18.75
Jack	2024-12-03	18.75
Jack	2024-12-04	18.75
Jack	2024-12-05	22.75
Jack	NULL	97.75
Jerry	2024-12-01	21.75
Jerry	2024-12-02	19.50
Jerry	2024-12-04	21.50
Jerry	NULL	62.75
Jill	2024-12-01	18.50
Jill	2024-12-03	19.50
Jill	2024-12-05	43.00
Jill	NULL	81.00
Tom	2024-12-02	19.50
Tom	2024-12-04	20.75
Tom	2024-12-05	22.75
Tom	NULL	63.00
NULL	NULL	304.50

*Use the **WITH ROLLUP** keywords to calculate subtotals (not supported in SQLite)*

ROLLING CALCULATIONS

- Duplicate Values
- Min / Max Value Filtering
- Pivoting
- **Rolling Calculations**
- Imputing NULL Values

Rolling calculations including subtotals, cumulative sums, and moving averages allow you to perform calculations across rows of data

order_id	customer_name	order_date	pizza_name	price
1	Jack	2024-12-01	Pepperoni	18.75
2	Jack	2024-12-02	Pepperoni	18.75
3	Jack	2024-12-03	Pepperoni	18.75
4	Jack	2024-12-04	Pepperoni	18.75
5	Jack	2024-12-05	Spicy Italian	22.75
6	Jill	2024-12-01	Five Cheese	18.50
7	Jill	2024-12-03	Margherita	19.50
8	Jill	2024-12-05	Garden Delight	21.50
9	Jill	2024-12-05	Greek	21.50
10	Tom	2024-12-02	Hawaiian	19.50
11	Tom	2024-12-04	Chicken Pesto	20.75
12	Tom	2024-12-05	Spicy Italian	22.75
13	Jerry	2024-12-01	California Chi…	21.75
14	Jerry	2024-12-02	Margherita	19.50
15	Jerry	2024-12-04	Greek	21.50

Cumulative sum

order_date	cumulative_sum
2024-12-01	59.00
2024-12-02	116.75
2024-12-03	155.00
2024-12-04	216.00
2024-12-05	304.50

*Use **SUM()** as a window function to calculate the cumulative sum*

ROLLING CALCULATIONS

- Duplicate Values
- Min / Max Value Filtering
- Pivoting
- **Rolling Calculations**
- Imputing NULL Values

Rolling calculations including subtotals, cumulative sums, and moving averages allow you to perform calculations across rows of data

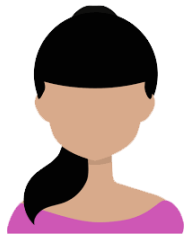
country	year	happiness_score
Afghanistan	2015	3.575
Afghanistan	2016	3.360
Afghanistan	2017	3.794
Afghanistan	2018	3.632
Afghanistan	2019	3.203
Afghanistan	2020	2.567
Afghanistan	2021	2.523
Afghanistan	2022	2.404
Afghanistan	2023	1.859
Albania	2015	4.959
Albania	2016	4.655
Albania	2017	4.644
Albania	2018	4.586
Albania	2019	4.719
Albania	2020	4.883

Moving average

country	year	happiness_score	moving_average
Afghanistan	2015	3.575	3.575
Afghanistan	2016	3.360	3.468
Afghanistan	2017	3.794	3.576
Afghanistan	2018	3.632	3.595
Afghanistan	2019	3.203	3.543
Afghanistan	2020	2.567	3.134
Afghanistan	2021	2.523	2.764
Afghanistan	2022	2.404	2.498
Afghanistan	2023	1.859	2.262
Albania	2015	4.959	4.959
Albania	2016	4.655	4.807
Albania	2017	4.644	4.753
Albania	2018	4.586	4.628
Albania	2019	4.719	4.650
Albania	2020	4.883	4.729

*Use **AVG()** as a window function to calculate the moving average*

ASSIGNMENT: ROLLING CALCULATIONS



NEW MESSAGE

December 5, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Monthly sales report**

Hi again,

We have one final request for you.

Can you help us generate a report that shows the total sales for each month, as well as the cumulative sum of sales and the six-month moving average of sales?

Thanks for all your help over the past month!
Mandy

Reply Forward

Results Preview

yr	mnth	total_sales	cumluative_sum	six_month_ma
2021	1	1065.24	1065.24	1065.240000
2021	2	547.09	1612.33	806.165000
2021	3	2245.63	3857.96	1285.986667
2021	4	2058.97	5916.93	1479.232500
2021	5	1855.21	7772.14	1554.428000
2021	6	1838.43	9610.57	1601.761667
2021	7	1908.23	11518.80	1742.260000
2021	8	2258.46	13777.26	2027.488333
2021	9	3927.08	17704.34	2307.730000
2021	10	2544.88	20249.22	2388.715000
2021	11	4357.18	24606.40	2805.710000
2021	12	4347.15	28953.55	3223.830000
2022	1	1028.66	29982.21	3077.235000
2022	2	832.78	30814.99	2839.621667
2022	3	1893.27	32708.26	2500.653333

SOLUTION: ROLLING CALCULATIONS

NEW MESSAGE

December 5, 2024

From: **Mandy Smore** *(Analyst, Maven Candies)*

Subject: **Monthly sales report**

Hi again,

We have one final request for you.

Can you help us generate a report that shows the total sales for each month, as well as the cumulative sum of sales and the six-month moving average of sales?

Thanks for all your help over the past month!
Mandy

Reply Forward

Solution Code

```sql
-- Cumulative sum and 6 month moving average
WITH ms AS (SELECT   YEAR(o.order_date) AS yr,
                     MONTH(o.order_date) AS mnth,
                     SUM(o.units * p.unit_price)
                         AS total_sales
            FROM     orders o LEFT JOIN products p
                     ON o.product_id = p.product_id
            GROUP BY YEAR(o.order_date), MONTH(o.order_date)
            ORDER BY YEAR(o.order_date), MONTH(o.order_date))

SELECT  yr, mnth, total_sales,
        SUM(total_sales) OVER (ORDER BY yr, mnth)
            AS cumluative_sum,
        AVG(total_sales) OVER (ORDER BY yr, mnth
            ROWS BETWEEN 5 PRECEDING AND CURRENT ROW)
            AS six_month_ma
FROM    ms;
```

ORDER BY in CTE is not needed and can be omitted

DEMO: IMPUTING NULL VALUES

- Duplicate Values
- Min / Max Value Filtering
- Pivoting
- Rolling Calculations
- Imputing NULL Values

Imputing values means replacing NULL values in the data with other values

We'll cover four different approaches on how to do this in SQL:

1. Hard coded value (*integer*)
2. Average of a column (*subquery*)
3. Prior row's value (*window function*)
4. Smoothed value (*two window functions*)

In this final demo, we'll be writing a single query that contains techniques learned in every section of this course! It includes a JOIN, UNION, subquery, recursive CTE, window function, numeric function and NULL function.

KEY TAKEAWAYS

- There are many ways to identify and handle **duplicate values**
  - *Use HAVING to view duplicate rows, and DISTINCT or window functions to exclude duplicate rows*
- **Min / max value filtering** allows you to filter data within each group
  - *This can be accomplished with a combination of GROUP BY and JOIN, or with a window function*
- **Pivoting** transforms row values into columns to summarize your data
  - *This can be accomplished by using CASE statements with aggregate functions, or PIVOT in some tools*
- **Rolling calculations** include subtotals, cumulative sums & moving averages
  - *This can be done using the WITH ROLLUP keywords or window functions with SUM() and AVG()*
- There are many options for **imputing NULL values**, or filling in missing data
  - *Options include using hard coded values, column aggregations, relative row values and more*

FINAL PROJECT

FINAL PROJECT

THE SITUATION

You've just been hired as a Data Analyst Intern for **Major League Baseball** (MLB), who has recently gotten access to a large amount of historical player data

THE ASSIGNMENT

You have access to decades worth of data including player statistics like schools attended, salaries, teams played for, height and weight, and more

Your task is to use **advanced SQL querying techniques** to track how player statistics have changed over time and across different teams in the league

THE OBJECTIVES

1. What **schools** do MLB players attend?
2. How much do teams spend on player **salaries**?
3. What does each player's **career** look like?
4. How do player **attributes** compare?

Data source: *http://seanlahman.com*