**SQL – NOTES from Studies:**

**--REVIEW QUERIES written throughout week, and copy takeaways/example snippets here  
  
Select Schema Info--To get information on columns of a table, you query the information\_schema.columns catalog.**

SELECT table\_name, column\_name, data\_type

FROM information\_schema.columns

WHERE table\_name = 'orders';

**MYSQL example using employees table:**

SELECT table\_name, column\_name, data\_type

FROM information\_schema.columns

WHERE table\_name='employees';

**Commands**

You have already learned a lot about writing code in SQL! Let's take a moment to recap all that we have covered before moving on:

| **Statement** | **How to Use It** | **Other Details** |
| --- | --- | --- |
| SELECT | SELECT **Col1**, **Col2**, ... | Provide the columns you want |
| FROM | FROM **Table** | Provide the table where the columns exist |
| LIMIT | LIMIT **10** | Limits based number of rows returned |
| ORDER BY | ORDER BY **Col** | Orders table based on the column. Used with **DESC**. |
| WHERE | WHERE **Col > 5** | A conditional statement to filter your results |
| LIKE | WHERE **Col LIKE '%me%'** | Only pulls rows where column has 'me' within the text |
| IN | WHERE **Col IN ('Y', 'N')** | A filter for only rows with column of 'Y' or 'N' |
| NOT | WHERE **Col NOT IN ('Y', 'N')** | **NOT** is frequently used with **LIKE** and **IN** |
| AND | WHERE **Col1 > 5 AND Col2 < 3** | Filter rows where two or more conditions must be true |
| OR | WHERE **Col1 > 5 OR Col2 < 3** | Filter rows where at least one condition must be true |
| BETWEEN | WHERE **Col BETWEEN 3 AND 5** | Often easier syntax than using an **AND** |

**NOTE: LIKE is case sensitive, ILIKE is case insensitive.  
  
DISTINCT – remove duplicates, ie.**

SELECT   
DISTINCT customer\_id,   
SUM(DISTINCT customer\_id) as unique\_customers,   
COUNT(DISTINCT month) AS unique\_months, **etc  
  
IS NULL - ie.** WHERE employee\_id IS NULL  **(opposite would be** WHERE employee\_id IS NOT NULL**)**

**LIMIT & OFFSET** – limit used to limit results (ie, limit 10 for top 10 records in result set); The OFFSET clause skips the offset rows before beginning to return the rows.

The **OFFSET** clause is optional. If you omit it, the query will return the rows from the first row returned by the SELECT clause.Example:   
  
Say you want to get 5 artists, but not the first five. You want to get rows 3 through 8. You’ll want to add an OFFSET of 2 to skip the first two rows:

SELECT \* FROM artists LIMIT 5 OFFSET 2;

**WINDOW functions – ie, RANK() OVER PARTITION BY (restart every)**

Example:  
SELECT company, profits

FROM (SELECT

company,

profits,

RANK() OVER(ORDER BY profits DESC) AS ranking

FROM forbes\_global\_2010\_2014) AS a

WHERE ranking<4

ORDER BY profits DESC

**Create a running total of standard\_amt\_usd (in the orders table) over order time with no date truncation.** Your final table should have two columns: one with the amount being added for each new row, and a second with the running total. --https://learn.udacity.com/courses/ud198/lessons/30325c41-887d-4247-a128-e267fe28a1fa/concepts/bdca5060-b080-427f-ae73-21aae9da4155

SELECT

standard\_amt\_usd AS amount\_added,

SUM(standard\_amt\_usd) OVER (ORDER BY occurred\_at) AS running\_total

FROM

orders;

**3 Day Moving Average of a Transaction**

SELECT  ct.date,

        cty.card\_type\_name,

        SUM(ct.amount) AS daily\_sum,

        AVG(SUM(ct.amount)) OVER (ORDER BY ct.date ASC ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS transaction\_running\_average

FROM card\_transaction ct

JOIN card\_number cn ON ct.card\_number\_id = cn.id

JOIN card\_type cty ON cn.card\_type\_id = cty.id

WHERE ct.date > '2020-11-30' AND date <= '2020-12-31'

AND cty.card\_type\_name = 'visa-electron'

GROUP BY ct.date, cty.card\_type\_name

ORDER BY cty.card\_type\_name;

**ALIASING for MULTIPLE WINDOW FUNCTIONS**

SELECT id,

account\_id,

DATE\_TRUNC('year',occurred\_at) AS year,

DENSE\_RANK() OVER account\_year\_window AS dense\_rank,

total\_amt\_usd,

SUM(total\_amt\_usd) OVER account\_year\_window AS sum\_total\_amt\_usd,

COUNT(total\_amt\_usd) OVER account\_year\_window AS count\_total\_amt\_usd,

AVG(total\_amt\_usd) OVER account\_year\_window AS avg\_total\_amt\_usd,

MIN(total\_amt\_usd) OVER account\_year\_window AS min\_total\_amt\_usd,

MAX(total\_amt\_usd) OVER account\_year\_window AS max\_total\_amt\_usd

FROM orders

WINDOW account\_year\_window AS (PARTITION BY account\_id ORDER BY DATE\_TRUNC('year',occurred\_at))

**ROW\_NUMBER example with CTE**

wITH StudentRanks AS

(

  SELECT \*, ROW\_NUMBER() OVER( ORDER BY Marks) AS Ranks

  FROM ExamResult

)

SELECT StudentName , Marks

FROM StudentRanks

WHERE Ranks >= 1 and Ranks <=3

ORDER BY Ranks

**LEAD** – Identify NEXT row value

**LAG** – Identify PREVIOUS row value  
Postgres Example: <https://learn.udacity.com/courses/ud198/lessons/30325c41-887d-4247-a128-e267fe28a1fa/concepts/b66551e4-cc1c-4d94-9a1f-03c6d31964d2>

Imagine you're an analyst at Parch & Posey and you want to determine how the current order's total revenue ("total" meaning from sales of all types of paper) compares to the next order's total revenue.

SELECT

occurred\_at,

total\_amt\_usd,

LEAD(total\_amt\_usd) OVER (ORDER BY occurred\_at) AS lead,

LEAD(total\_amt\_usd) OVER (ORDER BY occurred\_at) - total\_amt\_usd AS lead\_difference

FROM orders

| ORDER BY occurred\_at;  OUTPUT **occurred\_at** | **total\_amt\_usd** | **lead** | **lead\_difference** |
| --- | --- | --- | --- |
| 2013-12-04T04:22:44.000Z | 627.48 | 2646.77 | 2019.29 |
| 2013-12-04T04:45:54.000Z | 2646.77 | 2709.62 | 62.85 |
| 2013-12-04T04:53:25.000Z | 2709.62 | 277.13 | -2432.49 |
| 2013-12-05T20:29:16.000Z | 277.13 | 3001.85 | 2724.72 |

SELECT account\_id,

standard\_sum,

LAG(standard\_sum) OVER (ORDER BY standard\_sum) AS lag,

LEAD(standard\_sum) OVER (ORDER BY standard\_sum) AS lead,

standard\_sum - LAG(standard\_sum) OVER (ORDER BY standard\_sum) AS lag\_difference,

LEAD(standard\_sum) OVER (ORDER BY standard\_sum) - standard\_sum AS lead\_difference

FROM (

SELECT account\_id,

SUM(standard\_qty) AS standard\_sum

FROM orders

GROUP BY 1

| ) AS sub OUTPUT: **account\_id** | **standard\_sum** | **lag** | **lead** | **lag\_difference** | **lead\_difference** |
| --- | --- | --- | --- | --- | --- |
| 1901 | 0 |  | 79 |  | 79 |
| 3371 | 79 | 0 | 102 | 79 | 23 |
| 1961 | 102 | 79 | 116 | 23 | 14 |
| 3401 | 116 | 102 | 117 | 14 | 1 |

**CREATE VIEW AS** – definition: Constructs a virtual table that has no physical data based on the result-set of a SQL query. ALTER VIEW and DROP VIEW only change metadata.  
Example:  
-- Create or replace view for `experienced\_employee` with comments.

> CREATE OR REPLACE VIEW experienced\_employee

(id COMMENT 'Unique identification number', Name)

COMMENT 'View for experienced employees'

AS SELECT id, name

FROM all\_employee

WHERE working\_years > 5;

-- Create a temporary view `subscribed\_movies`.

> CREATE TEMPORARY VIEW subscribed\_movies

AS SELECT mo.member\_id, mb.full\_name, mo.movie\_title

FROM movies AS mo

INNER JOIN members AS mb ON mo.member\_id = mb.id;

**Create Materialized View (Databricks) - Unlike traditional implementations, Databricks materialized views retain the data state at the time of the last refresh, rather than updating every time they are queried.**

CREATE MATERIALIZED VIEW customer\_orders

AS

SELECT

  customers.name,

  sum(orders.amount),

  orders.orderdate

FROM orders

  LEFT JOIN customers ON

    orders.custkey = customers.c\_custkey

GROUP BY

  name,

  orderdate;

**Create Temporary Table   
Example 1:**

-- Create a temp table of Canadians

CREATE TEMPORARY TABLE canadians AS

    SELECT \*

    FROM athletes\_recent

    WHERE country\_code = 'CAN'

    AND season = 'Winter'; -- The table has both summer and winter athletes

-- Find the most popular sport

SELECT sport

  , COUNT(DISTINCT athlete\_id) as no\_athletes

FROM canadians

GROUP BY sport

ORDER BY no\_athletes DESC;

**Example 2:**

-- Create temp countries table

CREATE TEMPORARY TABLE countries AS

    SELECT DISTINCT o.region, a.country\_code, o.country

    FROM athletes a

    INNER JOIN oregions o

      ON a.country\_code = o.olympic\_cc;

ANALYZE countries; -- Collect the statistics

-- Count the entries

SELECT COUNT(\*) FROM countries;

**STORED PROCEDURES -** A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

**Example:**

CREATE PROCEDURE SelectAllCustomers  
AS  
SELECT \* FROM Customers  
GO;

EXEC SelectAllCustomers;

**Stored Procedure with Input Parameters**

CREATE PROCEDURE uspUpdateEmpSalary

(

@empId int

,@salary money

)

AS

BEGIN

UPDATE dbo.Employee

SET Salary = @salary

WHERE EmployeeID = @empId

END

**Example: Passing INPUT Parameters**

EXEC dbo.uspUpdateEmpSalary @EmpId = 4, @Salary = 25000

-- or

EXEC dbo.uspUpdateEmpSalary 4, 25000

**Subqueries – Queries within other queries -   
Scalar Example based on paintings table with id, name, artist\_id, and listed\_price fields  
We want to list paintings that are priced higher than the average.**

SELECT name, listed\_price

FROM paintings

WHERE listed\_price > (

    SELECT AVG(listed\_price)

    FROM paintings

);

**Multi-Row Subquery with Multiple Columns - we want to see the total amount of sales for each artist who has sold at least one painting in our gallery. We may start with a subquery that draws on the sales table and calculates the total amount of sales for each artist ID. Then, in the outer query, we combine this information with the artists’ first names and last names to get the required output:**

SELECT

  artists.first\_name,

  artists.last\_name,

  artist\_sales.sales

FROM artists

JOIN (

    SELECT artist\_id, SUM(sales\_price) AS sales

    FROM sales

    GROUP BY artist\_id

  ) AS artist\_sales

  ON artists.id = artist\_sales.artist\_id;

**Correlated Sub-Query – Show names who had zero sales:**

SELECT first\_name, last\_name

FROM artists

WHERE NOT EXISTS (

  SELECT \*

  FROM sales

  WHERE sales.artist\_id = artists.id

);

**Sub-Query / Join Returning Same result example**

**Sub-Query**

SELECT first\_name, last\_name

FROM collectors

WHERE id IN (

    SELECT collector\_id

    FROM sales

);

**Join**

SELECT DISTINCT collectors.first\_name, collectors.last\_name

FROM collectors

JOIN sales

  ON collectors.id = sales.collector\_id;

|  |
| --- |
| **EXAMPLE WHERE JOIN IS PREFERRED for PERFORMANCE  Using Subquery (slower)**  SELECT name, cost  FROM product  WHERE id=(SELECT product\_id    FROM sale      WHERE price=2000      AND product\_id=product.id    );  **Using JOIN** |
| SELECT p.name, p.cost  FROM product p  JOIN sale s ON p.id=s.product\_id  WHERE s.price=2000; |

**WHEN a SUBQUERY CANNOT BE REPLACED WITH A JOIN:**

**-Subquery in FROM with a GROUP BY**

SELECT city, sum\_price

 FROM

(

  SELECT city, SUM(price) AS sum\_price FROM sale

  GROUP BY city

) AS s

WHERE sum\_price < 2100;

**-Subquery returning an aggregate value in a WHERE clause**

SELECT name FROM product

WHERE cost<(SELECT AVG(price) from sale);

**-Subquery in an ALL clause**

SELECT name FROM product WHERE cost > ALL(SELECT price from sale);

**-Subquery Step by Step**

**First, we needed to group by the day and channel. Then ordering by the number of events (the third column) gave us a quick way to answer the first question.**

SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2

ORDER BY 3 DESC;

**Here you can see that to get the entire table in question 1 back, we included an \* in our\* SELECT\* statement. You will need to be sure to alias your table.**

SELECT \*

FROM (SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2

ORDER BY 3 DESC) sub;

**Finally, here we are able to get a table that shows the average number of events a day for each channel.**

SELECT channel, AVG(events) AS average\_events

FROM (SELECT DATE\_TRUNC('day',occurred\_at) AS day,

channel, COUNT(\*) as events

FROM web\_events

GROUP BY 1,2) sub

GROUP BY channel

ORDER BY 2 DESC;

**CTE (better for performance, readability vs. subqueries) – Common table expression; replace aggregates, subqueries using WITH name AS ( query ), then Select from** **name**  
**Example**:

WITH CTE AS(

SELECT

person\_id,

COUNT(\*) visits

FROM facebook\_event\_checkin

WHERE date BETWEEN 20171201 AND 20171231

AND event\_name = 'SQL Symphony Concert'

GROUP BY person\_id

HAVING COUNT(\*)>=3

)

SELECT p.\*, fb.\*

FROM drivers\_license dl

JOIN person p on dl.id = p.license\_id

JOIN CTE as fb on fb.person\_id = p.id

WHERE hair\_color='red'

AND height>=65

AND height <=67

AND car\_make='Tesla'

AND car\_model like '%Model S%'

AND gender = 'female' ;

**Example from earlier with WINDOW function**

wITH StudentRanks AS

(

  SELECT \*, ROW\_NUMBER() OVER( ORDER BY Marks) AS Ranks

  FROM ExamResult

)

SELECT StudentName , Marks

FROM StudentRanks

WHERE Ranks >= 1 and Ranks <=3

ORDER BY Ranks

**Revenues – Costs (per Eatery) – Profit per eatery returned**

WITH revenue AS (

-- Calculate revenue per eatery

SELECT eatery,

SUM(meal\_price \* order\_quantity) AS revenue

FROM meals

JOIN orders ON meals.meal\_id = orders.meal\_id

GROUP BY eatery

),

cost AS (

-- Calculate cost per eatery

SELECT eatery,

SUM(meal\_cost \* stocked\_quantity) AS cost

FROM meals

JOIN stock ON meals.meal\_id = stock.meal\_id

GROUP BY eatery

),

profit AS (

-- Calculate profit per eatery by combining revenue and cost

SELECT r.eatery,

revenue - cost AS profit

FROM revenue r

JOIN cost c ON r.eatery = c.eatery

)

SELECT eatery,

profit

FROM profit

ORDER BY eatery ASC;

**MORE WITH/CTE Solutions**

**Below, you will see each of the previous solutions restructured using the**WITH**clause. This is often an easier way to read a query.**

**Provide the**name**of the**sales\_rep**in each**region**with the largest amount of**total\_amt\_usd**sales.** WITH t1 AS (

SELECT s.name rep\_name, r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY 1,2

ORDER BY 3 DESC),

t2 AS (

SELECT region\_name, MAX(total\_amt) total\_amt

FROM t1

GROUP BY 1)

SELECT t1.rep\_name, t1.region\_name, t1.total\_amt

FROM t1

JOIN t2

ON t1.region\_name = t2.region\_name AND t1.total\_amt = t2.total\_amt;

**For the region with the largest sales**total\_amt\_usd**, how many**total**orders were** **placed?**

WITH t1 AS (

SELECT r.name region\_name, SUM(o.total\_amt\_usd) total\_amt

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY r.name),

t2 AS (

SELECT MAX(total\_amt)

FROM t1)

SELECT r.name, COUNT(o.total) total\_orders

FROM sales\_reps s

JOIN accounts a

ON a.sales\_rep\_id = s.id

JOIN orders o

ON o.account\_id = a.id

JOIN region r

ON r.id = s.region\_id

GROUP BY r.name

HAVING SUM(o.total\_amt\_usd) = (SELECT \* FROM t2);

**For the account that purchased the most (in total over their lifetime as a customer)**standard\_qty**paper,**how many accounts**still had more in**total**purchases?**

WITH t1 AS (

SELECT a.name account\_name, SUM(o.standard\_qty) total\_std, SUM(o.total) total

FROM accounts a

JOIN orders o

ON o.account\_id = a.id

GROUP BY 1

ORDER BY 2 DESC

LIMIT 1),

t2 AS (

SELECT a.name

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY 1

HAVING SUM(o.total) > (SELECT total FROM t1))

SELECT COUNT(\*)

FROM t2;

**For the customer that spent the most (in total over their lifetime as a customer) total\_amt\_usd, how many web\_events did they have for each channel?**

WITH t1 AS (

SELECT a.id, a.name, SUM(o.total\_amt\_usd) tot\_spent

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY 3 DESC

LIMIT 1)

SELECT a.name, w.channel, COUNT(\*)

FROM accounts a

JOIN web\_events w

ON a.id = w.account\_id AND a.id = (SELECT id FROM t1)

GROUP BY 1, 2

ORDER BY 3 DESC;

**What is the lifetime average amount spent in terms of**total\_amt\_usd**for the top 10 total spending**accounts**?**

WITH t1 AS (

SELECT a.id, a.name, SUM(o.total\_amt\_usd) tot\_spent

FROM orders o

JOIN accounts a

ON a.id = o.account\_id

GROUP BY a.id, a.name

ORDER BY 3 DESC

LIMIT 10)

SELECT AVG(tot\_spent)

FROM t1;

**CAST -- convert data types (like String to Date) – Syntax: CAST(*expression* AS *datatype(length)*)**Example: SELECT CAST('2017-08-25' AS datetime);

**WHERE** – used with typical select to filter records (ie, WHERE region=’West’) – cannot use Aggregates in WHERE clause

**Superstore – orders basic example**: SELECT \* FROM Orders WHERE Region=’West’;

**HAVING – used when aggregated data needs filtering**Examples:

SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5  
ORDER BY COUNT(CustomerID) DESC;

SELECT dept, avg(salary)

FROM employee

GROUP BY dept

HAVING avg(salary) > 20000;

**JOINS-Inequality, Self Joins, Left, Right, Full Outer (to find NULLs/missing values)  
Examples:**

**INEQUALITY JOIN**

SELECT

c.Name AS country\_name,

c.SurfaceArea,

l.Language AS official\_language,

LENGTH(c.Name) AS country\_name\_length,

LENGTH(l.Language) AS language\_name\_length

FROM

country c

JOIN

countrylanguage l ON LENGTH(c.Name) < LENGTH(l.Language);

**UNION (eliminates duplicate rows) / UNION ALL (retains duplicate rows)**

SELECT

first\_name,

last\_name

FROM

employees

UNION

SELECT

first\_name,

last\_name

FROM

dependents

ORDER BY

last\_name;

**UNION ALL – Basic Example:**

SELECT City FROM Customers  
UNION ALL  
SELECT City FROM Suppliers  
ORDER BY City;

**CASE logic**Examples:  
<https://www.w3schools.com/sql/sql_case.asp>

SELECT OrderID, Quantity,  
CASE  
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'  
    WHEN Quantity = 30 THEN 'The quantity is 30'  
    ELSE 'The quantity is under 30'  
END AS QuantityText  
FROM OrderDetails;  
  
SELECT CustomerName, City, Country  
FROM Customers  
ORDER BY  
(CASE  
    WHEN City IS NULL THEN Country  
    ELSE City  
END);

**More Complex (From Danny Ma SQL Case Study 1) - If each $1 spent equates to 10 points and sushi has a 2x points multiplier - how many points would each customer have**

SELECT

customer\_id,

SUM(CASE

WHEN product\_name='sushi' THEN price \* 10 \* 2

ELSE price \* 10

END) as points

FROM dannys\_diner.menu m

INNER JOIN dannys\_diner.sales s ON s.product\_id = m.product\_id

GROUP BY customer\_id;

**IS NULL - identify NULL values for filtering, etc.**Example:

SELECT CustomerName, ContactName, Address  
FROM Customers  
WHERE Address IS NULL;

**IS NOT NULL**

SELECT CustomerName, ContactName, Address  
FROM Customers  
WHERE Address IS NOT NULL;

**DATE\_TRUNC, DATE\_ADD, and others like EXTRACT**Examples:

SELECT EXTRACT(Month from submit\_date)as month, product\_id, ROUND(AVG(stars),2)

FROM reviews

GROUP BY 1,2

ORDER BY 1,2;

/\*

SELECT

EXTRACT(MONTH FROM submit\_date) AS mth,

product\_id,

ROUND(AVG(stars), 2) AS avg\_stars

FROM reviews

GROUP BY

EXTRACT(MONTH FROM submit\_date),

product\_id

ORDER BY mth, product\_id;

\*/

**Group Data by Year and Quarter:**

SELECT  EXTRACT(YEAR FROM date) AS year,

        EXTRACT(QUARTER FROM date) AS quarter,

        COUNT(amount) AS number\_of\_transactions

FROM card\_transaction

GROUP BY EXTRACT(YEAR FROM date), EXTRACT(QUARTER FROM date)

ORDER BY EXTRACT(YEAR FROM date) ASC, EXTRACT(QUARTER FROM date);

**Examples from Danny Ma’s 8 Week Challenge - Pizza Runner (Case Study 2)**

**-- Question 9 What was the total volume of pizzas ordered for each hour of the day?**

/\*

SELECT

DATE\_PART('hour', order\_time) as hour,

COUNT(pizza\_id) pizzasOrdered

FROM pizza\_runner.customer\_orders

GROUP BY DATE\_PART('hour', order\_time)

\*/

**--Question 10 What was the volume of orders for each day of the week?**

/\*

SELECT

--DATE\_PART('dow', order\_time) as weekday,

to\_char(order\_time , 'Day' )as day2,

COUNT(order\_id) orderVolume

FROM pizza\_runner.customer\_orders

GROUP BY

--DATE\_PART('dow', order\_time),

to\_char(order\_time , 'Day' )

\*/

**SQL Server Pivot in Subquery Example:**

select Doctor, Professor, Singer, Actor

from (

select

Occupation,

Name,

row\_number() over (partition by Occupation order by Name) as rown

from Occupations

) as SourceTable

pivot

(

max(name)

for Occupation IN (Doctor, Professor, Singer, Actor)

) as pvt

order by rown

**String/Text CONCATENATION:**

|| represents string concatenation. Unfortunately, string concatenation is not completely portable across all sql dialects:

ansi sql: || (infix operator)

mysql: concat ( vararg function ). caution: || means 'logical or' ([It's configurable](http://dev.mysql.com/doc/refman/5.7/en/sql-mode.html#sqlmode_pipes_as_concat), however; thanks to [@hvd](https://stackoverflow.com/users/743382/hvd) for pointing that out)

oracle: || (infix operator), concat ( caution: function of arity 2 only ! )

postgres: || (infix operator)

sql server: + (infix operator), concat ( vararg function )

sqlite: || (infix operator)

**FINANCE Examples  
  
Example 1: Creating Views and Using CASE Statements**

**In this example, we'll create a view that categorizes members based on their account balances.**

-- Create a view to categorize members by account balance

CREATE VIEW MemberBalanceCategories AS

SELECT member\_id,

CASE

WHEN balance >= 10000 THEN 'High Balance'

WHEN balance >= 5000 THEN 'Medium Balance'

ELSE 'Low Balance'

END AS balance\_category

FROM Accounts;

-- Query the view to see member balance categories

SELECT \* FROM MemberBalanceCategories;

**Example 2: Using CTEs for Advanced Queries**

**This example involves using a Common Table Expression (CTE) to calculate the average transaction amount per member.**

-- Calculate the average transaction amount per member using CTE

WITH TransactionAverages AS (

SELECT

member\_id,

AVG(amount) AS avg\_transaction\_amount

FROM Transactions

GROUP BY member\_id

)

-- Join the CTE with the Members table to display member names

SELECT M.member\_id, M.name, TA.avg\_transaction\_amount

FROM Members M

JOIN TransactionAverages TA ON M.member\_id = TA.member\_id;

**Example 3: Joining Multiple Tables for Data Enrichment**

**Here, we'll perform a join operation to enrich the member data with loan information.**

sqlCopy code

-- Join Members, Accounts, and Loans tables

SELECT

M.name, A.account\_id, A.balance, L.loan\_id, L.loan\_amount

FROM Members M

JOIN Accounts A ON M.member\_id = A.member\_id

LEFT JOIN Loans L ON M.member\_id = L.member\_id;

**These examples demonstrate your SQL skills in data transformation, using CASE statements, creating views, leveraging CTEs for advanced queries, and performing joins to enrich data. You can further build upon these scripts and incorporate Tableau for data visualization by connecting to the resulting SQL views or queries to create meaningful dashboards and reports.**