GENERAL ASSEMBLY

APPLYING FUNCTIONS IN SQL

INTRODUCTION

LEARNING OBJECTIVES

1. Apply string functions to manipulate how data are presented.
2. Apply math functions to add value to the data you’re working with.
3. Apply date logic to your SQL.

INTRODUCTION: STRING, MATH, & DATE FUNCTIONS

STRING, MATH, AND DATE FUNCTIONS

There are many different **function categories** within SQL. We're going to build a foundation with the functions used most commonly across environments.

Some of the various types of functions include:

- String functions.
- Numeric functions.
- Time and date functions.
- Set functions.
- Distinct set functions.
- `JOIN` operators.
- Predicate operators.
- Expression operators.
- Boolean operators.
- Data-type conversion.
- Operators
- Value expressions.

STRING, MATH, AND DATE FUNCTIONS

Let's begin with terms commonly used in reference materials to describe the syntax and appropriate use of SQL functions.

- **Operators** are the ***maths*** of SQL.
- **Delimiters** are the ***grammar*** of SQL.

STRING, MATH, AND DATE FUNCTIONS

Operators are divided into four categories:

1. Arithmetic
2. Comparison
3. Logical
4. String

All categories follow the same rules found in math.

Operator Type	Characters	Description

Arithmetic	`+`
	`-`
	`*`
	`/`
	`**`
		Addition or prefix plus.
		Subtraction or prefix minus.
		Multiplication.
		Division.
		Exponent (to the power of).

Comparison	`=`
	`!= or <>`
	`<`
	`<= or !>`
	`>`
	`>= or !<`
		Equal to.
		Not equal to.
		Less than.
		Less than or equal to (not greater than).
		Greater than.
		Greater than or equal to (not less than).

Logical	`ALL / AND`
	`ANY / OR`
	`BETWEEN`
	`EXISTS`
	`IN`
	`LIKE`
	`NOT`
	`IS NULL`
	`UNIQUE`
		Does the value meet **ALL** criteria in list?
		Does the value meet **ANY** criteria in list?
		Is the value **BETWEEN** listed values?
		Does a row meeting the criteria **EXIST** in the data?
		Is the value found in the listed **literal** values?
		Compares the value to listed values using wildcards.
		Reverses the meaning of a logical operator.
		Checks if the value is **NULL**.
		Searches all rows for **duplicates**.

String	`CHAR(n)`
	`CONCAT()`
	`FORMAT(n)`
	`LOWER()`
	`UPPER()`
	`REPEAT()`
	`TRIM()`
		Returns the character at index **n**.
		Concatenates (puts together) items.
		Returns a number formatted to **n** decimal places.
		Returns the argument in **lowercase**.
		Returns the argument in **uppercase**.
		Repeats the string a certain number of times.
		Removes leading and trailing spaces.

STRING, MATH, AND DATE FUNCTIONS

Delimiters are used to separate or mark the start and end of items of data.

This is similar to regular punctuation. If delimiters are omitted, it results in error conditions.

Examples include unbalanced parentheses or quotes, unmatched comment breaks, and missing semicolons.

Operator	Characters	Description
Comma	`,`	Separates list elements.
Period	`.`	Connects elements of a qualified name/decimal.
Semicolon	`;`	Terminates a statement.
Equals Sign	`=`	Assignment operator/equality in conditional statement.
Colon	`:`	Connects prefix to statements, lower bound to upper bound, and is used in RANGE statements.
Blank	`whitespace`	Separates elements.
Parentheses	`( )`	Encloses lists, expressions, iteration factors, repetition factors, and information associated with a keyword.

Comments	`--`
	`/* text */`
		Basic comment.
		In-line comment.
		Multi-line comment.

Quotes	`‘text’`	Denotes a string, as opposed to text being used as a variable name.

STRING, MATH, AND DATE FUNCTIONS

Two other important terms are **expressions** and **arguments**.

Expression operators create a method used to change or modify the returned values. Expressions in SQL generally fall into one of four categories:

1. Boolean
2. Numeric
3. Character
4. Date expressions

Arguments can be **literal** values or **variables** in a SQL statement.

GUIDED PRACTICE: STRING FUNCTIONS

STRING FUNCTIONS

Keep in mind that different SQL dialects have varied syntax, but their principles are consistent. Cross-reference resources for functions and specific vendors are readily available online (for example: techonthenet.com).

Let’s review a basic query as our starting point:

```
SELECT item, description
FROM sales
LIMIT 100;
```

STRING FUNCTIONS

CONCAT: Combines two fields or expressions together.

Syntax: `CONCAT(field1, field2, field3…)`

Example:

```
SELECT CONCAT(item,' - ',description)
FROM sales
LIMIT 100;
```

STRING FUNCTIONS

LENGTH: Counts the length of characters in a field.

Syntax: `LENGTH(field1)`

Example:

```
    SELECT LENGTH(CONCAT(item,' - ',description))
    FROM sales
    LIMIT 100;
```

STRING FUNCTIONS

REPLACE: Similar to the Excel function **SUBSTITUTE**; replaces a value in a field with another value.

Syntax:
`REPLACE(field_to_change, content_to_replace, new_content)`

Example:

```
    SELECT REPLACE(description,'Absolut', 'Grey Goose')
    FROM sales
    LIMIT 100;
```

STRING FUNCTIONS

Combining a **LENGTH** function with a **REPLACE** function can also work as a word counter for text fields.

Let's use "the quick brown fox jumped over the lazy dog" as an example.

```
LENGTH('the quick brown fox jumped over the lazy dog') = 44
LENGTH('thequickbrownfoxjumpedoverthelazydog') = 36
```

STRING FUNCTIONS

If we combine the two statements, we'd see that the difference in queries is:
44-36 = 8.

Next, since we are counting spaces to get a word count, we'd add one:

```
LENGTH('the quick brown fox jumped over the lazy
dog')-REPLACE(LENGTH('the quick brown fox jumped over the
lazy dog'),' ',''')
```

So in this example, **REPLACE** deletes every instance of a space, which is counted as a character, and lets you do a quick word count of a text field by counting the number of characters (including spaces) and subtracting the number of characters without spaces.

STRING FUNCTIONS

Changing case:

- Changing a case is useful if you have data inconsistencies in a table or across tables and need to create a common key for a `JOIN`.
- For example, if a company is called “apple inc,” “APPLE inc,” or “Apple Inc” in the same table, then a `WHERE` or `LIKE` clause may match one but not the other.
- In the same instance, if you need to join on a company name as a common key, there could be a mismatch across different tables.
- Normalizing the name allows for an easier match.

STRING FUNCTIONS

LOWER: Converts a field or expression to lowercase.
Syntax: `LOWER(field1)`

UPPER: Converts a field or expression to uppercase.
Syntax: `UPPER(field1)`

Example:
```
SELECT UPPER(CONCAT(item,' -  ',description))
FROM sales
LIMIT 100;
```

STRING FUNCTIONS

`LEFT`/`RIGHT` substring selection: Selects a given number of characters from one side of the string.

- `LEFT`: Selects characters from left side.
- `RIGHT`: Selects characters from right side.

Syntax: `LEFT(field1, length)`

Example:

```
    SELECT CONCAT(item,' - ',description),
    LEFT(UPPER(CONCAT(item,' - ',description)),5)
    FROM sales
    LIMIT 100;
```

STRING FUNCTIONS

SUBSTRING allows you to retrieve specific characters within a field.

Syntax:

```
SUBSTRING(field1,starting position,number of characters to
retrieve from starting positions)
```

Example:

```
SELECT SUBSTRING(CONCAT(item,' -',description),9,35)
FROM sales
LIMIT 100;
```

STRING FUNCTIONS

`LEFT/RIGHT TRIM`: Removes blanks from the specified side.

- `LTRIM`: Trims all blanks from the left side.
- `RTRIM`: Trims all blanks from the right side.

Syntax: `LTRIM(field1)`

Example:

```
      SELECT LTRIM(CONCAT(item,' -  ',description))
      FROM sales
      LIMIT 100;
```

STRING FUNCTIONS

TRIM: Removes specific characters from the start of the field (**leading** characters), end of the field (**trailing** characters), or both.

Syntax:

```
TRIM(leading ‘characters’, from field1)
TRIM(trailing ‘characters’, from field1)
TRIM(both ‘characters’, from field1)
```

Example:

```
SELECT description,
   TRIM(Leading 'A' from description),
   TRIM (TRAILING 'a' from description),
   TRIM(BOTH 'A' FROM description)
FROM sales
LIMIT 100;
```

GUIDED PRACTICE: MATH FUNCTIONS

MATH FUNCTIONS

Math functions return calculated values from your data. Many of these functions will require a `GROUP BY` clause to clearly identify which rows to select.

Pro Tip: The default will be to use all rows, but some functions aren’t able to do this.

MATH FUNCTIONS

`COUNT`: Returns the number of rows that match some specified criteria.

- If the criteria include only a column name, `COUNT` will return the number of `non-NULL` values in that column.

Syntax: `Count(column1)`

Example:

```
SELECT category_name, count(item_no)
FROM products
GROUP BY category_name
LIMIT 100;
```

MATH FUNCTIONS

`AVERAGE`: The **`AVG()`** function returns the average value of a numeric column.

Syntax: **`AVG(field)`**

Example:

```
    SELECT store, AVG(total)
    FROM sales
    GROUP BY store
    LIMIT 100;
```

MATH FUNCTIONS

`MIN`: The `MIN()` function returns the smallest value of the selected column.

Syntax: `MIN(field1)`

Example:

```
    SELECT store, MIN(total)
    FROM sales
    GROUP BY store
    LIMIT 100;
```

MATH FUNCTIONS

`MAX`: The **`MAX()`** function returns the largest value of the selected column.

Syntax: **`MAX(field1)`**

Example:

```
    SELECT store, MAX(total)
    FROM sales
    GROUP BY store
    LIMIT 100;
```

MATH FUNCTIONS

`SUM`: This function is used to find the sum of a field in various records.

Syntax: `SUM(field1)`

Example:

```
SELECT store, SUM(total)
FROM sales
GROUP BY store
LIMIT 100;
```

MATH FUNCTIONS

ROUND: This function returns a given number rounded **n** places to the right of the decimal point.

- If **n** is negative, it will be rounded **n** places to the left of the decimal point.
- This function operates within queries, not with hard-coded numbers.

Syntax: `ROUND(numeric_expression, n)`

Example: In the example below, assume x = 177.3589

`ROUND(x, 2)` would return 177.3600.
`ROUND(x, -2)` would return 200.

DATA TYPES

Before we continue, let’s make a distinction regarding data types:

- **Integers**: An integer is a *whole number*. It doesn’t have fractions, decimals, etc.
  - They are expressed as 1, 2, 3, 5000, 38983498, etc.

- **Floats**: A float value lets you express up to approximately 15,000 digits after the decimal point for accuracy.
  - They are expressed as 1.0000, 2.0134, 3.1419, 3944.39, etc.

Pro tip: Most of the time, storing 10 digits is enough.

DATA TYPES

When completing a “math” calculation, SQL uses the same data type for both inputs and outputs:

- integer + integer: integer 3+5 = 8
- integer * integer: integer 3*9 = 27
- integer / integer: integer 5/3 = 2

DATA TYPES: CONVERTING

To convert between two data types, we need to call out which data type we want. To change data types, we can use the **`CAST`** command.

To apply the **`CAST`** function:

- **`CAST`**`(sum(total) as int)` would change the sum(total) of our liquor store data from a decimal to an integer.
- **`CAST`**`(count(total) as decimal)` would change the number of sales from an integer to a decimal.
- We can also use a shorthand version by typing `::[datatype]` at the end.
  - For example, we can change an integer, “3,” to a decimal, “3.0,” via `3::decimal`.

MATH FUNCTIONS

To return to our example, **`SELECT`** `5/3` returns 2, not the correct answer of 1.666.

We can **`CAST`** the numeric values as **floats**, rewriting the query as:

```
      SELECT 4::float/3::float
```

We could also multiply the terms by 1.0 to convert each one to a float:

```
      SELECT (1.0*4)/(1.0*3)
```

GUIDED PRACTICE: DATE FUNCTIONS

DATE FUNCTIONS

Syntax varies depending on the SQL dialect and vendor.

- For example, IBM’s current query tool uses a `TIMESTAMPDIFF` function, while pgAdmin uses an `AGE` function to the same effect.

The best approach is to have an overall understanding of what `DATEs` can do, and consult your vendor's `DATE` documentation.

We’ll take a look at `CURRENT_DATE` and `AGE` — the most popular date functions.

DATE FUNCTIONS

`CURRRENT_DATE`: Brings back the current date from the system.

Syntax:

```
‣CURRENT_DATE
```

Example:

```
‣SELECT item, total, date, CURRENT_DATE
‣FROM sales
‣LIMIT 100;
```

`GETDATE()` is another handy function that will return the current date and time.

DATE FUNCTIONS

AGE: Returns the difference between two dates.

Syntax:

```
‣AGE(date1, date2)
```

Example:

```
‣SELECT item, total, date, CURRENT_DATE, AGE(date,
 CURRENT_DATE)
‣FROM sales
‣LIMIT 100;
```

DATE FUNCTIONS

Let’s consider how dates are used in your work:

- Billing date by day of week.
- Changes in day of the week by year.
- Comparison of days of the week by two dates.
- Estimation based on day of week.
- Estimation based on previous year.
- How many customers on a given day?
- Order date to ship date.

MORE ON DATES: SELECTING DATE

Most transaction-level databases will timestamp an entry down to the millisecond, but generally, questions are asked about sales by day, month, year, etc.

We'll need to extract and aggregate this data.

We'll start off with basic data exploration. What range of dates are we using in the Iowa Liquor Sales Database? `MIN/MAX` functions will give us some useful information.

```
Select min(date), max(date)
From sales
```

SELECTING DATE

Now how about sales by date?

```
SELECT date
, COUNT(date) AS sales_count
FROM sales
GROUP BY date
```

What day has the highest amount of sales?

```
SELECT date
, COUNT(date) AS sales_count
FROM sales
GROUP BY date
ORDER BY 2 DESC
```

RE-AGGREGATING DATES

Next, we'll look at the sales by month. There are two ways to get the month from the date. The first is the `DATE_TRUNC` function; the second is the `EXTRACT` function.

The main difference between the two is that `DATE_TRUNC` aggregates to the level of "date detail" you specify.

In other words, `EXTRACT` aggregates data at a **combined** level of date detail.

RE-AGGREGATING DATES

Run both of these codes and explain the difference between the results:

```
SELECT
EXTRACT(month FROM date)
, COUNT(date) AS sales_count
FROM sales
GROUP BY 1
ORDER BY 1
```

```
SELECT
DATE_TRUNC(‘month’,date)
, COUNT(date) AS sales_count
FROM sales
GROUP BY 1
ORDER BY 1
```

RE-AGGREGATING DATES

The `EXTRACT` function is aggregated **at** the level of detail you specified. It returns all sales by month, regardless of the year.

The `DATE_TRUNC` function is aggregated **to** the level of detail you specified. It returns sales by year and month.

RE-AGGREGATING DATES

Taking a look at the documentation. What levels of date detail can we use?

https://www.postgresql.org/docs/9.1/static/functions-datetime.html

RE-AGGREGATING DATES

What code would we use to get sales by week?

Work with a partner for five minutes to get the answer.

```
SELECT DATE_TRUNC(‘week’,date)
, COUNT(date) AS sales_count
FROM sales
GROUP BY 1
```

What order are the data in? What do you need to do next?

```
SELECT DATE_TRUNC(‘week’,date)
, COUNT(date) AS sales_count
FROM sales
GROUP by 1
```

RE-AGGREGATING DATES

How do you get rid of time when you don’t need hour/minute?

We can wrap the TO_CHAR function around our DATE_TRUNC to clean up the time information.

The format is:

```
Select TO_CHAR(date_trunc('month',date),'YYYY-MM-DD')
, sum(total)
From sales
Group by 1
```

DATES & WHERE CLAUSES

Unlike in Excel, dates are non-numeric and are considered strings.

Because of this, dates need to be wrapped in tick marks/single quotes. Most SQL formats treat dates using the YYYY-MM-DD format.

Let’s say we want to find all of the sales after a specific date; we would use:

```
Select * from sales
Where date>='2015-01-01'
```

DATES & WHERE CLAUSES

Pair up again and determine the proper code to use for the following criteria:

1. The number of sales by month.
2. The number of sales that occurred on the Fourth of July.
3. The number of sales between Memorial Day and Labor Day.
4. The number of sales for Black Hawk county in January and February 2014 and Des Moines county in July and August 2014.

INDEPENDENT PRACTICE: PUTTING IT ALL TOGETHER

PUTTING IT ALL TOGETHER

- `SQL` functions are plentiful. As you practice writing queries, you'll discover others useful tools, including `VARCHARS,` data types, and `SQRT` functions.

- This course has given you a solid functional foundation of tools and intermediate skills to use in production environments.

- The next step is to find out the specific database management vendor you'll be using. Recall that there are many different data management systems for SQL databases (e.g., SQLite, MySQL, PostgreSQL, Oracle's SQL & PL SQL, Toad, and Alteryx).

- Research your vendor and its special capabilities while continuing to practice and develop your skills.

ACTIVITY: PUTTING IT ALL TOGETHER

EXERCISE-2

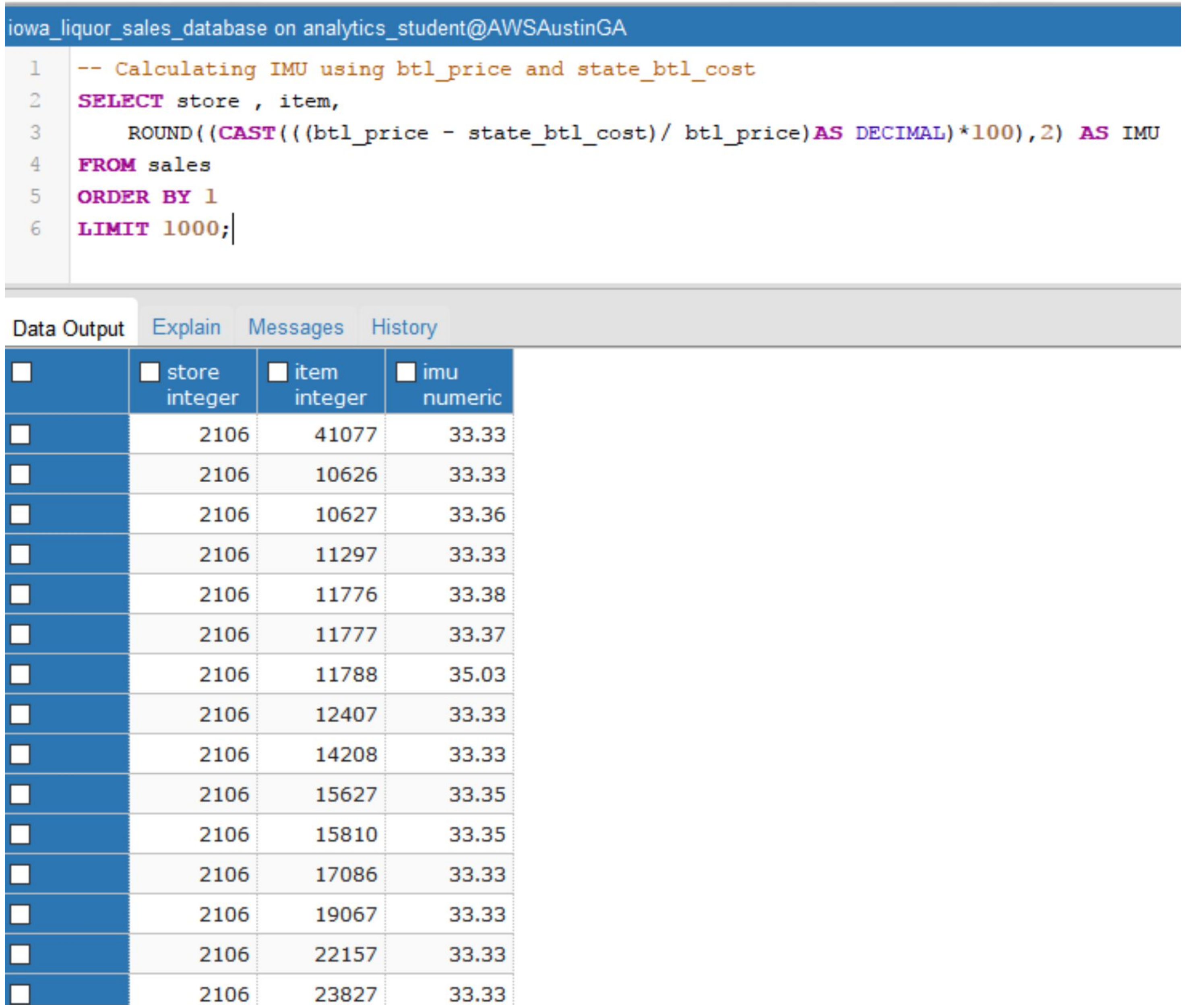
DIRECTIONS

Let's apply our SQL math and string manipulation skills to complete the following request.

For each store, list each item and its initial markup percentage using the formula below. Use btl_price and state_btl_cost for the calculation. In the output of the query, round the remainder to two digits. Limit the output to 100 rows.

$$\frac{(price - cost)}{price} \times 100$$

PUTTING IT ALL TOGETHER: SOLUTION



iowa_liquor_sales_database on analytics_student@AWSAustinGA

```
-- Calculating IMU using btl_price and state_btl_cost
SELECT store , item,
    ROUND((CAST(((btl_price - state_btl_cost)/ btl_price)AS DECIMAL)*100),2) AS IMU
FROM sales
ORDER BY 1
LIMIT 1000;
```

Data Output | Explain | Messages | History

store integer	item integer	imu numeric
2106	41077	33.33
2106	10626	33.33
2106	10627	33.36
2106	11297	33.33
2106	11776	33.38
2106	11777	33.37
2106	11788	35.03
2106	12407	33.33
2106	14208	33.33
2106	15627	33.35
2106	15810	33.35
2106	17086	33.33
2106	19067	33.33
2106	22157	33.33
2106	23827	33.33

CONCLUSION

WHAT DID WE LEARN?

To recap, in today’s lesson we:

1. Showed how to apply **string functions** to manipulate and transform data.
2. Explored **math functions** that can add value to the data you’re working with.
3. Reviewed date functions and logic within SQL.

APPLYING FUNCTIONS IN SQL

Q&A

RESOURCES

RESOURCES

- “What are the SQL Database Functions?”: https://docs.microsoft.com/en-us/sql/t-sql/functions/functions
- SQL Server: Functions – listed by category – from TechOnTheNet: https://www.techonthenet.com/sql_server/functions/index.php
- Useful SQL Functions – from TutorialPoints: https://goo.gl/Zg9yz1