SQL S10 Conditional Expressions and Procedures

My Course Notes and Code

Section Overview

- CASE
- COALESCE
- NULLIF
- CAST
- Views
- Import and Export Functionality

CASE statement

- Executing SQL code *only* when certain conditions are met.
- Similar to IF/ELSE in other programming languages

General CASE Syntax:

```
CASE
        WHEN condition2 THEN result1
        WHEN condition2 THEN result2
        ELSE some_other_result
END
```

- Example:

  ```
  SELECT a,
          CASE
                  WHEN a = 1 THEN 'one'
                  WHEN a = 2 THEN 'two'
                  ELSE 'other'
          END AS label -- 'CASE' is the default
  FROM test;
  ```

  - In this case, `CASE` is like a substitute for calling a columnl...
  - More flexible than *CES* (below), which only checks for equality

CASE Expression Syntax

```
CASE expression
        WHEN value1 THEN result1
        WHEN value2 THEN result2
        ELSE some_other_result
END
```

- Rewriting the previous example:

  ```
  SELECT a,
          CASE a
                  WHEN 1 THEN 'one'
  ```

```
                WHEN 2 THEN 'two'
        ELSE 'other' AS label -- 'CASE' is the default
        END
FROM test;
```

- Useful for checking for equality across many columns

COALESCE function

- ... accepts an unlimited number of arguments (usually columns are used as arguments).
- Returns the first argument that is not null.

```
SELECT COALESCE(1, 2, 3, NULL); --> 1
SELECT COALESCE(5, 6, 8, NULL); --> 5
SELECT COALESCE(NULL, NULL, 15, NULL); --> 15
```

- If all arguments are null, it will return null.

```
SELECT COALESCE(NULL, NULL, NULL, NULL); --> NULL
```

- **Example use**:

```
-- Example table includes columns: item, pre_discount_price, discount
SELECT item, (pre_discount_price - COALESCE(discount, 0)) AS final_price
FROM table;
```

`COALESCE` is useful when a column includes null values, yet we want to perform operations on it without having to make changes on the table.

CAST

The `CAST` operator lets us convert from one data type into another. However, it must be *reasonable* to convert the data in desired way.5

- Syntax for `CAST` function:

```
SELECT CAST('5' AS INTEGER)
```

- PostgreSQL `CAST` operator:

```
SELECT '5'::INTEGER
```

Typically, it is used with columns as arguments to be converted into a different data type.

NULLIF

Takes in 2 inputs.

- If both are equal → NULL
- Otherwise, it returns the *first* argument passed

```
SELECT NULLIF(10,10); --> NULL
SELECT NULLIF(10,12); --> 10
```

- Can be very usefull in cases where a *zero* value would cause an error or unwanted result.
- In other words, it can serve as a check against returning *zeros*.

Views

There are often specific combinations of tables and conditions that we use very often for a certain project. `VIEW` allows to quickly see these queries, without having to write them over and over again.

- It is a stored query. It can be assessed as a *virtual* table in PostgreSQL.
- Existing *views* can be updated and altered.

Import and Export

- Functionalities which alow to import/export data from/into *.csv* files.
- Not every outside data file will work - due to variations in formatting, macros, data types...
  - In such cases, the datafiles should first be edited to be compatible with SQL.
  - Details and examples on compatible file types: https://www.postgresql.org/docs/12/sql-copy.html
- Path files must be 100% correct
- The *Import* command doesn't create tables for us. It assumes that tables are acready created.

Jose's suggested additional resources:

- https://stackoverflow.com/questions/2987433/how-to-import-csv-file-data-into-a-postgresql-table
- https://www.enterprisedb.com/postgres-tutorials/how-import-and-export-data-using-csv-files-postgresql
- https://stackoverflow.com/questions/21018256/can-i-automatically-create-a-table-in-postgresql-from-a-csv-file-with-headers

CODE - The entire course segment

```
-- DVDrental database ---------------------------------------------------------------------

SELECT *
FROM customer
LIMIT 3;

-- GENERAL CASE SYNTAX --------------------------------------------------------------------

SELECT customer_id,
        CASE
                WHEN (customer_id <= 100) THEN 'Premium'
                WHEN (customer_id BETWEEN 100 AND 200) THEN 'Plus'
                ELSE 'Normal'
        END AS customer_class
FROM customer;

-- CASE EXPRESSION SYNTAX -----------------------------------------------------------------

SELECT customer_id,
        CASE customer_id
                WHEN 2 THEN 'Winner'
                WHEN 5 THEN '2nd place'
                ELSE 'Normal'
        END AS raffle_results
```

```
FROM customer;

-- CALCULATIONS on results of CASE statements ---------------------------------------------

SELECT *
FROM film
LIMIT 5;

SELECT
SUM(CASE rental_rate
                WHEN 0.99 THEN 1
                ELSE 0
        END) AS sum_of_bargains,
SUM(CASE rental_rate
                WHEN 2.99 THEN 1
                ELSE 0
        END) AS regular,
SUM(CASE rental_rate
                WHEN 4.99 THEN 1
                ELSE 0
        END) AS premium
FROM film;

-- CHALLENGE TASK - My Solution -----------------------------------------------------------

SELECT DISTINCT rating
FROM film;

SELECT
SUM(CASE rating
                WHEN 'R' THEN 1
                ELSE 0
        END) AS r,
SUM(CASE rating
                WHEN 'PG' THEN 1
                ELSE 0
        END) AS pg,
SUM(CASE rating
                WHEN 'PG-13' THEN 1
                ELSE 0
        END) AS pg13
FROM film;

-- CAST -----------------------------------------------------------------------------------

SELECT CAST('5' AS INTEGER) AS new_int;
SELECT CAST('five' AS INTEGER) AS new_int; -- not reasonable

SELECT '5'::INTEGER; -- PostgreSQL operator

SELECT *
FROM rental
LIMIT 2;

SELECT DISTINCT(CHAR_LENGTH(CAST(inventory_id AS VARCHAR)))
```

```
FROM rental;

-- NULLIF ----------------------------------------------------------------------------------
-- I first created a new 'testme' database -------------------------------------------------

CREATE TABLE departments(
        first_name VARCHAR(50) NOT NULL,
        department VARCHAR(50) NOT NULL
);

INSERT INTO departments(first_name, department)
VALUES
        ('Marion', 'A'),
        ('Caren', 'A'),
        ('Anya', 'B');

SELECT *
FROM departments;

SELECT (
        SUM(CASE department WHEN 'A' THEN 1 ELSE 0 END) /
        SUM(CASE department WHEN 'B' THEN 1     ELSE 0 END)
) AS department_ratio
FROM departments;

DELETE FROM departments
WHERE department = 'B';

SELECT (
        SUM(CASE department WHEN 'A' THEN 1 ELSE 0 END) /
        NULLIF(SUM(CASE department WHEN 'B' THEN 1 ELSE 0 END), 0)
) AS department_ratio
FROM departments; -- We get back NULL, instead of 'division by ZERO' error
                                  -- This makes sense, since division by NULL
                                  -- gives back NULL.

-- VIEW ------------------------------------------------------------------------------------
-- We're using DVDrental database again ----------------------------------------------------

CREATE VIEW customer_info AS
SELECT first_name, last_name, address
FROM customer AS c
INNER JOIN address AS a
ON c.address_id = a.address_id;

SELECT * FROM customer_info;

CREATE OR REPLACE VIEW customer_info AS
SELECT first_name, last_name, address, district
FROM customer AS c
INNER JOIN address AS a
ON c.address_id = a.address_id;

SELECT * FROM customer_info;
```

```sql
ALTER VIEW customer_info
RENAME TO c_info;

SELECT * FROM c_info;

DROP VIEW IF EXISTS customer_info;

-- IMPORT and EXPORT ----------------------------------------------------------------------
-- testme database ------------------------------------------------------------------------

-- C:\Users\PC\Desktop\simple_table.csv
-- Import and Export are not in the query but using the
-- PGAdmin interface on the left hand side

CREATE TABLE simple(
        A INTEGER,
        B INTEGER,
        C INTEGER
);

SELECT *
FROM simple;
```