Assignment 2_SQL_Aman Aug 17

Student: Aman Kaushik

/* ASSIGNMENT 2 */

/* SECTION 2 */

/* Student: Aman Kaushik

-- COALESCE

/* 1. Our favourite manager wants a detailed long list of products, but is afraid of tables!

We tell them, no problem! We can produce a list with all of the appropriate details.

Using the following syntax you create our super cool and not at all needy manager a list:

SELECT

product_name || ', ' || product_size|| ' (' || product_qty_type || ')'

FROM product

But wait! The product table has some bad data (a few NULL values).

Find the NULLs and then using COALESCE, replace the NULL with a

blank for the first problem, and 'unit' for the second problem.

HINT: keep the syntax the same, but edited the correct components with the string.

The `||` values concatenate the columns into strings.

Edit the appropriate columns -- you're making two edits -- and the NULL rows will be fixed.

All the other rows will remain the same.) */

----Corrcted:replace NULL values in the product_size and product_qty_type columns::With COALESCE

```
SELECT
  product_name || ', ' || COALESCE(product_size, '') || ' (' || COALESCE(product_qty_type, 'unit') || ')' AS product_details
FROM
  product;
```

--Windowed Functions

/* 1. Write a query that selects from the customer_purchases table and numbers each customer's

visits to the farmer's market (labeling each market date with a different number).

Each customer's first visit is labeled 1, second visit is labeled 2, etc.

You can either display all rows in the customer_purchases table, with the counter changing on

each new market date for each customer, or select only the unique market dates per customer

(without purchase details) and number those visits.

HINT: One of these approaches uses ROW_NUMBER() and one uses DENSE_RANK(). */

```
SELECT
  customer_id,
  market_date,
  ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY market_date) AS visit_number
FROM
  customer_purchases;
```

/* 2. Reverse the numbering of the query from a part so each customer's most recent visit is labeled 1,

then write another query that uses this one as a subquery (or temp table) and filters the results to

only the customer's most recent visit. */

---To see reverse the numbering & find the most recent visit, use the ROW_NUMBER()

```
 WITH RankedVisits AS (
 SELECT
  customer_id,
  market_date,
  DENSE_RANK() OVER (PARTITION BY customer_id ORDER BY market_date DESC) AS visit_rank
 FROM
  customer_purchases
)
SELECT
 customer_id,
 market_date
FROM
 RankedVisits
WHERE
 visit_rank = 1;
```

/* 3. Using a COUNT() window function, include a value along with each row of the

customer_purchases table that indicates how many different times that customer has purchased that product_id. */

```
SELECT
 customer_id,
 market_date,
 product_id,
 COUNT(*) OVER (PARTITION BY customer_id, product_id) AS
total_purchases_of_product
FROM
 customer_purchases;
```

-- String manipulations

/* 1. Some product names in the product table have descriptions like "Jar" or "Organic".

These are separated from the product name with a hyphen.

Create a column using SUBSTR (and a couple of other commands) that captures these, but is otherwise NULL.

Remove any trailing or leading whitespaces. Don't just use a case statement for each product!

| product_name | description |

|---|---|

| Habanero Peppers - Organic | Organic |

Hint: you might need to use INSTR(product_name,'-') to find the hyphens. INSTR will help split the column. */

```
SELECT
  product_name,
  CASE
   WHEN INSTR(product_name, '-') > 0 THEN TRIM(SUBSTR(product_name,
INSTR(product_name, '-') + 1))
   ELSE NULL
  END AS description
FROM
  product;
```

-- UNION

/* 1. Using a UNION, write a query that displays the market dates with the highest and lowest total sales.

HINT: There are a possibly a few ways to do this query, but if you're struggling, try the following:

1) Create a CTE/Temp Table to find sales values grouped dates;

2) Create another CTE/Temp table with a rank windowed function on the previous query to create

"best day" and "worst day";

3) Query the second temp table twice, once for the best day, once for the worst day,

with a UNION binding them. */

```
WITH DailySales AS (
  SELECT
```

```
    market_date,
    SUM(price_per_unit) AS total_sales
  FROM
    customer_purchases
  GROUP BY
    market_date
),
RankedSales AS (
  SELECT
    market_date,
    total_sales,
    DENSE_RANK() OVER (ORDER BY total_sales DESC) AS sales_rank_high,
    DENSE_RANK() OVER (ORDER BY total_sales ASC) AS sales_rank_low
  FROM
    DailySales
)
SELECT
  market_date,
  total_sales,
  'Highest Sales Day' AS sales_category
FROM
  RankedSales
WHERE
  sales_rank_high = 1
UNION ALL
SELECT
```

market_date,

total_sales,

'Lowest Sales Day' AS sales_category

FROM

RankedSales

WHERE

sales_rank_low = 1;

/* SECTION 3 */

-- Cross Join

/*1. Suppose every vendor in the `vendor_inventory` table had 5 of each of their products to sell to **every**

customer on record. How much money would each vendor make per product?

Show this by vendor_name and product name, rather than using the IDs.

HINT: Be sure you select only relevant columns and rows.

Remember, CROSS JOIN will explode your table rows, so CROSS JOIN should likely be a subquery.

Think a bit about the row counts: how many distinct vendors, product names are there (x)?

How many customers are there (y).

Before your final group by you should have the product of those two queries (x*y). */

SELECT

v.vendor_name,

p.product_name,

SUM(vi.original_price * 5) AS total_revenue

```
FROM
  vendor_inventory AS vi
JOIN
  vendor AS v ON vi.vendor_id = v.vendor_id
JOIN
  product AS p ON vi.product_id = p.product_id
CROSS JOIN (
  SELECT DISTINCT customer_id FROM customer
) AS c
GROUP BY
  v.vendor_name,
  p.product_name;
```

-- INSERT

/*1. Create a new table "product_units".

This table will contain only products where the `product_qty_type = 'unit'`.

It should use all of the columns from the product table, as well as a new column for the `CURRENT_TIMESTAMP`.

Name the timestamp column `snapshot_timestamp`. */

```
CREATE TABLE product_units AS
SELECT
  *,
  CURRENT_TIMESTAMP AS snapshot_timestamp
FROM
  product
WHERE
  product_qty_type = 'unit';
```

/*2. Using `INSERT`, add a new row to the product_units table (with an updated timestamp).

This can be any product you desire (e.g. add another record for Apple Pie). */

```
INSERT INTO product_units (
  product_id,
  product_name,
  product_size,
  product_category_id,
  product_qty_type,
  snapshot_timestamp
)
VALUES (
  444,
  'New Test Product-Apple',
  'Large',
```

```
 'unit',
 12.50,
 CURRENT_TIMESTAMP
);
```

-- DELETE

/* 1. Delete the older record for the whatever product you added.

HINT: If you don't specify a WHERE clause, you are going to have a bad time.*/

```
DELETE FROM
 product_units
WHERE
 product_id = 444;
```

-- UPDATE

/* 1.We want to add the current_quantity to the product_units table.

First, add a new column, current_quantity to the table using the following syntax.

```
ALTER TABLE
        product_units
ADD current_quantity INT;
```

Then, using UPDATE, change the current_quantity equal to the last quantity value from the vendor_inventory details.

```
UPDATE product_units
SET
 current_quantity = (
  SELECT
   COALESCE(quantity, 0)
  FROM
   vendor_inventory
  WHERE
   vendor_inventory.product_id = product_units.product_id
  ORDER BY
   vendor_inventory.market_date DESC
  LIMIT 1
 )
WHERE
 product_units.product_id IN (
  SELECT DISTINCT
   product_id
  FROM
   vendor_inventory
 );
```

HINT: This one is pretty hard.

First, determine how to get the "last" quantity per product.

Second, coalesce null values to 0 (if you don't have null values, figure out how to rearrange your query so you do.)

Third, SET current_quantity = (...your select statement...), remembering that WHERE can only accommodate one column.

Finally, make sure you have a WHERE statement to update the right row,

you'll need to use product_units.product_id to refer to the correct row within the product_units table.

When you have all of these components, you can run the update statement. */