SQL-06 | D&T contd.
CTEs, Views, Union and Self Joins

Lecture Queries

Question: Write a query that gives us the days between each purchase a customer makes.

```
SELECT
    x.customer_id,
    x.market_date,
    RANK() OVER (PARTITION BY x.customer_id ORDER BY
x.market_date) AS purchase_number,
    LEAD(x.market_date, 1) OVER (PARTITION BY x.customer_id
ORDER BY x.market_date) AS next_purchase,
    DATEDIFF(LEAD(x.market_date, 1) OVER (PARTITION BY
x.customer_id ORDER BY x.market_date),
                x.market_date
            ) AS days_between_prch
FROM (SELECT DISTINCT customer_id, market_date
    FROM customer_purchases) AS x
```

Question: today’s date is April 30, 2019, and the marketing director of the farmer’s market wants to give infrequent customers an incentive to return to the market in April.

```
SELECT x.customer_id,
        COUNT(x.market_date) AS market_count
FROM
(SELECT DISTINCT customer_id,
    market_date,
    DATEDIFF(market_date, '2019-04-30')
FROM farmers_market.customer_purchases
WHERE market_date >= '2019-04-01' AND
DATEDIFF(market_date, '2019-04-30') < 0) AS x
GROUP BY x.customer_id
HAVING COUNT(x.market_date) = 1
```

Question: if we wanted to reuse the previous query we wrote to generate the dataset of sales summarized by date and vendor for a report that summarizes sales by market week, we could put that query inside a WITH clause.

```
WITH sales_by_day_vendor AS (
SELECT
    cp.market_date,
    md.market_day,
    md.market_week,
    md.market_year,
    cp.vendor_id,
    v.vendor_name,
    v.vendor_type,
    ROUND(SUM(quantity * cost_to_customer_per_qty),
2) AS total_sales
FROM farmers_market.customer_purchases AS cp
    LEFT JOIN farmers_market.market_date_info AS md
        ON cp.market_date = md.market_date
    LEFT JOIN farmers_market.vendor AS v
        ON cp.vendor_id = v.vendor_id
GROUP BY
        cp.market_date,
        cp.vendor_id,
        md.market_day,
        md.market_week,
        md.market_year,
        v.vendor_name,
        v.vendor_type
ORDER BY cp.market_date, cp.vendor_id
)

SELECT s.market_year,
    s.market_week,
    SUM(s.total_sales) AS weekly_sales
FROM sales_by_day_vendor AS s
GROUP BY s.market_year, s.market_week
```

Views

```
CREATE VIEW farmers_market.vw_sales_by_day_vendor AS
 SELECT
     cp.market_date,
     md.market_day,
     md.market_week,
     md.market_year,
     cp.vendor_id,
     v.vendor_name,
     v.vendor_type,
     ROUND(SUM(cp.quantity * cp.cost_to_customer_per_qty),2) AS sales
 FROM farmers_market.customer_purchases AS cp
     LEFT JOIN farmers_market.market_date_info AS md
         ON cp.market_date = md.market_date
     LEFT JOIN farmers_market.vendor AS v
         ON cp.vendor_id = v.vendor_id
 GROUP BY cp.market_date, cp.vendor_id
 ORDER BY cp.market_date, cp.vendor_id
```

Views vs CTEs

Although there are some differences between them, common table expressions and views seem to perform very similarly. So, when should you use each one?

- **Ad-hoc queries.** For queries that are referenced occasionally (or just once), it’s usually better to use a CTE. If you need the query again, you can just copy the CTE and modify it if necessary.
- **Frequently used queries.** If you tend to reference the same query often, creating a corresponding view is a good idea. However, you’ll need **create view permission** in your database to create a view.
- **Access management.** A view might be used to restrict particular users’ database access while still allowing them to get the information they need. You can give users access to specific views that query the data they’re allowed to see without exposing the whole database. In such a case, a view provides an additional access layer.

Unions

Let’s look at one more important clause the Union clause.

- Using a UNION, you can combine any two queries that result in the same number of columns with the same data types.
- The columns must be in the same order in both queries.
- There are many possible use cases for UNION queries, but the syntax is simple: write two queries with the same number and type of fields, and put a UNION keyword between them:

Let’s say you want to get all the cities from two different tables.

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

Question Extract all the customers from the same Zip code.

This is where you’d need to use self join.

A self-join in SQL is when a table is joined to itself (you can think of it like two copies of the table joined together) in order to compare rows to one another.

```
SELECT
        A.customer_first_name AS CN1,
        B.customer_first_name CN2,
        A.customer_zip
From farmers_market.customer as A, farmers_market.customer B
WHERE A.customer_id <> B.customer_id
        AND
        A.customer_zip = B.customer_zip
```