SQL-07 | CTEs, Views, Union and Self Joins

Lecture Queries

Question: We want to design the dataset to have one row per date and vendor, we do not need to include detailed information about the customers or products.

OR

Question : Get total sales as per vendor across complete duration?

Question 1 : Get total sales as per vendor across complete duration ?
And
Question 2: We want to design the dataset to have one row per date and vendor, we do not need to include detailed information about the customers or products.

```
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
```

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.

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
```

Q: Get the empno with the highest salary

depname	empno	salary
develop	11	5200
develop	7	4200
develop	9	4500
develop	8	6000
develop	10	5200
personnel	5	3500
personnel	2	3900
sales	3	4800
sales	1	5000
sales	4	4800

Q: Get the empno with the highest salary

depname	empno	salary
develop	11	5200
develop	7	4200
develop	9	4500
develop	8	6000
develop	10	5200
personnel	5	3500
personnel	2	3900
sales	3	4800
sales	1	5000
sales	4	4800

```
WITH sal_rank AS
(SELECT empno,
        RANK() OVER(ORDER BY salary DESC) rnk
FROM salaries
)
SELECT empno FROM sal_rank
WHERE rnk = 1;
```

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.