Relational databases : Querying with aggregations

Databases can represent a very large volume of information, and we need to determine some aggregations (summaries) of data to handle them correctly in our "human" analysis. We may need to

- Count ('count()'), Sum (sum())
- Calculate: average (avg()), median and percentiles (percentile_disc(0.5))
- Determine partitions (categories of data) and do the aggregation on those

Relational databases : Querying with aggregations (2)

- The results of aggregations queries will be always a single value (a projection with one row and one column)
- You can observe what will be the result of the following query against the booking database

```
SELECT count(*) FROM facilities
```

Relational databases : Querying with aggregations (3)

- As for other SELECT queries, it will be necessary to **filter** data to aggregate only data of a certain kind
- You can then apply a filter using a regular WHERE clause
- try to execute the following query:

```
SELECT count(*) FROM facilities WHERE name LIKE 'Tennis%'
```

Relational databases : sorting and limiting data

- `ORDER BY` allows to give an order based on a column sort
- `RANK() OVER ()` allows to calculate a rank, and handles rank equality

Relational databases: example for ORDER BY

it can be combined with ASC and DESC to organize result in either ascending or descending order

```
SELECT * FROM members
ORDER BY joindate desc;
```

Relational databases: summarizing with group by

the following will group bookings by memberid and count each group sub

total bookings

```
SELECT bookings.memid as memberid, count(bookings.bookid) as cnt FROM bookings
LEFT JOIN members ON members.memid = bookings.memid
GROUP BY bookings.memid
ORDER BY cnt desc
```

Relational databases : sub SELECTs

One can consider using the result of a SELECT as a regular table

```
SELECT memberid, cnt, FROM (
    SELECT bookings.memid as memberid, count(bookings.bookid) as cnt FROM bookings
    LEFT JOIN members ON members.memid = bookings.memid
    GROUP BY bookings.memid
    ORDER BY cnt desc
) mem_bookings
```

Relational databases : sub SELECTs with rank

One can consider using the result of a SELECT as a regular table

```
SELECT
    memberid, cnt, pos
FROM (
        SELECT
            bookings.memid as memberid,
            COUNT(bookings.bookid) as cnt
            RANK () OVER (ORDER BY cnt desc) pos
        FROM bookings
        LEFT JOIN members ON members.memid = bookings.memid
        GROUP BY bookings.memid
        ORDER BY cnt desc
    ) mem_bookings
WHERE pos <= 4
```

Query processing order

the example with `RANK () OVER ()` shows an interesting fact about processing order

- If you noticed, the rank has to be filtered from outside the first query
- It is because of the processing order, which is the following :

  `FROM` --> `WHERE` --> `SELECT`
- As the rank function belongs to the `SELECT` phase (the column is computed at this time), the `WHERE` clause can't apply on it
- This is why in those cases we need subselects