COMP 330: SQL 2

Chris Jermaine and Kia Teymourian

Rice University

Aggregations

Can compute simple statistics using SQL

- SUM
- AVERAGE
- COUNT
- VARIANCE
- MAX
- MIN
- etc.

Question: What do all of these aggregates have in common?

Our First Aggregation

RATES (DRINKER, BEER, SCORE)

What is the average beer rating given by Chris?

Our First Aggregation

RATES (DRINKER, BEER, SCORE)

What is the average beer rating given by Chris?

```
SELECT AVERAGE (r.SCORE)
FROM RATES r
WHERE r.DRINKER = 'Chris'
```

COUNT DISTINCT

RATES (DRINKER, BEER, SCORE)

How many beers has Chris rated?

COUNT DISTINCT

RATES (DRINKER, BEER, SCORE)

How many beers has Chris rated?

Does this work?

```
SELECT COUNT (*)
FROM RATES r
WHERE r.DRINKER = 'Chris'
```

- Counts the number of ratings due to Chris.

COUNT DISTINCT

RATES (DRINKER, BEER, SCORE)

How many beers has Chris rated?

This gives us the actual number rated:

```
SELECT COUNT DISTINCT (r.BEER)
FROM RATES r
WHERE r.DRINKER = 'Chris'
```

GROUP BY

RATES (DRINKER, BEER, SCORE)

It is often desirable to compute an aggregate at a finer level of granularity.

Example: what is the average rating for each beer?

GROUP BY

RATES (DRINKER, BEER, SCORE)

It is often desirable to compute an aggregate at a finer level of granularity.

Example: what is the average rating for each beer?

```
SELECT r.BEER, AVERAGE (r.RATING)
FROM RATES r
GROUP BY r.BEER
```

- This first groups the relation into sets
- Every tuple in the set has the same value for r.BEER
- Then aggregate run over each set independently

GROUP BY

```
SELECT r.BEER, AVERAGE (r.RATING)
FROM RATES r
GROUP BY r.BEER
```

- Example input:

```
('Chris', 'PBR', 1)
('Chris', 'SSTP', 10)
('Kia', 'PBR', 2)
('Kia', 'Bud', 4)
```

GROUP BY

```
SELECT r.BEER, AVERAGE (r.RATING)
FROM RATES r
GROUP BY r.BEER
```

- Example input:

```
('Chris', 'PBR', 1)
('Chris', 'SSTP', 10)
('Kia', 'PBR', 2)
('Kia', 'Bud', 4)
```

- Output:

```
('PBR', 1.5)
('SSTP', 10)
('Bud', 4)
```

- Take care with integer arithmetic!

GROUP BY

```
SELECT r.BEER, AVERAGE (r.RATING)
FROM RATES r
GROUP BY r.BEER
```

- Also note: If you have an attribute outside of an agg function in an agg query
- Example: r.BEER here
- Then you must have grouped by that attribute
- Or query will not compile
- Why?

Subquery in FROM

RATES (DRINKER, BEER, SCORE)

Can have a subquery in FROM clause, treat as a temporary table

Example: What is the highest rated beer, on average?

Subquery in FROM

RATES (DRINKER, BEER, SCORE)

Can have a subquery in FROM clause, treat as a temporary table

Example: What is the highest rated beer, on average?

```
SELECT a.BEER
FROM (SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
      FROM RATES r
      GROUP BY BEER) a
WHERE a.AVG_RATING = (SELECT MAX (a.AVG_RATING)
                      FROM (SELECT r.BEER, AVERAGE (r.SCORE)
                            FROM RATES r
                            GROUP BY BEER) a)
```

Subquery in FROM

RATES (DRINKER, BEER, SCORE)

Note: a lot cleaner with a view!

```
CREATE VIEW AGG AS
SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY BEER

SELECT a.BEER
FROM AGG a
WHERE a.AVG_RATING = (SELECT MAX (a.AVG_RATING)
                      FROM AGG a)
```

Top k

RATES (DRINKER, BEER, SCORE)

Example: What is the highest rated beer, on average?

Actually, can be a lot easier with top k.

```
CREATE VIEW AGG AS
SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY BEER
```

Top k

RATES (DRINKER, BEER, SCORE)

Example: What is the highest rated beer, on average?

Actually, can be a lot easier with top k.

```
CREATE VIEW AGG AS
SELECT r.BEER, AVERAGE (r.SCORE) AS AVG_RATING
FROM RATES r
GROUP BY BEER

SELECT TOP (1) a.BEER
FROM AGG a
ORDER BY a.AVG_RATING DESC;
```

Top k

RATES (DRINKER, BEER, SCORE)

Example: What is the highest rated beer, on average?

Actually, can be a lot easier with top k.

- Can optionally use the PERCENT keyword
- Can add WITH TIES
- Can choose ASC or DESC
- Finally: note that ORDER BY can be used without TOP

Questions?