

Recap

- Data-savviness is the future!
- Notion of a DBMS
- The relational data model and algebra: bags and sets
- SQL queries:
 - SFW and its semantics
 - LIKE, AS, *, %, ...
 - Null values
 - Single and multiple relations
 - Subqueries
 - Bag algebra and set operations
 - Next: grouping



Aggregations

- **SUM, MAX, MIN, COUNT, AVG** can be applied to a column in a **SELECT** clause to produce that aggregation
- **COUNT (*)** is a special syntax to count number of tuples
- Number of films:
 - **SELECT COUNT(*) FROM Film**
- Max and average film lengths:
 - **SELECT MAX(length), AVG (length) FROM Film**



Removing Duplicates

- Adding **DISTINCT** removes duplicates prior to aggregation
- Count the number of distinct lengths in the Film relation:
 - **SELECT COUNT(DISTINCT length) FROM Film**



NULL values

- NULL values are not involved in aggregations
- But if there are no non-NULL values, result will be a NULL
- Number of movies
 - `SELECT COUNT (*) FROM Film;`
- Number of movies with a non-null length
 - `SELECT COUNT (length) FROM Film;`



Grouping

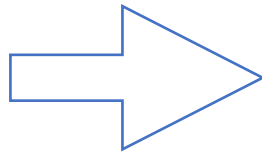
- In some cases, we may want to compute an aggregate for each “group” of tuples as opposed to an overall **COUNT**, **MAX** or **SUM**
- You do so by adding a **GROUP BY** clause after **SELECT FROM WHERE**
 - The results of SFV are then grouped according to the grouping attributes, and aggregation is applied per group
 - Example: find average and minimum length for each movie rating
SELECT rating, AVG(length) AS avgl, MIN (length) as minl
FROM Film GROUP BY rating;



Visualizing Grouping

`SELECT rating,AVG(length) AS avgl, MIN (length) AS minl
FROM Film GROUP BY rating;`

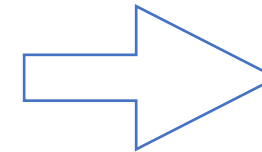
title	rating	length
...	G	...
...	PG	...
...	PG-13	...
...	PG	...
...	G	...



title	rating	length
...	G	...
...	G	...

title	rating	length
...	PG	...
...	PG	...

title	rating	length
...	PG-13	...



rating	avgl	minl
G
PG
PG-13



Grouping (contd.)

- Relations:
 - Film_actor (actor_id, film_id, last_update),
 - Actor (actor_id, first_name, last_name, last_update),
 - Film (film_id, ...)
- Q: Count the number of movies per actor



Grouping (contd.)

- Relations:
 - Film_actor (actor_id, film_id, last_update),
 - Actor (actor_id, first_name, last_name, last_update)
- Q: Count the number of movies per actor

```
SELECT Actor.first_name, Actor.last_name, COUNT(*)  
FROM Actor, Film_actor  
WHERE Actor.actor_id = Film_actor.actor_id  
GROUP BY Actor.first_name, Actor.last_name
```



Restriction of SELECT list with aggregation

- If aggregation is used, then each element of the **SELECT** clause must either be:
 - An aggregate, or
 - An attribute in the **GROUP BY** list
- Q: Why this restriction?
 - If an attribute is not being aggregated or being grouped, then you need some way to “squish” the values down per group.



Exercise

- Film (film_id, title, release_year, length, ...)
- Q: Find the movie with the maximum length



Exercise

- Film (film_id, title, release_year, length, ...)
- Q: Find the movie with the maximum length

```
SELECT FI.title, FI.length
```

```
FROM Film AS FI
```

```
WHERE FI.length >= (SELECT MAX (length) FROM Film)
```



HAVING Clauses

- **HAVING** <condition> may follow a **GROUP BY** clause
- If so, the condition is applied to each group, and groups not satisfying the condition are eliminated
- Example:
- Show actors who have starred in at least 30 movies

```
SELECT Actor.first_name, Actor.last_name, COUNT(*)  
FROM Film, Film_actor  
WHERE Actor.actor_id = Film_actor.actor_id  
GROUP BY Actor.first_name, Actor.last_name  
HAVING COUNT (*) > 30
```



Restrictions on HAVING Clauses

- Similar to **SELECT** clauses: each attribute mentioned must either be part of the **GROUP BY** or be aggregated



General Form of GROUPING

```
SELECT S  
FROM R1, R2, ...  
WHERE C1  
GROUP BY A1, A2, ...  
HAVING C2
```

- **S** and **C2** can contain **A1, A2, ...** or any other aggregated attributes
- **C1**: any condition



General Form of GROUPING

```
SELECT S  
FROM R1, R2, ...  
WHERE C1  
GROUP BY A1, A2, ...  
HAVING C2
```

- Order of evaluation:
 - Compute the “FROM-WHERE” part:
 - For each combination of tuples in the cross product of R1, R2, ...
 - Keep only those tuples that satisfy C1
 - Group by A1, A2, ...
 - For each group, check if C2 is satisfied
 - If so: compute aggregates in S and add to output



Recap

- Data-savviness is the future!
- Notion of a DBMS
- The relational data model and algebra: bags and sets
- SQL queries:
 - SFW and its semantics
 - LIKE, AS, *, %, ...
 - Null values
 - Single and multiple relations
 - Subqueries
 - Bag algebra and set operations
 - Grouping
 - Next: Ordering



Ordering: ORDER BY

- Use an **ORDER BY** clause to enforce ordering of the result
- **ORDER BY <attr> ASC | DESC**

- Order films by descending length:

```
SELECT title, length
```

```
FROM Film
```

```
ORDER BY length DESC;
```

- Order films by descending length, then title ascending

```
SELECT title, length
```

```
FROM Film
```

```
ORDER BY length DESC, title ASC;
```



Restrict output: LIMIT & OFFSET

- Sometimes you want to limit the result to a few tuples
- **LIMIT k**
 - Order films by descending length, ascending title, return top 15
 - **SELECT title, length FROM Film ORDER BY length DESC, title ASC LIMIT 15**
- And sometimes you want to start the output at a particular point
- **OFFSET k**
 - Order films by descending length, ascending title, return from position 11 to 15
 - **SELECT title, length FROM Film ORDER BY length DESC, title ASC LIMIT 5 OFFSET 10**



Recap

- Data-savviness is the future!
- Notion of a DBMS
- The relational data model and algebra: bags and sets
- SQL queries:
 - SFV and its semantics
 - LIKE, AS, *, %, ...
 - Null values
 - Single and multiple relations
 - Subqueries
 - Bag algebra and set operations
 - Grouping
 - Ordering
 - Next: Case



Case Statements

- **CASE** statements allow you to create new derived data in the select clause
- Suppose you want to categorize movies into long, medium, and short based on length

```
SELECT title, CASE
```

```
    WHEN length > 180 THEN 'long'
```

```
    WHEN length > 120 AND length <= 180 THEN 'medium'
```

```
    WHEN length < 120 AND length >= 60 THEN 'short'
```

```
    ELSE 'super short'
```

```
END AS type
```

```
FROM Film
```



Quick Demos

- Number of films:

```
SELECT COUNT(*) FROM Film
```

- Max and average film lengths:

```
SELECT MAX(length),AVG (length) FROM Film
```

- Count the number of distinct lengths in the Film relation:

```
SELECT COUNT(DISTINCT length) FROM Film
```

- Example: find average and minimum length for each movie rating

```
SELECT rating,AVG(length) AS avgl, MIN (length) as minl  
FROM Film GROUP BY rating;
```

- Show actors who have starred in at least 30 movies

```
SELECT Actor.first_name,Actor.last_name, COUNT(*)  
FROM Actor, Film_actor  
WHERE Actor.actor_id = Film_actor.actor_id  
GROUP BY Actor.first_name,Actor.last_name  
HAVING COUNT (*) > 30
```



Quick Demos (contd.)

- Order films by descending length:
`SELECT title, length FROM Film ORDER BY length DESC;`
- Order films by descending length, then title ascending
`SELECT title, length FROM Film ORDER BY length DESC, title ASC;`
- Order films by descending length, ascending title, return top 15
`SELECT title, length FROM Film ORDER BY length DESC, title ASC LIMIT 15`
- Order films by descending length, ascending title, return from position 11 to 15
`SELECT title, length FROM Film ORDER BY length DESC, title ASC LIMIT 5 OFFSET 10`
- Suppose you want to categorize movies into long, medium, and short based on length
`SELECT title, CASE
 WHEN length > 180 THEN 'long'
 WHEN length > 120 AND length <= 180 THEN 'medium'
 WHEN length < 120 AND length >= 60 THEN 'short'
 ELSE 'super short'
END AS type
FROM Film`

