Lab8 Grouping & Aggregates, Subqueries, Conditional Logic

Lab8 Agenda

- Grouping & Aggregates
- Subqueries
- Conditional Logic
- Εργαστηριακές Ασκήσεις
- Εξαμηνιαία Εργασία

Grouping & Aggregates

Aggregate Functions

- max()
- min()
- avg()
- sum()
- count()

```
SELECT MAX(amount) max,  MIN(amount) min,  AVG(amount) avg,
SUM(amount) sum,  COUNT(*) num FROM payment; /* no group by clause */
```

```
+-------+------+----------+----------+-------+
| max   | min  | avg      | sum      | num   |
+-------+------+----------+----------+-------+
| 11.99 | 0.00 | 4.200667 | 67416.51 | 16049 |
+-------+------+----------+----------+-------+
```

- implicit group (all rows in the payment table)

Explicit Groups

```
SELECT customer_id, MAX(amount) max,  MIN(amount) min,  AVG(amount) avg,
SUM(amount) sum,  COUNT(*) num  FROM payment
GROUP BY customer_id;
```

```
+-------------+-------+------+----------+--------+-----+
| customer_id | max   | min  | avg      | sum    | num |
+-------------+-------+------+----------+--------+-----+
|           1 |  9.99 | 0.99 | 3.708750 | 118.68 |  32 |
|           2 | 10.99 | 0.99 | 4.767778 | 128.73 |  27 |
                              ...
|         598 |  7.99 | 0.99 | 3.808182 |  83.78 |  22 |
|         599 |  9.99 | 0.99 | 4.411053 |  83.81 |  19 |
+-------------+-------+------+----------+--------+-----+
```

- group together rows having the same value in the customer_id column

Single-Column Grouping

- number of films associated with each actor

```
SELECT actor_id, count(*)  FROM film_actor  GROUP BY actor_id;
```

```
+----------+----------+
| actor_id | count(*) |
+----------+----------+
|        1 |       19 |
|        2 |       25 |
|        3 |       22 |
           ...
```

Multicolumn Grouping

- groups that span more than one column
  - number of films associated with each actor for each film rating (G, PG, ...)

```
SELECT fa.actor_id, f.rating, count(*) FROM film_actor fa
INNER JOIN film f  ON fa.film_id = f.film_id
GROUP BY fa.actor_id, f.rating
ORDER BY fa.actor_id, f.rating;
```

```
+----------+--------+----------+
| actor_id | rating | count(*) |
+----------+--------+----------+
|        1 | G      |        4 |
|        1 | PG     |        6 |
|        1 | PG-13  |        1 |
|        1 | R      |        3 |
|        1 | NC-17  |        5 |
|        2 | G      |        7 |
|        2 | PG     |        6 |
          ...
```

Grouping via Expressions

```
SELECT extract(YEAR FROM rental_date) sel_year,  COUNT(*) how_many
FROM rental  GROUP BY sel_year;
```

```
+----------+----------+
| sel_year | how_many |
+----------+----------+
|     2005 |    15862 |
|     2006 |      182 |
+----------+----------+
```

- `help extract;`

Group Filter Conditions

- where the filter acts
  - on raw data, it belongs in the where clause
  - on grouped data, it belongs in the having clause

```
SELECT fa.actor_id, f.rating, count(*)  FROM film_actor fa
INNER JOIN film f  ON fa.film_id = f.film_id
WHERE f.rating IN ('G','PG')
GROUP BY fa.actor_id, f.rating
HAVING count(*) > 9;
```

```
+----------+--------+----------+
| actor_id | rating | count(*) |
+----------+--------+----------+
|        7 | G      |       10 |
|       14 | G      |       10 |
|       17 | G      |       12 |
|       26 | PG     |       11 |
```

Subqueries

Subquery

- query contained within another SQL statement

```
SELECT customer_id, first_name, last_name FROM customer
WHERE customer_id = (SELECT MAX(customer_id) FROM customer);
```

```
+-------------+------------+-----------+
| customer_id | first_name | last_name |
+-------------+------------+-----------+
|         599 | AUSTIN     | CINTRON   |
+-------------+------------+-----------+
```

Subquery Types

- result set returned by a subquery
  - single row/column, single row/multicolumn, or multiple columns
- completely self-contained (**noncorrelated** subqueries)
- reference columns from the containing statement (**correlated** subqueries)

Noncorrelated Subqueries

- a scalar subquery

```
SELECT city_id, city  FROM city WHERE country_id <>
(SELECT country_id FROM country WHERE country = 'India');
```

```
+---------+----------------------------+
| city_id | city                       |
+---------+----------------------------+
|       1 | A Corua (La Corua)         |
|       2 | Abha                       |
           ...
|     600 | Ziguinchor                 |
+---------+----------------------------+
```

Multiple-Row, Single-Column Subqueries

- in and not in operators
  - an expression can (not) be found within a set of expressions
- find all cities that are not in Canada or Mexico

```
SELECT city_id, city FROM city
WHERE country_id NOT IN
 (SELECT country_id FROM country  WHERE country IN ('Canada','Mexico'));
```

```
+---------+----------------------------+
| city_id | city                       |
+---------+----------------------------+
|       1 | A Corua (La Corua)         |
|       2 | Abha                       |
|       3 | Abu Dhabi                  |
```

Multiple-Row, Single-Column Subqueries

- all operator
  - make comparisons between a single value and every value in a set
- finds all customers who have never gotten a free film rental

```
SELECT first_name, last_name FROM customer
WHERE customer_id <> ALL
(SELECT customer_id  FROM payment  WHERE amount = 0);
```

```
+-------------+--------------+
| first_name  | last_name    |
+-------------+--------------+
| MARY        | SMITH        |
| PATRICIA    | JOHNSON      |
| LINDA       | WILLIAMS     |
```

Multiple-Row, Single-Column Subqueries

- any operator
  - make comparisons between a single value and every value in a set
- find all customers whose total film rental payments exceed the total payments for all customers in Bolivia, Paraguay, or Chile 😬

Multiple-Row, Single-Column Subqueries 😫

```
SELECT customer_id, sum(amount)  FROM payment  GROUP BY customer_id
HAVING sum(amount) > ANY
(SELECT sum(p.amount) FROM payment p
INNER JOIN customer c  ON p.customer_id = c.customer_id
INNER JOIN address a  ON c.address_id = a.address_id
INNER JOIN city ct  ON a.city_id = ct.city_id
INNER JOIN country co ON ct.country_id = co.country_id
WHERE co.country IN ('Bolivia','Paraguay','Chile')
GROUP BY co.country );
```

```
+-------------+-------------+
| customer_id | sum(amount) |
+-------------+-------------+
|         137 |      194.61 |
|         144 |      195.58 |
|         148 |      216.54 |
|         178 |      194.61 |
|         459 |      186.62 |
|         526 |      221.55 |
+-------------+-------------+
```

Conditional Logic

Case Statement

```
CASE
WHEN condition1 THEN result1
WHEN condition2 THEN result2
WHEN conditionN THEN resultN
ELSE result
END;
```

Usage

```
SELECT first_name, last_name,
CASE
  WHEN active = 1
  THEN 'ACTIVE'
  ELSE 'INACTIVE'
END
activity_type FROM customer;
```

```
| first_name  | last_name     | activity_type |
+-------------+---------------+---------------+
| MARY        | SMITH-ALLEN2  | ACTIVE        |
| AUSTIN      | CINTRON       | ACTIVE        |
| NICK        | THE GREEK     | INACTIVE      |
+-------------+---------------+---------------+
```

Schenarios

- Result Set Transformations
- Checking for Existence
- Division-by-Zero Errors
- Handling Null Values

Result Set Transformations

```
SELECT monthname(rental_date) rental_month, count (*) num_rentals
FROM rental
WHERE rental_date BETWEEN '2005-05-01' AND '2005-08-01'
GROUP BY monthname(rental_date);
```

```
| rental_month | num_rentals |
+--------------+-------------+
| July         |        6709 |
| June         |        2311 |
| May          |        1156 |
+--------------+-------------+
```

Result Set Transformations

```
SELECT SUM ( CASE WHEN monthname(rental_date) ='May' THEN 1 ELSE 0 END ) May_rentals,
SUM ( CASE WHEN monthname(rental_date) = 'June' THEN 1 ELSE 0 END ) June_rentals,
SUM ( CASE WHEN monthname(rental_date) = 'July' THEN 1 ELSE 0 END ) July_rentals
FROM rental WHERE rental_date  BETWEEN '2005-05-01' AND '2005-08-01' ;
```

```
| May_rentals | June_rentals | July_rentals |
+-------------+--------------+--------------+
|        1156 |         2311 |         6709 |
+-------------+--------------+--------------+
```

Checking for Existence

```
SELECT a.first_name,
a.last_name,  CASE
WHEN EXISTS ( SELECT 1 FROM film_actor fa INNER JOIN film f ON fa.film_id = f.film_id WHERE fa.actor_id = a.actor_id AND f.rating='G' )
THEN 'Y'
ELSE 'N'
END
g_actor, CASE
WHEN EXISTS ( SELECT 1 FROM film_actor fa INNER JOIN film f ON fa.film_id = f.film_id WHERE fa.actor_id = a.actor_id AND f.rating = 'PG')
THEN 'Y'
ELSE 'N'
END
pg_actor, CASE
WHEN EXISTS ( SELECT 1 FROM film_actor fa INNER JOIN film f ON fa.film_id = f.film_id WHERE fa.actor_id = a.actor_id AND  f.rating = 'NC-17' )
THEN 'Y'
ELSE 'N'
END
nc17_actor
FROM actor a WHERE a.last_name LIKE 'S%' OR a.first_name LIKE 'S%' ;
```

Checking for Existence

first_name	last_name	g_actor	pg_actor	nc17_actor
JOE	SWANK	Y	Y	Y
SANDRA	KILMER	Y	Y	Y
CAMERON	STREEP	Y	Y	Y
SANDRA	PECK	Y	Y	Y
SISSY	SOBIESKI	Y	Y	N
NICK	STALLONE	Y	Y	Y
SEAN	WILLIAMS	Y	Y	Y
...				
JOHN	SUVARI	Y	Y	Y
JAYNE	SILVERSTONE	Y	Y	Y

Checking for Existence

```
SELECT f.title, CASE (SELECT count (*) FROM inventory i WHERE i.film_id  = f.film_id)
WHEN 0 THEN 'Out Of Stock'
WHEN 1 THEN 'Scarce'
WHEN 2 THEN 'Scarce'
WHEN 3 THEN 'Available'
WHEN 4 THEN 'Available'
ELSE 'Common'
END
film_availability FROM film f;
```

```
| title                        | film_availability |
+------------------------------+-------------------+
| ACADEMY DINOSAUR             | Common            |
| ACE GOLDFINGER               | Available         |
| ADAPTATION HOLES             | Available         |
| AFFAIR PREJUDICE             | Common            |
```

Conditional Updates

```
UPDATE customer SET active =
CASE
WHEN 90 <= ( SELECT datediff(now(), max (rental_date)) FROM rental r
WHERE r.customer_id = customer.customer_id)
THEN 0
ELSE 1
END
WHERE active = 1;
```

Εργαστηριακές Ασκήσεις 💻

1. Use subqueries to display the titles of movies starting with the letters K and Q whose language is English.
2. List customer ID, number of payments made and the total amount paid for each customer for customers with more than 30 payments.
3. List the most frequently rented movies in descending order.
4. Write a query to display how much money, each store brought in. Compare your results with the sales_by_store view.
5. List the last names of actors, as well as how many actors have that last name.
6. List the top five genres in revenue in descending order.
7. List the average length of films by category.
8. Which film categories are longer (on average than the average length of films)?
9. Which actor has appeared in the most films?

Εξαμηνιαία Εργασία 💻

- Database Schema Design
  1. Start thinking about the entities you need
     - Identify entities, attributes and relationships from the problem description
     - identify cardinality ratios of the relationships found
  2. Design an E/R diagram for your database
     - Look for any issues that are apparent in the E/R diagram

Εξαμηνιαία Εργασία 💻

- Materialize Schema: DDL statements
  1. Create your tables
     - create a table for each entity
     - a table (representing an entity) should have:
       - a column for each attribute, with appropriate data type
       - a primary key and possibly some candidate keys
     - include a foreign key (one-to-many relationships)
     - add indexes & constraints to your tables
  2. - Create views as needed
     - Create triggers for your tables
     - Create Stored Procedures & Functions for your application

Εξαμηνιαία Εργασία 💻

- Add Information to the Database: DML script
  - Populate the database with data
  - Write needed queries
  - Test and adapt offered functionality
- Finetune tables, queries, views, triggers, stored procedures & functions.

“ hint: start running your SQL commands from a separate file. This makes it much easier to alter and change your SQL code ”

Wrap Up

1. [x] `Grouping & Aggregates`
2. [x] `Subqueries`
3. [x] `Conditional Logic`
4. [x] `Εργαστηριακές Ασκήσεις`
5. [x] `Εξαμηνιαία Εργασία`

Wrap Up

😱 **Απορίες** https://discord.gg/g3fFxWVPfD

Εργαστηριακές Ασκήσεις / Απαντήσεις

1. Use subqueries to display the titles of movies starting with the letters K and Q whose language is English.

```
select f.title  from film as f
where f.language_id = (select language_id from language where name = 'English')
and f.title like 'K%' or 'Q%' ;
```

```
select f.title  from film as f  join language as l
on f.language_id = l.language_id
where f.title like 'K%' or 'Q%' and l.name = 'English';
```

Εργαστηριακές Ασκήσεις / Απαντήσεις

2. List customer ID, number of payments made and the total amount paid for each customer for customers with more than 30 payments.

```
SELECT customer_id, count(*), sum(amount) FROM payment GROUP BY customer_id  HAVING count(*) >= 30;
```

```
+-------------+----------+-------------+
| customer_id | count(*) | sum(amount) |
+-------------+----------+-------------+
|           1 |       32 |      118.68 |
|           5 |       38 |      144.62 |
|           7 |       33 |      151.67 |
```

Εργαστηριακές Ασκήσεις / Απαντήσεις

3. List the most frequently rented movies in descending order.

```
select f.title as 'Movie', count(r.rental_date) as 'Times Rented'
from film as f
join inventory as i on i.film_id = f.film_id
join rental as r on r.inventory_id = i.inventory_id
group by f.title
order by count(r.rental_date) desc;
```

Εργαστηριακές Ασκήσεις / Απαντήσεις

4. Write a query to display how much money, each store brought in. Compare your results with the sales_by_store view.

```
select concat(c.city,', ',cy.country) as `Store`, sum(p.amount) as `Total Sales`
from payment as p
join rental as r on r.rental_id = p.rental_id
join inventory as i on i.inventory_id = r.inventory_id
join store as s on s.store_id = i.store_id
join address as a on a.address_id = s.address_id
join city as c on c.city_id = a.city_id
join country as cy on cy.country_id = c.country_id
group by s.store_id;
```

```
select store as 'Store', total_sales as 'Total Sales' from sales_by_store;
```

Εργαστηριακές Ασκήσεις / Απαντήσεις

5. List the last names of actors, as well as how many actors have that last name.

```
select last_name, count(*) actor_count
from actor
group by last_name
order by actor_count desc, last_name;
```

Εργαστηριακές Ασκήσεις / Απαντήσεις

6. List the top five genres in revenue in descending order.5

```
select c.name as 'Film', sum(p.amount) as 'Gross Revenue'
from category as c
join film_category as fc on fc.category_id = c.category_id
join inventory as i on i.film_id = fc.film_id
join rental as r on r.inventory_id = i.inventory_id
join payment as p on p.rental_id = r.rental_id
group by c.name
order by sum(p.amount) desc
limit 5;
```

Εργαστηριακές Ασκήσεις / Απαντήσεις

7. List the average length of films by category.

```
select category.name, avg(length)
from film join film_category using (film_id) join category using (category_id)
group by category.name
order by avg(length) desc;
```

Εργαστηριακές Ασκήσεις / Απαντήσεις

8. Which film categories are longer (on average than the average length of films)?

```
select category.name, avg(length)
from film join film_category using (film_id) join category using (category_id)
group by category.name
having avg(length) > (select avg(length) from film)
order by avg(length) desc;
```

Εργαστηριακές Ασκήσεις / Απαντήσεις

9. Which actor has appeared in the most films?

```
SELECT a.actor_id, a.first_name, a.last_name, count(*) as film_count
FROM actor a
INNER JOIN film_actor fa  ON a.actor_id= fa.actor_id
GROUP BY a.actor_id
ORDER BY film_count DESC
limit 1;
```

```
+----------+------------+-----------+------------+
| actor_id | first_name | last_name | film_count |
+----------+------------+-----------+------------+
|      107 | GINA       | DEGENERES |         42 |
+----------+------------+-----------+------------+
1 row in set (0.002 sec)
```