Databases

BCS1510

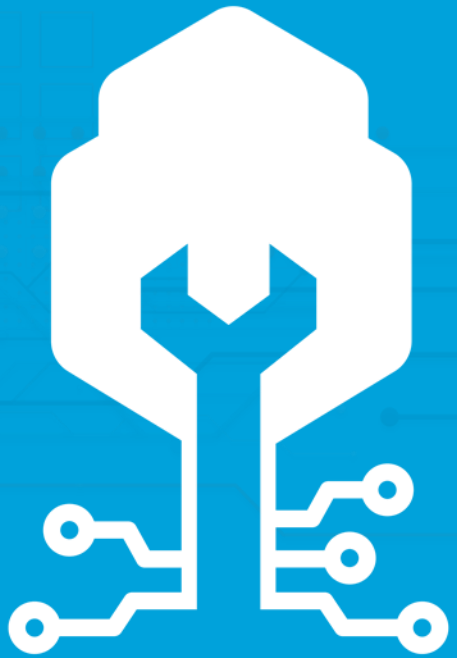
Dr. Katharina Schneider &
Dr. Tony Garnock-Jones*

🗓️ Week 2 - Lecture 2

📍 EPD150 MSM Conference Hall

Adapted with gratitude from the original lectures by Dr. Ashish Sai

*CC both of us when emailing please!



Maastricht University

Open question from last lecture

- What does DEFAULT do in MySQL?
  - Setting a default value in create table:

```
CREATE TABLE students (
      id INT primary key,
      name VARCHAR(100),
      grade INT DEFAULT 10);
```

  - Insert a default value (10 is added):

```
INSERT INTO students (id, name, grade)
 VALUES (2, 'Bob', DEFAULT);
```

What have we looked at so far?

- Database
  - Collection of logically related data
  - Two formal definitions
- Database Management System
  - Program used for defining and maintaining this database.
  - DBMS Evolution
  - From file systems to blockchains and big data.
- Data Models
  - Conceptual and Physical Data Models
- Relational Data Model
  - An overview of entity and relationship model
- Relational Algebra
  - Introduction to Relational Algebra.
  - Understanding the foundations of relational algebra as it applies to operations withing SQL queries
- SQL as data definition language
  - Creating and altering tables
- SQL as data manipulation language
  - Singel-relation and multi-relation queries, three-valued logic, joins, patterns, constants, tuple-variables (self-join), NULL values

Learning Objectives

- Understand the Role and Structure of Subqueries:
  - Learn how subqueries function as a tool in SQL to perform nested operations, where they can be used in SELECT, FROM and WHERE clauses to refine data retrieval.
- Master SQL Aggregations and Grouping
  - Gain proficiency in using SQL aggregate functions like `SUM`, `AVG`, `COUNT`, `MIN`, and `MAX` to perform calculations over sets of data,
  - and understand how these aggregations can be used with `GROUP BY` clauses to organize data into meaningful groups
- Apply Advanced SQL Techniques:
  - Develop skills to implement complex SQL queries involving the use of subqueries, the `IN` and `EXISTS` operators, and conditions applying to aggregates using `HAVING` clauses.
- Practical Application and Problem Solving:
  - Apply learned SQL techniques to solve specific problems such as identifying unique customer attributes, calculating total values, and effectively using SQL syntax to manage data involving conditions and joins.

SQL (Part 3)

SQL as Data Manipulation Language

Subqueries

Subquery

- A parenthesized SELECT-FROM-WHERE statement (subquery) can be used as a value in a number of places, including in FROM and WHERE clauses

- Also in the SELECT clause, a subquery is possible

Subquery in SELECT

Subquery in SELECT statement

- We saw already expressions in the SELECT clause:

```
SELECT model,price,price*1.08 AS priceUSD
FROM PCs
```

- The value 1.08 could also be a value gotten from another relation

Subquery in SELECT clause

- Imagine a relation “currencies” in addition to our pc shop example:

Currency	Conversion_factor
USD	1.08
EUR	1
GBP	0.84

Subquery in SELECT clause

- Instead of:

```
SELECT model,price,price*1.08 AS priceUSD
FROM PCs
```

- We can query:

```
SELECT model,price,price * (SELECT conversion_factor
      FROM currencies WHERE currency='USD') AS
      priceUSD
FROM PCs
```

Subquery in FROM

Subquery in FROM clause

- Instead of relation in FROM clause: use subquery
- Must use a tuple-variable (alias) to name tuples of the result
- Think about it when seeing the error “every derived table must have its own alias”

Remember tuple-variable:

```
SELECT c1.customer_id, c2.customer_id
FROM customers c1, customers c2
WHERE c1.city = c2.city AND c1.customer_id < customer_id;
```

Example: Subquery in FROM clause

- From Products and PCs, find the PC models made by maker A and with speed at least 2.0

```
SELECT model
FROM Products JOIN PCs USING (model)
WHERE maker = 'A'  AND speed >= 2.0;
```

- Or alternatively:

```
SELECT model
FROM Products JOIN (SELECT model FROM PCs WHERE speed >=
     2.0) atleast2 USING (model)
WHERE maker = 'A';
```

atleast2: Tuple Variable

Subquery in WHERE

Subquery in WHERE clause

- If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value (also in the SELECT clause)

- A run-time error occurs if there is no tuple or more than one tuple.

Example: Subquery in WHERE clause

- Q: Using customers, find the name of all customers from Limerick who have the same lastname as the customer with ID 1111111111

```
SELECT firstname, lastname
FROM customers
WHERE city = 'Limerick'  AND lastname = ???;
```

Example: Subquery in WHERE statement

- Q: Using customers, find the name of all customers from Limerick who have the same lastname as the customer with ID 1111111111

```
SELECT firstname, lastname
FROM customers
WHERE city = 'Limerick'  AND lastname = (SELECT lastname
      FROM customers WHERE customer_id = '1111111111');
```

Operators in Subqueries

IN / NOT IN Operator

- <tuple> IN (<subquery>)
- is true iff the tuple is a member of the relation produced by the subquery
- Opposite: <tuple> NOT IN (<subquery>)
- IN-expressions appear in WHERE clauses

Example: IN Operator

- Q: From Products and PCs find the PC models made by maker B and with speed at least 2.0 and at most 2.8

```
SELECT model
FROM Products
WHERE maker = 'B'  AND model IN (SELECT model FROM PCs
      WHERE speed >= 2.0 AND speed <=2.8)
```

- Alternatively:

```
SELECT model
FROM Products
WHERE maker = 'B'  AND model IN (SELECT model FROM PCs
      WHERE speed BETWEEN 2.0 AND 2.8)
```

Example: IN Operator

- Why can’t we use:

```
SELECT model
FROM Products
WHERE maker = 'B'  AND model = (SELECT model FROM PCs
      WHERE speed >= 2.0 AND speed <=2.8)
```

Example: IN Operator

- Why can't we use:

```
SELECT model
FROM Products
WHERE maker = 'B'  AND model = (SELECT model FROM PCs
      WHERE speed >= 2.0 AND speed <=2.8)
```

- Subquery returns more than 1 row
- Comparison only works to one value

Remember, last lecture

- Q2: Retrieve all customer ID and full name of customers who have made at least one purchase

- We solved it using a join:

```
SELECT DISTINCT c.customer_id, c.firstname, c.lastname
FROM Customers c
JOIN Sales s ON c.customer_id = s.customer_id;
```

- Can we solve it using the IN operator?

Remember, last lecture

- Q2: Retrieve all customer ID and full name of customers who have made at least one purchase

- Using IN operator:

```
SELECT customer_id, firstname, lastname
FROM Customers
WHERE customer_id IN (SELECT DISTINCT customer_id FROM
      Sales);
```

EXISTS Operator

- EXISTS (<subquery>)

- is true if and only if the subquery result is not empty

Example: EXISTS Operator

- Q: From Customers, find the IDs of those customers whose first names are unique, i.e., no other customers has the same first name

```
SELECT customer_id
FROM Customers c1
WHERE NOT EXISTS (
      SELECT *
      FROM Customers
      WHERE firstname = c1.firstname AND customer_id <> c1.customer_id);
```

ANY Operator

- X = ANY (<subquery>)
- is a Boolean condition that is true if and only if X equals at least one tuple in the subquery result
- = could be any standard comparison operator (=, <>, !=, >, >=, <, or <=)

ALL Operator

- X<> ALL (<subquery>)
- is a Boolean condition that is true if and only if X is different from all tuples in the subquery result
- <> could be any standard comparison operator (=, <>, !=, >, >=, <, or <=)

Example: ALL Operator

- Q: From PCs find the fastest PC model(s) with hd less than or equal to 250

```
SELECT model
FROM PCs
WHERE hd <= 250 AND speed >= ALL (SELECT speed
      FROM PCs
      WHERE hd <= 250);
```

Short Break

- **10** minutes

SQL (Part 4)

Aggregations and Grouping

Aggregate Functions

- For example: sum, avg, count, min, max
- They can be applied to a single attribute in a SELECT clause to produce that aggregation on the attribute
- Also, count(*) counts the number of tuples
- Aggregations are NOT allowed in the WHERE clause (outside of subqueries)

Example Aggregation

- Q: From PCs find the average speed of PC models with hd less than or equal to 250.

```
SELECT AVG (speed)
FROM PCs
WHERE hd <= 250;
```

Example – Illegal Use of Aggregation

- Q: From PCs find the fastest PC model(s) with hd less than or equal to 250.

- Intuitive would be:

```
SELECT model
FROM PCs
WHERE hd <= 250 AND speed = MAX(speed);
```

- BUT IT DOES NOT WORK (no aggregate functions in WHERE clause outside of subqueries!!)

Example – Illegal Use of Aggregation

- Q: From PCs find the fastest PC model(s) with hd less than or equal to 250.

- Maybe, intuitive would be:

```
SELECT model, MAX(speed)
FROM PCs
WHERE hd <= 250;
```

- BUT IT DOES NOT WORK (model has 11 tuples, MAX(speed) only 1)

Example – Any ideas of what would work?

- Q: From PCs find the fastest PC model(s) with hd less than or equal to 250.

Example – Correct Solution

- Q: From PCs find the fastest PC model(s) with hd less than or equal to 250.

```
SELECT model
FROM PCs
WHERE hd <= 250 AND speed =
      (SELECT MAX(speed)
      FROM PCs
      WHERE hd <= 250)
```

- Returns multiple PCs if there are multiple that have hd <= 250 and speed = 3.2

Eliminating Duplicates in Aggregations

- Use DISTINCT inside an aggregation

- Example:
- Q: find the number of different RAM sizes of PC models

```
SELECT COUNT (DISTINCT ram)
FROM PCs
```

Aggregations: What happens will NULL values?

- NULL never contributes to a sum, average, or count
- NULL can never be the minimum or maximum of a column
- If there are no non-NULL values in a column, then the result of the aggregation is NULL
  - Exception: COUNT of an empty set is 0

Example: Effect of NULL

```
SELECT COUNT (*)
FROM Customers
```

- Gives you the number of rows in table Customers (5)

```
SELECT COUNT (email)
FROM Customers
```

- Gives you the number of rows in table Customer with non-NULL email (3)

Grouping

Grouping

- Done with GROUP BY clause following the SELECT-FROM-WHERE expression
- The relation that results from the SELECT-FROM-WHERE is grouped according to the values of all those attributes
- Aggregations are then applied only within each group

Example: Grouping

- Q: From Products(maker, model, type) find the total number of laptops per maker

```
SELECT COUNT(*)
FROM Products
WHERE type = 'laptop'
GROUP BY maker;
```

- What result would you expect?

Count(*)
3
1
3
2
1

Example: Grouping

- Q: From Products(maker, model, type) find the total number of laptops per maker

```
SELECT COUNT(*)
FROM Products ORDER BY maker
WHERE type = 'laptop'
GROUP BY maker;
```

- But the result is:

Count(*)
3
3
1
2
1

WHAAAAT?

- Ordering the resulting tuples is not guaranteed

- So, what you can do is:

```
SELECT COUNT(*)
FROM Products
WHERE type = 'laptop'
GROUP BY maker
ORDER BY maker;
```

Count(*)
3
1
3
2
1

LIMIT

- Used to restrict the number of rows returned by a query

```
SELECT column1, column 2
FROM table_name
LIMIT N;
```

ORDER BY + LIMIT + OFFSET

- OFFSET makes that a certain number of rows are skipped

- Q: Get the 2nd most expensive laptop

```
SELECT *
FROM laptops
ORDER BY price DESC
LIMIT 1 OFFSET 1;
```

Grouping: Restriction on SELECT

- If grouping is used, then each element of the SELECT list must be either

  - Aggregated

  OR

  - An attribute on the GROUP BY list

GROUPING + HAVING

- HAVING <condition> may follow a GROUP BY clause

- If so, the condition applies to each group, and groups not satisfying the condition are eliminated

Example 1 HAVING

- Q: From sales find the customers who have spent more than 2000 in total. List the ID and the total sum paid for each of them.

```
SELECT customer_id, SUM(paid)
FROM Sales
GROUP BY customer_id
HAVING SUM(paid) > 2000;
```

Example 2 HAVING

- Q: From sales and customer find the customers from Limerick who have spent more than 2000 in total. List the ID and the total sum paid for each of them.

- Suggestions?

Example 2 HAVING – Solution 1

- Q: From sales and customer find the customers from Limerick who have spent more than 2000 in total. List the ID and the total sum paid for each of them.

```
SELECT customer_id, SUM(paid)
FROM Sales
GROUP BY customer_id
HAVING SUM(paid) > 2000
      AND customer_ID in
          (SELECT customer_id
           FROM customers
           WHERE city = 'Limerick');
```

Example 2 HAVING – Solution 2

- Q: From sales and customer find the customers from Limerick who have spent more than 2000 in total. List the ID and the total sum paid for each of them.

```
SELECT customer_id, SUM(paid)
FROM Sales JOIN Customers USING (customer_id)
WHERE city = 'Limerick'
GROUP BY customer_id
HAVING SUM(paid) > 2000;
```

Requirements on HAVING conditions

- Anything goes in a subquery

- Outside subqueries, they may refer to attributes only if they are either
  - A grouping attribute

  OR
  - Aggregated (same condition as for SELECT clauses with aggregation)

In Class Exercises

- Q1: Find the average speed, RAM, and hard drive size for laptops
- Q2: Find the total quantity sold for a product with model 1007. Call the resulting column total_quantity
- Q3: Assuming that you only buy a certain model once, retrieve all customer ID and full name of customers who have made at least two purchases.
- Q4: Find the customer who bought the most products
- Q5: Find customers who have made purchases on more than one day

What will come next week

- Set operations in SQL (UNION, INTERSECT, MINUS)
- Built-in Functions
- More Data Modification Statements (INSERT< DELETE, UPDATE)
- SQL in Host Language Environment

Contact details

- If you have any issues throughout the course, contact us via email:
- Tony: tony.garnock-jones@maastrichtuniversity.nl
- Katharina: k.schneider@maastrichtuniversity.nl
- Or send a message on Discord
- (https://discord.gg/Be2KSF8QG6)

Questions?

See you in two weeks