Relational Databases and Datawarehousing – SQL

Subqueries, Insert-Update-Delete-Merge, Views,
Common Table Expressions

SUBQUERIES

SUBQUERIES basic form

- Nested subqueries
  - Basic form

```
SELECT
FROM
WHERE condition
```

→ Contains in its left and/or right hand side statement another SELECT

    - Outer level query = the first SELECT. This is the main question.
    - Inner level query = the SELECT in the WHERE clause (or HAVING clause). This is the subquery:
      - Always executed first
      - Always between ().
      - Subqueries can be nested at > 1 level.
  - A subquery can return one value or a list of values

SUBQUERY that returns a single value

- The result of the query can be used anywhere you can use an expression.
  - With all relational operators: =, >, <, <=,>=,<>
  - Example:
    - What is the UnitPrice of the most expensive product?

```
SELECT MAX(UnitPrice) As MaxPrice FROM Products
```

	MaxPrice
1	263,50

    - What is the most expensive product?

```
SELECT ProductID, ProductName, UnitPrice As MaxPrice
FROM Products
WHERE UnitPrice = (SELECT MAX(UnitPrice) FROM Products)
```

	ProductID	ProductName	MaxPrice
1	38	Côte de Blaye	263,50

    - Returns the previous query always the same resultset as the following?

```
SELECT TOP 1 ProductID, ProductName, UnitPrice As MaxPrice
FROM Products ORDER BY UnitPrice DESC
```

First the table Products is searched to determine the highest salary (= subquery). Then the table Products is searched a second time (= main query) to evaluate each unitprice against the determined maximum.

SUBQUERY that returns a single value

- Other examples
  - Give the products that cost more than average

```
SELECT ProductID, ProductName, UnitPrice As MaxPrice
FROM Products
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products)
```

  - Who is the youngest employee from the USA?

```
SELECT LastName, FirstName
FROM Employees
WHERE Country = 'USA'
AND BirthDate = (SELECT MAX(BirthDate) FROM Employees WHERE Country = 'USA')
```

SUBQUERY that returns a single column

- The resulting column can be used as a list
  - Operators IN, NOT IN, ANY, ALL
  - IN operator (=ANY operator)
    - Example: Give all employees that have processed orders

```
SELECT e.EmployeeID, e.FirstName + ' ' + e.LastName As Name
FROM Employees e
WHERE e.EmployeeID IN (SELECT DISTINCT EmployeeID FROM Orders)
```

    - This can also be accomplished with JOIN

```
SELECT DISTINCT e.EmployeeID, e.FirstName + ' ' + e.LastName As Name
FROM Employees e JOIN Orders o ON e.EmployeeID = o.EmployeeID
```

SUBQUERY that returns a single column

- The resulting column can be used as a list
  - Operators IN, NOT IN, ANY, ALL
  - NOT IN operator
    - Example: Give all customers that have not placed any orders yet

```
SELECT *
FROM Customers
WHERE CustomerID NOT IN (SELECT DISTINCT CustomerID FROM Orders)
```

    - This can also be accomplished with JOIN

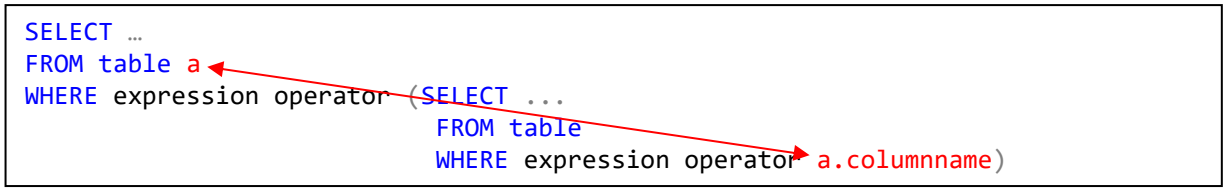
```
SELECT *
FROM Customers c LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE o.CustomerID is NULL
```

Correlated subqueries

- In a correlated subquery the inner query depends on information from the outer query.
  The subquery contains a search condition that refers to the main query, which makes the subquery depends on the main query
- The subquery is executed for each row in the main query. => $O(n^2)$
- The order of execution is from top to bottom, not from bottom to top as in a simple subquery, which is O(n).

Correlated subqueries

- For performance reasons use joins or CTE (see further) or simple subquery if possible
- Principle



Correlated subqueries

- Example: Give employees with a salary larger than the average salary

```
SELECT FirstName + ' ' + LastName As FullName, Salary
FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees)
```

0. Row 1 in the outer query

1. Outer query passes column values for that row to inner query

2. Inner query uses those values to evaluate inner query.

3. Inner query returns value to outer query, which decides if row in outer query will be kept.

4. This process repeats for each row in outer query.

Back to step 1.

This is a completely different (and more complex) task.

- Example: Give the employees whose salary is larger than the average of the salary of the employees who report to the same boss.

```
SELECT FirstName + ' ' + LastName As FullName, ReportsTo, Salary
FROM Employees As e
WHERE Salary > (SELECT AVG(Salary) FROM Employees WHERE ReportsTo = e.ReportsTo)
```

Remark: in the inner query you can use fields from the tables in the outer query but NOT vice versa.

HO GENT

Subqueries and the EXISTS operator

- The operator EXISTS tests the existence of a result set
  - Example: Give all customers that already placed an order

```
SELECT *
FROM Customers As c
WHERE EXISTS
(SELECT * FROM Orders WHERE CustomerID = c.customerID)
```

- There is also NOT EXISTS
  - Example: Give all customers that have not placed any orders yet

```
SELECT *
FROM Customers As c
WHERE NOT EXISTS
(SELECT * FROM Orders WHERE CustomerID = c.customerID)
```

HO GENT

3 ways to accomplish the same result

- Example: Give all customers that did not place any orders yet
  - OUTER JOIN

```
SELECT *
FROM Customers c LEFT JOIN Orders o ON c.CustomerID = o.CustomerID
WHERE o.CustomerID is NULL
```

Which one will, in general, be the slowest?

  - Simple subquery

```
SELECT *
FROM Customers
WHERE CustomerID NOT IN (SELECT DISTINCT CustomerID FROM Orders)
```

  - Correlated subquery

```
SELECT *
FROM Customers As c
WHERE NOT EXISTS
(SELECT * FROM Orders WHERE CustomerID = c.customerID)
```

HO GENT

Subqueries in the SELECT and FROM-clause

- The previous examples showed how subqueries can be used in the WHERE-clause
- Since the result of a query is a table it can be used in the FROM-clause.
- We will be using CTE's (see further) instead

Subqueries in the SELECT-clause

- In a SELECT clause scalar (simple or correlated) subqueries can be used
  - Example: Give for each employee how much they earn more (or less) than the average salary of all employees with the same supervisor.
  - This example is just by way of example. We will be using CTE's instead.

```
SELECT Lastname, Firstname, Salary,
Salary -
(
    SELECT AVG(Salary)
    FROM Employees
    WHERE ReportsTo = e.ReportsTo
)
FROM Employees e
```

Some exercises

```
-- 1. Give the id and name of the products that have not been purchased yet.
-- 2. Select the names of the suppliers who supply products that have not been ordered yet.
-- 3. Give a list of all customers from the same country as the customer Maison Dewey
-- 4. Give for each product how much the price differs from the average price of all products of the same
category
-- 5. Give per title the employee that was last hired
-- 6. Which employee has processed most orders?
-- 7. What's the most common ContactTitle in Customers?
-- 8. Is there a supplier that has the same name as a customer?
-- 9. Give all the orders for which the ShipAddress is different from the CustomerAddress,
```

DML

HO
GENT

SQL - DML basic tasks

- SELECT → consulting data
- INSERT → adding data
- UPDATE → changing data
- DELETE → removing data

HO
GENT

Tip for not destroying your database

- The statements in this chapter are destructive.
- SQL has no UNDO by default!
- BUT you can 'simulate' UNDO if you take precautions.

Transactions are discussed in detail in one of the next chapters.

```
/* Tip for not destroying your database */
BEGIN TRANSACTION -- starts a new "transaction" -> Saves previous state of DB in buffer

-- several "destructive" commands can go here:
INSERT INTO Products(ProductName)
values ('TestProduct');

-- only you (in  your session) can see changes
SELECT * FROM Products WHERE ProductID = (SELECT MAX(ProductID) FROM Products)

ROLLBACK;   --> ends transaction and restores database in previous state
-- COMMIT;  --> ends transaction and makes changes permanent
```

DML

INSERT: add new records

HO
GENT

Adding data - INSERT

- The INSERT statement adds data in a table
  - Add one row through via specification
  - Add selected row(s) from other tables

HO GENT

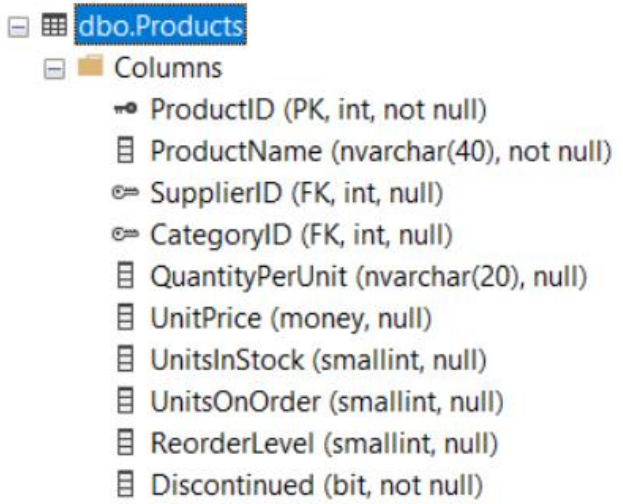
INSERT of 1 row

- Method 1: specify only the (not NULL) values for specific columns.
- Method 2: specify all column values
- If the identity is generated automatically, this column can't be mentioned.

```
INSERT INTO Products (ProductName, CategoryID, Discontinued)
VALUES ('Toblerone', 3, 0)

INSERT INTO Products
VALUES ('Sultana', null, 3, null, null, null, null, null, 1)
```

HO GENT



INSERT of 1 row

- The number of specified columns corresponds to the number of values.
- The specified values and correspond ding columns have compatible data types.
- If no column names are specified the values are assigned in the column order as specified by the CREATE TABLE statement.
- Unmentioned columns get the value NULL or the DEFAULT value if any.
- NULL can also be specified as a value.

HO GENT

INSERT of rows selected from other tables

- Mandatory fields have to be specified, unless they have a DEFAULT value.
- Constraints (see further) are validated.
- Unmentioned columns get the value NULL or the DEFAULT value if any.

```
INSERT INTO Customers (CustomerID, ContactName, ContactTitle, CompanyName)
SELECT substring(FirstName,1,2) || substring(LastName,1,3), FirstName || ' ' ||
LastName, Title, 'EmployeeCompany'
FROM Employees
```

HO GENT

DML

UPDATE: modify values

HO GENT

Changing data - UPDATE

- Changing all rows in a table
  - Example: Increase the price of all products with 10%

```
UPDATE Products
SET UnitPrice = UnitPrice * 1.1
```

- Changing 1 row or a group of rows
  - Example: Increase the price of all the 'Bröd' products with 10%

```
UPDATE Products
SET UnitPrice = UnitPrice * 1.1
WHERE ProductName LIKE '%Bröd%'
```

  - Example: Increase the price of all the 'Bröd' products with 10% and set all units in stock to 0

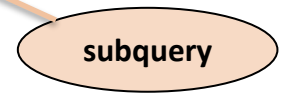
```
UPDATE Products
SET UnitPrice = UnitPrice * 1.1, UnitsInStock = 0
WHERE ProductName LIKE '%Bröd%'
```

HO GENT

Changing data - UPDATE

- Change rows based on data in another table
  - Standard SQL does not offer JOINs in an update statement → you can only use subqueries to refer to another table
  - Example: Due to a change in the euro – dollar exchange rate, we have to increase the unit price of products delivered by suppliers from the USA by 10%.

```
UPDATE Products
SET UnitPrice = (UnitPrice * 1.1)
WHERE SupplierID IN
(SELECT SupplierID FROM Suppliers WHERE Country = 'USA')
```



HO GENT

DML

DELETE: remove records

HO
GENT

Removing data - DELETE

- Delete some rows
  - Example: Delete the 'Bröd' products

```
DELETE FROM Products
WHERE ProductName LIKE '%Bröd%'
```

- Delete all rows in a table
  - via DELETE the identity values continues
  - via TRUNCATE the identity value restarts from 1
    TRUNCATE is also more performant, but does not offer WHERE clause: it's all or nothing

```
-- the identity value continues
DELETE FROM Products
```

```
-- the identity value restarts from 1
TRUNCATE TABLE Products
```

HO GENT

DELETE - based on data in another table

- Change rows based on data in another table
  - Again no JOIN, only subquery
  - Example: Delete the orderdetails for all orders from the most recent orderdate

```
DELETE FROM OrderDetails
WHERE OrderID IN
(SELECT OrderID FROM Orders WHERE OrderDate = (SELECT MAX(OrderDate) from Orders))
```

HO GENT

Views

HO
GENT

Views – Introduction

- Definition
  - A view is a saved SELECT statement
  - A view can be seen as a virtual table composed of other tables & views
  - No data is stored in the view itself, at each referral the underlying SELECT is re-executed;

HO GENT

Views – Introduction

- Advantages
  - Hide complexity of the database
    - Hide complex database design
    - Make large and complex queries accessible and reusable
    - Can be used as a partial solution for complex problems
  - Used for securing data access: revoke access to tables and grant access to customised views.
  - Organise data for export to other applications

HO GENT

Definition of a view

```
CREATE VIEW view_name [(column_list)]AS select_statement
```

- number of columns in (column_list) = # colunms in select
  - If no column names are specified, they are taken from the select
  - Column names are mandatory if the select statement contains calculations or joins in which some column names appear more than once
- the select statement may not contain an order by

COMMON TABLE EXPRESSIONS

HO GENT

Common Table Expressions: the WITH component

- Example: Give per category the minimum price and all products with that minimum price

```
-- Solution 1 -> with subqueries

SELECT CategoryID, ProductID, UnitPrice
FROM Products p
WHERE UnitPrice = (SELECT MIN(UnitPrice) FROM Products WHERE CategoryID = p.CategoryID)
```

- Not performant! Loops through all products and calculates the MIN(unitprice) for the category of that specific product: $O(n^2)$
- The MIN(unitprice) is calculated multiple times for each category!

Common Table Expressions: the WITH component

- Example: Give per category the minimum price and all products with that minimum price
- Solution 2 → CTE's (**C**ommon **T**able **E**xpression)

```
-- Solution 2 -> with CTE's
WITH CategoryMinPrice(CategoryID, MinPrice)
AS (SELECT CategoryID, MIN(UnitPrice)
    FROM Products AS p
    GROUP BY CategoryID)
SELECT c.CategoryID, p.ProductID, MinPrice
FROM Products AS p
JOIN CategoryMinPrice AS c ON p.CategoryID = c.CategoryID AND p.UnitPrice = c.MinPrice;
```

Common Table Expressions: the WITH component

- Solution 2 → CTE's (**C**ommon **T**able **E**xpression)

```
-- Solution 2 -> with CTE's
WITH CategoryMinPrice(CategoryID, MinPrice)
AS (SELECT CategoryID, MIN(UnitPrice)
    FROM Products AS p
    GROUP BY CategoryID)

SELECT c.CategoryID, p.ProductID, MinPrice
FROM Products AS p
JOIN CategoryMinPrice AS c ON p.CategoryID = c.CategoryID AND p.UnitPrice = c.MinPrice;
```

- Using the WITH-component you can give the subquery its own name (with column names) and reuse it in the rest of the query (possibly several times!)

Common Table Expressions: the WITH component

The columns in the CTE should have a name, so
you can refer to these columns.
(1) If not given a name, it will use the 'default' name (e.g. CategoryID, MIN(UnitPrice))
(2) Or you can specify the columnname in the 'header' of the CTE
(3) Or you can give each column a name in the CTE.

```
-- Solution 2
WITH CategoryMinPrice(CategoryID, MinPrice)
AS (SELECT CategoryID, MIN(UnitPrice)
    FROM Products AS p
    GROUP BY CategoryID)

SELECT c.CategoryID, p.ProductID, MinPrice
FROM Products AS p
JOIN CategoryMinPrice AS c ON p.CategoryID = c.CategoryID AND p.UnitPrice = c.MinPrice;
```

```
-- Solution 2
WITH CategoryMinPrice
AS (SELECT CategoryID As CategoryID, MIN(UnitPrice) AS MinPrice
    FROM Products AS p
    GROUP BY CategoryID)

SELECT c.CategoryID, p.ProductID, MinPrice
FROM Products AS p
JOIN CategoryMinPrice AS c ON p.CategoryID = c.CategoryID AND p.UnitPrice = c.MinPrice;
```

Common Table Expressions: the WITH component

- the WITH-component has two application areas:
  - Simplify SQL-instructions, e.g. simplified alternative for simple subqueries or avoid repetition of SQL constructs
  - Traverse recursively hierarchical and network structures

CTE’s versus Views

- Similarities
  - WITH ~ CREATE VIEW
  - Both are virtual tables:
    the content is derived from other tables
- Differences
  - A CTE only exists during the SELECT-statement
  - A CTE is not visible for other users and applications

CTE’s versus Subqueries

- Similarities
  - Both are virtual tables:
    the content is derived from other tables
- Differences
  - A CTE can be reused in the same query
  - A subquery is defined in the clause where it is used (SELECT/FROM/WHERE/…)
  - A CTE is defined on top of the query
  - A simple subquery can always be replaced by a CTE

CTE's with more than 1 WITH - component

- Example: Give the employees that process more orders than average
- Step 1: the number of processed orders per employee

```
-- Step 1 -> the number of processed orders per employee

SELECT EmployeeID, COUNT(DISTINCT OrderID)
FROM Orders
GROUP BY EmployeeID
```

CTE’s with more than 1 WITH - component

- Step 2: Calculate the average

```
-- Step 2 -> the average

SELECT AVG(NumberOfOrders)
FROM <???>
```

CTE's with more than 1 WITH - component

- Step 3: Combine both

```
-- Step 3 -> Combine both
WITH cte1 (EmployeeID, NumberOfOrders)
AS
(SELECT EmployeeID, COUNT(DISTINCT OrderID)
FROM Orders
GROUP BY EmployeeID),

cte2 (AverageNumberOfOrders)
AS
(SELECT AVG(NumberOfOrders)
FROM cte1)

SELECT *
FROM cte1 c1 JOIN cte2 c2
ON c1.NumberOfOrders >= c2.AverageNumberOfOrders
```

Exercises

```
-- 1. Give all employees that started working as an employee in the same year as Robert King

-- 2 Make a histogram of the number of orders per customer, so show how many times each number occurs.
-- E.g. in the graph below: 1 customer placed 1 order, 2 customers placed 2 orders, 7 customers placed 3
orders, etc.

/*

nrNumberOfCustomers
11
22
37
46
510
68
77
...

*/
```