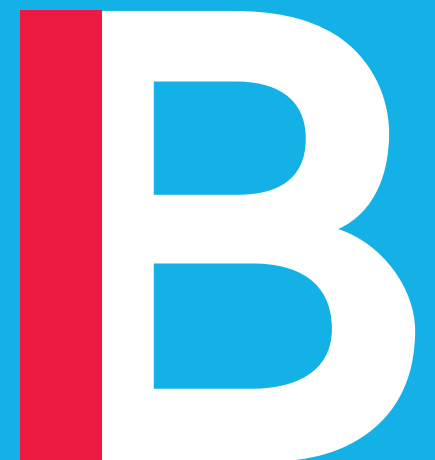


Lecture 6

Subqueries: splitting up queries

Dr Fintan Nagle
f.nagle@imperial.ac.uk



Reading

Video lectures:

6.2 - Overview of methods of splitting queries.mp4

6.3 - Subqueries.mp4

6.4 - CTEs and the WITH statement.mp4

6.5 - Views.mp4

Tutorial on subqueries: <https://www.techonthenet.com/postgresql/subqueries.php>

Postgres documentation on expressions in subqueries:

<https://www.postgresql.org/docs/current/static/functions-subquery.html>

Postgres documentation on WITH: <https://www.postgresql.org/docs/current/static/queries-with.html>

Postgres documentation on CREATE VIEW: <https://www.postgresql.org/docs/10/static/sql-createview.html>

Wikipedia article on views: [https://en.wikipedia.org/wiki/View_\(SQL\)](https://en.wikipedia.org/wiki/View_(SQL))

Splitting up queries

- There are three ways to split up queries:
- **Subqueries**
Smaller queries nested within the main query.
- **Common Table Expressions (CTEs)**
using the **WITH** keyword
Smaller queries placed at the top of the main query.
- **Views**
Saved queries which you can access as if they were tables.

Subqueries

A blue-tinted photograph of a modern glass-walled building. The building's facade is composed of large glass panels that reflect the surrounding environment. In the foreground, a person is sitting on a white, rounded, modular bench. The ground is paved with light-colored tiles. The overall scene is a modern, urban setting.

Subqueries

How do we show films whose rental rate is higher than the average?

We could do two queries, first

```
SELECT AVG (rental_rate)  
FROM film;
```

...and then

```
SELECT film_id, title, rental_rate  
FROM film  
WHERE rental_rate > 2.98;
```

Subqueries

Or we can use a single query with another query nested inside:

```
SELECT *  
FROM film  
WHERE rental_rate >  
      (SELECT AVG(rental_rate) FROM film)
```

Duplicated subqueries

```
SELECT *  
FROM film  
INNER JOIN rental  
WHERE film.rental_rate >  
(SELECT AVG(rental_rate) FROM film)  
AND rental.rental_rate >  
(SELECT AVG(rental_rate) FROM film)
```

Duplicated subqueries

```
SELECT *  
FROM film  
INNER JOIN rental  
WHERE film.rental_rate >  
  (SELECT AVG(rental_rate) FROM film)  
AND rental.rental_rate >  
  (SELECT AVG(rental_rate) FROM film)
```


Duplicated subqueries

```
WITH average_rental_rate AS(  
    SELECT AVG(rental_rate) AS rate  
    FROM film  
    WHERE length < 100 AND rating = 'PG-13'  
)
```

```
SELECT *  
FROM film  
INNER JOIN rental  
ON film.film_id = rental.film_id  
WHERE film.rental_rate >  
    (SELECT rate FROM average_rental_rate)  
AND rental.rental_rate >  
    (SELECT rate FROM average_rental_rate)
```

Uses for subqueries

- If HAVING is difficult:
just wrap in a subquery and treat like a table, selecting with
WHERE

Tables vs. single bits of data

Sometimes, subqueries return a single piece of data (**SELECT** MAX(price) **FROM** products). This can be used with WHERE, etc.

They can also return a full table (**SELECT** * **FROM** products). These cannot be used where a single number/value/datum is required.

It is important to distinguish between the two cases.

CTEs (WITH)

A blue-tinted photograph of a modern glass-walled building. The building's facade is composed of large glass panels that reflect the surrounding environment. In the foreground, a person is sitting on a white, rounded, modular bench. The ground is paved with light-colored tiles. The overall scene is a modern, urban setting.

WITH (Common Table Expressions or CTEs)

WITH allows another expression to be included as a smaller part.
It's very similar to subqueries.
From the Postgres docs:

```
WITH regional_sales AS (  
    SELECT region, SUM(amount) AS total_sales  
    FROM orders  
    GROUP BY region  
) , top_regions AS (  
    SELECT region  
    FROM regional_sales  
    WHERE total_sales > (SELECT SUM(total_sales) / 10 FROM  
regional_sales)  
)  
SELECT region,  
    product,  
    SUM(quantity) AS product_units,  
    SUM(amount) AS product_sales  
FROM orders  
WHERE region IN (SELECT region FROM top_regions)  
GROUP BY region, product;
```

WITH (Common Table Expressions or CTEs)

WITH allows another expression to be included as a smaller part.
It's very similar to subqueries.

```
WITH new_name AS (  
    query1  
) ,  
query2
```

WITH (Common Table Expressions or CTEs)

WITH allows another expression to be included as a smaller part.
It's very similar to subqueries.

```
WITH new_name AS (  
    query1  
), top_regions AS (  
    query2  
)  
  
query3
```

WITH (Common Table Expressions or CTEs)

What's the difference between WITH and subqueries?

WITH lets you give a subquery a name, allowing it to be easily reused in several parts of the main query.

This also allows the SQL processor to optimise further, as it can re-use the same subquery once it has been given a name using WITH.

However, modern processors are getting better and can often notice that a subquery is repeated and do the optimisation anyway, even without WITH.

Views

A blue-tinted photograph of a modern glass-walled building. The building's facade is composed of large glass panels that reflect the surrounding environment. In the foreground, a person is sitting on a row of white, rectangular outdoor seating units. The ground is paved with light-colored tiles. The overall scene is captured in a monochromatic blue tone.

Views

(Northwind)

- Read only

```
CREATE VIEW french_suppliers AS
```

```
SELECT * FROM suppliers  
WHERE country='France'
```

Views

CREATE TEMPORARY VIEW

french_suppliers **AS**

SELECT * FROM suppliers
WHERE country='France'

CREATE OR REPLACE VIEW

french_suppliers **AS**

SELECT * FROM suppliers
WHERE country='France'

- Read only
- Temporary views expire at the end of your database session
- **CREATE OR REPLACE** will replace the view if it is already in place

DVD rental

Create a view showing all films from 2006.

DVD rental

Create a view showing all films from 2006.

```
CREATE VIEW films_2006 AS  
SELECT * FROM film  
WHERE release_year = 2006
```


Subquery example

Northwind

Find the total value of all products ordered from each supplier.

Northwind

Find the total value of all products ordered from each supplier.

First we get the total value of products in each order:

```
SELECT order_details.ProductID,  
SUM(Order_Details.UnitPrice*order_details.Quantity)  
FROM Order_details INNER JOIN Products  
ON Order_Details.ProductID = Products.ProductID  
GROUP BY order_details.ProductID
```

This gets the total value of each product ID.

Note: what's the difference between SUM and *?

Now, we need the SupplierID in order to add up by supplier...

Can we just add it to the list of columns returned?

Northwind

Now we need the supplier ID.

Northwind

Now we need the supplier ID:

```
SELECT order_details.ProductID,  
products.supplierID, SUM(Order_Details.UnitPrice*order_details.Quantity)  
FROM Order_details INNER JOIN Products  
ON Order_Details.ProductID = Products.ProductID  
GROUP BY order_details.ProductID
```

Will this work?

Northwind

Now we need the supplier ID:

```
SELECT order_details.ProductID,  
products.supplierID, SUM(Order_Details.UnitPrice*order_details.Quantity)  
FROM Order_details INNER JOIN Products  
ON Order_Details.ProductID = Products.ProductID  
GROUP BY order_details.ProductID
```

Will this work?



```
1 SELECT order_details.ProductID, products.SupplierID, SUM(Order_Details.UnitPrice*order_details.Quantity)
```

```
ERROR: column "products.supplierid" must appear in the GROUP BY clause or be used in an aggregate function
```

```
2 FROM Order_details INNER JOIN Products  
3 ON Order_Details.ProductID = Products.ProductID  
4 GROUP BY order_details.ProductID
```

Northwind

```
1 SELECT order_details.ProductID, products.SupplierID, SUM(Order_Details.UnitPrice*order_details.Quantity)
2 FROM Order_details INNER JOIN Products
3 ON Order_Details.ProductID = Products.ProductID
4 GROUP BY order_details.ProductID
```

ERROR: column "products.supplierid" must appear in the GROUP BY clause or be used in an aggregate function

So we could either

- Group by SupplierID instead
- Join this with the Products table to get the SupplierID

(you can join a table to any table including itself)

Northwind

Join with the Products table to get the SupplierID:

```
SELECT t.ProductID, SupplierID, total_sales  
FROM
```

```
(SELECT order_details.ProductID,  
SUM(Order_Details.UnitPrice*order_details.Quantity) AS total_sales  
FROM  
Order_details INNER JOIN Products  
ON Order_Details.ProductID = Products.ProductID  
GROUP BY order_details.ProductID) as t
```

```
INNER JOIN products ON t.ProductID = products.ProductID
```

Northwind

Join with the Products table to get the SupplierID:

```
SELECT SupplierID,  
SUM(total_sales) AS total_supplier_sales  
FROM
```

```
(SELECT t.ProductID, SupplierID, total_sales  
FROM
```

```
(SELECT order_details.ProductID,  
SUM(Order_Details.UnitPrice*order_details.Quantity) AS total_sales  
FROM Order_details  
INNER JOIN Products  
ON Order_Details.ProductID = Products.ProductID  
GROUP BY order_details.ProductID) as t
```

```
INNER JOIN products ON t.ProductID = products.ProductID) AS t2
```

```
GROUP BY SupplierID
```

Northwind

Join with the Products table to get the SupplierID:

With WITH:

```
SELECT SupplierID,  
SUM(total_sales) AS total_supplier_sales  
FROM
```

```
(SELECT t.ProductID, SupplierID, total_sales  
FROM
```

```
(SELECT order_details.ProductID,  
SUM(Order_Details.UnitPrice*order_details.Quantity) AS total_sales  
FROM Order_details  
INNER JOIN Products  
ON Order_Details.ProductID = Products.ProductID  
GROUP BY order_details.ProductID) as t
```

```
INNER JOIN products ON t.ProductID = products.ProductID) AS t2
```

```
GROUP BY SupplierID
```

Northwind

Join with the Products table to get the SupplierID:

```
SELECT SupplierID,  
to_char(ROUND(SUM(total_sales / 1000)::numeric,1), '99.9') || 'k' AS  
total_supplier_sales  
FROM
```

```
(SELECT t.ProductID, SupplierID, total_sales  
FROM
```

```
(SELECT order_details.ProductID,  
SUM(Order_Details.UnitPrice*order_details.Quantity) AS total_sales  
FROM Order_details  
INNER JOIN Products  
ON Order_Details.ProductID = Products.ProductID  
GROUP BY order_details.ProductID) as t
```

```
INNER JOIN products ON t.ProductID = products.ProductID) AS t2
```

```
GROUP BY SupplierID
```


Northwind

Join with the Products table to get the SupplierID:

```
WITH supplier_sales AS
( SELECT SupplierID, to_char(ROUND(SUM(total_sales / 1000)::numeric,1),
'99.9') || 'k' AS total_supplier_sales
FROM
  ( SELECT t.ProductID, SupplierID, total_sales
    FROM
      ( SELECT order_details.ProductID,
        SUM(Order_Details.UnitPrice*order_details.Quantity) AS total_sales
        FROM
          Order_details INNER JOIN Products
        ON Order_Details.ProductID = Products.ProductID
        GROUP BY order_details.ProductID) as t
      INNER JOIN products ON t.ProductID = products.ProductID) AS t2
  GROUP BY SupplierID)
SELECT * FROM supplier_sales
```

Northwind

Join with the Products table to get the SupplierID:

```
SELECT SupplierID, to_char(ROUND(SUM(total_sales / 1000)::numeric,1), '99.9')  
|| 'k' AS total_supplier_sales  
FROM  
  (SELECT t.ProductID, SupplierID, total_sales  
  FROM  
    (SELECT order_details.ProductID,  
    SUM(Order_Details.UnitPrice*order_details.Quantity) AS total_sales  
  FROM  
    Order_details INNER JOIN Products  
    ON Order_Details.ProductID = Products.ProductID  
  GROUP BY order_details.ProductID) as t  
  INNER JOIN products ON t.ProductID = products.ProductID) AS t2  
GROUP BY SupplierID
```



UNION

UNION

Generates the union (list with duplicates removed) of two tables.

(**SELECT** LastName **FROM** Employees)

UNION

(**SELECT** FirstName **FROM** Employees)

UNION

(**SELECT** EmployeeID, LastName **FROM** Employees)

UNION

(**SELECT** EmployeeID, FirstName **FROM** Employees)

UNION

You can use UNION ALL to concatenate tables:

```
(SELECT EmployeeID, LastName FROM Employees)
```

UNION ALL

```
(SELECT EmployeeID, FirstName FROM Employees)
```

UNION vs. UNION ALL

(**SELECT** EmployeeID, LastName **FROM** Employees)

UNION

(**SELECT** EmployeeID, FirstName **FROM** Employees)

(**SELECT** EmployeeID, LastName **FROM** Employees)

UNION ALL

(**SELECT** EmployeeID, FirstName **FROM** Employees)

UNION vs. UNION ALL

SELECT COUNT(*) FROM

(
 SELECT EmployeeID, LastName **FROM** Employees)

UNION ALL

(**SELECT** EmployeeID, FirstName **FROM** Employees)

) **AS** t