Tomas Sandnes
tomas.sandnes@kristiania.no

Week 5

DB1102 / PGR 111 – DATABASES

Today's topics

(Today's chapters: 5.1-5.3 in Norwegian book, 3.8 & 4.2 in English book)

- Exercise walkthrough of task 5 from last week
- Subqueries
- VIEWs

Exercise walkthrough

Exercise walkthrough, 1 task from week 4

- We're looking into task 5 from the previous week:
  - *Retrieve all the countries in the world, and all their cities, where the population of the country is less than 1000 people. We want to retrieve the name of the country (AS CountryName), the population of the country (AS CountryPop), which continent the country belongs to, the name of all cities (AS CityName) and the population of the cities (AS CityPop). We want the result sorted alphabetically by the country names. (Hint: My answer returns 10 rows, is that the same number of rows you get?)*

Subqueries

Subqueries

- As mentioned in previous lessons, the result of a SELECT is presented as an SQL table.
  - It forms columns and rows the same way as existing tables in the database do.

- Therefore, it is ok to use the result of one SELECT as part of another SQL statement. Using a SELECT statement inside another SQL statement in this way is called a subquery.
  - (For the exercises on table `mail` and `person` in a previous week, you used a SELECT as part of an INSERT INTO statement.)

Subqueries – cont.

- Sometimes we need the answer from one query to complete another SQL statement.
  - We have already done this regarding insert into / update / delete from.
  - We can also do this when only SELECT statements are involved.

- *Example:* We want to figure out how many cities that have a population equal to or greater than the average population.
  - Before we can complete this query, we need to figure out what the average city population is.

Subqueries – cont.

- Task: *"How many cities have a population equal to or greater than the average city population?"*
  - We can solve it with a subquery (a SELECT) inside a SELECT, like this:

```
SELECT COUNT(*)
FROM city
WHERE Population >=
(SELECT AVG(Population) FROM city);
```

Subqueries – cont.

- Another one: *"List all European cities"*
  - We can solve it like this:

```
SELECT Name
FROM city
WHERE CountryCode IN
(SELECT Code
 FROM country
 WHERE Continent = 'Europe');
```

  - Explanation of IN: Lets us compare a column with a group of values.
  - *(Note: This one could have been solved with a JOIN as well.)*

Subqueries – cont.

- Yet another example: *"What size, as a percentage of the world's largest country, does each country have?"*
  - We can solve it with a subquery (a SELECT) inside a SELECT, like this:

```
SELECT Name, SurfaceArea / (SELECT MAX(SurfaceArea) FROM country) * 100
FROM country
ORDER BY SurfaceArea DESC;
```

- Or what about: *"We want to see what percentage the inhabitants of a country make up of the total population of Earth."*
  - This is one of today's exercises. :-)

VIEW

What is view?

- A VIEW is a pre-created query for one or more tables.
  - *Example:* We only want ID, NAME, and ADDRESS, for "STAVANGER"

ID	NAME	ADDRESS	CITY	POSITION	SALARY	HIREDATE	BNO
3	JON	BRUVEIEN 7	STAVANGER	MANAGER	70000	08-SEP-15	1
2	MARIE	STRILEGATEN 8	BERGEN	MANAGER	60000	01-JAN-15	2
1	SUSANNE	SKRAPLODDVEIEN 62	OSLO	MANAGER	90000	01-JUL-14	3
20	OLAV	GALMANNSVEIEN 4	STAVANGER	BROKER	52000	07-JUL-17	1
5	DAVID	GULERLEVEIEN 43	STAVANGER	SECRETARY	36000	14-JUN-15	1
4	ANNE	STRANDGATEN 5	STAVANGER	BROK	24000	12-DEC-16	1
8	GUSTAV	NORDLYSVEIEN 78	STAVANGER	BROK			
9	OLAVA	LOMVIVEIEN 57	STAVANGER	BROKER			
11	LEONORA	RØDVEIEN 6	OSLO	BROKER			
10	TEODOR	TULIPAN 12	OSLO	SECRETARY			
6	JONNAS	KIRKEVEIEN 7	BERGEN	BROKER			
12	TULLA	BLÅVEIEN 7a	OSLO	BROKER			
18	FREDRIK	LASSOVEIEN 37	OSLO	BROKER			
7	KARL	OLAVSGATE 7	OSLO	BROKER			
16	SMUKKA	GRAUTSTIEN 43	OSLO	BROKER			
17	KARL	BLÅVEISVEIEN 7	OSLO	BROKER	41000	12-MAY-96	3

ID	NAME	ADDRESS
3	JON	BRUVEIEN 7
20	OLAV	GALMANNSVEIEN 4
5	DAVID	GULERLEVEIEN 43
4	ANNE	STRANDGATEN 5
8	GUSTAV	NORDLYSVEIEN 78
9	OLAVA	LOMVIVEIEN 57

Why use views?

- Security: Restrict data access in the database.

- Reduce complexity when writing SQL queries later:
  - Can make complex queries easier.
  - Reduces the amount of data for the user.

- Customization: Achieve different views of the database.
  - user groups / applications

Disadvantages of views

- More complexity related to updates and changes:
  - There are restrictions on when you can change underlying data through a view.
  - The structure is determined when the VIEW is created, and will not automatically change later (VIEW based on SELECT * FROM ...)

- Performance:
  - A view can join many tables, and thus be a relatively heavy query.
  - This is not always easy to see for the user.

Create a view

```
CREATE [OR REPLACE] VIEW view_name AS
SELECT subquery;
```

- OR REPLACE → overwrites a VIEW if it already exists
- subquery → the SELECT query we want to store

Example, the world schema

```sql
CREATE OR REPLACE VIEW EuropeCountry_view AS
SELECT Code, Name, Population
FROM country
WHERE Continent = 'Europe';
```

```sql
SELECT *
FROM EuropeCountry_view;
```

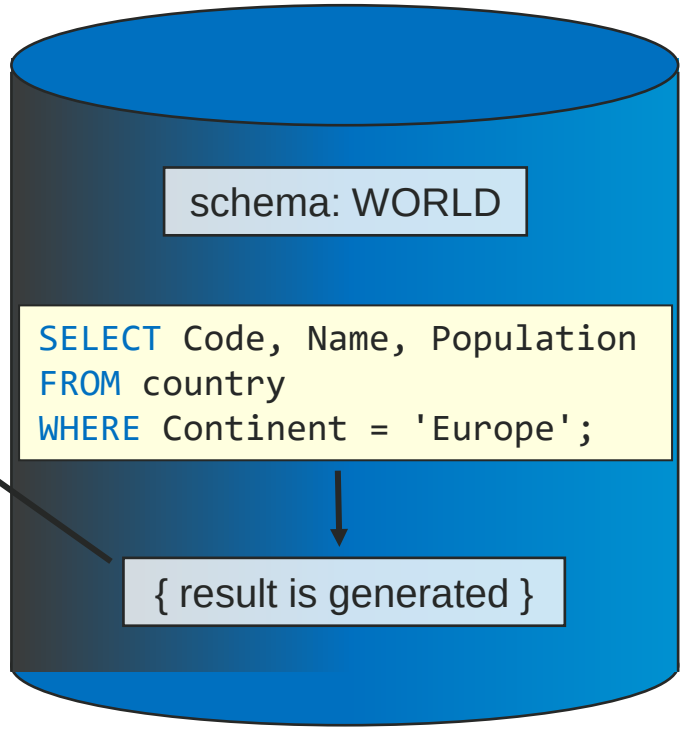
```
Code Name                  Population
---- ---------------------- ----------
ALB  Albania                3401200
AND  Andorra                78000
AUT  Austria                8091800
BEL  Belgium                10239000
BGR  Bulgaria               8190900
BIH  Bosnia and Herzegovina 3972000
BLR  Belarus                10236000
CHE  Switzerland            7160400
CZE  Czech Republic         10278100
DEU  Germany                82164700
DNK  Denmark                5330000
ESP  Spain                  39441700
...
```

- Views are treated the same way as tables are.

This is how querying a view works

```
SELECT *
FROM EuropeCountry_view;
```

```
Code Name         Population
---- ------------ ----------
ALB  Albania      3401200
AND  Andorra      78000
AUT  Austria      8091800
BEL  Belgium      10239000
BGR  Bulgaria     8190900
...
```



- The definition of the VIEW itself (*not* the result) is stored in the database.

Example, some variations

- We can give views their own column names:

```
CREATE OR REPLACE VIEW EuropeCountry_view AS
SELECT Code AS ID, Name AS Country, Population
FROM country
WHERE Continent = 'Europe';
```

```
CREATE OR REPLACE VIEW EuropeCountry_view
(ID, Country, Population) AS
SELECT Code, Name, Population
FROM country
WHERE Continent = 'Europe';
```

```
SELECT *
FROM EuropeCountry_view;
```

```
ID   Country                 Population
---- ----------------------- ----------
ALB  Albania                 3401200
AND  Andorra                 78000
AUT  Austria                 8091800
BEL  Belgium                 10239000
BGR  Bulgaria                8190900
BIH  Bosnia and Herzegovina  3972000
BLR  Belarus                 10236000
CHE  Switzerland             7160400
CZE  Czech Republic          10278100
DEU  Germany                 82164700
DNK  Denmark                 5330000
ESP  Spain                   39441700
...
```

Group functions used in views

```sql
CREATE OR REPLACE VIEW ContinentPopulation_view AS
SELECT Continent, SUM(Population) AS Population
FROM country
GROUP BY Continent
ORDER BY SUM(Population) ASC;
```

```sql
SELECT *
FROM ContinentPopulation_view;
```

```
Continent        Population
---------------- -----------
Antarctica       0
Oceania          30401150
South America    345780000
North America    482993000
Europe           730074600
Africa           784475000
Asia             3705025700
```

- Group functions can be used in views.
  - It is a good idea to use AS (alias) to customize the column names.

Exercise

- Create a view to display the cities in the world with population > 5 million.
  - It should display city, population and which country the city is located in.
  - It should be sorted on population descending.

```
CREATE OR REPLACE VIEW LargeCities_view
(city, population, country) AS
SELECT city.name, city.population, country.name
FROM city
JOIN country ON CountryCode = Code
WHERE city.Population > 5000000
ORDER BY city.Population DESC;
```

city	population	country
Mumbai (Bombay)	10500000	India
Seoul	9981619	South Korea
São Paulo	9968485	Brazil
Shanghai	9696300	China
Jakarta	9604900	Indonesia
Karachi	9269265	Pakistan
Istanbul	8787958	Turkey

- Then, lets retrieve the 7 largest cities:

```
SELECT *
FROM largeCities_view
LIMIT 7;
```

New SQL word: LIMIT

- LIMIT limits (d'oh!) the number of rows in your result.

- Example:
  - I want to find the three physically largest countries in the world.

```
SELECT Name, SurfaceArea
FROM Country
ORDER BY SurfaceArea DESC
LIMIT 3;
```

Result Grid | Filter Rows:

Name	SurfaceArea
Russian Federation	17075400.00
Antarctica	13120000.00
Canada	9970610.00
NULL	NULL

Tips for using CREATE VIEW

- Same tip for creating VIEW as for doing insert / update / delete based on queries:
  - Always test the SELECT first, so you know if it works before you use it in the view!

```
CREATE OR REPLACE VIEW largeCities
(city, population, country)
AS SELECT ci.name, ci.population, co.name
FROM city ci LEFT JOIN country co ON ci.countryCode = co.Code
WHERE ci.Population>5000000
ORDER BY population DESC;
```

Updates through views

- Views can be used to update data in an underlying table.

- *Note:* There are ISO restrictions that must be followed before a view can be used for updates:
  - The view can only reference one table.
  - DISTINCT cannot be part of the view.
  - All elements in the select part of the view must be columns (not constants, summations, etc.)
  - No GROUP BY or HAVING.
  - Rows that are added must follow the integrity rules for the underlying table (not null, etc.).

Updates through views – cont.

- Note that a view updates the data in the table! (Not just the view itself.)

```
UPDATE EuropeCountry_view
SET ID = 'A_Z'
WHERE Country = 'Austria';
```

```
SELECT ID, Country
FROM EuropeCountry_view;
```

```
ID   Country
---- ---------
ALB  Albania
AND  Andorra
A_Z  Austria
BEL  Belgium
BGR  Bulgaria
...
```

```
SELECT Code, Name, Population
FROM country
WHERE Name = 'Austria';
```

```
CODE Name      Population
---- --------- ----------
A_Z  Austria   8091800
```

Deleting views

- Syntax to delete a view is almost the same as for a table:
  - Just replace the word "table" with "view".

```
DROP VIEW EuropeCountry_view;
```

Today's exercises & next lesson

- Now: 2 hours of exercises.

- Exercises are found on Canvas. Short summary:
  - Try to make some subqueries on your own.
  - Combine what we just learned about VIEW with SQL knowledge syntax from our earlier lessons.

- Main contents for the next lesson:
  - Not new content, but a sum-up of what we have learned so far!
  - Walkthrough of exercises *YOU* want to see solved form previous weeks. Send me those week- & task-numbers. :-)

The