Almacéns e Minaría de Datos

Práctica 1: PostgreSQL

Hugo Gómez Sabucedo
hugo.gomez.sabucedo@rai.usc.es

Horas dedicadas: 10

Índice

1-Importar datos do censo de poboación..... 3
2-Buscar as 4 parroquias con máis habitantes da provincia de Pontevedra 3
3-Mostrar, para cada provincia, os concellos con poboación superior a 15.000 habitantes..... 4
4-Instalar a extensión cstore_fdw, importar os rexistros e comparar o tamaño das táboas..... 5
5-Realizar consultas sobre ambas táboas e análise dos tempos. 5
6-Instalar a BD Adventure Works. Pasos e problemas atopados. Seccións do ficheiro install.sql e propósito..... 6
7-Importar a BD e realizar consultas. Analizar a estrutura e comparar coa estrutura da anterior BD 7

1-Importar datos do censo de poboación

En primeiro lugar, para importar os datos, deberemos de ter creada unha base de datos en PostgreSQL coa súa correspondente táboa. Isto farémolo empregando o comando que se ve na seguinte imaxe, para crear unha táboa que conteña tódolos campos necesarios, como aparecen nos arquivos dos que extraeremos os nosos datos.

```
create table Parroquias (
    PRO int not null,
    CON int not null,
    cp int not null,
    PROVINCIA varchar(12) not null,
    CONCELLO varchar (35) not null,
    PARROQUIA varchar (55) not null,
    POBOACION int not null
);
```

A continuación, descargaremos o arquivo da web. Debido a que se descarga como formato Excel (xls), é preciso convertelo ó formato CSV, fixándonos que estea codificado con UTF-8. Os datos da poboación vamos a reflectilos na táboa coma un número enteiro; por tanto, é preciso que, no propio arquivo CSV, antes de importalo, eliminemos o . separador dos millares xa que, se non, obteremos un erro.

Unha vez feito isto, podemos importar os datos do arquivo CSV, que é importante que estea gardado na carpeta axeitada. Para importar os datos, empregaremos o seguinte comando:

```
COPY Parroquias(PRO, CON, cp, PROVINCIA, CONCELLO, PARROQUIA, POBOACION)
from 'C:/Program Files/PostgreSQL/14/data/parroquias.csv'
delimiter ';' CSV HEADER;
```

Isto devolveranos como resultado o número de liñas copiadas, que neste caso son 3797. Desta forma, podemos comprobar que tódolos datos se engadisen correctamente. Se estivésemos en Linux, un exemplo de directorio no cal almacenar o arquivo, de forma que poida ser lido por Postgres, sería /tmp.

Outra opción para importar os datos sería empregar a opción de importar que nos ofrece pgAdmin. Unha vez que temos creada a táboa da base de datos, facemos clic dereito sobre ela e seleccionamos “Import/Export Data...”. Ábrese unha pestana na que teremos que seleccionar a localización do arquivo, o seu formato (csv) e a codificación (UTF8). En *options*, debemos establecer como delimitador o ;. En *columns*, poderemos seleccionar en que columnas queremos que se importen os datos.

2-Buscar as 4 parroquias con máis habitantes da provincia de Pontevedra

Para facer isto, debemos empregar unha consulta SQL. Pódese facer de dúas formas: empregando a opción `limit X` ou `fetch first X rows only`. Os comandos que empregaríamos serían os seguintes:

```
select parroquia, poboacion
from Parroquias
where provincia='PONTEVEDRA'
order by poboacion desc
limit 4
```

```
select parroquia, poboacion
from Parroquias
where provincia='PONTEVEDRA'
order by poboacion desc
fetch first 4 rows only
```

Desta forma, en ámbolos dous casos estaríamos seleccionando as parroquias, coa súa poboación, da provincia de Pontevedra, e ordenándoas por poboación de forma descendente (é dicir, de maior a menor). Desta forma, podemos empregar a opción `limit 4`, para limitar os resultados a 4 liñas, ou `fetch first 4 rows only`, para seleccionar só as primeiras catro liñas (é dicir, as 4 parroquias con máis poboación). Así, os resultados obtidos son os seguintes:

	parroquia character varying (55)	poboacion integer
1	VIGO	198212
2	PONTEVEDRA	52641
3	LAVADORES (SANTA CRISTINA)	15965
4	MARÍN (SANTA MARÍA DO PORT…	15747

3-Mostrar, para cada provincia, os concellos con poboación superior a 15.000 habitantes

Neste caso, é preciso empregar unha subconsulta, como se ve na seguinte imaxe. Os datos dos que dispoñemos son de poboación por parroquias pero, neste caso, precisamos a poboación total por concellos. Para isto, na subconsulta, o que facemos é sumar, para cada concello, a poboación das súas parroquias (seleccionando a suma como *Total*). Empregamos o *group by* para agrupar os datos por provincias e concellos e quedarnos só con unha fila por concello. Unha vez feito isto na subconsulta, podemos seleccionar só aqueles concellos cuxa poboación supere as 15.000 persoas, e ordenar os datos de forma descendente. Desta forma, obteríamos o seguinte resultado:

```
SELECT provincia, concello, Total
FROM (
    SELECT provincia, concello, sum(poboacion) as Total
     from parroquias
     group by provincia, concello
) as sub
where Total>15000
order by Total desc
```

	provincia character varying (12)	concello character varying (35)	total bigint
1	PONTEVEDRA	VIGO	280186
2	A CORUÑA	CORUÑA, A	236379
3	OURENSE	OURENSE	107510
4	A CORUÑA	SANTIAGO DE COMP…	90188
5	LUGO	LUGO	88414
6	A CORUÑA	FERROL	77950
7	PONTEVEDRA	PONTEVEDRA	74942
8	PONTEVEDRA	VILAGARCÍA DE ARO…	33496
9	A CORUÑA	NARÓN	32204
10	PONTEVEDRA	REDONDELA	29003
11	A CORUÑA	CARBALLO	28142
12	A CORUÑA	OLEIROS	27252
13	A CORUÑA	RIBEIRA	26086
14	PONTEVEDRA	MARÍN	24997
15	PONTEVEDRA	CANGAS	23981
16	A CORUÑA	ARTEIXO	23306
17	A CORUÑA	CULLEREDO	22348
18	PONTEVEDRA	ESTRADA, A	22308
19	PONTEVEDRA	LALÍN	19869
20	A CORUÑA	CAMBRE	19262
21	LUGO	MONFORTE DE LEMOD	19091
22	PONTEVEDRA	PONTEAREAS	19011
23	A CORUÑA	AMES	18782
24	PONTEVEDRA	MOAÑA	17887
25	A CORUÑA	BOIRO	17748
26	PONTEVEDRA	NIGRÁN	16110
27	PONTEVEDRA	SANXENXO	16098
28	PONTEVEDRA	TUI	16042
29	PONTEVEDRA	PORRIÑO, O	15960
30	A CORUÑA	TEO	15476
31	LUGO	VILALBA	15365
32	LUGO	VIVEIRO	15240

4-Instalar a extensión cstore_fdw, importar os rexistros e comparar o tamaño das táboas

Unha vez instalamos a extensión, creamos un servidor onde empregaremos dita extensión. Empregando o comando `CREATE FOREIGN TABLE parroquias_col(PRO INT, CON INT, CP INT, PROVINCIA TEXT, CONCELLO TEXT, PARROQUIA TEXT, POBOACION INT) SERVER cstore_server;` , crearemos a táboa onde importaremos os datos. Unha vez feito isto, podemos importar os datos empregando o comando `\COPY parroquias_col from '/tmp/parroquias.csv' WITH CSV;` . Unha vez executado, vemos que se copiaron 3797 liñas, o cal nos indica que a operación se completou correctamente.

En primeiro lugar, para realizar as análises, debemos saber como funciona o almacenamento columnar. Contrario ó almacenamento por filas que usamos habitualmente, no cal se almacena cada fila de datos nun bloque, o almacenamento por columnas almacena cada columna da táboa en bloques. É dicir, o método tradicional almacena, no mesmo bloque, tódolos atributos dun dato (no noso caso, provincia, concello, parroquia, poboación...), de forma que se queremos acceder a varios datos deberemos ler varias filas da táboa. Pola contra, o almacenamento por columnas almacena, no mesmo bloque, cada atributo; desta forma, teríamos un bloque con tódolos concellos, outro con tódalas parroquias, outro con tódolos datos de poboación...

Isto, ademais de ser máis eficiente con operacións que só requiren acceder a unha columna (para, por exemplo, calcular a súa media ou suma), permite tamén aforrar espazo, xa que, ó almacenar o mesmo tipo de datos no mesmo bloque, a compresión é maior. Isto tamén fai posible que o acceso ós datos en memoria sexa máis rápido xa que, ó estaren máis comprimidos, podemos gardar máis datos en memoria e acceder a eles máis rápido.

No que respecta ó tamaño das táboas, podemos achalo executando o comando `SELECT pg_size_pretty(pg_table_size('tablename'));`, onde *tablename* é o nome da táboa para a cal queremos saber o seu tamaño. Se executamos este comando para a primeira táboa, obtemos un tamaño de 360kB. Para obter o tamaño da táboa almacenada por columnas, debemos executar o mesmo comando, pero cambiando *pg_table_size* por *cstore_table_size*. Se facemos isto, obtemos un tamaño de 269kB. Como xa explicamos, isto é o que deberíamos esperar, posto que, ó almacenar o mesmo tipo de datos no mesmo bloque, podemos comprimir máis os datos e facer que ocupen menos espazo en disco.

5-Realizar consultas sobre ambas táboas e análise dos tempos.

A primeira consulta realizada foi para seleccionar as parroquias e a súa poboación da provincia de Pontevedra, como se ve na imaxe. Se o executamos co comando EXPLAIN ANALYZE, vemos que na primeira táboa (a normal) temos un tempo planificado de 5.528ms e un tempo de execución de 3.200ms, mentres que para a segunda táboa o tempo planificado é de 6.864ms e o de execución, de 7.415ms.

```
Select parroquia, poboacion
FROM parroquias
WHERE provincia='pontevedra'
```

A segunda consulta, como se ve na imaxe, selecciona o número total de parroquias de cada concello, agrupando por concello. Neste caso, para a primeira táboa, o tempo planificado é de 0.233ms e o de execución, 2.521ms, mentres que para a segunda táboa o tempo planificado é de 0.352 e o de execución, de 3.032ms.

```
SELECT count(parroquia)
FROM parroquias
GROUP BY concello
```

A terceira e última consulta é similar á anterior, pero seleccionando tamén o concello e ordenando os resultados polo número total de parroquias, de forma descendente. Para esta consulta, na primeira táboa temos un tempo planificado de 0.214ms e un tempo de execución de 2.806ms, e na segunda táboa, un tempo planificado de 0.181ms e un tempo de execución de 2.450ms.

```
SELECT concello, count(parroquia) as total
FROM parroquias
GROUP BY concello
ORDER BY total DESC
```

En xeral, podemos observar que, para tódolos exemplos, o tempo de execución na táboa por columnas é superior ó tempo de execución na táboa normal. Isto pode deberse, no primeiro caso, a que a consulta ten unha cláusula condicional *where*, a cal non se selecciona para o resultado final; isto pode facer que se incremente o tempo de execución, posto que nunha táboa normal só teriamos que ler as distintas filas e quedarnos coas que cumpran a condición, mentres que ó almacenar por columnas temos que ler o bloque onde se almacena a provincia, ver qué filas cumpren a condición, e logo seleccionar os datos que queremos mostrar na columna.

Por outra parte, nos outros dous casos, temos unhas consultas sinxelas, só con cláusulas *group by*. Malia que podería resultar esperable unha redución do tempo de execución na base de datos columnar respecto á base de datos por fila, que o tempo aumente pódese explicar por un motivo. Ó ter unha cantidade relativamente pequena de datos (pouco menos de 4000 datos), non podemos esperar unha redución excesiva do tempo de execución, xa que as bases de datos columnares son especialmente eficientes con grandes cantidades de datos.

6-Instalar a BD Adventure Works. Pasos e problemas atopados. Seccións do ficheiro install.sql e propósito

Para instalar esta base de datos, en primeiro lugar debemos descargar os arquivos dispoñibles en Github. Se abrimos o README, atopamos as instrucións de instalación. Debemos descargar os arquivos do enderezo que se nos indica e combinalos nunha carpeta cos arquivos update_csvs.rb e install.sql . É importante que esta carpeta a gardemos no directorio /tmp ou noutro que poida ser accedido por postgres. Unha vez feito isto, usamos o comando `ruby update_csvs.rb` para modificar os CSVs e creamos a base de datos co comando `psql -c "CREATE DATABASE \"Adventureworks\";"`.

O seguinte paso é situarnos no directorio onde gardamos a carpeta cos arquivos, para importar os datos. Unha vez aí, debemos executar o arquivo install.sql, empregando o comando `psql -d AdventureWorks < /tmp/install.sql` . Ó executar dito arquivo iremos obtendo mensaxes segundo as distintas operacións que se foron realizando, podendo comprobar que ningunha delas é unha mensaxe de erro. Ademais, ó rematar a execución, obtemos un listado con tódalas relacións que se crearon.

No que respecta ó arquivo install.sql, este contén todo o código necesario para crear a base de datos e importar os datos. Consta das seguintes seccións:

- **Instrucións de instalación.** Nesta primeira sección, explícansenos os pasos que debemos seguir para instalar correctamente a base de datos.
- **Tipos de datos.** Nesta sección, defínense 6 tipos de datos personalizados que se usarán na base de datos.
- **Esquemas.** Esta é a sección principal do arquivo. Nela, créanse os cinco esquemas, con tódalas súas correspondentes táboas. Para cada esquema, créanse tódalas táboas que pertencen a dito esquema (definindo as distintas columnas de cada táboa e, de ser o caso, as posibles restricións), e insírense os datos dende os distintos arquivos CSV que hai na

carpeta. Nalgúns casos, tamén se converten algúns campos representados con números binarios ó formato decimal, empregando o comando `UPDATE`.

- **Comentarios en táboas e columnas.** Esta sección está orientada a engadir comentarios ás distintas táboas (e ás súas columnas), de forma que se explica que tipos de datos se almacenan en dita táboa e as devanditas columnas, así como se algunha delas é unha chave primaria ou foránea.
- **Chaves primarias.** Nesta sección, emprégase o comando `ALTER TABLE` para definir as chaves primarias das táboas; isto é, o campo que identifica a cada fila única da táboa, o cal non se pode repetir dentro de cada táboa. Para algunhas táboas, tamén se emprega `CLUSTER ... USING`, o cal sirve para almacenar no mesmo bloque de datos físicos os datos de diferentes táboas.
- **Chaves foráneas.** Esta sección é similar á anterior, que emprega tamén o comando `ALTER TABLE` para as chaves foráneas; é dicir, os campos dunha táboa que se refiren ós campos de chave primaria doutras táboas.
- **Vistas.** Nesta sección, créanse e defínense as vistas nun formato apto para Postgres (tamén se explica, nun código comentado, como se faría en MSSQL). Isto faise empregando o comando `CREATE VIEW`, que crea unha "táboa virtual" cos resultados dunha consulta que se define no propio comando, de forma que podemos acceder a datos como, por exemplo, os produtos e as súas descricións en varios idiomas.
- **Esquemas a partir de vistas.** Nesta última seccións, créanse os mesmos cinco esquemas que se crearon na 3ª sección, pero a partir das vistas definidas anteriormente.

7-Importar a BD e realizar consultas. Analizar a estrutura e comparar coa estrutura da anterior BD

Ó importar esta base de datos, o primeiro problema que nos atopamos é un erro no tipo *money*. Isto débese a que os datos que temos no arquivo de instalación para este tipo son no formato $90,000.00. Porén, para que sexan válidos, debemos eliminar o símbolo $ e a coma separadora de miles. No primeiro caso, podemos eliminar este símbolo de forma sinxela, abrindo calquera editor de texto e substituíndo o símbolo por un espazo en branco. No caso da coma, podemos executar o comando `sed 's/\([0-9]\),\([0-9]\)/\1\2/g'`, que nos eliminará as comas, e gardar o resultado no mesmo arquivo. Unha vez feito isto, teremos a base de datos cos datos correctamente importados.

```
SELECT avg(yearlyincome::numeric)
FROM dimcustomer
WHERE totalchildren=0
```

No que respecta ás consultas, realizaremos tres. A primeira é achar a media do ingreso anual dos clientes que teñen fillos e dos clientes que non teñen fillos. Esta consulta é como se ve na imaxe. No caso dos clientes sen fillos, teremos que usar un = na cláusula condicional; no caso dos clientes con fillos, usaremos o símbolo >. Para a media, temos que usar `::numeric` para convertir o tipo de dato *money* a un tipo numérico, posto que, se non, teremos un erro. Se executamos esta consulta, obtemos unha media de 5148112.29$ para os clientes sen fillos e de 5956453.19$ para os clientes sen fillos.

```
SELECT factinternetsales.customerkey, firstname, lastname,
count(factinternetsales.customerkey) as total
FROM factinternetsales, dimcustomer
WHERE dimcustomer.customerkey=factinternetsales.customerkey
GROUP BY factinternetsales.customerkey, dimcustomer.firstname, dimcustomer.lastname
ORDER BY total DESC LIMIT 5
```

A segunda consulta será obter o código de cliente, nome e número total de compras dos cinco clientes con máis compras por Internet. Isto facémolo coa consulta que se ve na imaxe de arriba. Temos que seleccionar, da táboa de vendas por Internet, a suma das vendas de cada cliente, mentres que da táboa cos datos dos clientes seleccionaremos o ID e o nome e apelidos. Temos que poñer a condición de que coincidan os IDs dos clientes na táboa de vendas e na dos datos, para poder obter correctamente os datos que queremos. Así, o resultado que obtemos é o da imaxe.

	customerkey integer	firstname character varying (50)	lastname character varying (50)	total bigint
1	11185	Ashley	Henderson	68
2	11300	Fernando	Barnes	67
3	11277	Charles	Jackson	65
4	11262	Jennifer	Simmons	63
5	11287	Henry	Garcia	62

```
SELECT movementdate, sum(unitsin*unitcost)-sum(unitsout*unitcost) as balance
FROM factproductinventory
GROUP BY movementdate
ORDER BY balance DESC LIMIT 1
```

Por último, a última consulta será para atopar o día con máis balance positivo no inventario. Para isto, teremos que facer a consulta que vemos na imaxe. Nesta consulta, para achar o balance, simplemente facemos unha resta das unidades de cada produto que entraron, multiplicado polo prezo por unidade, menos as unidades de cada produto que saíron, multiplicado polo prezo por unidade, e ordenamos o resultado de forma descendente para quedarnos só co primeiro resultado. Así, obtemos que o día con maior balance positivo no inventario é o 2 de agosto de 2007, cun balance de 261,658,540.00$.

No que respecta á estrutura da base de datos, unha das principais diferencias é que a primeira base de datos (á cal nos referiremos como 2012) consta de 5 esquemas diferentes, mentres que a segunda base de datos (á cal nos referiremos como 2014) consta dun único esquema. Na base de datos 2012, temos esquemas para as vendas (Sales), compras (Purchasing), produción (Production), recursos humanos (HumanResources) e persoas (Person). Isto tamén provoca cambios na estrutura per se das táboas, tanto interna como externa. Se contamos o número de táboas, na base de datos 2012 temos 68, fronte ás 72 táboas da base de datos 2014.