Data Base Administrator PostgreSQL

Ernesto Quiñones A.
ernesto@eqsoft.net

Sesión 5

Mantenimiento Recurrente
Indexamiento
Interpretando un Plan de Ejecución
Qureys Avanzados

Backup y Restore

a) pg_dump, permite generar un backup de una db específica.

Todos los parámetros donde se especifican nombre de objetos (tablas, schemas, etc.) pueden usar el comodín “*”, por ejemplo todas la tabla que empiezan con EQSOFT es “--table=EQSOFT*”

Todos se pueden usar solos o en combinación, además están los parametros de conexión los cuales son:

```
-h HOST_NAME <-- ip del servidor o nombre del host
-p PUERTO    <-- número del puerto
-U USER_NAME <-- nombre del usuario
-w           <-- no pide password
-W           <-- fuerza a pedir el password
```

Se pueden usar estas variables de entorno: PGDATABASE, PGHOST, PGOPTIONS, PGPORT y PGUSER

```
ernesto@depeche:~/aaa$ pg_dump prueba5 > archivo.dump
pg_dump: [archivador (bd)] fallÃ³ la conexiÃ³n a la base de datos Â«prueba5Â»: FATAL:  no se especifica un nombre de usuario en el paquete de inicio

ernesto@depeche:~/aaa$ export PGUSER=dbadmin
ernesto@depeche:~/aaa$ pg_dump prueba5 > archivo.dump
```

Backup y Restore

```
ernesto@depeche:~/aaa$ pg_dump prueba5 > archivo.dump
pg_dump: [archivador (bd)] fallÃ³ la conexiÃ³n a la base de datos
Â«prueba5Â»: FATAL:  no se especifica un nombre de usuario en el
paquete de inicio

ernesto@depeche:~/aaa$ export PGUSER=dbadmin
ernesto@depeche:~/aaa$ pg_dump prueba5 > archivo.dump
```

Backup y Restore

a) pg_dump, permite generar un backup de una db específica.

```
pg_dump NOMBRE_DB -a                    <-- solo datos

pg_dump NOMBRE_DB -b                    <-- incluir objetos grandes

pg_dump NOMBRE_DB -c                    <-- incluye limpieza de objetos
                                            de la db, inserta drops antes
                                            De crearlos

pg_dump NOMBRE_DB -C                    <-- incluye un create database

pg_dump NOMBRE_DB --encoding=LATIN1     <-- backup con otra codificación

pg_dump NOMBRE_DB --file=backup.dump    <-- el backup se crear en este
                                            archivo

pg_dump NOMBRE_DB --format=p            <-- backup en texto plano
```

Backup y Restore

a) pg_dump, permite generar un backup de una db específica.

```
pg_dump NOMBRE_DB --format=c          <-- backup en formato que soporta
                                          pg_restore, comprimido

pg_dump NOMBRE_DB --schema=NOMBRE_SC <-- backup solo del esquema
                                          especificado

pg_dump NOMBRE_DB --exclude-schema=NO<-- backup excluyendo el esquema
                                          especificado

pg_dump NOMBRE_DB -o                  <-- incluir el OID de los
                                          objetos, útil solo si se usa
                                          los OID específicamente

pg_dump NOMBRE_DB -j                  <-- permite paralelizar en
                                          múltiples tareas el backup de
                                          la DB, esto mejora el tiempo
                                          de backupo pero satura el
                                          servidor con más trabajo.
```

Backup y Restore

a) pg_dump, permite generar un backup de una db específica.

```
pg_dump NOMBRE_DB -O                    <-- no se backupean los OWNER de
                                            los objetos, es útil cuando
                                            Cuando se mueve una db de un
                                            servidor a otro diferente.

pg_dump NOMBRE_DB -S                    <-- solo backupea estructura de
                                            la db más no la data

pg_dump NOMBRE_DB --table=NOMBRE_TAB <-- backupea solo la tabla
                                            especificada

pg_dump NOMBRE_DB --exclude-table=TA <-- backupea sin la tabla
                                            especificada, si se desea la
                                            Estructura pero no la data
                                            puede usar
                                            –exclude-table-data

pg_dump NOMBRE_DB -x                    <-- no se backupean los permisos
```

Backup y Restore

a) pg_dump, permite generar un backup de una db específica.

```
pg_dump NOMBRE_DB –compres=0..9       <-- especifica el nivel de
                                          compresión del backup, útil
                                          cuando se usa el parámetro:
                                          --format=c

pg_dump NOMBRE_DB --inserts           <-- genera inserts de la data en
                                           vez de copys, añadir
                                          --column-inserts si se desea
                                          que se especifiquen en los
                                          Insert los nombres de los
                                          campos

pg_dump NOMBRE_DB --disable-triggers <-- desabilita los triggers,
                                          útil solo si copiamos data y
                                          no estructura de la db.

pg_dump NOMBRE_DB --no-tablespaces    <-- no incluye las definiciones
                                          de los tablespaces en el
                                          backup
```

https://www.postgresql.org/docs/10/static/app-pgdump.html

Backup y Restore

Restaurar un backup en texto plano es bastante sencillo:

1. **Cree una base de datos donde desee levantar el backup**
2. **Ejecute lo siguiente:**

```
[user@host] psql BASE_NUEVA < ARCHIVO_BACKUP.BAK -U USUARIO
```

Tenga en cuenta que los backups llevan dentro de si los accesiso y revocaciones definidos en la base de datos, por lo tanto al momento de levantar el backup ya debe contar en el servidor con los usuarios definidos en la base de datos original.

Backup y Restore

b) pg_dumpall, permite generar un backup de todo el dbms.

Implementa los mismos parámetros de pg_dump, añade el parámetro "-l NOMBRE_DB" para especificar el nombre de la base de datos a backupear (mejor usar pg_dump).

pg_dumpall además copia usuarios de la base de datos además de la data y estructura cosa que no hace pg_dump

Para comprimir un dump normal (en texto plano) podemos usar:

```
ernesto@depeche:~/aaa$ pg_dump prueba5 -U dbadmin > dump.dump
ernesto@depeche:~/aaa$ ls dump.dump -la
-rw-r--r-- 1 ernesto ernesto 24522 2010-10-09 12:07 dump.dump

ernesto@depeche:~/aaa$ pg_dump prueba5 -U dbadmin | gzip >
dump.dump.gz
ernesto@depeche:~/aaa$ ls dump.dump.gz -la
-rw-r--r-- 1 ernesto ernesto 3940 2010-10-09 12:06 dump.dump.gz
```

Backup y Restore

c) Backup físico de los directorios de PostgreSQL.

Esta opción de Backup es posible, es tan simple como:

```
[postgres@host]tar -cf dump.tar /var/lib/postgresql/9.3/main <-- copia
                                                               todos los
                                                               Files

[postgres@host]tar -zcf dump.tar.gz /var/lib/postgresql/9.3/main <--
                                                                 copia y
                                                                 comprime
```

Debe tomar en cuenta que para hacer esto el motor de base de datos debe estar detenido, sino la consistencia de los datos podría no ser el deseado.

Se puede tomar en consideración “backups alternativos” como con Rsync hacía otros servidores, pero el consumo de recursos sería bastante elevado.

Backup y Restore

Existen otros medios de obtener backup como pg_basebackup y pg_start_backup(), ambos funcional a nivel de copiar los archivos WAL, esto permite simular un backup incremental pero genera muchas configuraciones utilizadas para replicación, podría ser recomendado para dbms muy grandes, además el proceso de recuperación es bastante manual. Los archivos WAL son los intercambios de data transaccional en la base de datos a nivel binario

Backup y Restore

d) pg_restore, permite cargar un backup a la dbms, siempre y cuando el backup haya sido generado por el parámetro “--format=c”

Algunas ventajas de usar pg_restore sobre el método alternativo es que tenemos mayor velocidad en restore de bases de datos enormes debido al uso de multihilos para el restore.

```
ernesto@depeche:~/aaa$ pg_dump prueba5 --format=c -U dbadmin > pgrestore.dump

ernesto@depeche:~/proyectos$ dropdb prueba5 -U dbadmin
ernesto@depeche:~/proyectos$ createdb prueba5 -U dbadmin

ernesto@depeche:~/aaa$ pg_restore pgrestore.dump -U dbadmin –dbname=prueba5
```

Backup y Restore

No es posible levantar un backup en texto plano usando PG_RESTORE

```
ernesto@depeche:~/aaa$ pg_dump prueba5 > otro.dump -U dbadmin

ernesto@depeche:~/proyectos$ createdb prueba6 -U dbadmin

ernesto@depeche:~/aaa$ pg_restore otro.dump -U dbadmin
--dbname=prueba6
pg_restore: [archivador] el archivo de entrada no parece ser un
archivador vÃ¡lido
```

Backup y Restore

d) pg_restore

Pg_restore permite el uso de muchos parámetros, básicamente los mismos de pg_dump en sentido inverso (no baja data sino sube obviamente), la lista completa puede ser revisada aquí:
https://www.postgresql.org/docs/10/static/app-pgrestore.html

Para especificar cuantos hilos deseamos que se lancen en el proceso de restore usamos el parámetro -j

```
ernesto@depeche:~/aaa$ pg_restore pgrestore.dump -U dbadmin -j 4
--dbname=prueba8
```

El tiempo mejorado depende de: cantidad de data, estructura de la db, cantidad de jobs, disponibilidad de la db, etc. Eventualmente se obtendrán beneficios de 25% de reducción del tiempo hasta 85% (lo máximo reportado) de ahorro.

Vacuum

El Vacuum es una de las más importantes tareas de administración, lo que hace es limpiar las “páginas” no usadas por el sistema y actualiza las estadísticas de las tablas e índices para una mejor resolución de querys.

La dbms ejecuta periódicamente (definido en postgresql.conf) un Lazy Vacuum, esto es libera páginas no usadas por data, más no por índice, este vacuum no genera demasiado tiempo de bloqueo en la tabla (depende el tamaño).

Antes de un Vacuum

Plato 1		
1	2	ED
ED	3	ED
OF	OF	OF
7	8	OF
OF	OF	SD

Plato 2		
EI	EI	5
4	OF	OF
OF	OF	OF
13	14	6
SD	SD	SD

Plato 3		
10	12	11
ED	ED	SD
SD	EI	SD
SD	SD	SD
9	SD	SD

Despues de un Vacuum

Plato 1		
1	2	3
4	5	6
OF	OF	OF
7	8	OF
OF	OF	9

Plato 2		
EI	EI	10
11	OF	OF
OF	OF	OF
12	13	14
SD	SD	SD

Plato 3		
SD	SD	SD
SD	SD	SD
SD	EI	SD
SD	SD	SD
SD	SD	SD

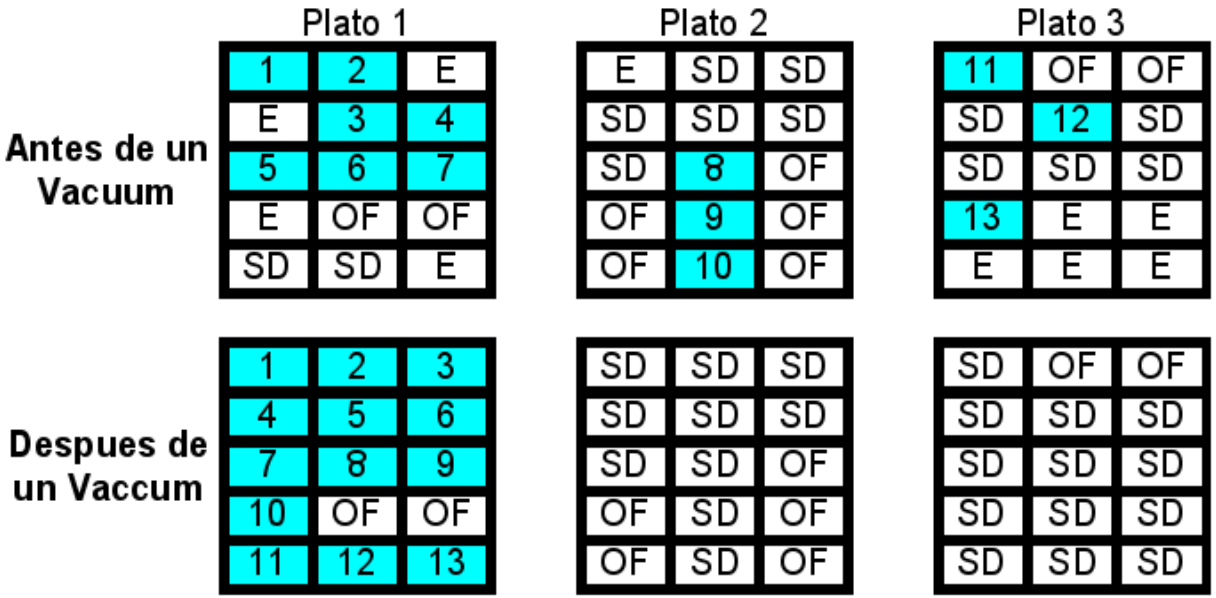
- Los recuadros con números son datos del postgresql (índice o data)
- Los recuadros con OF son de Otros Archivos (Other Files)
- Los recuadros con ED son páginas ocupadas por datos , ya no usadas
- Los recuadros con EI son páginas ocupadas por índices, ya no usadas

Esto es lo que sucede en un Lazy Vaccum, PostgreSQL recupera las páginas de datos y los ocupa con información “viva”.

Vacuum

El full vacuum requiere acceso exclusivo a la tabla durante el tiempo que demoré la operación, esta limpiara totalmente las páginas no usadas.

Mejora notablemente el tiempo de acceso a los datos.



- Recuadros con números ocupados por PostgreSQL.
- E páginas de PostgreSQL ocupadas pero no usadas..
- OF páginas con otros archivos.
- SD páginas sin datos

Vacuum

Se pueden aplicar Vaccums a nivel de tabla o base de datos.

- **Vacuum TABLA <-- solo limpia data y reorganiza las páginas.**
- **Vacuum freeze TABLA <-- libera espacio y reescribe la tabla.**
- **Vacuum analyze TABLA<-- actualiza las estadísticas para el generador de plan de ejecución.**
- **Vacuum full TABLA <-- limpia todo el espacio no utilizado y actualiza estadísticas, reescribe totalmente la tabla.**

```
DBNAME=# vacuum (verbose,full) tabla;
```

Se puede añadir de precisión un Vacuum sobre un campo.

```
DBNAME=# vacuum (verbose,analyze) tabla(campo1,campo2);
```

Vacuum

“Vacuumdb” se utiliza externamente desde la linea de comandos del sistema operativo para ejecutar la operación sobre la base de datos seleccionada.

```
ernesto@depeche:~$ vacuumdb --full prueba5 -U dbadmin
```

La “estadísticas”, es información de las tablas que permite generar al dbms buenos planes de ejecución de los querys, el proceso de autovacuum realiza este análisis si el contenido de una tabla ha cambiado lo suficiente para justificarlo.

Vacuum

Los parámetros del autovacuum pueden ser afinados en el archivo de configuración “postgresql.conf”.

```
autovacuum_naptime              <-- expresado en minutos, intervalo
                                    en el que se
                                    ejecuta un vacuum
autovacuum_vacuum_threshold     <-- cantidad mínima de filas
                                    actualizadas antes de
                                    hacer vacuum
autovacuum_analyze_threshold    <-- cantidad mínima de filas
                                    actualizadas antes de
                                    hacer un analyze
autovacuum_vacuum_scale_factor  <-- % de cambio de la tabla antes
de
                                    hacer un vacuum
autovacuum_analyze_scale_factor <-- % de cambio de la tabla antes
de
                                    hacer un analyze
autovacuum_freeze_max_age       <-- antigüedad máxima en
                                    transacciones en que se
                                    mantienen los files en pg_clog
```

Vacuum

autovacuum_vacuum_cost_delay <-- tiempo mínimo en milisegundos para reiniciar un trabajo de vacuum

autovacuum_vacuum_cost_limit <-- “costo” máximo que tendrá un Vacumm

autovacuum_max_workers <-- cantidad de procesos en paralelo que ejecutarán un vacuum

Para optimizar el proceso de autovacuum este se subdivide en varias operaciones encoladas, así se evita bloquear las tablas y permitir el ingreso de transacciones mientras se ejecuta en caliente el vacuum.

El “costo” es una medida matemática de ejecución de una operación.

Mucha gente piensa que un autovacuum constante puede ayudarles a mantener una base de datos afinada y un planeer de querys mucho más eficiente, esto no es tan cierto, el costo de procesamiento y de trabajo en disco que estaría pagando podría ser demasiado alto, el resultado final seguramente sería totalmente lo contrario al esperado.

Analyze

Una manera de actualizar las estadísticas sin necesidad de tener que ejecutar un VACUUM es usar ANALYZE:

```
ANALYZE [ VERBOSE ] [ table_name [ ( column_name [, ...] ) ] ]
```

Si se ejecuta sin mencionar alguna tabla entonces se actualizarán las estadísticas de todas las tabla de la base de datos.

https://www.postgresql.org/docs/10/static/sql-analyze.html

Los índices

Los índices nos permiten generar ordenamientos de datos para realizar una búsqueda mucho más rápida sobre los datos.

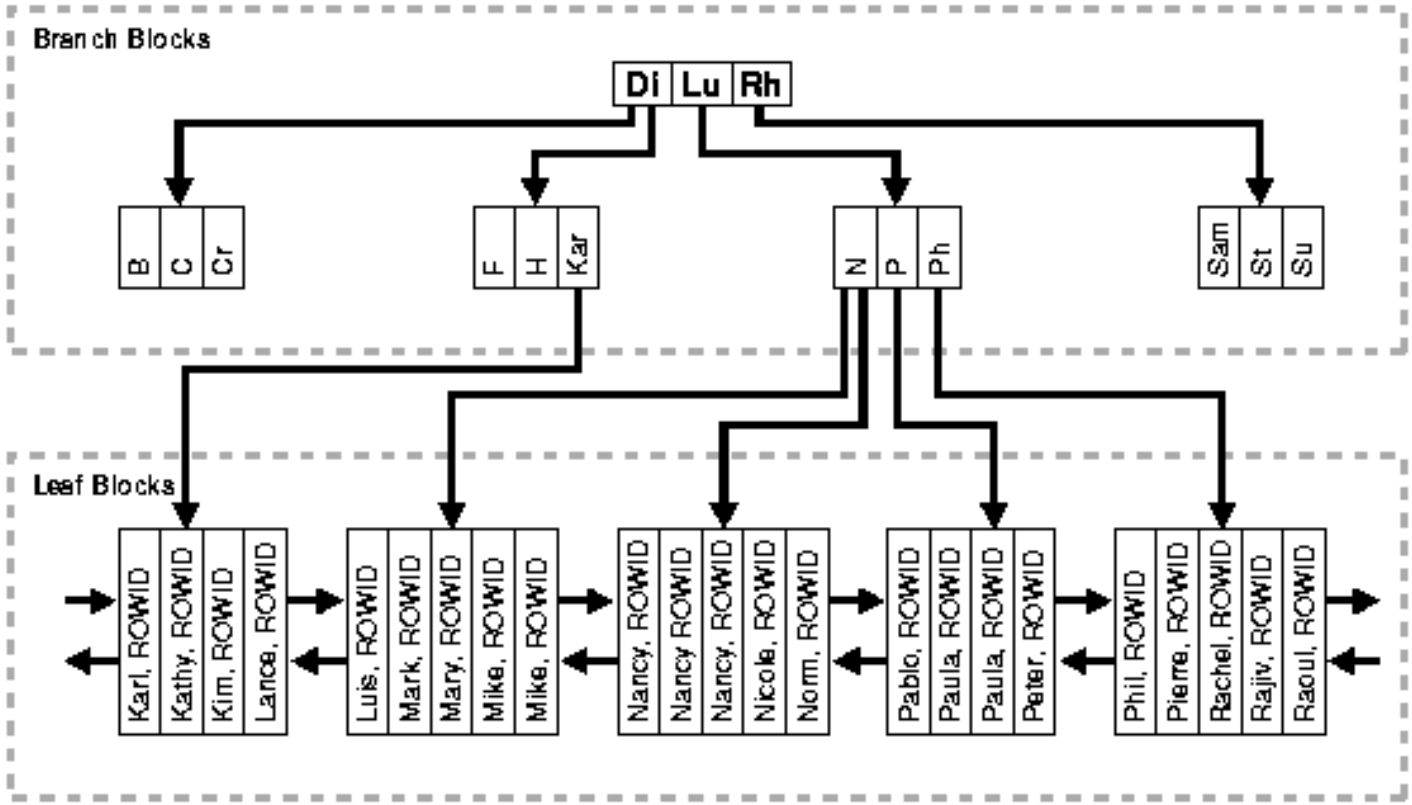
Los índices en sencillo un ordenamiento de los datos (ascendente o descendente) en páginas especiales que tienen punteros a los registros físicos.

El algoritmo más utilizado para los índices es el BTREE (árboles binarios).

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ name ] ON table_name
[ USING method ]
    ( { column_name | ( expression ) } [ COLLATE collation ]
[ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
    [ WITH ( storage_parameter = value [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ]
```

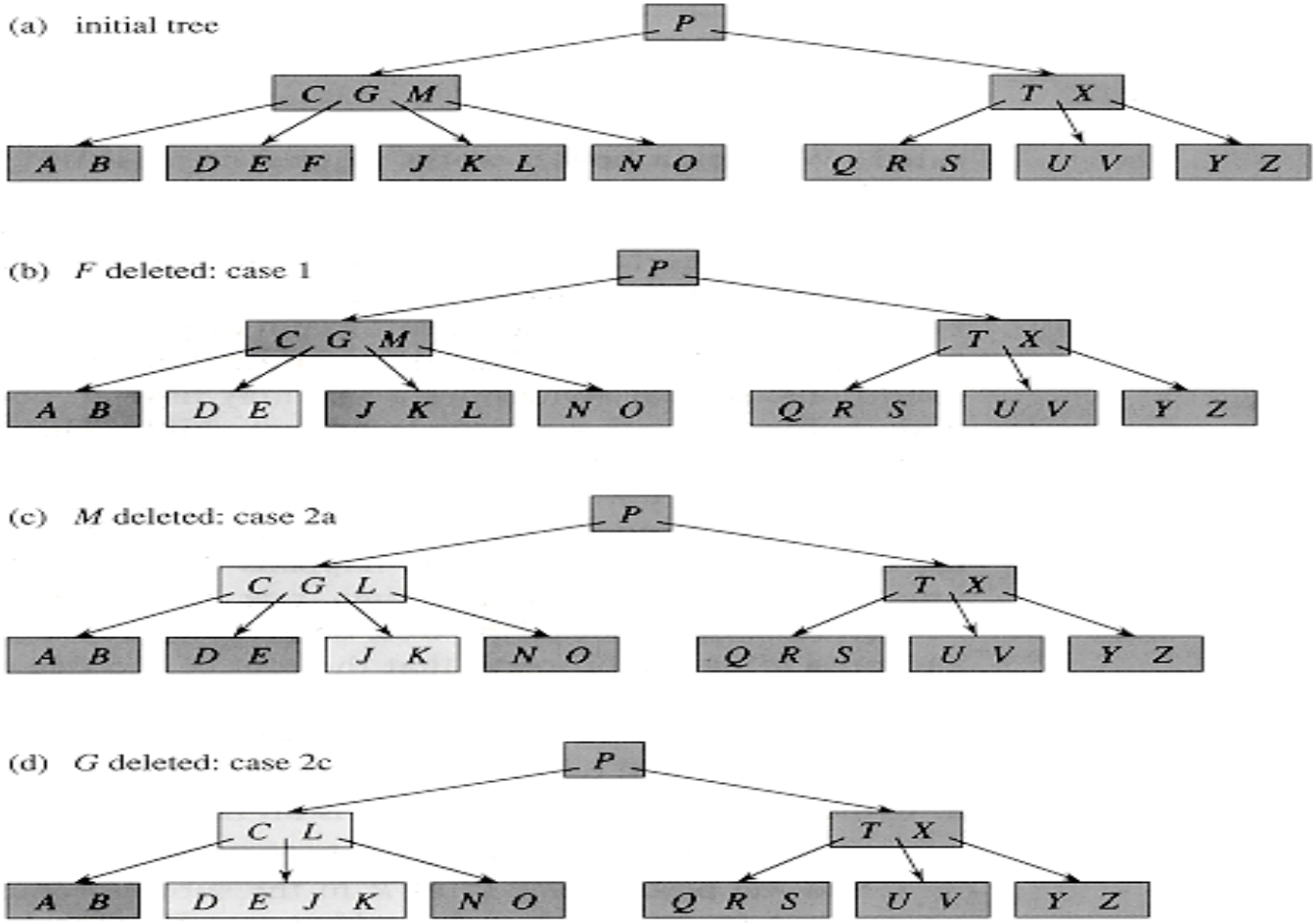
Los índices

Los índices son estructuras de datos, generalmente árboles binarios, creados para acelerar el acceso a los datos.



Los índices

Generalmente el algoritmo más utilizado es el de B-TREE.



Los índices

```
prueba2=> create table tablita (id serial, valores int);
CREATE TABLE


prueba2=> insert into tablita (valores) values
( generate_series(1,10000) * random());
INSERT 0 10000


prueba2=> select count(distinct valores) from tablita;
count
-------
 4944     ← a cada uno le mostrará un número diferente
(1 row)
```

En este ejemplo hemos creado una tabla y la llenamos con valores correlativos en un campo y aleatorios en el otro, luego solicitamos cuantos valores aleatorios únicos se han creado.

Los índices

```
prueba2=> explain analyze select * from tablita where valores = 10;
                          QUERY PLAN
-----------------------------------------------------------------------
Seq Scan on tablita  (cost=0.00..170.00 rows=13 width=8) (actual
time=0.037..0.939 rows=13 loops=1)
  Filter: (valores = 10)
  Rows Removed by Filter: 9987
Planning time: 0.053 ms
Execution time: 0.952 ms
```

Como se puede ver en esta consulta, la base de datos a realizado una búsqueda secuencial sobre nuestros datos, esto se ve en l parte que menciona: “Seq Scan on tablita”

Los índices

```
prueba2=> create index idxTablita on tablita(valores);
CREATE INDEX

prueba2=> explain analyze select * from tablita where valores = 10;
                           QUERY PLAN
-----------------------------------------------------------------
Index Scan using idxtablita on tablita  (cost=0.29..32.94 rows=13
width=8) (actual time=0.017..0.024 rows=13 loops=1)
  Index Cond: (valores = 10)
Planning time: 0.121 ms
Execution time: 0.038 ms
```

Ahora lo que hemos realizado es crear un índice a la columna "valores", al hacerlo nuestra consulta demora muchísimo menos tiempo en ejecutarse, esto se puede visualizar al comprar los datos de "Execution time".

Los índices

Una tabla puede tener muchos índices, pero debe tomarse en cuenta que si tiene demasiados cualquier actualización de datos (insert, update o delete) tomará mucho tiempo, cada vez que ejecutamos una de estas operaciones aparte de actualizar datos es necesario actualizar los índices.

Un índice puede estar conformado por más de un campo.

```
prueba2=> alter table tablita add column valores2 int;
ALTER TABLE

prueba2=> update tablita set valores2 = 10000 * random();
UPDATE 10000
                                        ^
prueba2=> select count(distinct valores2) from tablita;
count
-------
 6294
```

Los índices

```
prueba2=> explain analyze select * from tablita where valores = 10
and valores2=20;
                        QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using idxtablita on tablita  (cost=0.29..35.19 rows=1
width=12) (actual time=0.016..0.016 rows=0 loops=1)
   Index Cond: (valores = 10)
   Filter: (valores2 = 20)  Rows Removed by Filter: 13

prueba2=> explain analyze select * from tablita where valores2 = 20
and valores=10;
                        QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using idxtablita on tablita  (cost=0.29..35.19 rows=1
width=12) (actual time=0.032..0.032 rows=0 loops=1)
   Index Cond: (valores = 10)
   Filter: (valores2 = 20)  Rows Removed by Filter: 13
```

En este caso se toma el índice como apoyo a la consulta aunque los dos campos no estén indexados, el REWRITE se encarga de reordenar la consulta

Los índices

```
prueba2=> drop index idxtablita;
prueba2=> create index idxtablita on tablita(valores, valores2);

prueba2=> explain analyze select * from tablita where valores = 10
and valores2=20;
                              QUERY PLAN
-----------------------------------------------------------------------
Index Scan using idxtablita on tablita  (cost=0.29..8.30 rows=1
width=12) (actual time=0.011..0.011 rows=0 loops=1)
  Index Cond: ((valores = 10) AND (valores2 = 20))

prueba2=> explain analyze select * from tablita where valores2 = 20
and valores=10;
                              QUERY PLAN
-----------------------------------------------------------------------
Index Scan using idxtablita on tablita  (cost=0.29..8.30 rows=1
width=12) (actual time=0.010..0.010 rows=0 loops=1)
  Index Cond: ((valores = 10) AND (valores2 = 20))
```

Aquí borramos el índice y lo volvemos a crear pero ahora con los dos campos, la búsqueda se optimiza.

Los índices

```
prueba2=> explain analyze select * from tablita where valores = 10;
                                QUERY PLAN
-----------------------------------------------------------------------
Bitmap Heap Scan on tablita  (cost=4.39..42.42 rows=13 width=12)
(actual time=0.018..0.026 rows=13 loops=1)
  Recheck Cond: (valores = 10)
  Heap Blocks: exact=6
  ->  Bitmap Index Scan on idxtablita  (cost=0.00..4.38 rows=13
width=0) (actual time=0.012..0.012 rows=13 loops=1)
        Index Cond: (valores = 10)

prueba2=> explain analyze select * from tablita where valores2 = 20;
                                QUERY PLAN
-----------------------------------------------------------------------
Index Scan using idxtablita on tablita  (cost=0.29..202.30 rows=2
width=12) (actual time=0.221..0.221 rows=0 loops=1)
  Index Cond: (valores2 = 20)
```

En casos de que una consulta se pueda apoyar en un índice existente la base de datos lo tomará, aunque puede ser que determine que usarlo no es la mejor ruta de búsqueda.

Los índices

Los índices compuestos tiene el límite de hasta 32 campos en su composición, considere al momento de las consultas usar el mismo orden de columnas con el cual creo el índice.

En versiones anteriores a la 9 de PostgreSQL, los datos NULL no se indexaban.

Podemos crear índices más complejos como por ejemplo:

- `Create index idx_tabla on tabla ( campo1 || campo 2);`
- `Create index idx_tabla on tabla ( substr(campo1,5,10) );`
- `Create index idx_tabla on tabla ( upper(campo1));`

Para obtener provecho de esto las consultas deben hacer el pedido de la data en la estructura en la que se ha creado el índice.

Los índices

Se pueden aplicar ordenes especificos a los índices:

```
Create index idx_nombre on tabla1 ( campo1 DESC);

Create index idx_nombre on tabla1 ( campo1 ASC);

Create index idx_nombre on tabla1 ( campo1 NULLS FIRST);

Create index idx_nombre on tabla1 ( campo1 NULLS LAST);
```

Los índices - CONCURRENTLY

El proceso de generación de un índice normalmente bloquea la tabla para esrituras al momento de crearlo, sin embargo en un entorno de producción esto podría ser poco conveniente. Para solventar este problema existe la opción CONCURRENTLY la cual nos permite generar los índices en “2 pasadas”, sin afectar las escrituras, la generación del índice demora más pero no interfiere con las operaciones de escritura en la tabla, obviamente el índice no esta disponible hasta terminar el proceso.

```
CREATE INDEX CONCURRENTLY INDEX_NAME ON TABLA(CAMPO);
```

TIMING, permite visualizar el tiempo de ejecución de un query.

```
prueba=# \timing
El despliegue de duración está activado.
prueba=# select sum(id) from tablita;
 sum
-----
  80
(1 fila)

Duración: 35,951 ms
prueba=# explain analyze select sum(id) from tablita;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------
Aggregate  (cost=40.00..40.01 rows=1 width=4) (actual time=0.040..0.041 rows=1
loops=1)
   ->  Seq Scan on tablita  (cost=0.00..34.00 rows=2400 width=4) (actual
time=0.007..0.018 rows=8 loops=1)
 Total runtime: 0.103 ms
(3 filas)

Duración: 26,842 ms
```

Como se puede ver el tiempo de ejecución de un query realmente tenemos que capturarlo con un EXPLAIN ANALYZE.

Interpretando un Plan de Ejecución

Explain permite visualizar el plan de ejecución de un query, el plan de ejecución son los pasos que sigue la dmbs para procesar la consulta, tabla por tabla con la que trabaja, las uniones, tipos de índices que utiliza, tuplas movidas, etc.

https://www.postgresql.org/docs/10/static/sql-explain.html

```
prueba4=# explain select * from regla;
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on regla  (cost=0.00..18.00 rows=800 width=72)
(1 row)
```

Interpretando un Plan de Ejecución

Para ver los comandos usados y los tiempos de respuesta

```
prueba4=# explain analyze select * from regla;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------
Seq Scan on regla  (cost=0.00..18.00 rows=800 width=72) (actual time=0.012..0.016
rows=2 loops=1)
 Total runtime: 0.070 ms
```

Informa los campos requeridos en la consulta

```
prueba4=# explain verbose select * from regla;
                            QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on public.regla  (cost=0.00..18.00 rows=800 width=72)
   Output: id, nombre

prueba4=# explain analyze verbose select * from regla;
                                                      QUERY PLAN
-------------------------------------------------------------------------------------------------
Seq Scan on public.regla  (cost=0.00..18.00 rows=800 width=72) (actual
time=0.010..0.013 rows=2 loops=1)
   Output: id, nombre
 Total runtime: 0.050 ms
```

Interpretando un Plan de Ejecución

Los Explain siempre se deben leer de “adentro hacia afuera”.

```
prueba4=# explain select * from factura_cab a  join factura_det b on a.id
=b.fac_id;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Merge Join  (cost=260.93..525.73 rows=17120 width=29)
   Merge Cond: (b.fac_id = a.id)
   ->  Sort  (cost=111.15..115.15 rows=1600 width=21)
         Sort Key: b.fac_id
         ->  Seq Scan on factura_det b  (cost=0.00..26.00 rows=1600 width=21)
   ->  Sort  (cost=149.78..155.13 rows=2140 width=8)
         Sort Key: a.id
         ->  Seq Scan on factura_cab a  (cost=0.00..31.40 rows=2140 width=8)
(8 rows)
```

Interpretando un Plan de Ejecución

El primer explain fue hecho sin hacer un VACUMM a las tablas, en el segundo se procedió a ello, nótese las diferencia en las cifras, esto se debe a las estadísticas.

```
prueba4=# explain select * from factura_cab a  join factura_det b on a.id
=b.fac_id;
                                QUERY PLAN
-------------------------------------------------------------------------
 Hash Join  (cost=1.07..2.16 rows=4 width=25)
   Hash Cond: (b.fac_id = a.id)
   ->  Seq Scan on factura_det b  (cost=0.00..1.04 rows=4 width=17)
   ->  Hash  (cost=1.03..1.03 rows=3 width=8)
         ->  Seq Scan on factura_cab a  (cost=0.00..1.03 rows=3 width=8)
(5 rows)
```

Interpretando un Plan de Ejecución

Para determinar si un índice debe ser utilizado, PostgreSQL debe tener estadísticas sobre la tabla. Estas estadísticas se recolectan mediante VACUUM ANALYZE, o simplemente ANALYZE. Usando las estadísticas, el optimizador sabe cuántas son las filas en la tabla, y puede determinar mejor si los índices deben utilizarse. Las estadísticas son también valiosas en la determinación de un orden óptimo y métodos de unión. La recolección de Estadísticas debe ser realiza periódicamente como el cambio de contenido de la tabla.

El costo:

```
(cost=149.78..155.13 rows=2140 width=8)
149.78    <-- costo inicial de traer la primera tupla
155.13    <-- costo total estimado
Rows=2140 <-- filas de salida
Width=8   <-- peso promedio en bytes de las filas de salida
```

El costo total estimado se calcula sobre la siguiente formula
(disk pages read * seq_page_cost) + (rows scanned * cpu_tuple_cost)

```
seq_page_cost  <-- costo de acceso a las páginas de la data
(1.00 default)
cpu_tuple_cost <-- costo del proceso de cada fila durante la consultas
(0.01 default)
```

Interpretando un Plan de Ejecución

```
DBNAME=# create table masivo ( id serial, numero bigint);
CREATE TABLE                    <-- Insertamos 1'572,864 registros.

DBNAME=# select count(*) from masivo;
  count
---------
 1572864

DBNAME=# select relpages, reltuples::bigint from pg_class where
relname = 'masivo';
 relpages |  reltuples
----------+-------------
     8502 | 1572864
```

Interpretando un Plan de Ejecución

```
DBNAME=# update masivo set numero = 3 where numero=2; <-- UPDATE 262144

DBNAME=# select relpages, reltuples::bigint from pg_class where
relname = 'masivo';
 relpages | reltuples
----------+-----------
     9919 |   1572864

DBNAME=# update masivo set numero = 2 where numero=1; <-- UPDATE 262144

DBNAME=# select relpages, reltuples::bigint from pg_class where
relname = 'masivo';
 relpages | reltuples
----------+-----------
    10057 |   1572864

DBNAME=# explain select * from masivo;
                          QUERY PLAN
-------------------------------------------------------------
 Seq Scan on masivo  (cost=0.00..25785.64 rows=1572864 width=12)
```

Interpretando un Plan de Ejecución

El tiempo:

```
(actual time=0.007..0.018 rows=8 loops=1)
0.007     <-- tiempo de espera inicial que toma para empezar a
              procesar el query
0.018     <-- tiempo total en segundos que le toma procesar la
              consulta
Rows=8    <-- filas esperadas de retorno, una diferencia entre este y
              los ROWS del COST indican un posible problema en el
              optimizador interno de PostgreSQL para resolver la
              consulta.
loops=1   <-- En algunos casos, especialmente en JOINs es necesario
              ejecutar la consulta varias veces, los tiempos y ROWs
              deben ser multiplicados por el número de LOOPs para
              tener los totales reales.
```

Interpretando un Plan de Ejecución

```
Seq Scan on factura_cab a  (cost=0.00..1.03 rows=3 width=8)

Seq Scan          <-- indica que esta llevando a cabo una búsqueda
                      secuencial en la tabla, bien porque no existe un
                      índice o bien porque no la necesita.
on factura_cab a  <-- nos indica que esta buscando sobre la tabla
                      “factura” con alias “a”.

Hash  (cost=1.03..1.03 rows=3 width=8)

Hash              <-- significa que esta creando un mapa de datos e
                      índices con el fin de facilitar búsquedas

Sort  (cost=149.78..155.13 rows=2140 width=8)
        Sort Key: a.id

SORT KEY          <-- nos índica por que campo esta realizando un
                      ordenamiento y posteriormente el costo del
                      ordenamiento
```

Interpretando un Plan de Ejecución

```
 Merge Join  (cost=260.93..525.73 rows=17120 width=29)
   Merge Cond: (b.fac_id = a.id)

 Hash Join  (cost=1.07..2.16 rows=4 width=25)
   Hash Cond: (b.fac_id = a.id)
```

Tanto en el caso Merge Join como en el caso Hash Join el dbms esta realizando la unión de las dos tablas, nos indica el costo para cada caso.

Lo que tenemos que buscar siempre es que los COSTOS de nuestras consultas sean lo más bajos posibles, no siempre una búsqueda secuencial puede tener un costo mayor que una búsqueda indexada, eso dependerá de la data y estructura de las tablas.

Es importante escribir querys limpios, dar el menor trabajo posible al analizador de querys de PostgreSQL para que la resolución del plan de ejecución sea la menor posible.

Interpretando un Plan de Ejecución

Otros operadores que podemos encontrar son:

Nested Loop	**<--**	**Se combinan dos tablas utilizando bucles anidados. Por cada fila de la tabla padre, se recorren todas las filas de la tabla hija.**
Index scan	**<--**	**búsqueda sobre un índice**
Unique	**<--**	**elimina registros de valores duplicados**
Limit	**<--**	**indica que se esta pidiendo una cantidad limitada de registros, aparece cuando se ha usado el operador en la consulta**
Aggregate	**<--**	**concatenación de columnas**
Subquery Scan	**<--**	**indica una subconsulta dentro del contexto de un UNION**
Subplan	**<--**	**indica una subconsulta dentro de un subquery**
Append	**<--**	**indica que se están uniendo 2 tablas, aparece cuando se uso el operador UNION en la consulta**
Tid Scan (raro)		
Group	**<--**	**indica que se hace un GROUP BY**
Result	**<--**	**indica una operación que no mueve datos**
Materialize	**<--**	**se da en caso de que existan subconsultas que se repiten y el dbms decide almacenar los resultados obtenidos para su reuso.**

Interpretando un Plan de Ejecución

La información de los explain se puede exportar a diversos formatos:

```
prueba=# explain (format json) select * from masivo;
            QUERY PLAN
---------------------------------
 [                               +
   {                             +
     "Plan": {                   +
       "Node Type": "Seq Scan",  +
       "Relation Name": "masivo",+
       "Alias": "masivo",        +
       "Startup Cost": 0.00,     +
       "Total Cost": 25785.64,   +
       "Plan Rows": 1572864,     +
       "Plan Width": 12          +
     }                           +
   }                             +
 ]
```

```
prueba=# explain (format yaml) select * from
masivo;
          QUERY PLAN
----------------------------
 - Plan:                     +
     Node Type: "Seq Scan"   +
     Relation Name: "masivo"+
     Alias: "masivo"         +
     Startup Cost: 0.00      +
     Total Cost: 25785.64    +
     Plan Rows: 1572864      +
     Plan Width: 12
(1 row)
```

```
prueba=# explain (format xml) select * from masivo;
                       QUERY PLAN
-------------------------------------------------------
 <explain xmlns="http://www.postgresql.org/2009/explain">+
   <Query>                                              +
     <Plan>                                             +
       <Node-Type>Seq Scan</Node-Type>                  +
       <Relation-Name>masivo</Relation-Name>            +
       <Alias>masivo</Alias>                            +
       <Startup-Cost>0.00</Startup-Cost>                +
       <Total-Cost>25785.64</Total-Cost>                +
       <Plan-Rows>1572864</Plan-Rows>                   +
       <Plan-Width>12</Plan-Width>                      +
     </Plan>                                            +
   </Query>                                             +
 </explain>
```

Interpretando un Plan de Ejecución

EXPLAIN (ANALYZE, BUFFERS) [Query]

Nos permite visualizar la cantidad de shared blockes usados.

HIT : lecturas eludidas porque la data estaba cacheada
READ : bloques leidos
DIRTIED: bloques previamente no-modificados que son cambiados por el query
WRITE : dirty-blocks dejados de lado del cache durante el proceso Del query.

```
db1=#explain (analyze,buffers) insert into t200(id) values
(generate_series(1,100000));

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
Insert on t200  (cost=0.00..5.01 rows=1000 width=0) (actual time=92.458..92.458
rows=0 loops=1)
   Buffers: shared hit=100885 read=442 dirtied=443
   ->  Result  (cost=0.00..5.01 rows=1000 width=0) (actual time=0.005..10.204
rows=100000 loops=1)
 Planning time: 0.017 ms
 Execution time: 105.709 ms
```

Interpretando un Plan de Ejecución

EXPLAIN (ANALYZE, BUFFERS) [Query]

```
db1=# explain (analyze,buffers) select * from t200 where id % 11 = 0;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------
 Seq Scan on t200  (cost=0.00..6018.00 rows=1500 width=4) (actual time=0.022..38.282 rows=26813
loops=1)
   Filter: ((id % 11) = 0)
   Rows Removed by Filter: 267034
   Buffers: shared hit=1518

db1=# explain (analyze,buffers) update t200 set id = 300*random() where id % 11 = 0;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------
 Update on t200  (cost=0.00..6029.25 rows=1500 width=6) (actual time=76.565..76.565 rows=0 loops=1)
   Buffers: shared hit=28354
   ->  Seq Scan on t200  (cost=0.00..6029.25 rows=1500 width=6) (actual time=0.015..53.414 rows=26813
loops=1)
         Filter: ((id % 11) = 0)
         Rows Removed by Filter: 267034
         Buffers: shared hit=1518

db1=# explain (analyze,buffers) delete from t200 where id % 11 = 0;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------
 Delete on t200  (cost=0.00..6018.00 rows=1500 width=6) (actual time=65.504..65.504 rows=0 loops=1)
   Buffers: shared hit=4008 dirtied=199
   ->  Seq Scan on t200  (cost=0.00..6018.00 rows=1500 width=6) (actual time=0.060..63.946 rows=2490
loops=1)
         Filter: ((id % 11) = 0)
         Rows Removed by Filter: 291357
         Buffers: shared hit=1518 dirtied=199
```

Interpretando un Plan de Ejecución

Un EXPLAIN puede ejecutarse para evaluar un INSERT, UPDATE o DELETE, un peuqeño truco para que cuando se ejecute un EXPLAIN en estos casos y no afecte los datos es hacer lo siguiente:

```
BEGIN;
  EXPLAIN ANALYZE ...;
ROLLBACK;
```

Queries Avanzados

Create Table - Select

Podemos crear tablas en base a otras tablas a través de una consulta SELECT.

```
pruebas=# create table tabla_base (codigo int, nombre char(10));
pruebas=# insert into tabla_base values (1,'A');
```

*** Crear una tabla de un select solo estructura**

```
pruebas=# create table tabla_base1 as select * from tabla_base where 1=2;
pruebas=# select * from tabla_base1;
  codigo | nombre
 --------+--------
```

*** Crear una tabla de un select con sus datos**

```
pruebas=# create table tabla_base2 as select * from tabla_base;
pruebas=# select * from tabla_base2;
  codigo |   nombre
 --------+------------
       1 | A
```

*** Crear una tabla de un select, solo ciertos campos**

```
pruebas=# create table tabla_base3 as select codigo from tabla_base;
pruebas=# select * from tabla_base3;
     codigo
    --------
        1
```

Queries Avanzados

Create Table - Herencias

INHERITS nos permite heredar a la estructura de nuestra nueva tabla la estructura de otra tabla.

```
prueba4=# create table tbl_padre (id integer, nombre varchar(100));
prueba4=# create table tbl_hijo (direccion varchar(200)) INHERITS
(tbl_padre);
CREATE TABLE
prueba4=# select * from tbl_hijo;
 id | nombre | direccion
----+--------+-----------

prueba4=# drop table tbl_padre cascade;
NOTICE:  drop cascades to table tbl_hijo
DROP TABLE
```

LIKE permite heredar la estructura de la tabla padre, más no existe dependencia directa con datos y si se borra la tabla padre no se pierde la estructura.

```
prueba4=# create table tbl_hijo2  (like tbl_padre);
prueba4=# create table tbl_hijo3  (like tbl_padre, casa varchar(300));
```

Queries Avanzados

Create Table - Herencias - SELECTs

Es posible limitar una consulta a que solo presente el contenido de la tabla principal sin considerar el contenido de sus tablas heredadas.

```
prueba=# create table ciudades ( nombre varchar(100));
prueba=# create table capitales ( capital boolean) inherits (ciudades);

prueba=# insert into ciudades values ('callao');
prueba=# insert into ciudades values ('piura');
prueba=# insert into capitales values ('lima', true);
prueba=# insert into capitales values ('buenos aires', true);

prueba=# select * from ciudades;
    nombre
--------------
 callao
 piura
 lima
 buenos aires

prueba=# select * from only ciudades;
 nombre
--------
 callao
 piura
```

Queries Avanzados

Casos especiales de Inserts

Es posible ejecutar una inserción de datos desde una consulta del tipo "Select".

```
prueba=# create table tbl_simple ( id serial, nombre varchar(10)); <-- insertar
                                                                      datos
prueba=# create table tbl_simple2 ( id serial, nombre varchar(10));

prueba=# insert into tbl_simple2 ( nombre) select nombre from tbl_simple;

prueba=# select * from tbl_simple2;
 id | nombre
----+---------
  1 | ernesto
  2 | juan
  3 | 111
  4 | 10chars

prueba=# insert into tbl_simple2 ( nombre) select nombre from tbl_simple
returning id;
 id
----
  5
  6
  7
  8
```

Queries Avanzados

Casos especiales de Inserts

Es posible insertar valores por defecto de esta forma:

```
prueba=# create table t1 (id int, valor char(10) default 'ernesto');
prueba=# insert into t1 (id, valor) values (1, DEFAULT);
prueba=# select * from t1;
 id |   valor
----+-----------
  1 | ernesto


prueba=# create table t2 (id int default -1, valor char(10) default 'ernesto');
prueba=# insert into t2  DEFAULT VALUES;
prueba=# select * from t2;
 id |   valor
----+-----------
 -1 | ernesto


prueba=# insert into t1  DEFAULT VALUES;
prueba=# select * from t1;
 id |   valor
----+-----------
  1 | ernesto
    | ernesto
```

Queries Avanzados

Casos especiales de Inserts

Es posible utilizar optimización de consultar temporales con querys del tipo WITH.

```
prueba=# create table T1 (id serial, dato int);
prueba=# create table T2 (id int, dato int);

prueba=# insert into t1 (dato) values (10), (20), (30);

prueba=# with tabla1 as (select * from t1) insert into t2 (id, dato)
select * from tabla1;

prueba=# select * from t2;
 id | dato
----+------
  1 |   10
  2 |   20
  3 |   30
```

En este caso se crea en tiempo de ejecución una tabla temporal llamada Tabla1.

Queries Avanzados

Casos especiales de Inserts

Se puede usar querys complejos y muchos cuando trabajamos con WITH.

```
prueba=# insert into t2 (dato) values  (11), (21), (31);
prueba=# with tabla1 as (select * from t1),
             tabla2 as (select * from t2)
          insert into t2 (id, dato) select * from tabla2 where
             dato not in  ( select dato from tabla1);

prueba=# select * from t2;
 id | dato
----+------
  1 |   10
  2 |   20
  3 |   30
    |   11
    |   21
    |   31
    |   11
    |   21
    |   31
```

Queries Avanzados

```
prueba=# explain analyze with tabla1 as (select * from t1), tabla2 as
(select * from t2) insert into t2 (id, dato) select * from tabla2
where dato not in  ( select dato from tabla1);
                                QUERY PLAN
-----------------------------------------------------------------------
Insert on t2  (cost=110.95..159.10 rows=1070 width=8) (actual
time=0.146..0.146 rows=0 loops=1)
   CTE tabla1
     ->  Seq Scan on t1  (cost=0.00..31.40 rows=2140 width=8) (actual
                  time=0.003..0.004 rows=3 loops=1)
   CTE tabla2
     ->  Seq Scan on t2 t2_1  (cost=0.00..31.40 rows=2140 width=8)
                  (actual time=0.011..0.014 rows=9 loops=1)
     ->  CTE Scan on tabla2  (cost=48.15..96.30 rows=1070 width=8)
                  (actual time=0.091..0.101 rows=6 loops=1)
         Filter: (NOT (hashed SubPlan 3))
         Rows Removed by Filter: 3
         SubPlan 3
           ->  CTE Scan on tabla1  (cost=0.00..42.80 rows=2140
                width=4) (actual time=0.008..0.011 rows=3 loops=1)
```

Casos especiales de Inserts

Esto hace lo mismo sin WITH:

```
prueba=# insert into t2 (id, dato) select * from t2 where dato not in  ( select dato
from t1);INSERT 0 12
Time: 16,261 ms
prueba=# explain analyze insert into t2 (id, dato) select * from t2 where dato not in
( select dato from t1);
                      QUERY PLAN
-----------------------------------------------------------------------
Insert on t2  (cost=36.75..73.50 rows=1070 width=8) (actual time=0.124..0.124 rows=0
loops=1)
   ->  Seq Scan on t2 t2_1  (cost=36.75..73.50 rows=1070 width=8) (actual
               time=0.048..0.066 rows=24 loops=1)
         Filter: (NOT (hashed SubPlan 1))
         Rows Removed by Filter: 3
         SubPlan 1
           ->  Seq Scan on t1  (cost=0.00..31.40 rows=2140 width=4) (actual
                  time=0.003..0.005 rows=3 loops=1)
```

Como puede observarse para similar resultado los patrones de de plan de ejecución varían.

WITH también puede usarse en un UPDATE, DELETE y por supuesto en un SELECT.

Queries Avanzados

Casos especiales de Inserts: ON CONFLICT (desde versión 9.5)

ON CONFLICT nos permite establecer que hacer cuando un INSERT viola alguno de los constraints de una tabla, de esta forma el motor no enviará un error cuando se viole este sino ejecutará una acción definida.

```
db1=# create table t1 (id int primary key);
db1=# insert into t1 values (1);

db1=# insert into t1 values (1);
ERROR:  duplicate key value violates unique constraint "t1_pkey"
                                                        ^
db1=# insert into t1 values (1) on conflict (id) do update set id = 2;
INSERT 0 1
db1=# select * from t1;
 id
----
  2
```

Casos especiales de Inserts: ON CONFLICT (desde versión 9.5)

ON CONFLICT nos permite establecer que hacer cuando un INSERT viola alguno de los constraints de una tabla, de esta forma el motor no enviará un error cuando se viole este sino ejecutará una acción definida.

```
db1=# insert into t1 values (3);
db1=# select * from t1;
 id
----
  2
  3

db1=# insert into t1 values (3) on conflict (id) do update set id = 4;
db1=# select * from t1;
 id
----
  2
  4

db1=# insert into t1 values (3) on conflict (id) do nothing;
db1=# select * from t1;
 id
----
  2
  4
  3
```

Queries Avanzados

Casos especiales de Update / Delete

Ejecutando un UPDATE de datos obtenidos por un SELECT en otra tabla.

```
prueba=# create table t1 ( id serial, dato int);
prueba=# insert into t1 (dato) values (10),(20),(30);
prueba=# create table t2 ( Xid serial, Xdato int);
prueba=# insert into t2 (xdato) values (100),(200),(300);

prueba=# select * from t1;
 id | dato
----+------
  1 |   10
  2 |   20
  3 |   30

prueba=# update t1 set dato = a.Xdato from (select Xid, Xdato from t2) a where
t1.id = a.Xid and t1.id<=2;

prueba=# select * from t1;
 id | dato
----+------
  3 |   30
  1 |  100
  2 |  200
```

Queries Avanzados

Casos especiales de Update / Delete

En DELETEs podemos usar USING para incluir en nuestra sentencia el uso de una tabla adicional sin tener que crear SUBQUERYs.

```
prueba=# select * from t1;
 id | dato
----+------
  3 |   30
  1 |  100
  2 |  200

prueba=# delete from t1 using t2 where t1.id = t2.Xid and t2.Xid = 3
returning id, dato;
 id | dato
----+------
  3 |   30

prueba=# select * from t1;
 id | dato
----+------
  1 |  100
  2 |  200
```

Queries Avanzados

Select – OVER

Permite un agrupamiento de datos a la vez que se disponen de las filas de la tabla al mismo tiempo, es una evolución de GROUP BY.

```
prueba=# select  * from tbl_window;
  mes   | nombre  | sueldo
--------+---------+--------
 201001 | ernesto |     10
 201002 | ernesto |     11
 201003 | ernesto |     12
 201001 | juan    |     13
 201003 | juan    |     15
 201002 | juan    |     14

prueba=# select nombre, mes, sueldo, sum(sueldo) as sueldo_total over (partition
by nombre) from tbl_window;
 nombre  |  mes   | sueldo | sueldo_total
---------+--------+--------+-------------
 ernesto | 201001 |     10 |  33
 ernesto | 201002 |     11 |  33
 ernesto | 201003 |     12 |  33
 juan    | 201001 |     13 |  42
 juan    | 201003 |     15 |  42
 juan    | 201002 |     14 |  42
```

Queries Avanzados

Select – OVER

Permite un agrupamiento de datos a la vez que se disponen de las filas de la tabla al mismo tiempo, es una evolución de GROUP BY.

```
prueba=# select nombre, mes, sueldo,
            sum(sueldo)over (partition by nombre) sueldo_total,
            sum(sueldo) over (partition by nombre) / 3 as
                sueldo_promedio,
            (sum(sueldo) over (partition by nombre) / 3 )- sueldo as
                desviacion_promedio
          from tbl_window;
```

nombre	mes	sueldo	sueldo_total	sueldo_promedio	desviacion_promedio
ernesto	201001	10	33	11	1
ernesto	201002	11	33	11	0
ernesto	201003	12	33	11	-1
juan	201001	13	42	14	1
juan	201003	15	42	14	-1
juan	201002	14	42	14	0

Queries Avanzados

Select - OVER

```
prueba=#select nombre, mes, sueldo,
             sum(sueldo) over (partition by nombre order by nombre desc, mes )
        from tbl_window;
 nombre  |  mes   | sueldo | sum
---------+--------+--------+-----
 juan    | 201001 |     13 |  13
 juan    | 201002 |     14 |  27
 juan    | 201003 |     15 |  42
 ernesto | 201001 |     10 |  10
 ernesto | 201002 |     11 |  21
 ernesto | 201003 |     12 |  33

prueba=# select nombre, mes, sueldo,
             rank() over (partition by nombre order by sueldo desc)
         from tbl_window;
 nombre  |  mes   | sueldo | rank
---------+--------+--------+------
 ernesto | 201003 |     12 |    1
 ernesto | 201002 |     11 |    2
 ernesto | 201001 |     10 |    3
 juan    | 201003 |     15 |    1
 juan    | 201002 |     14 |    2
 juan    | 201001 |     13 |    3
```

Queries Avanzados

Select – OVER - WINDOW

```
prueba=# select nombre, sueldo, mes,
            sum(sueldo) over ventana,
            avg(sueldo) over ventana
          from tbl_window window ventana as (partition by nombre);

 nombre  | sueldo |  mes   | sum |         avg
---------+--------+--------+-----+---------------------
 ernesto |     10 | 201001 |  33 | 11.000000000000000
 ernesto |     11 | 201002 |  33 | 11.000000000000000
 ernesto |     12 | 201003 |  33 | 11.000000000000000
 juan    |     13 | 201001 |  42 | 14.000000000000000
 juan    |     15 | 201003 |  42 | 14.000000000000000
 juan    |     14 | 201002 |  42 | 14.000000000000000
```

Queries Avanzados

Select – OVER - WINDOW

```
prueba=# select * from t3;
 persona | mes | sueldo
---------+-----+--------
 ernesto | 01  |    100
 ernesto | 02  |    140
 juan    | 01  |     99

prueba=# select persona, mes, sueldo, sum(sueldo)  over w
              from t3 window w as (partition by persona);
 persona | mes | sueldo | sum
---------+-----+--------+-----
 ernesto | 01  |    100 | 240
 ernesto | 02  |    140 | 240
 juan    | 01  |     99 |  99

prueba=# select persona, mes, sueldo, sum(sueldo)  over w, sum(sueldo) over j
              from t3
              window w as (partition by persona),
              j as (partition by mes);
 persona | mes | sueldo | sum | sum
---------+-----+--------+-----+-----
 ernesto | 01  |    100 | 240 | 199
 juan    | 01  |     99 |  99 | 199
 ernesto | 02  |    140 | 240 | 140
```

El detalle de las funciones aplicables con WINDOW puede ser encontrado aquí https://www.postgresql.org/docs/10/static/functions-window.html

Queries Avanzados

Select - Variaciones de Distinct

```
prueba=# select distinct nombre, id from tbl_simple2;
   nombre   | id
------------+----
 32110chars |  8
 321111     |  3
 321111     |  7
 321juan    |  2
 321ernesto |  5
 321juan    |  6
 321ernesto |  1
 32110chars |  4

prueba=# select distinct on (nombre) nombre, id from tbl_simple2 order by 1 desc, 2;
   nombre   | id
------------+----
 321juan    |  2
 321ernesto |  1
 321111     |  3
 32110chars |  4

prueba=# select distinct on (nombre) nombre, id from tbl_simple2 order by 1, 2 desc;
   nombre   | id
------------+----
 32110chars |  8
 321111     |  7
 321ernesto |  5
 321juan    |  6
```

Queries Avanzados

SELECT- FOR UPDATE / FOR SHARE

FOR UDATE bloqueda el acceso a los registros que se solicitan de tal manera que el query no desbloqueara los registros hasta que haya culminado la transacción en curso.

```
prueba=# select * from tbl_simple2 limit 2 for update;
```

FOR SHARE permite que se modifiquen los datos.

Queries Avanzados

SELECT- GROUPING SETS

Permite establecer criterios de agrupamiento independientes.

```
prueba=# select * from t3;
 persona | mes | sueldo
---------+-----+--------
 ernesto | 01  |    100
 ernesto | 02  |    140
 juan    | 01  |     99
 ernesto | 01  |    300
 ernesto | 02  |    300
 juan    | 01  |     10

prueba=# select persona, mes, sum(sueldo) from t3 group by persona, mes;
 persona | mes | sum
---------+-----+-----
 juan    | 01  | 109
 ernesto | 02  | 440
 ernesto | 01  | 400

prueba=# select persona, mes, sum(sueldo) from t3 group by grouping sets ((persona), (mes));
 persona | mes | sum
---------+-----+-----
 ernesto |     | 840
 juan    |     | 109
         | 01  | 509
         | 02  | 440
```

Queries Avanzados

SELECT- CUBE y ROLLUP

```
prueba=# select persona,mes, sum(sueldo) from t3 group by cube (persona, mes);
 persona | mes | sum
---------+-----+-----
         |     | 949
 juan    | 01  | 109
 ernesto | 02  | 440
 ernesto | 01  | 400
 ernesto |     | 840
 juan    |     | 109
         | 01  | 509
         | 02  | 440
(8 rows)

prueba=# select persona,mes, sum(sueldo) from t3 group by rollup (persona, mes);
 persona | mes | sum
---------+-----+-----
         |     | 949
 juan    | 01  | 109
 ernesto | 02  | 440
 ernesto | 01  | 400
 ernesto |     | 840
 juan    |     | 109
```

Queries Avanzados

SELECT - SIMILAR TO

Hacemos búsquedas por expresiones regulares

```
prueba4=# select * from log_regla;
           fecha            | id |   nombre   | estado
----------------------------+----+------------+--------
 2010-10-08 20:26:37.200866 |  2 | Ernesto    | I
 2010-10-08 20:49:24.691783 |  8 | pedro      | I
 2010-10-08 21:00:08.529349 |  1 | alejandro  | U
 2010-10-08 21:21:55.82194  |  5 | pepelucho  | U

prueba4=# select * from log_regla where nombre similar to '%(a|c)%';
                   // contiene a ó c
           fecha            | id |   nombre   | estado
----------------------------+----+------------+--------
 2010-10-08 21:00:08.529349 |  1 | alejandro  | U
 2010-10-08 21:21:55.82194  |  5 | pepelucho  | U

prueba4=# select * from log_regla where nombre similar to '%(x|f)%';
 fecha | id | nombre | estado
-------+----+--------+--------
```

Queries Avanzados

SELECT - SIMILAR TO

Inicia con d ó f

```
prueba4=# select * from log_regla where nombre similar to '(d|f)%';
 fecha | id | nombre | estado
-------+----+--------+--------
(0 rows)
```

```
prueba4=# select * from log_regla where nombre similar to '(p|f)%';
           fecha            | id |   nombre   | estado
----------------------------+----+------------+--------
 2010-10-08 20:49:24.691783 |  8 | pedro      | I
 2010-10-08 20:49:43.711889 |  9 | pedro      | I
 2010-10-08 21:21:55.82194  |  5 | pepelucho  | U
 2010-10-08 21:22:06.759216 |  5 | pepelucho1 | U
```

Alfabéticos de al menos 5 letras

```
prueba4=# select * from log_regla where nombre similar to '[a-z]{5}';
           fecha            | id | nombre | estado
----------------------------+----+--------+--------
 2010-10-08 20:49:24.691783 |  8 | pedro  | I
 2010-10-08 20:49:43.711889 |  9 | pedro  | I
```

Queries Avanzados

SELECT – ORDER BY + FUNCIONES

```
prueba=# create table orden ( campo int);
prueba=# insert into orden values (0),(8),(4),(3),(15),(7), (9);

create function divisor (valor int) returns int as $$
begin
 return valor/3;
end;
$$ language plpgsql;
```

```
prueba=# select *, divisor(campo) from orden order by divisor(campo);
 campo | divisor
-------+---------
     0 |       0
     4 |       1
     3 |       1
     8 |       2
     7 |       2
     9 |       3
    15 |       5
```

También podemos usar:

```
prueba=# select *, divisor(campo) from orden order by 2;
```

<-- apunta al 2do campo, el plan de ejecución es el mismo.

Queries Avanzados

CASE WHEN : Nos permite hacer evaluaciones de datos dentro de un Query para retornar uno u otro valor según sea el caso.

```
prueba=# create table opciones (campo int);
prueba=# insert into opciones values ( 1),(2),(1),(2),(3),(4);

prueba=# select
          case
           when campo = 1 then 'uno'
           when campo = 2 then 'dos'
           else 'ni uno ni dos'
          end
       from opciones;

     case
--------------
 uno
 dos
 uno
 dos
 ni uno ni dos
 ni uno ni dos
```

Copy

Copy es una excelente alternativa cuando debemos cargar información de millares de registros en una tabla, es mucho más eficiente que hacer varios insert o un insert de un select.

Copy además nos permite generar un archivo con datos de una tabla o de una consulta.

```
prueba=# copy tbl_simple2 to '/tmp/uno.txt';
COPY 8

prueba=# copy tbl_simple2(nombre) to '/tmp/uno.txt';
COPY 8

prueba=# copy (select * from tbl_simple2) to '/tmp/uno.txt';
COPY 8

prueba=# copy (select * from tbl_simple2) to stdin;
1       321ernesto
2       321juan
3       321111
4       32110chars
5       321ernesto
6       321juan
7       321111
8       32110chars
```

Copy

```
prueba=# select * from tbl_simple;
 id | nombre
----+--------
(0 filas)

prueba=# copy tbl_simple from '/tmp/uno.txt';
COPY 8

prueba=# select * from tbl_simple;
 id |   nombre
----+------------
  1 | 321ernesto
  2 | 321juan
  3 | 321111
  4 | 32110chars
  5 | 321ernesto
  6 | 321juan
  7 | 321111
  8 | 32110chars
(8 filas)
```

https://www.postgresql.org/docs/10/static/sql-copy.html