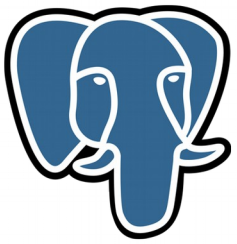
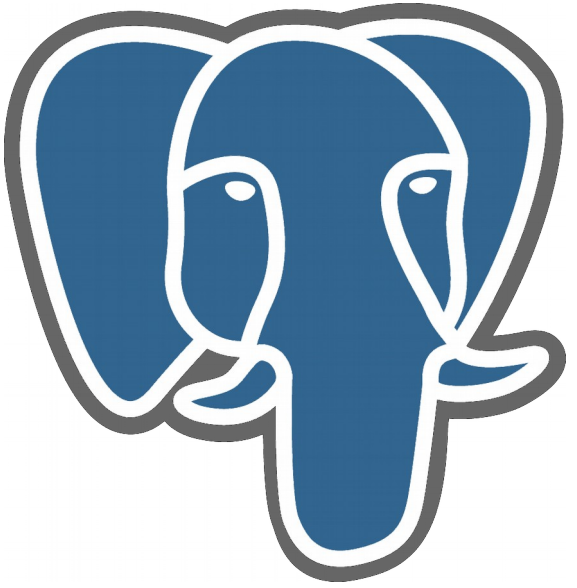


TodoPostgreSQL

Administración de PostgreSQL
MVCC



INSTRUCTOR:

José Segovia <info@todopostgresql.com>

MVCC - Outline

- En este módulo aprenderemos
  - Definición de Transacción
  - Efectos de la Concurrencia sobre las Transacciones
  - Niveles de Aislamiento de las Transacciones
  - Descripción del Control de Concurrencia Multiversión (MVCC)
  - Diferencias entre MVCC y Bloqueo
  - Ejemplo MVCC
  - Mostrar bloqueo
  - cmin/max xmin/max
  - Cómo se usan

MVCC: Transacciones

- Lo más importante de una transacción es que agrupa múltiples pasos en un único evento ACID caracterizado por:
  - Una operación todo o nada

    (*Atomicity*, atomicidad).
  - Sólo se escriben datos válidos en la base de datos (*Consistency*, consistencia).
  - Los pasos intermedios de otros estados no son visibles a otras transacciones concurrentes (*Isolation*, aislamiento).
  - En el caso en el que ocurriese algún fallo que impidiera completar la transacción, ninguno de los pasos afectaría de ninguna manera a la base de datos

    (*Durability*, durabilidad).

MVCC: Concurrencia y Transacciones

- Concurrencia – dos o más sesiones que acceden a los mismos datos de forma simultánea

- Cada transacción ve una foto de los datos (versión de la base de datos) como era anteriormente.

- Aislamiento de Transacción – Protege a la transacción de visualizar datos "inconsistentes" (que están siendo actualizados por otra transacción).

- Sin bloqueo – los lectores no bloquean a los escritores y los escritores no bloquean a los lectores.

MVCC: Niveles de Aislamiento

- Se necesitan los niveles de aislamiento para prevenir una serie de problemas no deseados
  - Se describen en las siguientes transparencias
  - Fundamental para permitir concurrencia
  - Afectan a la Consistencia de los datos
  - Relacionado con el Aislamiento de transacciones

- Niveles de aislamiento definidos por el estándar ANSI SQL
  - PostgreSQL los soporta todos a partir de 9.0

MVCC: Niveles de Aislamiento (cont.)

- Control de aislamiento:
  - SET TRANSACTION ISOLATION LEVEL
  - Sólo afecta a la transaccion actual
  - No puede ser cambiado tras la primera SELECT o instrucción DML de la transacción

```
SET TRANSACTION transaction_mode [, ...]
SET TRANSACTION SNAPSHOT snapshot_id
SET SESSION CHARACTERISTICS AS TRANSACTION transaction_mode [, ...]

where transaction_mode is one of:

    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ
COMMITTED | READ UNCOMMITTED }
   READ WRITE | READ ONLY
   [ NOT ] DEFERRABLE
```

MVCC: Aislamiento - Problemas

Lectura sucia (*dirty read*):

- Una transacción lee datos escritos por una transacción concurrente no consignada.

- **Lectura no-repetible** (*non-repeatable read*):
  - Una transacción re-lee datos que ha leído anteriormente y encuentra que los datos han sido modificados por otra transacción (que ha hecho un commit desde la lectura inicial).

MVCC: Aislamiento Problemas (cont.)

- **Lectura fantasma** (*phantom read*):
  - Una transacción re-ejecuta una consulta que devuelve cierto número de filas --- que satisfacen una serie de condiciones

    (o una consulta equivalente)
  - El conjunto de las filas que satisfacen las condiciones ha cambiado debido a otra transacción que ha hecho un commit recientemente.

MVCC: Niveles de Aislamiento

- Definidos por el estándar SQL

- READ COMMITED
  - Una transacción sólo puede ver tuplas consignadas antes de su inicio

- REPEATABLE READ
  - Todas las sentencias de la transacción sólo pueden ver las filas consignadas antes de la primera consulta o actualización de esta transacción

MVCC: Niveles de Aislamiento (cont.)

- Definidos por el estándar SQL (cont)
- SERIALIZABLE
  - Todas las sentencias de la transacción actual pueden ver exclusivamente filas consignadas antes de que se ejecute la primera consulta o modificación de datos en la transacción actual
  - Si un patrón de lecturas y escrituras entre transacciones concurrentes es tal que no se podría haber dado en una ejecución serie (una tras otra) de las mismas transacciones, una de ellas será abortada (rollback) con un error *serialization_failure*
- READ UNCOMMITED
  - No hay aislamiento
  - Postgres provee READ COMMITED

MVCC: Aislamiento de las transacciones

Nivel de Aislamiento	Lectura Sucia	Lectura No-Repetible	Lectura Fantasma
Lectura no consignada	Posible	Posible	Posible
Lectura consignada	No posible	Posible	Posible
Lectura reiterable	No posible	No posible	Posible
Serializable	No Posible	No Posible	No Posible

MVCC: ¿Qué bases de datos lo soportan?

- MVCC: Multiversion Concurrency Control
- Soportado en las bases de datos "serias":
  - Oracle
  - DB2
  - MySQL con InnoDB
  - Informix
  - Firebird
  - MSSQL (opcionalmente, desactivado por defecto)

MVCC: Snapshots

Instantáneas MVCC

- Las instantáneas MVCC controlan qué tuplas son visibles por las diferentes Sentencias SQL
  - Son "fotos" del estado de las tuplas
  - Requiere mantener estado de las tuplas

- A medida que se van modificando los datos (INSERT, UPDATE, DELETE)
  - Filas aparecen y dejan de estar visibles
  - Origina "tuplas muertas" (invisibles para TODAS las transacciones actuales y futuras)
  - Requiere de mantenimiento

MVCC: Snapshots (cont.)

- El aislamiento de transacciones Postgres se pone a LECTURA CONSIGNADA por defecto...
  - Cuando una transacción trabaja en este nivel de aislamiento...
  - una consulta SELECT sólo ve los datos consignados antes de que comenzara la consulta; nunca verá datos no consignados ni cambios hechos durante la ejecución de consultas por otras transacciones concurrentes.

MVCC: ¿Cómo funciona?

- Situación en la demo:
  - Crear Tabla MVCC
  - Conectar a la BD con 2 usuarios diferentes
  - Comenzar la Transacción
  - *Insert* (insertar), *update* (actualizar) y *delete* (borrar)
  - Rollback
  - Comparar resultados

- OJO: hay que encapsular las operaciones en una transacción

MVCC: demo

```
CREATE TABLE mvcc_demo (val int primary key);

\d mvcc_demo
   Table "public.mvcc_demo"
 Column |  Type   | Modifiers
--------+---------+-----------
 val    | integer | not null
Indexes:
    "mvcc_demo_pkey" PRIMARY KEY, btree (val)

INSERT INTO mvcc_demo VALUES (1);

select xmin, xmax,ctid,* from mvcc_demo;
 xmin | xmax | ctid  | val
------+------+-------+-----
 1017 |    0 | (0,1) |   1
(1 row)
```

MVCC – demo (cont.)

- Conectarse a testdb como user1

```
\c testdb user1
Password for user user1:
You are now connected to database "testdb" as user "user1".
```

- Sesión de user1

```
update mvcc_demo set val=2;
UPDATE 1

testdb=# select xmin, xmax,ctid,* from mvcc_demo;
 xmin | xmax | ctid  | val
------+------+-------+-----
 1020 |    0 | (0,4) |   2
```

MVCC – demo (cont.)

Conectarse a testdb como user2

```
\c testdb user2
Password for user user2:
You are now connected to base de datos "testdb" as user "user2".

testdb=# select xmin,xmax,ctid,* from mvcc_demo;
 xmin | xmax | ctid  | val
------+------+-------+-----
 1019 | 1020 | (0,3) |   1
(1 row)
```

** Nótese la diferencia de versión de la instantánea respecto a user1

User2 intenta actualizar una entrada que está bloqueada por user1

```
testdb=# update mvcc_demo set val=3;
```

MVCC – demo (cont.)

```
ps -ef | grep postgres
502      12131  6001  0 09:15 ?        00:00:00 postgres: user1
testdb [local] idle in transaction
502      13819  6001  0 09:49 ?        00:00:00 postgres: user2
testdb [local] UPDATE waiting
```

User1 hace un commit, User2 consigue un bloqueo y completa la transacción

*** NÓTESE el incremento de xmin***

```
testdb=# commit;
COMMIT
testdb=# select xmin, xmax,ctid,* from mvcc_demo;
 xmin | xmax | ctid  | val
------+------+-------+-----
 1022 |    0 | (0,5) |   3
(1 row)
```

MVCC: Resumen

- Foto de los datos en un momento dado.
  - Updates, inserts y deletes provocan la creación de una nueva versión de fila.
  - La nueva versión se guarda en la misma página.
  - Cada fila tiene 2 IDs de transacción: *created* (creado) y *expired* (caducado)

- Las consultas verifican que:
  - Se haya hecho un commit del ID de la transacción y sea menor que el actual contador de transacción
  - La línea no tenga ID de transacción de caducidad o que la caducidad se estuviera procesando al comienzo de la query

MVCC: Detalles de implementación

Identificadores de Objetos

- OID – Object identifier (entero de 32bits)

- **xmin**: número de transacción de creación, asignado por INSERT y UPDATE
- **xmax**: número de transacción de caducidad, asignado por UPDATE y DELETE; también se usa para bloqueo explícito de filas

cmin/cmax: se usa para identificar el número de comando que creó o expiró la tupla

- guardar los IDs de subtransacciones cuando la tupla se crea o expira en la misma transacción
- para el bloqueo explícito de filas.

MVCC: Consideraciones

- Las versiones de filas pueden causar "sobrecarga"
  - Hay que mantener datos de las instantáneas

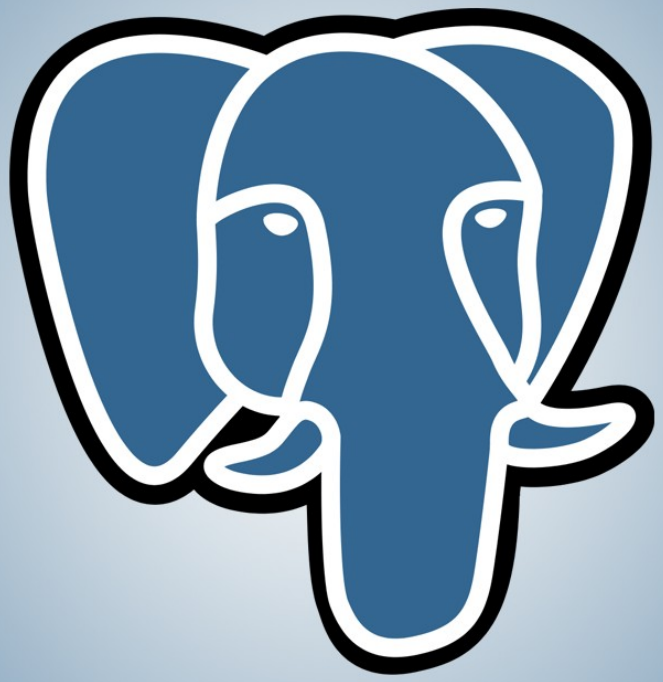
- Las filas que ya no se necesiten
  - se recuperan para ser reutilizadas
  - se eliminan mediante *vacuum*

- Previene "transacciones cruzadas"
  - Desbordamiento de ID de transacción
  - Se usa un ID de transacción "congelada" (frozenXID)

MVCC: Transacciones cruzadas

XIDs paras determinar qué versiones de la fila son “visibles”

- Contador XID limitado a aprox. 4E9
- Se puede hacer un cruce con cero – ¡las transacciones que ahora están en el pasado parecen estar en el futuro e invisibles!
- Corrupción de datos



todopostgresql.com