



Administración avanzada de PostgreSQL

Agenda del curso

- En este curso aprenderá:
 - Introducción y arquitectura
 - Transacciones y concurrencia
 - Performance Tuning
 - Replicación y Failover
 - Hot Standby
 - Particionamiento
 - PGPool-II
 - Pgbouncer
 - Monitorización
 - Lenguajes procedurales
 - Utilidades complementarias – Extensión



Módulo 1

Arquitectura del sistema

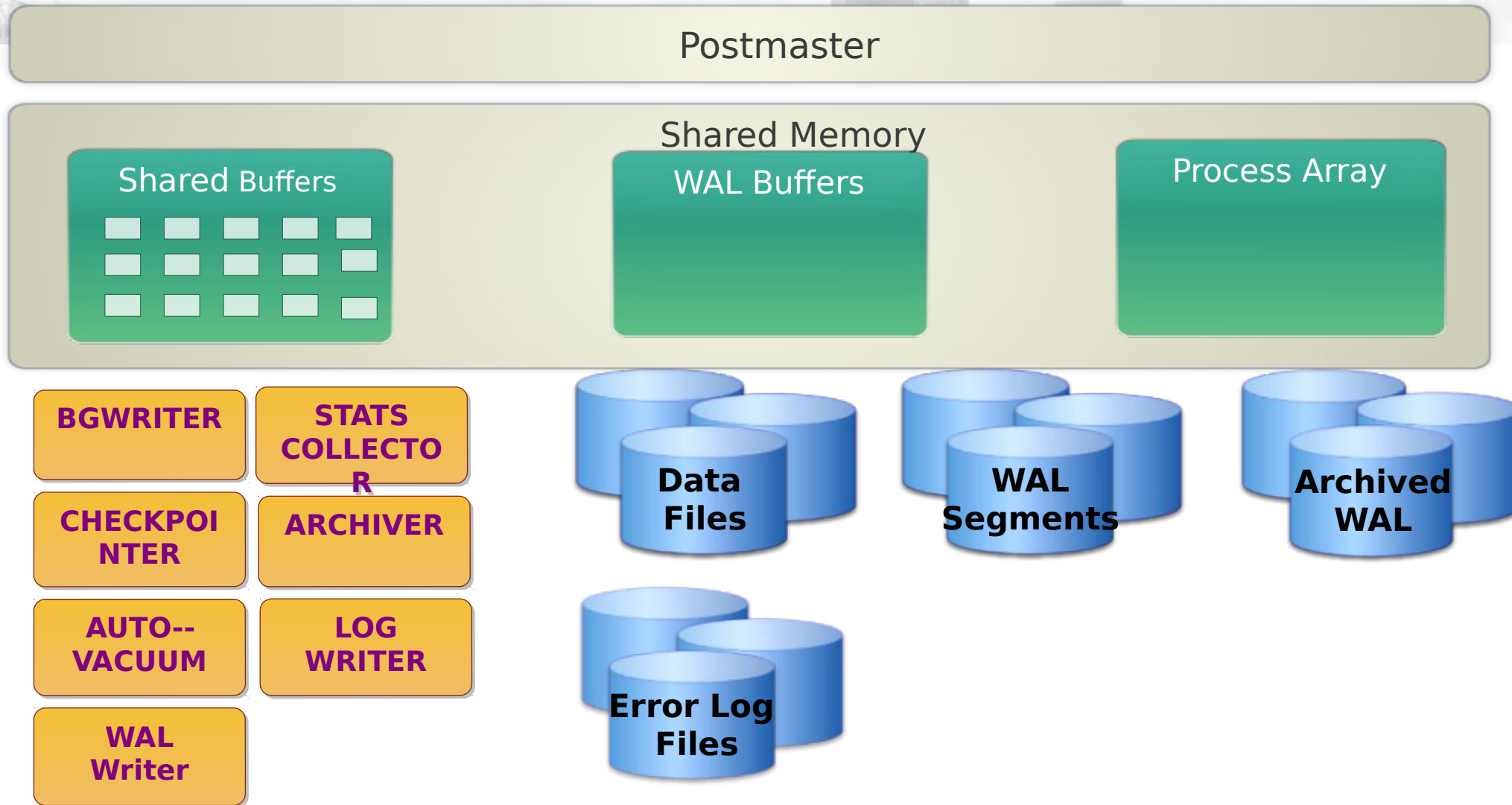
Objetivos

- Visión de la arquitectura
- Arquitectura de procesos y memoria
- Procesos de utilidad
- Conexión Petición-Respuesta
- Disk Read Buffering Disk
- Write Buffering
- Background Writer Cleaning Scan
- Commit y Checkpoint
- Procesamiento de sentencia
- Arquitectura física de la base de datos
- Estructura del directorio de datos
- Estructura del directorio de instalación
- Estructura de la página

Visión de la arquitectura

- PostgreSQL usa procesos, no hilos.
- El proceso Postmaster actúa como supervisor.
- Varios procesos de utilidad desempeñan tareas en segundo plano.
 - postmaster los inicia y los restaura si han muerto
- Un proceso de backend por sesión de usuario
 - postmaster escucha por nuevas conexiones

Arquitectura de procesos y memoria



Procesos de mantenimiento

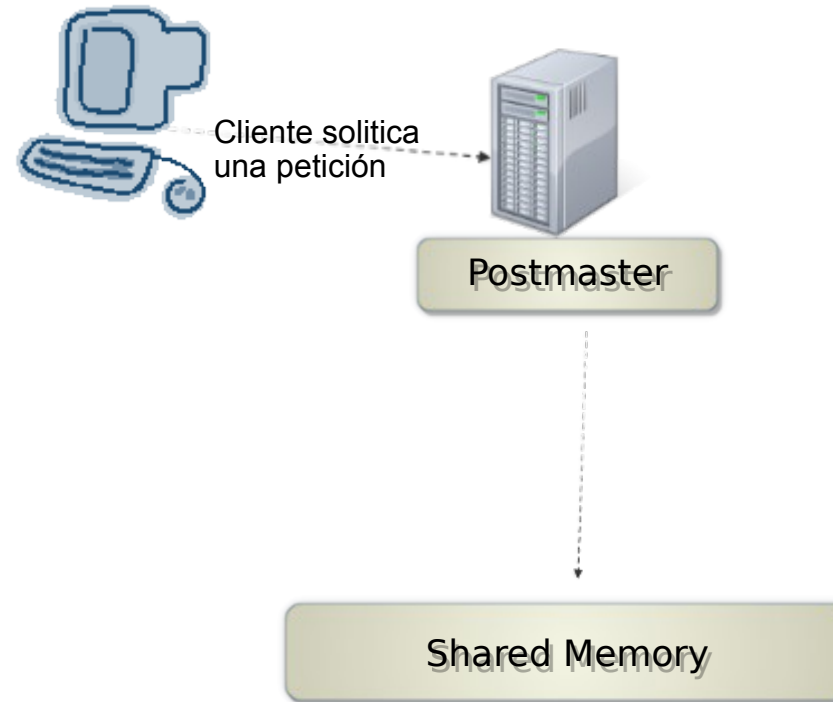
- Background writer
 - Consolida los bloques de datos sucios a disco
- WAL writer
 - Descarga el write-ahead log a disco
- Proceso Checkpointer
 - Automáticamente lleva a cabo un checkpoint basado en los parámetros de configuración
- Autovacuum Launcher
 - Arranca los Autovacuum workers según se necesite
- Autovacuum workers
 - Recuperan el espacio libre para reutilizarlo

Más procesos de mantenimiento

- Logging collector
 - Dirige los mensajes de log a syslog, eventlog o ficheros de log
- Stats collector
 - Recolecta las estadísticas de uso por relación y bloque
- Archiver
 - Archiva los ficheros de write-ahead log

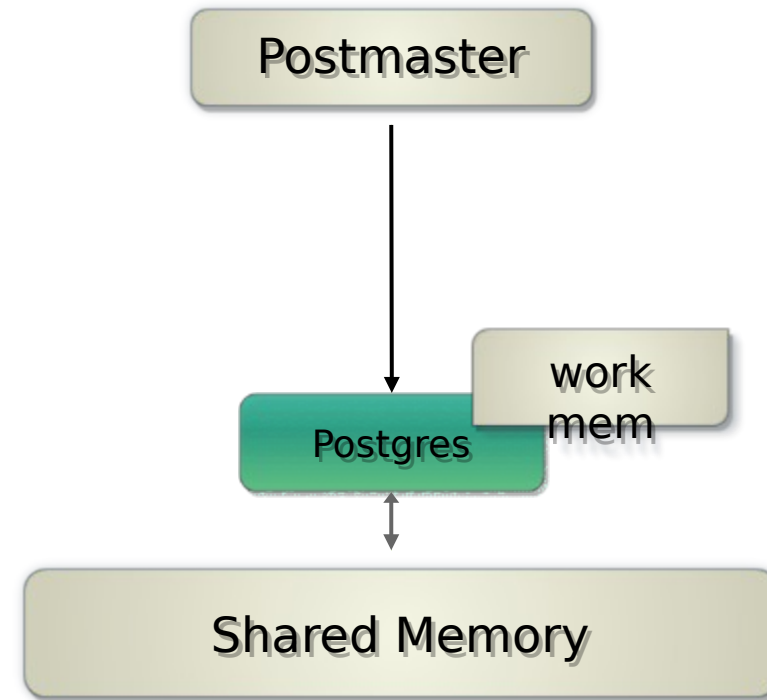
Postmaster como Listener

- Postmaster es el proceso principal llamado postgres
- Escucha en único puerto tcp
- Recibe peticiones de conexión del cliente



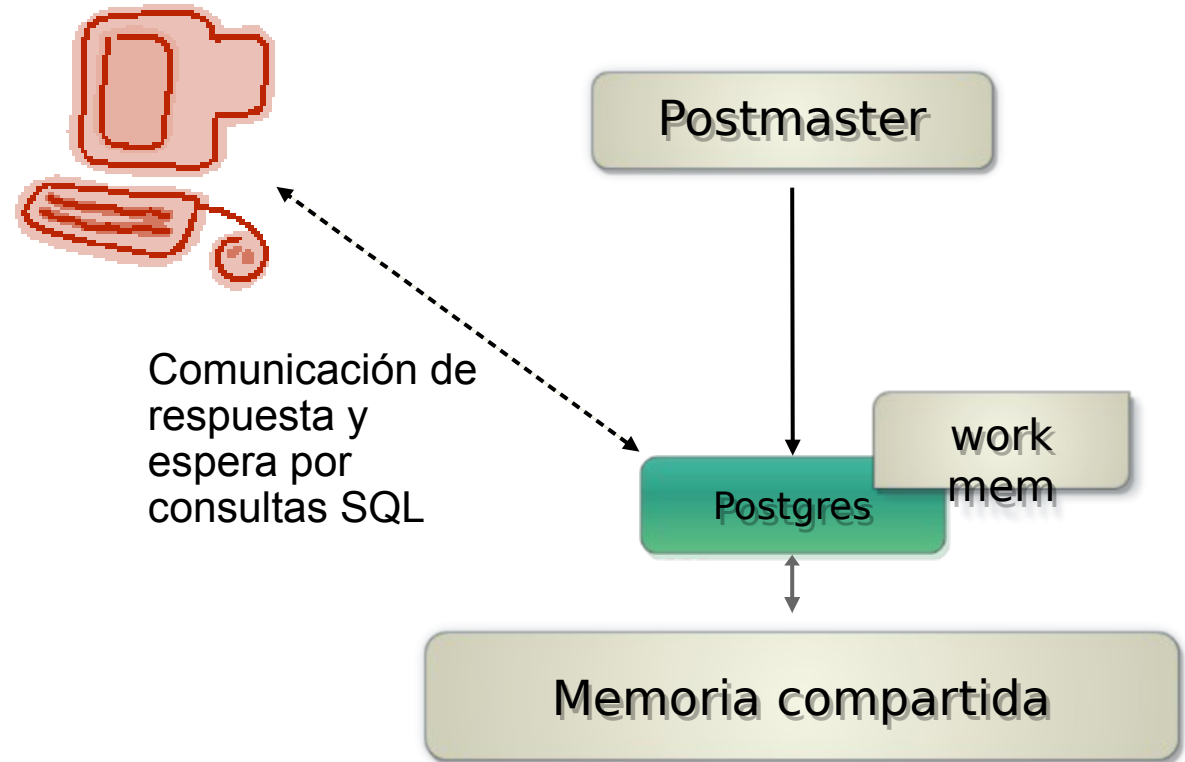
Reproducción de procesos de backend

- El proceso postmaster, genera un nuevo proceso servidor por cada una de las peticiones de conexión detectada.
- La comunicación entre se realiza utilizando semáforos y memoria compartida.
- Autenticación: IP, usuario y contraseña
- Autorización: Verifica los permisos



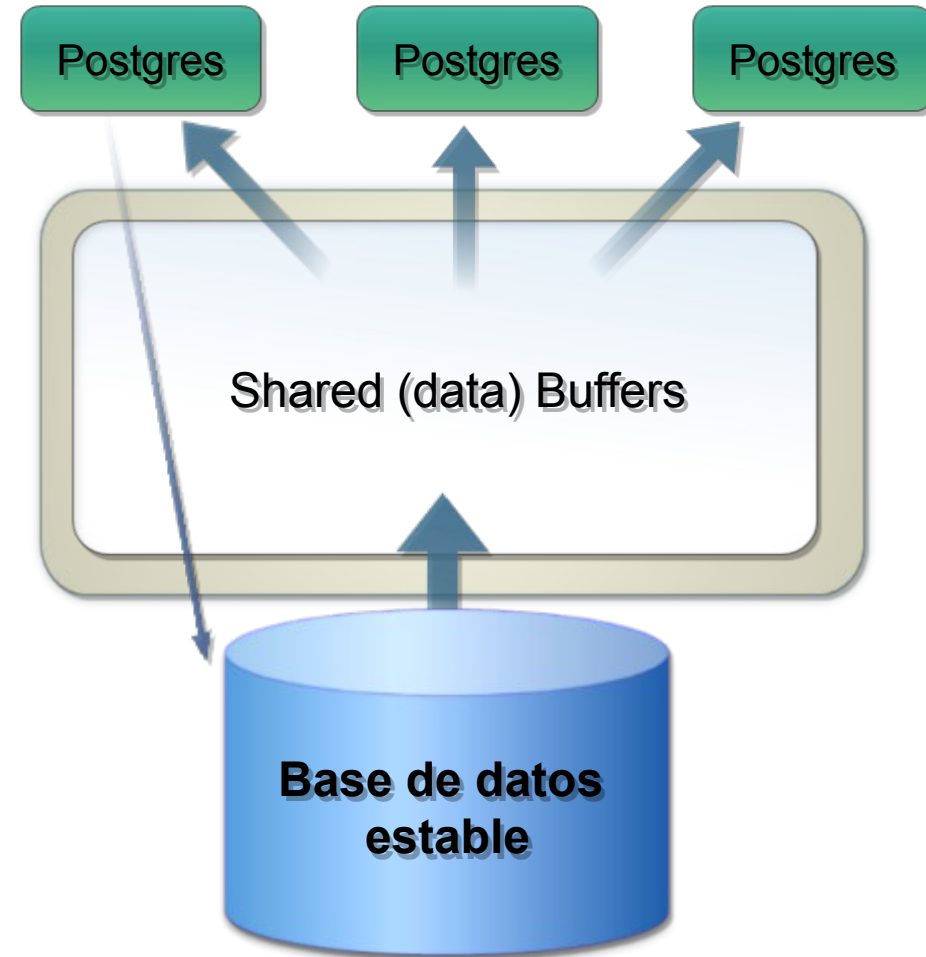
Respuesta al cliente

- El proceso de backend llamado postgres
- Comunicación de respuesta al cliente
- Espera por consultas SQL
- La consulta es transmitida en texto plano



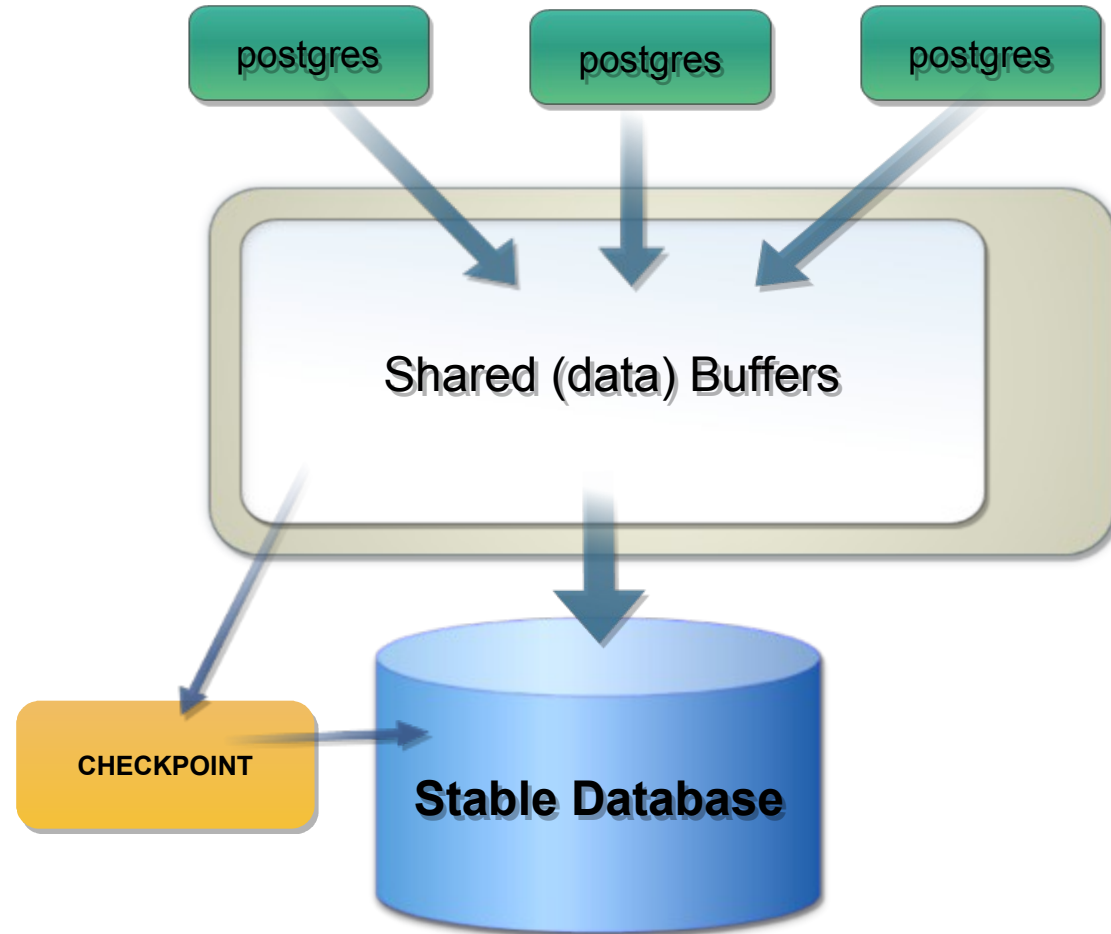
Buffering de lecturas a disco

- La caché de lectura de PostgreSQL (shared_buffers) reduce las lecturas del SO
- Se lee el bloque una vez y se consulta múltiples veces en caché.



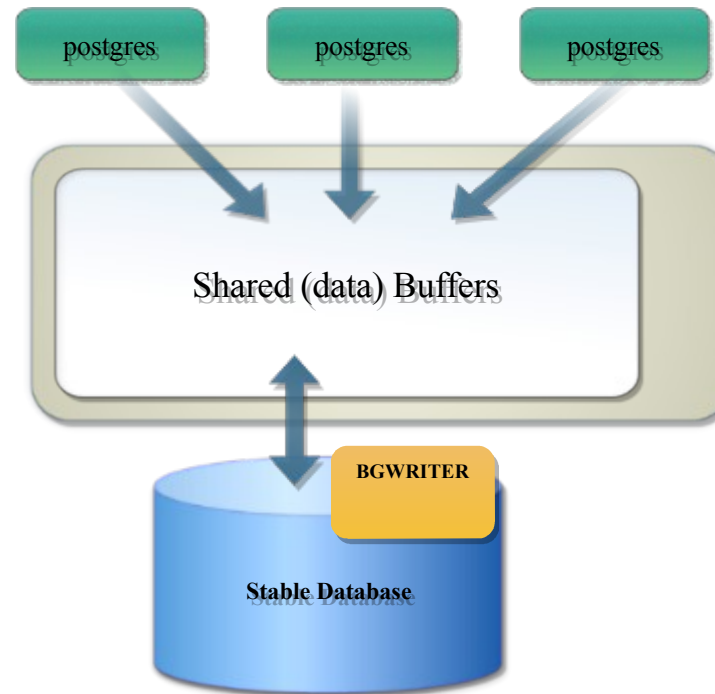
Buffering de escrituras a disco

- Los bloques se escriben a disco sólo cuando es necesario:
 - Para hacer espacio para nuevos bloques
 - En el momento de ejecutar un checkpoint



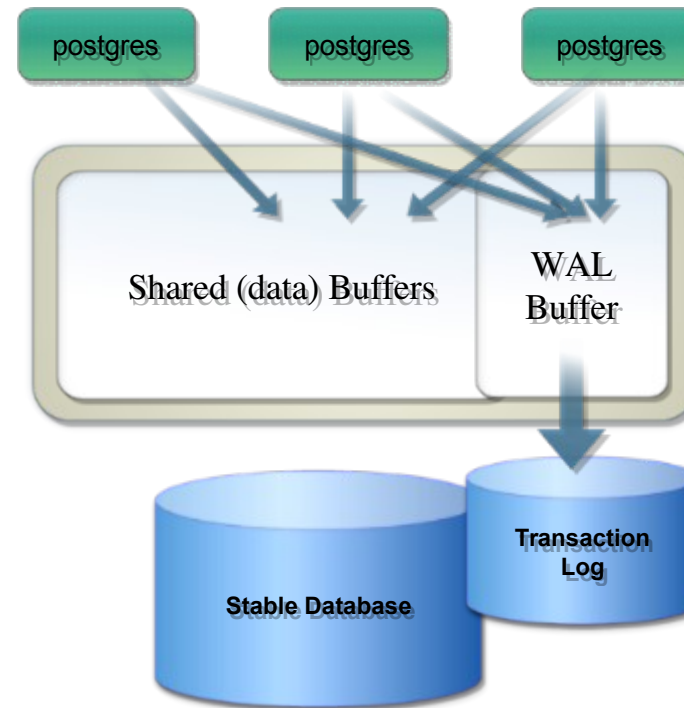
Background Writer Cleaning Scan

- El BGwriter asegura un suministro adecuado de buffers limpios al ir liberándolos eventualmente
- Escriben en segundo término los buffers sucios cuando es necesario



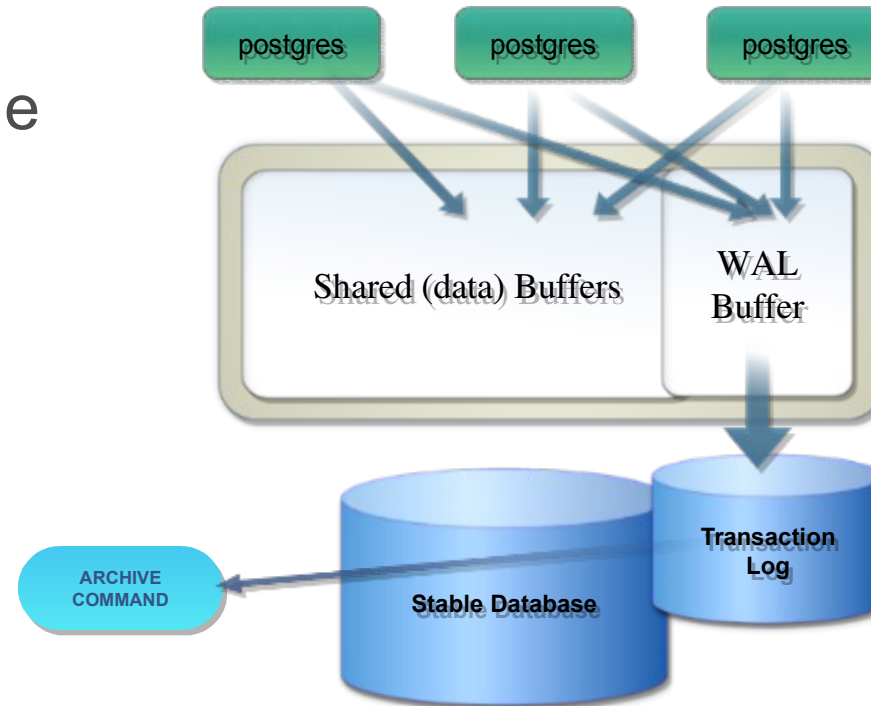
Write Ahead Logging (WAL)

- Backend que escribe la información a los buffers del WAL.
- Escribe los buffers del WAL a disco periódicamente (WAL writer), cuando se hace commit o cuando los buffers están llenos.
- Hacen el commit por grupos



Archivo del Log de transacciones

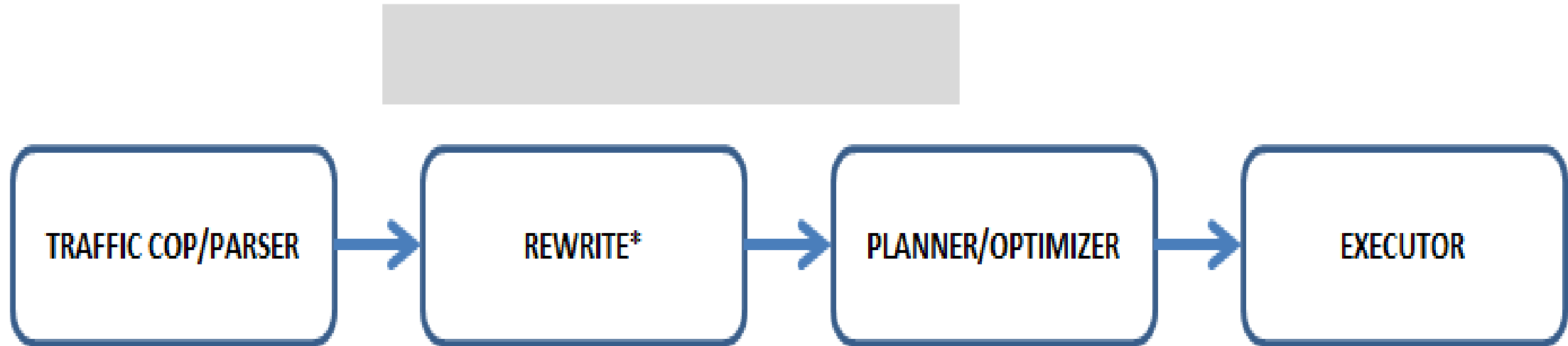
- Los archivadores generan una tarea para copiar los ficheros de log **pg_xlog** a disco cuando están llenos.



Commit y Checkpoint

- Antes del commit
 - Las actualizaciones antes del commit están en memoria
- Después del commit
 - Las actualizaciones después del commit pasan de la memoria a disco (write-ahead log file)
- Después del checkpoint
 - Las páginas de datos modificadas son escritas desde la memoria compartida a los ficheros de datos.

Procesamiento de sentencias SQL



Arquitectura física de la base de datos

- Un cluster es un conjunto de bases de datos gestionadas por una instancia de servidor
- Cada cluster tiene diferentes:
 - Directorio de datos
 - Puerto TCP
 - Conjunto de procesos
- Un cluster puede contener múltiples bases de datos

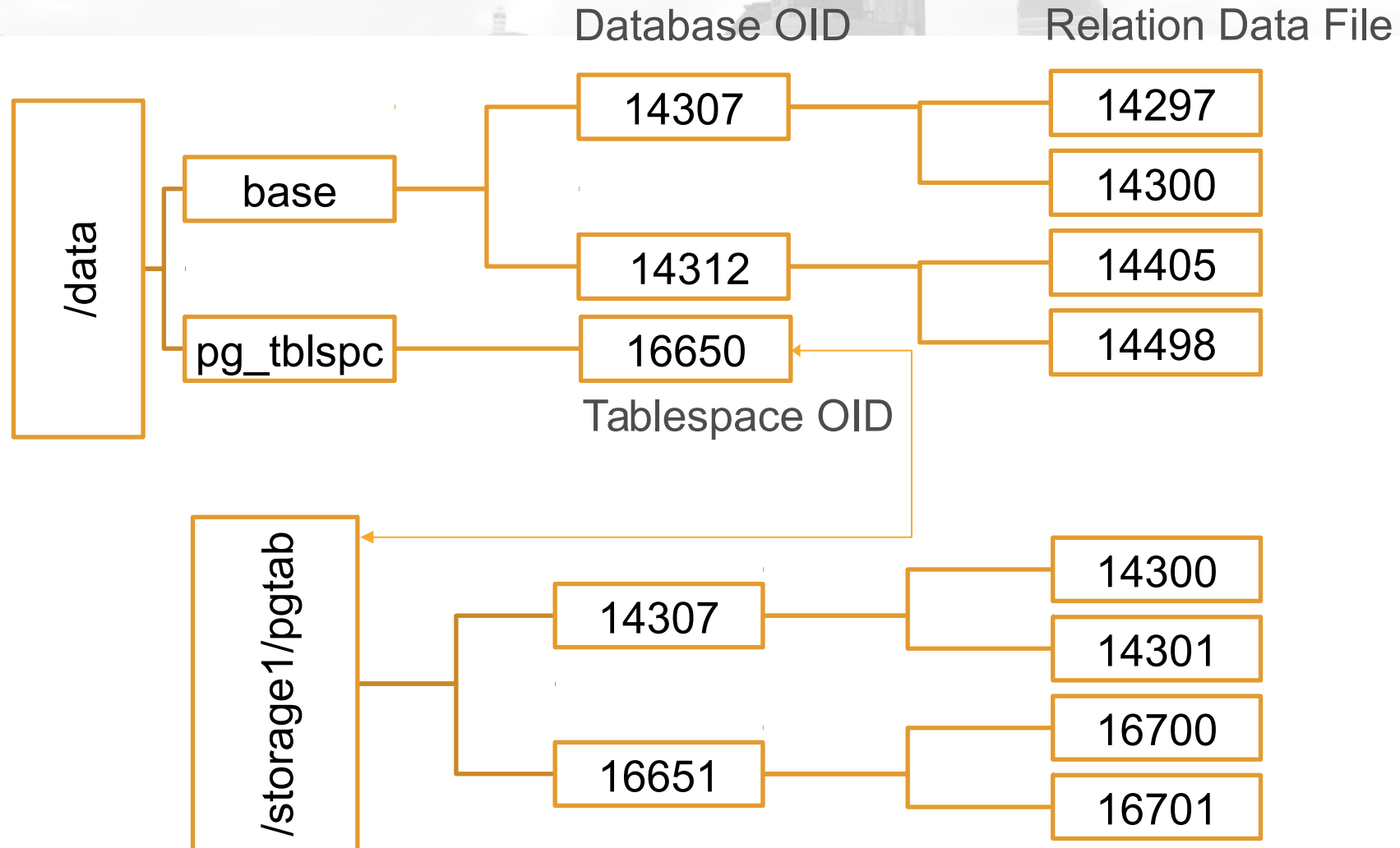
Directorios de datos del cluster de base de datos

- **global** – Objetos de la base de datos a nivel de cluster
- **base** – Contiene bases de datos
- **pg_tblspc** – Links simbólicos a tablespaces
- **pg_xlog** – Write ahead logs
- **pg_log** – Logs de errores
- Directorios que contienen diferentes informaciones de estado
- Ficheros de configuración del servidor
- Ficheros de información del postmaster

Arquitectura física de la base de datos

- Al menos un fichero por tabla, y un fichero por índice
- Un tablespace es un puntero lógico a un directorio
- Cada base de datos que utiliza ese tablespace, tiene un subdirectorio
- Cada relación que utiliza esa combinación de tablespace/base de datos, tiene uno o más ficheros de 1 GB.
- Existen ficheros adicionales para contener información auxiliar (mapa de espacio libre, mapa de visibilidad)
- Cada nombre de fichero es un número (`pg_class.relfilenode`)

Ejemplo de directorio de datos



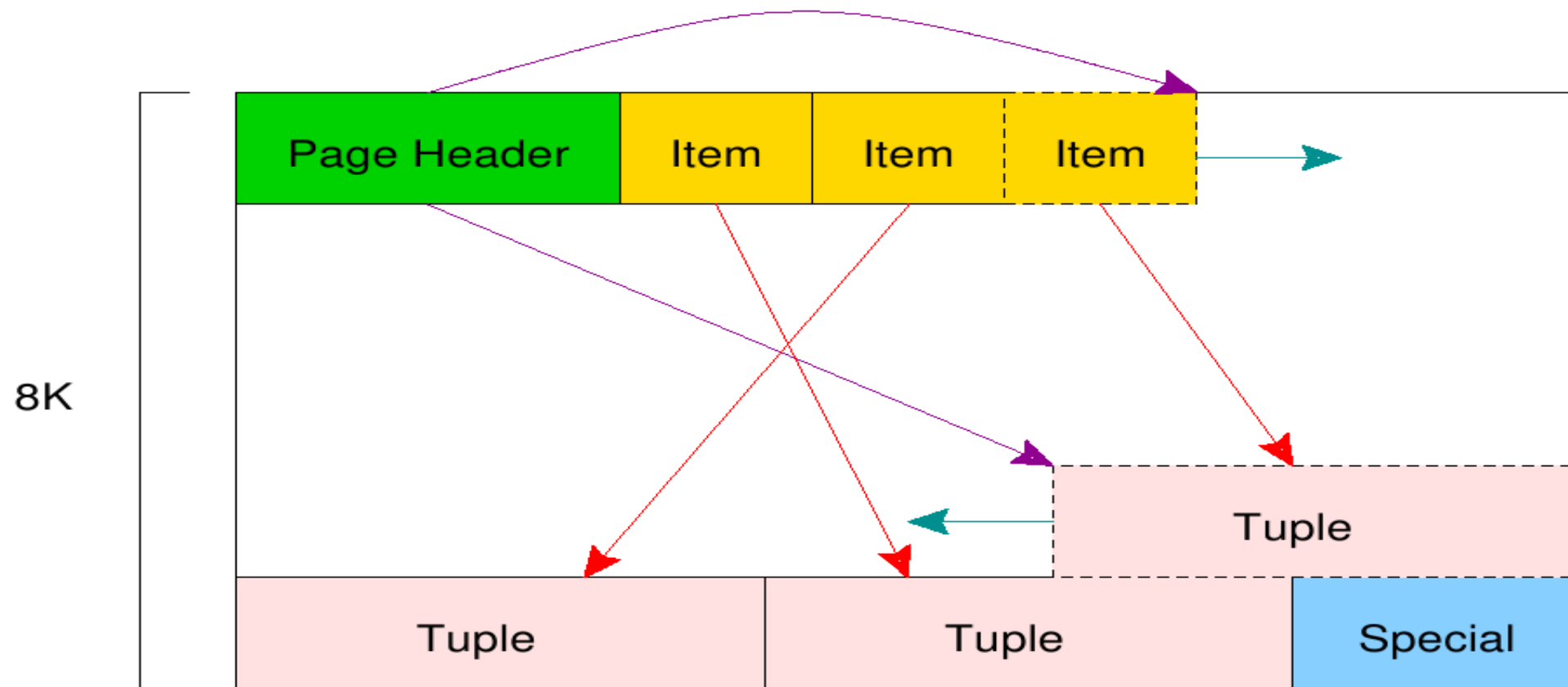
Directorios de instalación

- **bin** – Binarios ejecutables
- **data** – Directorio de datos
- **doc** – Documentación
- **include** – Cabeceras
- **installer, scripts** – Ficheros de instalación
- **lib** – Librerías
- **pgAdmin III** – Herramienta de administración gráfica
- **stackbuilder** (PostgreSQL) – Utilizado para instalar otros componentes
- **pg_env.bat/sh** – Variables de entorno para la instalación

Estructura de página

- Cabecera de página
 - Información general sobre la página
 - Punteros a espacios libres
 - 24 bytes
- Punteros a filas/índices
 - Vector de pares desplazamiento/longitud apuntando a las filas/entradas de índices
 - 4 bytes por objeto
- Espacio libre
 - Espacio no asignado
 - Los nuevos punteros crecen desde las primeras posiciones, las nuevas filas/entradas de índices crecen desde atrás
- Filas/entradas de índices
 - Datos reales de fila o índice
- Especiales
 - Datos específicos de métodos de acceso de índices
 - Vacíos en tablas

Estructura de página



Estructura de página

Overall Page Layout

Item	Description
PageHeaderData	24 bytes long. Contains general information about the page, including free space pointers.
ItemIdData	Array of (offset,length) pairs pointing to the actual items. 4 bytes per item.
Free space	The unallocated space. New item pointers are allocated from the start of this area, new items from the end.
Items	The actual items themselves.
Special space	Index access method specific data. Different methods store different data. Empty in ordinary tables.

Field	Type	Length	Description
pd_lsn	PageXLogRecPtr	8 bytes	LSN: next byte after last byte of xlog record for last change to this page
pd_checksum	uint16	2 bytes	Page checksum
pd_flags	uint16	2 bytes	Flag bits
pd_lower	LocationIndex	2 bytes	Offset to start of free space
pd_upper	LocationIndex	2 bytes	Offset to end of free space
pd_special	LocationIndex	2 bytes	Offset to start of special space
pd_pagesize_version	uint16	2 bytes	Page size and layout version number information
pd_prune_xid	TransactionId	4 bytes	Oldest unpruned XMAX on page, or zero if none

PageHeaderData Layout

HeapTupleHeaderData Layout

Field	Type	Length	Description
t_xmin	TransactionId	4 bytes	insert XID stamp
t_xmax	TransactionId	4 bytes	delete XID stamp
t_cid	CommandId	4 bytes	insert and/or delete CID stamp (overlays with t_xvac)
t_xvac	TransactionId	4 bytes	XID for VACUUM operation moving a row version
t_ctid	ItemPointerData	6 bytes	current TID of this or newer row version
t_infomask2	uint16	2 bytes	number of attributes, plus various flag bits
t_infomask	uint16	2 bytes	various flag bits
t_hoff	uint8	1 byte	offset to user data

Resumen del módulo

- Arquitectura
- Arquitectura de procesos y memoria
- Procesos de mantenimiento
- Conexiones Petición-Respuesta
- Buffering de lectura de disco
- Buffering de escritura
- Limpieza de escritura en background
- Commit y Checkpoint
- Procesamiento de consultas
- Arquitectura física de la base de datos
- Directorio de datos
- Instalación de directorio de datos
- Estructura de páginas

Ejercicio 1

- Previo estos laboratorios, usted debe haber creado la base de datos de pruebas hoplastore con algunos objetos.
- Habrá psql y:
 - Escriba una consulta que muestre el nombre del datafile en el que los la tabla *customers* están almacenados.
 - Escriba una query para encontrar el número de páginas y filas de la tabla *customers*.
 - Escriba una consulta que nos de el nombre del fichero de índice de la clave primaria de la tabla *customers*.
 - Abra una terminal y muévase al directorio donde se sitúa el datafile de la tabla *customers*.

Ejercicio 2

- Escriba un comando para encontrar los autovacuum workers que están actualmente ejecutándose en su servidor.
- Escriba un SQL statement para listar los autovacuum workers de su cluster.
- Encuentre un proceso con PID=10009 en su servidor consumiendo >80% de los recursos de CPU. Escriba un statement para encontrar este proceso en su cluster de base de datos.
- Escriba un SQL statement para proceder a matar este proceso.

Ejercicio - 3

- Escriba una consulta para mostrar la tercera fila de la tercera columna de la tabla *orders*.
- Escriba una consulta para mostrar todas las filas físicamente ubicadas antes de la página tercera de la fila tercera de la tabla *orders*.



Módulo 2

Transacciones y Concurrencia

Objetivos del módulo

- En éste módulo se tratara de:
 - Definición de transacción
 - Efectos de la concurrencia en las transacciones
 - Niveles de aislamiento en las transacciones
 - Visión del modelo Multi-Version Concurrency Control (MVCC)
 - MVCC Ejemplo
 - Identificadores internos
 - Transacción Wraparound
 - Mantenimiento del MVCC
 - Demostración del MVCC

¿Qué es una transacción?

- Una transacción es un conjunto de sentencias agrupadas en un sólo paso, en una operación de todo o nada.
- Una transacción debe tener las propiedades ACID:
 - Una operación de todo o nada (Atomicidad).
 - Únicamente datos válidos son escritos en la base de datos (Consistencia).
 - Los estados intermedios entre los pasos de la transacción no son visibles por otras transacciones concurrentes. (Aislamiento).
 - Los datos una vez modificados por la transacción no serán perdidos incluso si hay una caída del sistema (Durabilidad).

Concurrencia y transacciones

- Concurrencia – dos o más sesiones accediendo a los mismos datos al mismo tiempo.
- La coherencia de datos se mantiene mediante el uso del modelo Multiversion Concurrency Control(MVCC).
- Cada transacción ve una instantánea de los datos (versión de la base de datos) en el momento en el que se inicio la misma.
- Aislamiento de transacción – Protege que la transacción vea datos “inconsistentes” (actualmente siendo actualizados por otra transacción).
- MVCC ventaja – lectores no bloquean a escritores y escritores no bloquean a lectores.

Aislamiento transaccional

- El estándar ANSI/ISO SQL define cuatro niveles de aislamiento transaccional.
- Estos niveles de aislamiento están definidos en términos de tres fenómenos:
 - Dirty Read: Una transacción lee los datos de otra transacción que no ha sido committed.
 - Nonrepetible read: Una transacción lee dos veces la misma fila con diferente valor que ha sido committed por otra transacción concurrente.
 - Phantom read: Es una situación especial de Nonrepetible read donde ejecuta dos veces la misma consulta con valores diferentes.

Niveles de aislamiento transaccional

Nivel aislamiento\Fenómeno	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	<i>Posible</i>	<i>Posible</i>	<i>Posible</i>
Read committed	<i>No posible</i>	<i>Posible</i>	<i>Posible</i>
Repeatable read	<i>No posible</i>	<i>No posible</i>	<i>Posible</i>
Serializable	<i>No Posible</i>	<i>No Posible</i>	<i>No Posible</i>

Niveles de aislamiento transaccional (Continuación)

- Internamente, están disponibles tres niveles únicos de aislamiento: Read Committed, Repeatable Read, y Serializable.
- Cuando usemos el nivel Read Uncommitted PostgreSQL usará Read Committed.
- El nivel por defecto es Read Committed.
- El standard SQL - en los cuatro niveles de aislamiento solo se definen que fenómenos no deben aparecer pero no define que fenómenos deben aparecer.
- PostgreSQL sólo proporciona tres niveles de aislamiento sensibles a mapearse al modelo MVCC.

Multi-Version Concurrency Control (MVCC)

- Instantánea de los datos en un punto en el tiempo.
- Actualizaciones, inserciones y borrados provocan la creación de una nueva versión de la fila.
- La versión de fila se almacena en la misma página si es posible.
- MVCC utiliza el ID de transacción para conseguir la consistencia.
- Cada fila tiene dos identificadores de transacción: creación (XMIN) y finalización (XMAX).
- Las consultas comprueban:
 - `id_creacion_transacción` está committed y es \leq `contador_actual_transacción`
 - La fila que no tiene `id_finalización_transacción` o la finalización está en proceso cuando la consulta empezó.

Identificadores internos

- Identificadores de objetos
 - OID – Identificador de objeto (entero de 32-bit)
 - xmin – ID de transacción de creación de la nueva versión de la fila.
 - xmax – ID de transacción de borrado de la versión de la fila

Transaction Wraparound

- XIDs se usan para determinar que versiones de las filas son “visibles”
- XIDs tienen un tamaño limitado a (32 bits) y un contador de 4 mil millones
- Puede reiniciarse a cero – las transacciones del pasado parecen estar en el futuro y no visibles!
- Corrupción de datos
- Este problema se conoce como el fallo de transaction id wraparound

Mantenimiento MVCC

- MVCC crea varias versiones de una fila para soportar la concurrencia.
- Antiguas versiones de las filas pueden causar “bloat”
- Filas que no son necesarias, son recuperadas para reutilizarlas/borrarlas mediante el vacuum o autovacuum
- Para prevenir el fallo transaction wraparound cada tabla debe ser limpiada periódicamente (vacuumed)
- PostgreSQL reserva XID especiales como FrozenXID
- Este XID siempre es considerado más antiguo que cualquier XID normal.

MVCC Demostración

PSQL Terminal Sesión 1

```
drop table if exists mvc;  
create a table and insert rows:  
create table mvc(id numeric, name  
    varchar);  
insert into mvc values(1,'A'), (2,'B');
```

```
postgres=# select*from mvc;
```

id	name
----	------

1	A
2	B

(2 rows)

PSQL Terminal Sesión 2

```
postgres=# begin;  
BEGIN  
postgres=# update mvc set name='C' where  
id=2;
```

PSQL Terminal Sesión 1 – Puede ver una versión más antigua de la fila debido al MVCC

```
postgres=# select*from mvc;
```

id	name
----	------

1	A
2	B

(2 rows)

Resumen

- En este módulo ha aprendido:
 - Definición de transacción
 - Efectos de la concurrencia en transacciones
 - Niveles de aislamiento de transacciones
 - Multi-Version Concurrency Control Overview (MVCC)
 - MVCC ejemplo
 - Identificador Interno
 - Transaction Wraparound
 - Mantenimiento de MVCC
 - Demostración MVCC

Ejercicio - 1

- Simula el comportamiento del MVCC usando un ejemplo sobre la tabla *pdoducts*.
- Abre un psql en un terminal (sesión 1).
- Escriba una consulta para ver el precio total de los productos con *prod_id* entre 1 y 10.
- Abra otro psql en otro terminal (sesión 2).
- Comience una transacción explícita en el segundo psql. Escriba una consulta para actualizar el precio de todos los productos con *prod_id* entre 1 y 10 para que tenga un coste 10\$ superior al actual. Escriba una consulta para ver el precio total de los productos con *prod_id* entre 1 y 10. Este valor debe haber cambiado. NOTA: No haga *commit* de esta transacción.
- Vuelva al psql 1. Escriba una consulta para ver el precio total de los productos con *prod_id* entre 1 y 10. Debe ver el valor antiguo.
- Vuelva al psql 2 y haga *commit*.
- Vuelva al psql 1. Escriba una consulta para ver el precio total de los productos con *prod_id* entre 1 y 10. Debe ver el valor nuevo.



Módulo 3

Tuning de rendimiento

Objetivos del módulo

- Configuración del hardware
- Configuración de OS
- Tuning de los parámetros del servidor
- Configuración de las conexiones
- Parámetros de memoria
- Parámetros de memoria para el planificador
- Parámetros del WAL
- Explain Plan
- Ejemplo de un Explain
- Recolector de estadísticas
- Índices
- Examinar el uso de índices
- Consejos para insertar grandes cantidades de datos
- Algunos detalles sobre pg_dump
- Parámetros no-durables
- Laboratorios

Configuración del hardware

- Centrarse en la velocidad del disco y RAM sobre la velocidad de la CPU
 - Como otros RDBMSes, PostgreSQL es I/O intensivo.
- Más Spindles
 - PostgreSQL utilizará más discos para paralelizar las peticiones de lectura y escritura en la base de datos.
- Separar el log de transacciones e índices
 - Pon el log de transacciones de la base de datos (pg_xlog) en discos dedicados.
 - Tablespaces puede ser usado para crear índices en discos separados.
- SCSI o IDE
 - SCSI es más caro pero puede aceptar múltiples peticiones ayudando a mejorar la eficiencia.

Configuración de SO

- Usa la memoria disponible del servidor
 - PostgreSQL usa el sistema de segmento de memoria compartida así que establecer “shmmax” al valor apropiado basado en el tamaño de la RAM física del servidor.
 - El tamaño de shmmax depende de varios parámetros de configuración de PostgreSQL.
 - shmmax debe ser lo suficientemente grande para PostgreSQL y cualquier otra aplicación que esté usando segmento de memoria compartida.

Configuración de SO

Uso de memoria compartida en PostgreSQL

Uso	Bytes requeridos aproximadamente (a partir de 8.3)
Conexiones	$(1800 + 270 * \text{max_locks_per_transaction}) * \text{max_connections}$
Procesos de autovacuum	$(1800 + 270 * \text{max_locks_per_transaction}) * \text{autovacuum_max_workers}$
Prepared transactions	$(770 + 270 * \text{max_locks_per_transaction}) * \text{max_prepared_transactions}$
Shared disk buffers	$(\text{block_size} + 208) * \text{shared_buffers}$
WAL buffers	$(\text{wal_block_size} + 8) * \text{wal_buffers}$
Fixed space requirements	770 kB

Configuración de SO

- El sistema de ficheros marca la diferencia
 - Un sistema de ficheros con journaling no es necesario para el log de transacción
 - Múltiples opciones están disponibles para Linux:
 - EXT2
 - EXT3
 - EXT4 y XFS
 - Remote file system no están recomendados.
- Elige la mejor opción basada en mejores escrituras, recuperabilidad, soporte de múltiples fabricantes y confiables.
- Recomendado: controladoras RAID con cache con baterías (BBU) y configurado para write-back.

Tunning de los parámetros del servidor

- Los parámetros por defecto del servidor en postgresql.conf estan configurados para una amplia compatibilidad.
- Parámetros del servidor deben ser afinados a un valor óptimo para un mejor rendimiento.
- Los parámetros pueden ser evaluados de diferentes maneras y pueden ser permanentemente configurados en postgresql.conf.
- Algunos parámetros requieren reinicio.
- Información básica requerida para hacer el tuning pero no limitado a:
 - Tamaño de la base de datos
 - El tamaño de la tabla más grande
 - Tipo y frecuencia de las consultas
 - RAM disponibles
 - Número de conexiones concurrentes

Configuración de las conexiones

- max_connections
 - Establece el número máximo de conexiones concurrentes.
 - Cada conexión de usuario está asociada a un proceso backend de usuario en el servidor.
 - Los proceso de backend de usuario se terminan cuando el usuario cierra la sesión
 - Un pool de conexiones puede reducir la sobrecarga en el postmaster por reusar los procesos de usuario del backend existentes.

Parámetros de memoria

- `shared_buffers`
 - Establece el número de buffers de memoria compartida usada por el servidor de base de datos.
 - Cada buffer es de 8KB.
 - Valor mínimo debe ser al menos 128 KB.
 - 128 MB es el valor por defecto pero este debe ajustarse entre 25% y 40% de la RAM de la máquina para servidor dedicado.

Parámetros de memoria

- `work_mem`
 - Cantidad de memoria en KB a ser usada por las operaciones SORT y HASH.
 - Mínimo valor disponible es 64 KB.
 - Se establece un valor por defecto de 1024KB (1 MB).
 - Incrementar el `work_mem` a menudo ayudará en hacer las ordenaciones más rápidas

Parámetros de memoria

- `maintenance_work_mem`
 - Máxima memoria en KB que será usada en tareas de mantenimiento como VACUUM, CREATE INDEX, y ALTER TABLE ADD FOREIGN KEY. Mínimo valor permitido es 1024 KB.
 - Se establece en KB y el valor por defecto es 16384 KB (16 MB).
 - El rendimiento para las tareas de mantenimiento pueden ser mejoradas incrementando este valor.
 - Cuando autovacuum se ejecuta, hasta autovacuum_max_workers procesos pueden ser levantados para realizar esta tarea.

Parámetros de memoria para el planificador

- `effective_cache_size`
 - Parámetro que afecta en la decisión sobre realizar un seq scan o un index scan.
 - Un valor alto favorece los escaneos por índice.
 - Este parámetro no reserva memoria; es usado únicamente para propósitos estimativos.
 - $\frac{1}{2}$ de la RAM es un valor conservador
 - $\frac{3}{4}$ de la RAM es un valor más agresivo
 - Encontrar el valor óptimo mirando en las estadísticas de SO después de incrementar o disminuir este parámetro.

Ficheros temporales

- temp_file_limit
 - Cantidad máxima de espacio en disco que una sesión puede utilizar para los ficheros temporales.
 - Una transacción que intente superar este límite será cancelada.
 - Por defecto es -1 (sin limite).
 - Este parámetro restringe el espacio total usado en cualquier instante por todos los ficheros temporales en uso en una sesión de postgresql.

Parámetros del WAL

- wal_level
 - wal_level determina cuanta información es volcada al WAL
 - El valor por defecto es minimal, que escribe solamente la información necesaria para recuperarse de un crash o una parada inmediata.
 - El valor archive añade información de log necesaria para hacer WAL archiving
 - El valor hot_standby añade información necesaria para poder ejecutar sentencias de sólo lectura en el servidor de respaldo (standby)
 - Este parámetro sólo puede establecerse al arranque del servidor

Parámetros del WAL

- wal_buffers
 - Número de buffers reservados en la memoria compartida relativa a datos de WAL
 - Las páginas de disco son por defecto de 8 KB y el tamaño de este buffer por defecto es de 16 MB.
 - El valor mínimo es de 4.
 - Es necesario que todos los WAL buffers mantengan al menos la información necesaria para reproducir una única transacción antes del commit.

Parámetros del WAL

- Checkpoints
 - Escribe las páginas modificadas actuales en memoria (conocidas como páginas sucias) a disco.
 - Un checkpoint automático ocurre cada vez que se llega a un número de checkpoint `checkpoint_segments` o a un tiempo de `checkpoint_timeout`.
- `checkpoint_segments`
 - Máximo número de segmentos antes de realizar un checkpoint.
 - Cada segmento del fichero de log es 16 MB.
 - Un Checkpoint es forzado cada vez que es llenado (16 MB x `checkpoint_segments`)
 - Un valor más alto genera menor frecuencia de checkpoints.
 - El valor mínimo es 1.
 - Por defecto es 3.

Parámetros del WAL

- `checkpoint_timeout`
 - Máximo intervalo de tiempo en segundos antes de ejecutar un WAL checkpoint automático.
 - Un valor más alto llevará a menos frecuencia de checkpoints.
 - Por defecto es 300 segundos (5 min).

Parámetros del WAL

- fsync
 - Asegurar que todos los wal buffers sean escritos al wal log en cada commit.
 - Cuando está activo, fsync() o otro método establecido por wal_sync_method es invocado.
 - Su deshabilitación conllevará una mejora del rendimiento pero existe el riesgo de corrupción de datos.
 - Se puede deshabilitar durante la carga inicial de la instancia.

Utilidad pg_test_fsync

- El método especificado en `wal_sync_method` es usado para forzar la actualización en disco del WAL
- `pg_test_fsync` puede determinar cuál es el método más rápido de `wal_sync_method` en tu sistema
- `pg_test_fsync` informa del tiempo medio en sincronizar los ficheros
- Información diagnostica para identificar problemas de I/O

Timing

- Utilice `\timing` en `psql` para ver el tiempo que tarda en ejecutarse cualquier comando.

```
postgres=# \timing on
Timing is on.
postgres=# select count(*) from analytics.order_detail;
 count
-----
 780465
(1 row)

Time: 4037.000 ms
```


Plan de ejecución de consulta

- El planificador de PostgreSQL es el responsable de crear los planes de ejecución de las consultas
- El optimizador de PostgreSQL determina el plan de ejecución más eficiente para una consulta
- La optimización está basada en costes, los costes son estimados en función de los recursos utilizados por el plan
- EXPLAIN query – muestra el plan de ejecución para una sentencia sql sin ejecutarla.
- El plan de ejecución muestra en detalle los pasos necesarios para ejecutar una sentencia SQL.

Plan de ejecución de consulta

- Elementos de un plan de ejecución:
 - Cardinalidad: Estimación de filas
 - Método de acceso: Secuencial o indexado
 - Método de Join: Hash, Nested Loop etc.
 - Join Type, Join Order
 - Sort

- Sintaxis:

EXPLAIN [(option [, ...])] statement

Donde option puede ser alguno entre:

ANALYZE [boolean]

VERBOSE [boolean]

COSTS [boolean]

BUFFERS [boolean]

TIMING [boolean]

FORMAT { TEXT | XML | JSON | YAML }

Estadísticas de tablas

- El planificador y optimizador de PostgreSQL utilizan las estadísticas de las tablas para generar los planes de ejecución de consultas.
- La elección de un buen plan de ejecución depende totalmente de las estadísticas.
- Estadísticas de tablas.
 - Las estadísticas de las tablas almacenan el total de filas en tablas e índices así como el número de páginas de disco usados por cada tabla e índices.
 - La tabla de estadísticas `pg_class` almacena `reltuples` y `relpages` que contienen información importante por cada tabla en una base de datos.

Ejemplo de explain

```
create table city (cityid numeric(5) primary key, cityname varchar(30));
create table office(officeid numeric(5) primary key, cityid numeric(5) references
city(cityid));
```

- Vamos a ver el plan sin datos ni actualizando las estadísticas:

```
EXPLAIN ANALYZE SELECT city.cityname, office.officeid, office.cityid FROM public.city,
public.office WHERE office.cityid = city.cityid;
```

Salida:

```
"Hash Join  (cost=25.30..72.59 rows=1570 width=100) (actual time=0.003..0.003 rows=0
loops=1) "
"  Hash Cond: (office.cityid = city.cityid) "
"    ->  Seq Scan on office  (cost=0.00..25.70 rows=1570 width=22) (actual time=0.002..0.002
rows=0 loops=1) "
"    ->  Hash  (cost=16.80..16.80 rows=680 width=89) (never executed) "
"          ->  Seq Scan on city  (cost=0.00..16.80 rows=680 width=89) (never executed) "
"Total runtime: 0.052 ms"
```

Ejemplo de explain

- Cargamos datos:

```
insert into city values(1,'Edmonton'), (2,'Calgary'), (3,'Sherwood Park'), (4,'ST  
Albert');  
insert into office values(generate_series(1,100),4);  
insert into office values(generate_series(101,200),3);  
insert into office values(generate_series(201,300),2);  
insert into office values(generate_series(301,400),1);
```

- Actualizamos las estadísticas de las tablas city y office:

```
ANALYZE city;  
ANALYZE office;
```

Ejemplo de explain

- Plan de ejecución:

```
EXPLAIN ANALYZE SELECT city.cityname, office.officeid, office.cityid FROM
public.city, public.office WHERE office.cityid = city.cityid;
```

```
"Hash Join  (cost=1.09..13.59 rows=400 width=24) (actual time=0.032..0.338
rows=400 loops=1) "
```

```
"  Hash Cond: (office.cityid = city.cityid) "
```

```
"    ->  Seq Scan on office  (cost=0.00..7.00 rows=400 width=14) (actual
time=0.010..0.056 rows=400 loops=1) "
```

```
"    ->  Hash  (cost=1.04..1.04 rows=4 width=17) (actual time=0.009..0.009 rows=4
loops=1) "
```

```
"          Buckets: 1024  Batches: 1  Memory Usage: 1kB "
```

```
"          ->  Seq Scan on city  (cost=0.00..1.04 rows=4 width=17) (actual
time=0.002..0.004 rows=4 loops=1) "
```

```
"Total runtime: 0.397 ms"
```

Recolector de estadísticas

- PostgreSQL recolecta y mantiene las estadísticas a nivel de tabla y columnas.
- El nivel de recolección de estadísticas puede controlarse mediante:
 - `ALTER TABLE <table> ALTER COLUMN <column> SET STATISTICS <number>;`
- Donde <number> está entre 1 y 10000
- Mayor <number> indicará al servidor que recolecte y actualice más estadísticas y puede ralentizar las operaciones de auto vacuum y analyze sobre las tablas de estadísticas.

Índices

- PostgreSQL proporciona varios tipos de índices:
- B-tree, Hash, GiST y GIN.
 - B-trees puede tratar las consultas de igualdad y consultas sobre rangos de datos que pueden ordenarse.
 - Los índices Hash pueden solamente tratar comparaciones de igualdad sencillas.
 - Los índices GiST para búsquedas full text search sobre ficheros sencillos.
 - Índices GIN para búsquedas full text search sobre ficheros complejos.

Índices multicolumna

- Un índice puede definirse sobre más de una columna de una tabla.
- En estos momentos, solamente los tipos B-tree, GiST y GIN soportan los índices multicolumna.
- Ejemplo:
 - `CREATE INDEX test_idx1 ON test (id_1, id_2);`
 - Este índice solamente se usarán cuando se consulte:
 - `SELECT * FROM test WHERE id_1=1880 AND id_2= 4500;`
- Los índices multicolumna deben usarse con moderación debido a su espacio y tiempo.

Índices y ORDER BY

- Se puede ajustar la ordenación de un índice B-tree incluyendo las opciones ASC, DESC, NULLS FIRST, y/o NULLS LAST en la creación del índice; por ejemplo:
 - `CREATE INDEX test2_info_nulls_low ON test2 (info NULLS FIRST);`
 - `CREATE INDEX test3_desc_index ON test3 (id DESC NULLS LAST);`
- Esto ahorrará tiempo en la consulta.
- Por defecto, los índices B-tree almacenan sus entradas en orden ascendente y con los nulos al final.

Índices únicos

- Los índices se emplean también para forzar la unicidad de valores en columnas o combinación de varias columnas.
 - `CREATE UNIQUE INDEX name ON table (column [, ...]);` En estos momentos, sólo los índices B-tree pueden declararse como únicos.
- PostgreSQL crea de manera automática un índice único cuando se define una restricción de unicidad o clave primaria para una tabla.

Índices de funciones

- Un índice puede crearse sobre un valor calculado de las columnas de las tablas.
 - Por ejemplo:
 - `CREATE INDEX test1_lower_coll_idx ON test1 (lower(coll));`
- Las expresiones de los índices son relativamente costosas de mantener.
- Los índices de expresiones son útiles cuando se quiere priorizar la velocidad de las consultas con respecto a las inserciones y actualizaciones.
- También se pueden crear sobre funciones definidas por el usuario.

Índices parciales

- Un índice parcial está construido sobre un subconjunto de los datos de una tabla.
- El índice contiene entradas sólo para aquellas filas de la tabla que satisfacen el predicado.
 - Ejemplo:
 - `CREATE INDEX idx_test2 on test(id_1) where province='AB';`

Examinar el uso del índice

- Es difícil dar un procedimiento general que determine que índices crear.
 - Antes que nada, primero ejecutar ANALYZE .
 - Usar datos reales para probar.
 - Cuando los índices no se están usando, puede ser útil, probarlos forzando su uso.
 - El comando EXPLAIN ANALYZE puede ser útil aquí.

Consejos para cargas de datos grandes

- Si inserta datos con múltiples inserts utilice BEGIN al principio y COMMIT al final.
- Emplee el comando COPY para cargar todas las filas con un solo comando, en vez de una serie de sentencias INSERT.
- Si no puede usar COPY, podría ayudarle PREPARE para crear sentencias de INSERT preparados.
- Si está cargando una tabla nueva, el método más rápido es crear la tabla, hacer la carga masiva de datos usando COPY, y entonces crear cualquier índice que necesite la tabla.
- Podría ser útil borrar las restricciones de clave foránea, cargar los datos y volver a recrear las restricciones.

Consejos para cargas de datos grandes

- Temporalmente incremente las variables de configuración `maintenance_work_mem` y `checkpoint_segments` en la carga de datos grandes pueden proporcionarle un incremento de rendimiento.
- Deshabilite el archivado de WAL y el Streaming Replication
- Los triggers y el Autovacuum también pueden deshabilitarse.
- Ciertos comandos de base de datos se ejecutan más rápidamente si `wal_level` está a nivel minimal:
 - CREATE TABLE AS SELECT
 - CREATE INDEX (y también variantes como ALTER TABLE ADD PRIMARY KEY)
 - ALTER TABLE SET TABLESPACE
 - CLUSTER
 - COPY FROM, cuando la tabla destino ha sido creada o truncada recientemente en la misma transacción.

Algunos detalles sobre pg_dump

- Establecer valores apropiados (ej, más grandes de lo normal) para `maintenance_work_mem` y `checkpoint_segments`.
- Si estas usando archivado de WAL o streaming replication, mejor deshabilitarlos durante la restauración.
- Estudia si debieras de restaurar todo el dump en una transacción. Para hacerlo, tiene que pasar el parámetro `-1` o `--single-transaction` en los comandos `psql` o `pg_restore`.
- Si están disponibles varias CPUs en el servidor de base de datos, entonces es recomendable usar la opción `--jobs` de `pg_restore`.
- Después de esta tarea, ejecutar `ANALYZE` .

Ajustes sobre durabilidad

- La durabilidad asegura la grabación de las transacciones comiteadas pero añade una sobrecarga.
- PostgreSQL se puede configurar sin durabilidad
- Coloque el directorio de datos del cluster en un sistema de ficheros en memoria (ej disco de RAM).
- Deshabilite fsync; ya que no hay necesidad de consolidar los datos a disco.
- Deshabilite full_page_writes; ya que no hay necesidad de protegerse contra escritura de páginas parciales.
- Incremente checkpoint_segments y checkpoint_timeout ; reducirá la frecuencia de los checkpoints.
- Deshabilite synchronous_commit; no habría necesidad de escribir el wal en el disco en cada commit.

Ejercicio 1

- Los usuarios están preocupados por el bajo rendimiento de la base de datos *hoplastore*.
- Configura el `postgresql.conf` para un óptimo performance basado en el tamaño de la base de datos *hoplastore*, la tabla más grande y demás información recolectada de la base de datos *hoplastore*.

Ejercicio 2

- Chequee el uso de los índices para todos los índices de usuario en la base de datos *hoplastore*.
- Reindexe todos los índices.
- Manualmente actualice las estadísticas para todos los objetos en la base de datos *hoplastore*.
- Verifique si se ha recolectado estadísticas de manera automática para las tablas de usuario de la base de datos *hoplastore*.

Ejercicio - 3

- Tome un full backup de la base de datos *hoplastore* empleando *pg_dump*.
- Elimine la base de datos *hoplastore* con un *drop database*.
- Cree una base de datos *hoplastore* en blanco.
- Verifique que *fsync* está a ON y compruebe el tiempo que tarda la base de datos *hoplastore* en restaurarse.
- Elimine la base de datos *hoplastore* con un *drop database* y cree de nuevo la base de datos *hoplastore*.
- Verifique que *fsync* está a OFF y compruebe el tiempo que tarda la base de datos *hoplastore* en restaurarse.

Objetivos

- Configuración del hardware
- Configuración de OS
- Tuning de los parámetros del servidor
- Configuración de las conexiones
- Parámetros de memoria
- Parámetros de memoria para el planificador
- Parámetros del WAL
- Explain Plan
- Ejemplo de un Explain
- Recolector de estadísticas
- Índices
- Examinar el uso de índices
- Consejos para insertar grandes cantidades de datos
- Algunos detalles sobre pg_dump
- Parámetros no-durables
- Laboratorios



Módulo – 4

Alta disponibilidad y Replicación

Objetivos

- Alta disponibilidad en bases de datos
- Causas de pérdidas de datos
- Plan para errores comunes
- Elige un criterio
- Opciones de alta disponibilidad
- Replicación hot streaming, arquitectura y configuración
- Ejemplo de replicación de streams
- Replicación de Slony-I, arquitectura y configuración
- Switchovers and Failovers
- Limitaciones de la replicación
- Ejemplo de replicación con Slony
- Laboratorios

Alta disponibilidad de bases de datos

- Los fallos hardware y errores humanos son los principales causantes de pérdidas de datos.
- El objetivo de la alta disponibilidad es proporcionar una protección efectiva frente a la pérdida de datos.
- La alta disponibilidad nos da una protección completa de los datos y disponibilidad de éstos, asegurando la permanente continuidad del negocio.
- La alta disponibilidad puede ayudar a un servidor de base de datos a tomar el control en caso que el servidor primario se caiga.
- Diferentes tipos de herramientas y opciones están disponibles para implementar la alta disponibilidad en PostgreSQL.

Causas de pérdidas de datos

- Todo sistema afronta caídas del sistema planificadas y no planificadas.
- Las caídas planificadas incluyen operaciones rutinarias, periodos de mantenimiento y nuevos despliegues.
- Las caídas no planificadas pueden suceder debido a muy distintas causas.
- Borrados de tablas o actualizaciones erróneas son mucho más frecuentes de lo que la gente suele admitir.
- El 80% de las interrupciones se deben a errores humanos.

Plan para errores comunes

- La falta de una planificación y pruebas previas representan el 80% del tiempo total de parada de servicio.
- Continuos Archiving (“Point in Time Recovery”) es el mejor método para recuperarse de estos errores.
- Las técnicas de alta disponibilidad están orientadas a escenarios de fallos en sistemas o *sites*.
- Disponibilidad o recuperabilidad son conceptos distintos.

Elija un criterio

- Coste!
- Tiempo de recuperación objetivo. Recovery Time Objective (RTO)
- Punto de recuperación objetivo. Recovery Point Objective (RPO)
- Acceder a los datos en los nodos esclavos
 - Activo (RW) Nodo que permita leer y escribir datos
 - Activo (RO) Nodo que permita sólo leer datos
 - Pasivo Nodo que no permite leer o escribir datos
- Empeora el rendimiento del nodo Master
- Limitación de distancia entre maestro y esclavo/s
 - Site (< 100m)
 - Local (< 20km)
 - Metro (< 100km)
 - Mundial

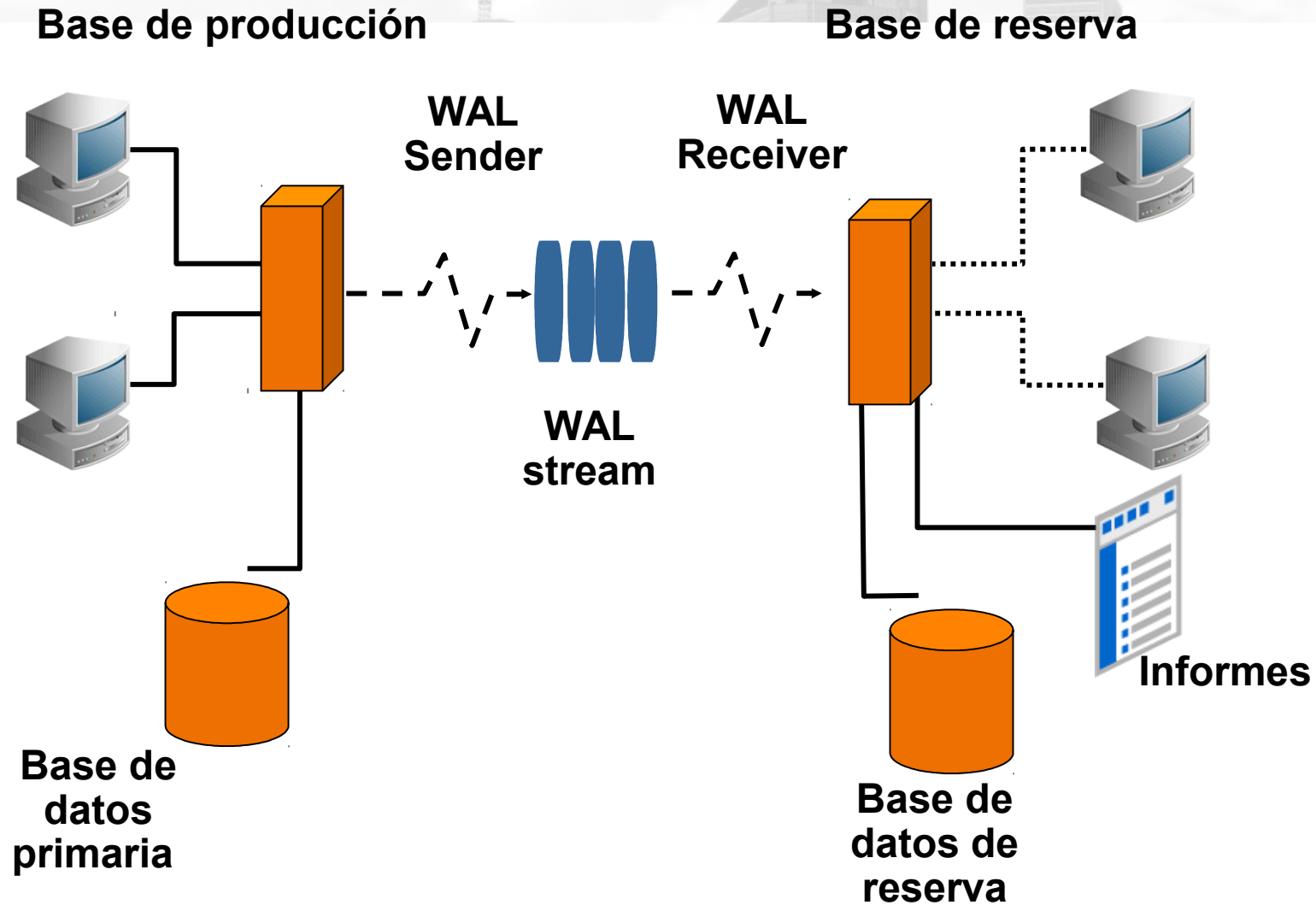
Opciones de alta disponibilidad

- PostgreSQL soporta varias utilidades para implementar la alta disponibilidad y escalar sobre múltiples servidores.
- En cluster de SO (Almacenamiento compartido)
- Replicado usando diferentes opciones disponibles:
 - Streaming Replication
 - Maestro-Esclavo usando Slony
- En este módulo se hablará como implementar
 - Streaming Replication
 - Replicación con Slony

Hot Streaming Replication

- Streaming Replication (Hot Standby) es una característica importante en PostgreSQL 9.0
- Facilita la gestión de servidores standby y distribuye las consultas de sólo lectura sobre estos.
- Los ficheros WAL files se transmiten a los servidores standby, reduciendo en gran medida los retardos gracias al log shipping.
- Los ficheros WAL también pueden ser archivados si se requiere PITR
- Dispone de opciones para replicación Síncrona/Asíncrona
- La versión PostgreSQL 9.2 provee la replicación en cascada, así un servidor de standby también puede enviar los cambios de replicación compartiendo la sobrecarga de un maestro.

Arquitectura de Hot Streaming



Preparar el servidor primario

- Configurar Continuos Archiving en el servidor primario.
- Los logs de archivado deben estar disponibles para los servidores de standby aunque el maestro esté parado.
- Modificar el parámetro *wal_level*:
 - `wal_level=hot_standby`
- Establecer el número mínimo adicional de segmentos que se mantienen en pg_xlog; los servidores standby van a necesitar buscarlos para la replicación:
 - `wal_keep_segments = 50`
- Habilita la posibilidad de transmitir WAL a los servidores standby vía procesos sender:
 - `max_wal_senders = 4`

Configurar replicación síncrona mediante streams

- El nivel por defecto de la replicación mediante streams es asíncrono.
- El nivel síncrono permite configurar la replicación para dar una doble seguridad de replicación.
- En el commit, las transacciones deben ser escritas en los ficheros del wal tanto en el primario como en hot standby.
- Se deben configurar parámetros adicionales:
 - `synchronous_commit=on`
 - `synchronous_standby_names`
 - Lista separada por comas de los nombre de los servidores standby que soportan replicación síncrona.
- Durante la configuración de la replicación síncrona `pg_start_backup()` y `pg_stop_backup()` son ejecutados en una sesión con `synchronous_commit = off`

Configurar la conexión desde el Standby

- Configurar la autenticación en el primario para permitir conexiones de replicación desde los servidores de reserva.
- Poner una o varias entradas en el fichero pg_hba.conf con el campo base de datos establecido a replication.
- Ejemplo de pg_hba.conf del primario:

```
host      replication  all      127.0.0.1/32  trust
```

Nota: Necesitará reiniciar/recargar el servidor

Hacer un backup completo del servidor primario

- Conectar a la base de datos como supersuario y lanzar el comando:
 - `SELECT pg_start_backup('label');`
Donde label es cualquier cadena que se quiera usar para identificar de manera única esta operación de backup.
- Realizar el backup, usando cualquier herramienta de backup de sistemas de ficheros como tar
 - `cp -rp /opt/PostgreSQL/9.2/data /backup/data1`
- De nuevo conectarse a la base de datos como superusuario y lanzar el comando:
 - `SELECT pg_stop_backup();`
- Copia el directorio de backup en el servidor de reserva
- Conviene cambiar el número de puerto a 5433 para el servidor de reserva en caso que configure el servidor de reserva en la misma máquina

Configurar servidor Standby

- Después de copiar el backup base en el servidor standby debe de borrar el fichero `postmaster.pid` del directorio de datos del servidor standby
- `rm /backup/data/postmaster.pid`
- Abrir el fichero `postgresql.conf` del directorio de datos del servidor standby y configura los siguientes parámetros:
 - **hot_standby** (booleano): Indica si se puede conectar y ejecutar consultas durante la recuperación.
 - **max_standby_archive_delay** (entero): Este parámetro determina cuanto tiempo esperará el servidor standby antes de cancelar las queries en caso de que entren en conflicto con sus archives.
 - **max_standby_streaming_delay** (entero): Este parámetro determina cuanto tiempo esperará el servidor standby antes de cancelar las queries en caso de que entren en conflicto con su replicación vía stream.

Configurar servidor Standby

- Crea el fichero recovery.conf dentro del directorio de datos del servidor standby
 - `vi /backup/data1/recovery.conf`
- Habilita la replicación mediante streams en recovery.conf:
 - `standby_mode = 'on'`
 - `primary_conninfo = 'host=localhost port=5432'`
 - El nombre de *host*, *número de puerto*, usuario y contraseña de la conexión al servidor primario se pueden especificar en el parámetro *primary_conninfo*.
 - `trigger_file = '/path_to/trigger'`
 - Indica un fichero trigger cuya presencia debe provocar el failover de la replicación de streams
 - `restore_command = 'cp /path_to/archive/%f "%p"'`
 - Indica el comando usado para cargar los segmentos archivados desde el archivo WAL
- Inicia el servidor usando el comando `pg_ctl`
 - `pg_ctl -D /backup/data1 -l /backup/data1/logfile start`

Añadir servidores standby replicados en cascada

- Clone el servidor primario usando pg_basebackup:
 - *pg_basebackup -h localhost -U replication_user -p 5433 -D /backup/data2*
- Un nuevo cluster se creará en el directorio de datos “data2”
- Cambie el puerto de cluster “data2” a 5434
- Cree el fichero recovery.conf en “data2”
 - *standby_mode = on*
 - *primary_conninfo = 'host=localhost port=5433 user=replication_user password=secret'*
- Modifique el fichero pg_hba.conf del cluster data1 que permita conexiones desde el cluster data2
 - host replication all 127.0.0.1/32 trust*
- Arranque el cluster:
 - *pg_ctl -D backup/data2 start*

Monitorización Hot Standby

- Se puede monitorizar la replicación de streams basada en la cantidad de registros WAL generados en el primario, pero que aún no se han aplicado en el standby.
- Puedes calcular este retardo, comparando la posición actual de escritura en WAL en el primario con la última posición de WAL recibida en el standby.
- Para ello usar `pg_current_xlog_location` en el maestro y la función `pg_last_xlog_receive_location` en el standby.
- También es posible obtener un listado de procesos WAL sender vía la vista `pg_stat_replication`.

Ejemplo de replicación en streams

- En este ejemplo nosotros configuraremos una replicación por streams asíncrona en una máquina local donde el cluster por defecto este levantado y funcionando También habilitaremos el archivado de WAL que permite hacer PITR cuando se requiera.
- Comprueba que el cluster de PostgreSQL está funcionando en el puerto 5432 y el directorio de datos está situado en el directorio /opt/PostgreSQL/9.2/data antes de continuar

Ejemplo de replicación en streams

- Pasos en el primario
 1. Entrar como usuario del sistema operativo postgres: `su - postgres`
 2. Abrir el fichero `postgresql.conf` con el editor `vi`
 3. Cambiar los siguientes parámetros:
 - `wal_level = hot_standby`
 - `max_wal_senders = 2`
 - `wal_keep_segments = 32`
 - `archive_mode = on`
 - `archive_command = 'cp %p /home/postgres/arch_dest/%f'`
 4. Guardar y cerrar el fichero `postgresql.conf`
 5. Crear el directorio para archivar los WAL
 - `mkdir /home/postgres/arch_dest`
 6. Abrir el fichero `pg_hba.conf` usando el editor `vi` y añadir la siguiente entrada:
 - `host replication all 127.0.0.1/32 trust`
 7. Reiniciar el servidor primario
 - `/etc/init.d/postgresql-9.2 restart` or `pg_ctl -D /opt/PostgreSQL/9.2/data -mf restart`

Ejemplo de replicación en streams

8. Hacer un backup base copiando el directorio de datos del servidor primario en el servidor standby.

- `pg_basebackup -h localhost -U postgres -D /home/postgres/data`

- Los siguientes pasos se pueden llevar a cabo en diferentes máquinas standby si existieran. En este caso no olvidar cambiar `pg_hba.conf` añadiendo la entrada mencionada en anteriores transparencias y copiar el backup en cada servidor de standby

Ejemplo de replicación en streams

- Pasos a realizar en el hot standby
 1. Entrar como usuario postgres del SO: `su - postgres`
 2. Abrir `postgresql.conf` con el editor `vi`
 - `Port=5433` - Esto no es necesario si los esclavos van a funcionar en diferentes servidores
 - `hot_standby=on`
 3. Borrar el fichero `postmaster.pid` del directorio de dato del servidor standby
 - `rm /backup/data/postmaster.pid`
 4. Crear el fichero `recovery.conf` dentro del directorio de datos del servidor standby
 - `vi /home/postgres/data/recovery.conf`
 - `standby_mode = 'on'`
 - `primary_conninfo = 'host=localhost port=5432'`
 - `trigger_file = '/home/postgres/trigger_hot_standby_failover'`
 5. Iniciar el servidor con el comando `pg_ctl`
 - `pg_ctl -D /home/postgres/data -l /home/postgres/data/logfile start`
 6. Comprueba que los cambios hechos en tiempo real están siendo transferidos mediante replicación por streams.

Replicación con Slony-I

- Slony-I es una herramienta de replicación asíncrona maestro-esclavo para PostgreSQL.
- Slony-I nos ofrece un servicio de continuidad (HA) gracias al uso de switchover y failover entre sistemas.
- Las bases de datos esclavas se pueden usar para reporting dando balanceo de lectura para el servidor primario.
- Los parámetros del servidor standby pueden configurarse independientemente.
- Slony-I no necesita de un costoso y complejo hardware o software de mirroring.

Componentes de replicación y vocabulario

- **Replication cluster** – Un conjunto de instancias de bases de datos en los que la replicación se llevará a cabo
- **Master node** – La base base primaria con las que las aplicaciones interactúan
- **Slave node** – Todos los nodos que forman parte del replication cluster exceptuando el master node
- **Replication set** – Las tablas y secuencias que se van a replicar
- **Origin** – Nodo donde la aplicación puede modificar una tabla
- **Subscribers** – Otros nodos que quieren recibir los datos
- **Cascaded replication** – Cuando un nodo se convierte en master node de otro
- **Switchover** – Una promoción de esclavo a maestro planificado
- **Failover** – Una promoción de esclavo a maestro no planificado

Componentes de replicación y vocabulario

- slon daemon
 - El proceso principal que funciona en cada nodo y es responsable de sincronizar los cambios entre maestro y esclavos
- slonik
 - La aplicación que ejecuta las ordenes que sirven para configurar y modificar las configuraciones del replication cluster
- Replication Schema
 - Mantiene toda la información necesaria para el replication cluster como son la información de configuración, transacciones pendientes, estados, etc.

Slonik Script

- Se emplea para lanzar ficheros shell scripting.
- Usos típicos
 - Crear clusters
 - Añadir tables
 - Switchover
 - Failover

Configurar replicación con Slony-I

- Preparar el servidor primario
- Añadir una entrada en el pg_hba del servidor standby y recargar el servidor primario
- Hacer un backup de estructura del primario con pg_dump
- Preparar los servidores esclavos
- Añadir una entrada para el servidor primario y recargar el servidor standby
- Restaurar en el standby el backup de estructura
- Crear los scripts slonik
- Crear el replication cluster

Añadir la información de conexión maestro esclavo

Configurar replicación con Slony-I

- Crear el replication set
- Añadir las tablas y secuencias que van a ser replicadas
- Añadir la información para la comunicación de slon
- Ejecutar el script en el maestro
- Crear un script de suscripción
- Añadir la información de conexión maestro esclavo
- Añadir la información de suscripción
- Ejecutar el script en el esclavo
- Iniciar el demonio/proceso de replicación en el maestro y esclavo

Switchover

- Un cambio de rol de maestro controlado originado por mantenimiento del sistema
- Es necesario que Slonik ejecute los siguientes comandos:
 - `lock set (id = 1, origin = 1);`
 - `wait for event (origin = 1, confirmed = 2);`
 - `move set (id = 1, old origin = 1, new origin = 2);`
 - `wait for event (origin = 1, confirmed = 2);`
- Reconectar las aplicaciones al nuevo nodo maestro

Failover

- Hay una posibilidad de recuperar un servidor fallido.
 - En un escenario de replicación maestro-esclavo, existe la posibilidad de pérdida de las transacciones que todavía no han sido replicadas.
- En Slonik es necesario ejecutar el siguiente comando:
 - `failover (id = 1, backup node = 2);`

Failover (continuación)

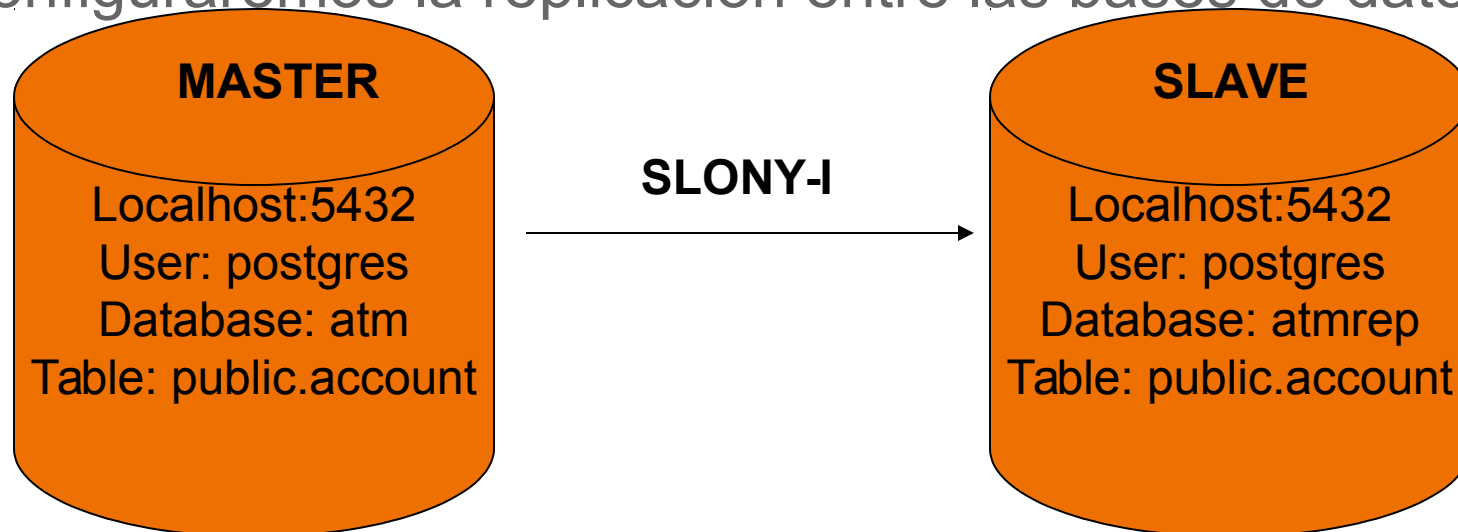
- Reconectar las aplicaciones al nuevo nodo maestro
- Después de que el failover esté completado y el nuevo nodo acepta operaciones de escritura contra las tablas, se borra los restos del nodo antiguo con el comando slonik:
 - `drop node (id = 1, event node = 2);`

Limitaciones de la replicación

- No detecta ni propaga los cambios en la definición de tablas
- Las tablas replicadas necesitan tener un índice único
- No replica objetos grandes (LO)
- No permite múltiples maestros
- No puede detectar un fallo de un esclavo
- No puede replicar cambios en roles y usuarios

Ejemplo de replicación con Slony

- En este ejemplo vamos a crear dos nuevas bases de datos llamadas atm y atmrep.
- Crearemos una nueva tabla account en el esquema público de la base de datos atm e insertamos algunas filas en ésta.
- Después configuraremos la replicación entre las bases de datos atm y atmrep



Ejemplo de replicación con Slony

1. Comprueba la instalación de Slony (instálelo usando stack builder en caso de no estar instalado)

- Entramos al sistema con el usuario postgres
- `which slon`
- `slonik -version`

2. Configuramos la base de datos maestra: atm

- `psql`
- `postgres=# create database atm;`
- `postgres=# \c atm`
- `atm=# create table account (id numeric primary key, name varchar);`
- `atm=# insert into account values (1, 'A'), (2, 'B'), (3, 'C');`

Ejemplo de replicación con Slony

3. Configuramos la base de datos esclava

- `atm=# create database atmrep;`
- `atm=# \q`

4. Haga un backup de estructura de la base de datos atm y restáurela sobre atmrep

- `pg_dump -s -p 5432 -h localhost atm | psql -h localhost -p 5432 atmrep`

5. Compruebe que no hay datos en la tabla account de la base de datos esclavo

- `psql atmrep`
- `atmrep=# select*from account;`

Ejemplo de replicación con Slony

6. Crear el script de slonik para configurar el maestro/esclavo

- `vi init_master.slونيك`
 - `#!/bin/sh`
 - `cluster name = mycluster;`
 - `node 1 admin conninfo = 'dbname=atm host=localhost port=5432 user=postgres password=postgres';`
 - `node 2 admin conninfo = 'dbname=atmrep host=localhost port=5432 user=postgres password=postgres';`
 - `init cluster (id=1);`
 - `create set (id=1, origin=1);`
 - `set add table(set id=1, origin=1, id=1, fully qualified name = 'public.account');`
 - `store node (id=2, event node = 1);`
 - `store path (server=1, client=2, conninfo='dbname=atm host=localhost port=5432 user=postgres password=postgres');`
 - `store path (server=2, client=1, conninfo='dbname=atmrep host=localhost port=5432 user=postgres password=postgres');`
 - `store listen (origin=1, provider = 1, receiver = 2);`
 - `store listen (origin=2, provider = 2, receiver = 1);`

Ejemplo de replicación con Slony

7. Crear el script slonik para la suscripción en el esclavo

- `vi init_slave.slonik`
 - `#!/bin/sh`
 - `cluster name = mycluster;`
 - `node 1 admin conninfo = 'dbname=atm host=localhost port=5432 user=postgres password=postgres';`
 - `node 2 admin conninfo = 'dbname=atmrep host=localhost port=5432 user=postgres password=postgres';`
 - `subscribe set (id = 1, provider = 1, receiver = 2, forward = no);`

Ejemplo de replicación con Slony

8. Ejecute el script slonik en el maestro

- `slonik init_master.slonik`

9. Ejecute el script slonik en el esclavo

- `slonik init_slave.slonik`

10. Inicie el demonio slon en el maestro

- `nohup slon mycluster "dbname=atm host=localhost port=5432 user=postgres password=postgres" &`

11. Inicie el demonio slon en el esclavo

- `nohup slon mycluster "dbname=atmrep host=localhost port=5432 user=postgres password=postgres" &`

12. Comprueba si están funcionando

- `ps -ef | grep slon`

13. En caso que no, buscar errores en el fichero nohup.out

Ejemplo de replicación con Slony

14. Comprueba que los datos son replicados desde la tabla maestra a la tabla esclava.

15. Conecta al maestro y consulta todos los registros de la tabla

- `psql atm`
- `atm=# select*from account;`

16. Añade una nueva fila en el maestro

- `atm=# insert into account (5, 'E');`

17. Conecta al esclavo y consulta todos los registros de la tabla

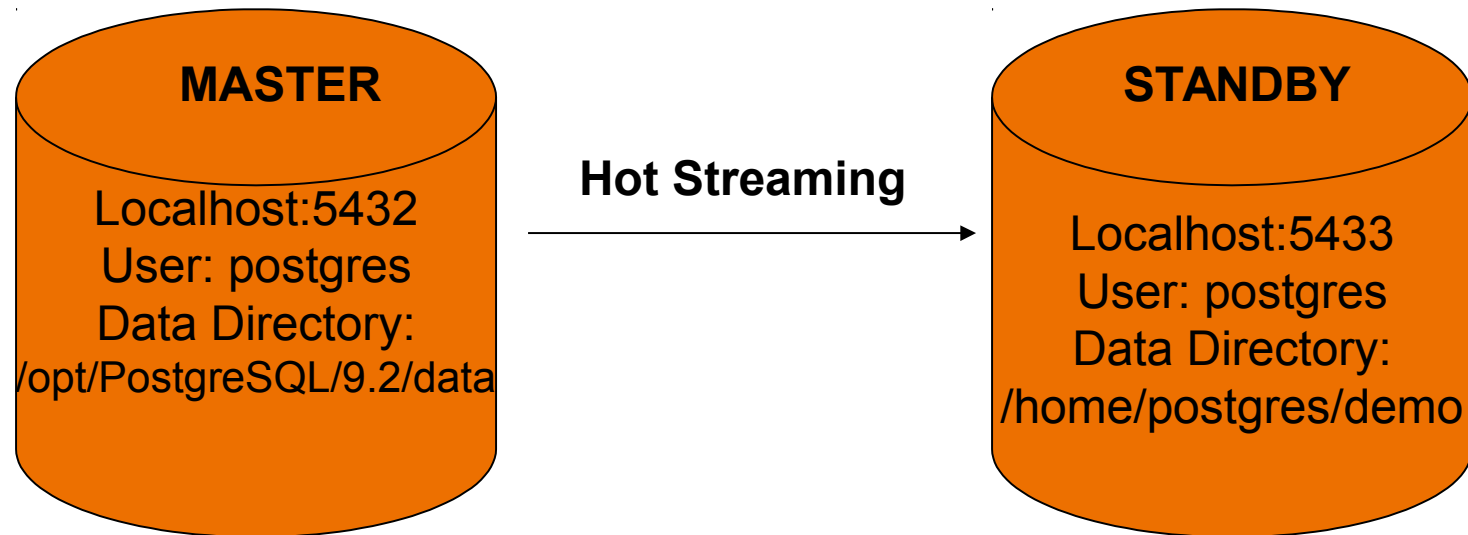
- `atm=# \c atmrep`
- `atmrep=# select*from account;`

18. Intenta insertar una fila en el esclavo

- `atmrep=# insert into account values (5, 'E');`
- `ERROR: Slony-I: Table account is replicated and cannot be modified on a subscriber node - role=0`

Ejercicio 1

- Implementa replicación mediante streaming replication entre dos clusters funcionando en tu servidor como se muestra:



Resumen

- En este módulo usted aprendió:
 - Alta disponibilidad de base de datos
 - Causas de pérdidas de datos
 - Plan para errores comunes
 - Elige un criterio
 - Opciones de alta disponibilidad
 - Replicación Hot Streaming arquitectura y configuración
 - Ejemplo de replicación de streams
 - Replicación de Slony-I, arquitectura y configuración
 - Switchovers y Failovers
 - Limitaciones de la replicación
 - Ejemplo de replicación con Slony
 - Laboratorios



Módulo – 5

Particionamiento de tablas

Objetivos

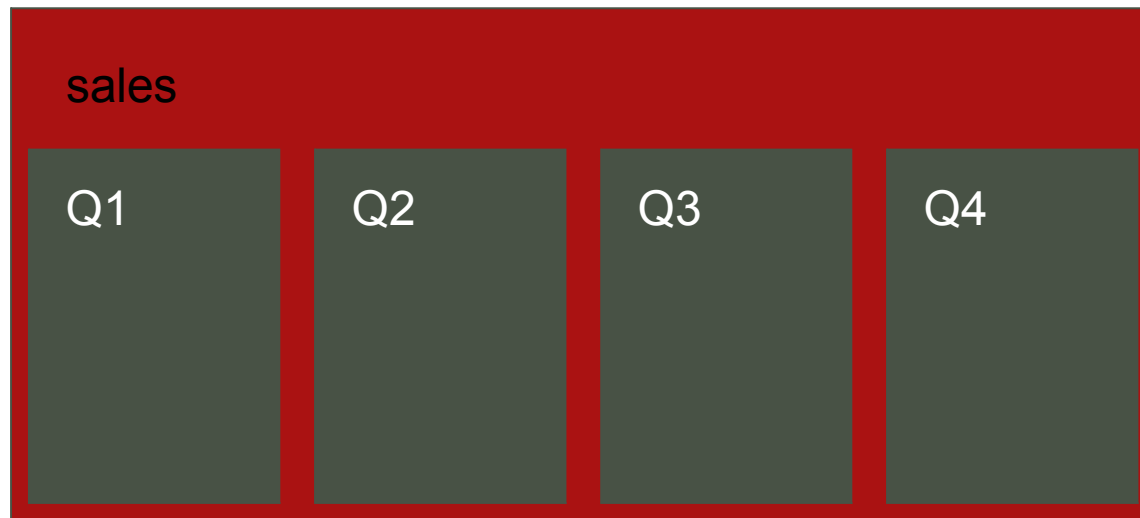
- En este módulo aprenderá:
 - Particionamiento
 - Métodos de particionamiento
 - Cuando particionar
 - Configurar el particionamiento
 - Ejemplo de particionamiento
 - Particionamiento y Constraint Exclusion
 - Advertencias
 - Laboratorio

Particionamiento

- Por particionar hacemos referencia a dividir físicamente una gran tabla lógica en partes más pequeñas.
- El rendimiento de consulta puede mejorar dramáticamente.
- Mejora del rendimiento de actualización al poder entrar particiones completas e índices completos en memoria.

Particionamiento

- PostgreSQL soporte particionamiento básico de tablas
- El particionamiento es totalmente transparente para las aplicaciones



Particionamiento

- Las eliminaciones masivas pueden llevarse a cabo por la simple eliminación de una de las particiones.
- En ocasiones, los datos menos utilizados pueden ser migrados a sistemas de almacenamiento mas baratos y lentos.
- PostgreSQL gestiona el particionamiento vía herencia de tablas. Cada partición tiene que ser creada como una tabla “hijo” o derivada de una única tabla padre.
- En sí misma, la tabla padre esta normalmente vacía, solo existe para representar el total del conjunto de datos.
- Todas las tablas “hijo” deben tener los mismos atributos lógicos como columnas, tipo de datos, pero pueden tener distintos atributos físicos como espacio de tablas, fill factor, etc.

Métodos de particionamiento

- Particionamiento por rangos
 - Las particiones por rangos están definidas por columnas clave, no deben existir superposiciones ni espacios entre rangos.
- Lista de particionamiento
 - Cada valor clave figura explícitamente para el esquema de particionamiento.

Cuando particionar

- Aquí tenemos algunas sugerencias sobre cuando realizar el particionamiento:
 - Cuando la tabla es muy grande y hay problemas de rendimiento de consultas/informes sobre la tabla.
 - En tablas que contienen datos históricos donde los nuevos datos pueden ser añadidos a nuevas particiones.
 - Un ejemplo típico es aquel en el que los datos referentes a un mes se actualizan con frecuencia y los datos referentes a los once meses restantes son sólo de lectura.
 - Cuando el contenido de una tabla necesita ser distribuido en distintos dispositivos de almacenamiento, dependiendo del tratamiento de los datos.

Configuración del particionamiento

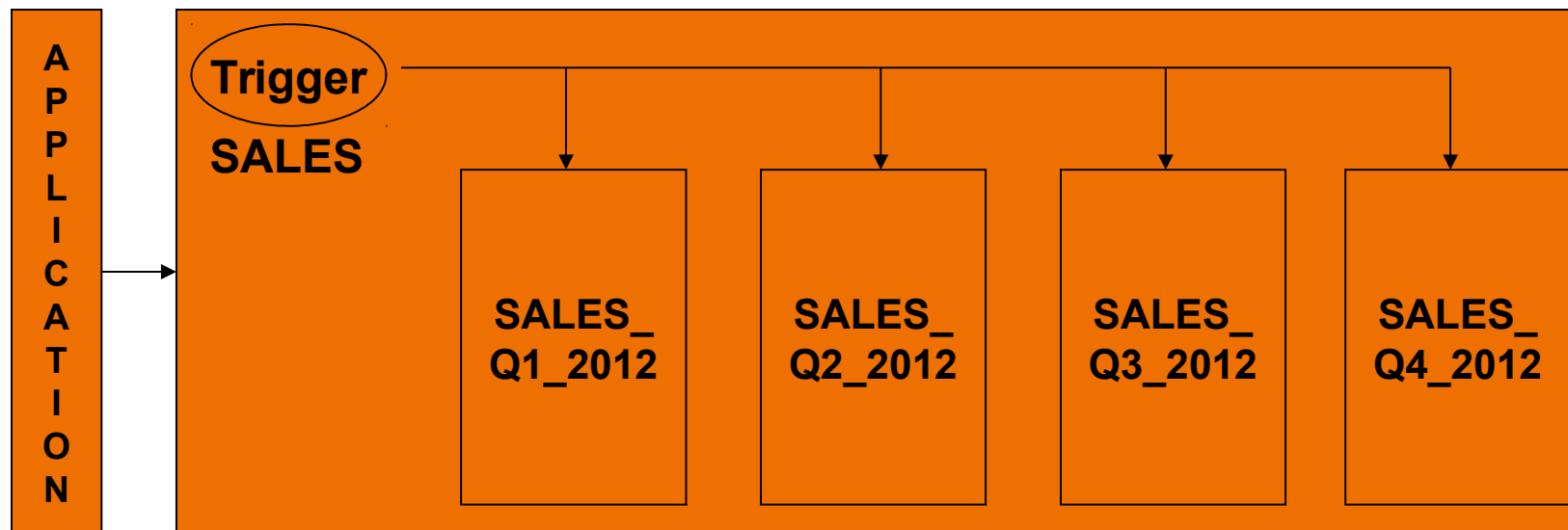
- El particionamiento puede ser implementado sin necesidad de requerir ninguna modificación para tus aplicaciones
- Crear la tabla base o tabla maestra
 - Esta tabla no contendrá datos.
 - Solo define restricciones que se aplicarán a todas las particiones.
- Crear la tabla “hijo” o tablas particionadas con su respectiva herencia de la tabla maestra
- Añadir restricciones a las tablas particionadas para definir los valores clave permitidos en cada partición.
- Asegurar que las restricciones garantizan que no hay superposiciones entre los valores clave permitidos en diferentes particiones.

Configuración de la partición

- Crear índices para cada partición
- Opcionalmente, definir una regla o trigger para redireccionar modificaciones de la tabla maestra a la partición adecuada.
- Asegúrese de que el parámetro de configuración `constraint_exclusion` está activado en `postgresql.conf`. Sin esto, las consultas no se optimizarán.

Ejemplo de particionamiento

- En este ejemplo configuraremos el particionamiento de la tabla “sales” en base a la columna “sale_date”
- Cada tabla particionada (tabla hijo) almacenará información sobre “ventas” para cada trimestre.
- Utilizaremos triggers



Ejemplo de particionamiento

- Creando la Tabla Maestra.

```
CREATE TABLE sales(  
    sale_id      NUMERIC    NOT NULL,  
    sale_date DATE,  
    amount      NUMERIC  
);
```

Ejemplo de particionamiento

- Creando particiones de tablas hijo

```
CREATE TABLE sales_q1_2012 (  
    CHECK (sale_date >= DATE '2012-01-01' AND  
           sale_date < DATE '2012-04-01')  
    ) INHERITS (sales);
```

```
CREATE TABLE sales_q2_2012 (  
    CHECK (sale_date >= DATE '2012-04-01' AND  
           sale_date < DATE '2012-07-01')  
    ) INHERITS (sales);
```

```
CREATE TABLE sales_q3_2012 (  
    CHECK (sale_date >= DATE '2012-07-01' AND  
           sale_date < DATE '2012-10-01')  
    ) INHERITS (sales);
```

```
CREATE TABLE sales_q4_2012 (  
    CHECK (sale_date >= DATE '2012-10-01' AND
```

Ejemplo de particionamiento

- Crear índices en cada tabla particionada

```
CREATE INDEX sales_10q1_sale_date ON sales_q1_2012 (sale_date);
```

```
CREATE INDEX sales_10q2_sale_date ON sales_q2_2012 (sale_date);
```

```
CREATE INDEX sales_10q3_sale_date ON sales_q3_2012 (sale_date);
```

```
CREATE INDEX sales_10q4_sale_date ON sales_q4_2012 (sale_date);
```

Ejemplo de particionamiento

- Crear función trigger que será llamada por el propio trigger

```
REPLACE FUNCTION sales_part_func () RETURNS trigger AS $$
BEGIN
    IF NEW.sale_date >= DATE '2012-01-01' and NEW.sale_date < DATE '2012-04-01' then
        INSERT INTO sales_q1_2012 VALUES (NEW.sale_id,NEW.sale_date,NEW.amount);
    ELSEIF NEW.sale_date >= DATE '2012-04-01' and NEW.sale_date < DATE '2012-07-01' then
        INSERT INTO sales_q2_2012 VALUES ( NEW.sale_id,NEW.sale_date,NEW.amount);
    ELSEIF NEW.sale_date >= DATE '2012-08-01' and NEW.sale_date < DATE '2012-10-01' then
        INSERT INTO sales_q3_2012 VALUES ( NEW.sale_id,NEW.sale_date,NEW.amount);
    ELSEIF NEW.sale_date >= DATE '2012-10-01' and NEW.sale_date < DATE '2013-01-01' then
        INSERT INTO sales_q4_2012 VALUES ( NEW.sale_id,NEW.sale_date,NEW.amount);
    END IF;
    RETURN NULL;
END; $$ language PLPGSQL;
```

Ejemplo de particionamiento

- Creando un trigger de particionamiento en la tabla ventas.

```
CREATE TRIGGER sales_part_trig  
  BEFORE INSERT ON sales  
  FOR EACH ROW  
  EXECUTE PROCEDURE sales_part_func();
```

Ejemplo de particionamiento

- Inserte algunas filas en la tabla de ventas y verifique que han sido insertadas en tablas particionadas.
- Recuerde que la aplicación no conocerá esta partición, pero podrá acceder a la tabla hijo directamente para cualquier carga o eliminación masiva directa.
- Ver siempre el plan de ejecución de la consulta para verificar que solo las tablas hijo son exploradas o escaneadas.

Exclusión de particionamiento y restricción

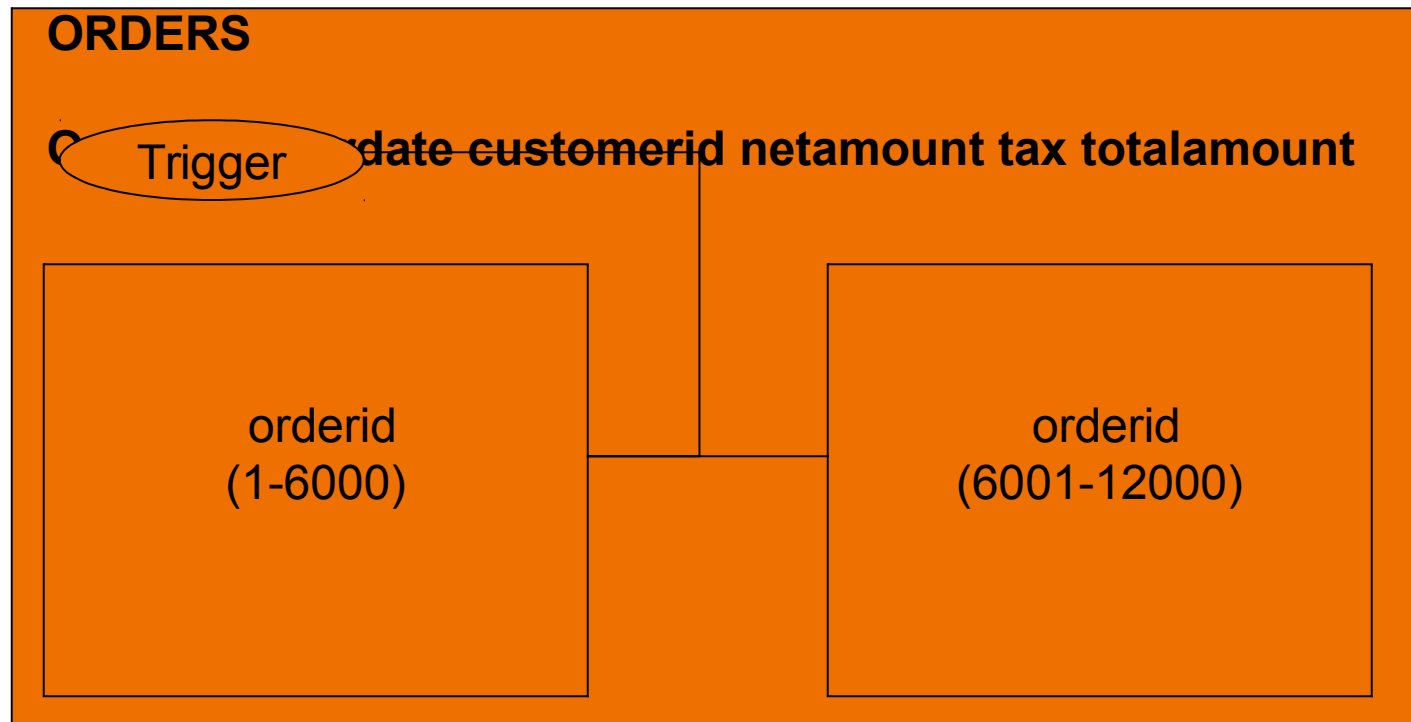
- Constraint Exclusion es una técnica de optimización de consulta que mejora el rendimiento para tablas particionadas.
- Ejemplo
- `SET constraint_exclusion = on;`
- `SELECT count(*) FROM sales WHERE sale_date >= DATE '2012-10-01';`
- Sin este parámetro activado la consulta antes mencionada escaneará cada una de las particiones de la tabla ventas.

Advertencias

- No hay forma de verificar que todas las restricciones “CHECK” son mutuamente excluyentes.
- No hay forma de especificar explícitamente que no se pueda insertar en la tabla maestra.
- Cada tabla “hijo” debe ser vacuumed y analyzed por separado.
- En un particionamiento basado en reglas, el comando COPY no puede ser utilizado en la tabla padre
- La clave externa no puede estar referida a ninguna columna de la tabla padre

Práctica Lab - 1

- Ver la estructura de la tabla orders en la base de datos de hoplastore
- Particionar la tabla según el esquema



Práctica lab - 2

- Crear clave primaria en la columna orderid de cada tabla particionada
- Ver el plan de ejecución de la siguiente consulta
 - `Select * from orders where orderid=3000;`
- Desconectar el constraint_exclusion y ver el plan de ejecución de la siguiente consulta
 - `Select * from orders where orderid=3000;`

Resumen

- En este módulo se han tratado los siguientes temas:
 - Particionamiento
 - Métodos de particionamiento
 - Cuando realizar un particionamiento
 - Configuración del particionamiento
 - Ejemplos de particionamiento
 - Exclusión de particionamiento y restricción
 - advertencias
 - Lab



Module - 6

Connection Pooling

Objectives

- En este módulo veremos:
 - Pgpool-II
 - Pgpool-II características
 - Configurar e instalar pgpool-II
 - Pgpool II Modes
 - Arrancar/parar pgpool-II
 - Pgpool-II ejemplo
 - Pgbouncer
 - Pgbouncer modos de conexión
 - Como están establecidas las conexiones
 - Configuración y gestión de Pgbouncer
 - Pgbouncer ejemplo
 - Lab

Pgpool-II

- pgpool-II es un middleware que trabaja entre los servidores de aplicaciones y la base de datos de PostgreSQL
- pgpool-II habla el protocolo de backend y frontend de PostgreSQL.
- Características:
 - Pool de conexiones
 - Replicación
 - Balanceo de carga
 - Limita las conexiones

Pooling de conexión

- pgpool-II ahorra conexiones al servidor de PostgreSQL
- Reutilizarlos cada vez que una nueva conexión con las mismas características (nombre de usuario , base de datos...) entra
- Reduce el tiempo de creación de conexión y mejora el rendimiento global del sistema.

Replicación

- pgpool-II puede gestionar numerosos servidores de PostgreSQL.
- Utilizar la función de replicación permite crear un backup en tiempo real con dos o más instancias, por lo que el servicio puede continuar sin parar los servidores en caso de fallo de alguna instancia.

Balanceo de carga

- Si una base de datos es replicada, ejecutando una consulta SELECT en cualquier otro servidor retornará el mismo resultado.
- Pgpool-II presenta ventajas en la característica de replicación a la hora de reducir la carga en cada servidor de PostgreSQL mediante la distribución de consultas SELECT entre múltiples servidores, mejorando el rendimiento total del sistema.
- El balanceo de carga funciona mejor en situaciones donde hay muchos usuarios ejecutando consultas al mismo tiempo.

Limitar las conexiones extras

- Hay un límite en el máximo número de conexiones concurrentes en PostgreSQL, y las conexiones son rechazadas cuando este se alcanza.
- Mantener un máximo número de conexiones concurrentes muy elevado incrementa el consumo de recursos y afecta al rendimiento del sistema.
- Pgpool-II también tiene un límite respecto al número máximo de conexiones, pero cualquier conexión extra será puesta en cola de espera en vez de devolver un error de forma inmediata.

pgpool-II Instalación

- pgpool-II puede descargarse en: <http://pgfoundry.org/projects/pgpool/>
- La instalación de pgpool-II necesita gcc 2.9 o superior y la librería libpq.

Configurando pgpool-II

- Los archivos de configuración de pgpool-II están por defecto en /usr/local/etc/pgpool.conf y /usr/local/etc/pcp.conf.
- Hay varios modos de operación en pgpool-II. Cada modo tiene asociado funciones que se pueden habilitar o deshabilitar, y parámetros de configuración específicos para controlar su comportamiento.
- Veamos los modos de Configuración en la siguiente diapositiva.

Configurando pcp.conf

- El pcp.conf es el fichero de usuario/contraseña para la autenticación con la interfaz.
- Se crea \$prefix/etc/pcp.conf.sample después de instalar pgpool-II.
 - `cp $prefix/etc/pcp.conf.sample $prefix/etc/pcp.conf`
 - `username:[password encrypted in md5]`
 - `[password encrypted in md5]` can be produced with the `$prefix/bin/pg_md5` command.
 - `pg_md5 -p password: <your password>`

Configurando pgpool.conf

- Cada modo de operación tiene parámetros específicos de configuración en pgpool.conf
- Se crea `$prefix/etc/pgpool.conf.sample` después de instalar pgpool.conf
 - `cp $prefix/etc/pgpool.conf.sample $prefix/etc/pgpool.conf`

Pgpool II Modos

- Raw:
 - En el modo raw, los clientes simplemente se conectan a los servidores PostgreSQL vía pgpool-II. Este modo es útil para limitar las conexiones al servidor o para habilitar *failover* con múltiples servidores.
- Modo de pool de conexión
 - En el modo de pool de conexión, se pueden usar tanto las funciones del modo raw como la función de pool de conexión.

Pgpool II Modos

- Modo de replicación
 - Este modo habilita la replicación de datos entre backends. se deberán de poner los parámetros de configuración inferiores además de todo lo anterior.
- Modo maestro/esclavo
 - Este modo es para usar pgpool-II junto a otro software de replicación maestro-esclavo (como Slony-I), para que se produzca una replicación de datos real. La información de los nodos de la base de datos se deberán establecer en modo de replicación. Además, se deberá poner `set master_slave_mode` y `load_balance_mode` a true. Pgpool-II mandará consultas que deberán ser replicadas en la base de datos máster y a las demás se les balanceará la carga si fuera posible.

Preparar pool_hba.conf para autenticación del cliente (HBA)

- Igual que con el pg_hba.conf en PostgreSQL, pgpool soporta una función de autenticación de cliente similar usando un fichero llamado "pool_hba.conf".
- Al instalar pgpool, pool_hba.conf.sample se instalará en "/usr/local/etc", que es el directorio por defecto de los archivos de configuración.
- Copiar pool_hba.conf.sample como pool_hba.conf y editar si fuera necesario.

Iniciar/Parar pgpool-II

- Se deberán reiniciar todos los backends y las bases de datos de Sistema (si fuera necesario) antes de iniciar pgpool-II
 - `pgpool [-c][-f config_file][-a hba_file][-F pcp_config_file][-n][-d]`
- Existen dos maneras de parar pgpool-II. Una es mediante el comando PCP (descrito posteriormente). Otra es con el comando pgpool-II. A continuación se muestra un ejemplo de uso del comando pgpool-II.
 - `pgpool [-f config_file][-F pcp_config_file] [-m {s[mart]|f[ast]|i[mmediate]}}] stop`

Pgpool-II Ejemplo

- En este ejemplo aprenderemos como instalar pgpool-II
- Los siguientes pasos se pueden realizar utilizando el usuario root para sudo
 1. Descargar pgpool-II
 - <http://pgfoundry.org/frs/download.php/3076/pgpool-II-3.0.4.tar.gz>
 2. Unzip la fila tar
 - `tar -zxvf pgpool-II-3.0.4.tar.gz`
 - `cd pgpool-II-3.0.4`
 3. Puede ver los parámetros que se pueden configurar antes de compilar las fuentes pgPool-II
 - `./configure --help`
 4. Compilar e instalar las fuentes
 - `./configure --with-pgsql=/opt/PostgreSQL/9.2/bin --with-pgsql-includedir=/opt/PostgreSQL/9.2/include --with-pgsql-libdir=/opt/PostgreSQL/9.2/lib --prefix=/opt/PostgreSQL/9.2`
 - `make`
 - `make install`

Pgpool-II Ejemplo

5. Después de la instalación creemos los ficheros de configuración a través de los de muestra.

- `cd /opt/PostgreSQL/9.2`
- `cp etc/pcp.conf.sample etc/pcp.conf`
- `cp etc/pgpool.conf.sample etc/pgpool.conf`

6. Generar clave:

- `bin/pg_md5 -p postgres`

7. Añadir usuario dentro de pcp.conf

- `vi etc/pcp.conf`
- `postgres:e8a48653851e28c69d0506508fb27fc5`

8. Salvar y cerrar

Pgpool-II Ejemplo

9. Abrir el fichero pgpool.conf y configurar pgpool para proveer pooling de conexiones para nuestro cluster PostgreSQL que por defecto se ejecuta en el puerto 5432

10. `vi etc/pgpool.conf`

11. Editar los siguientes parámetros

- `listen_addresses = '*'`
- `port = 9999`
- `backend_hostname0 = 'localhost'`
- `backend_port0 = 5432`
- `backend_weight0 = 1`
- `backend_data_directory0 = '/opt/PostgreSQL/9.2/data'`

12. Salvar y guardar

Pgpool-II Ejemplo

13. Inicia pgpool-II

- `/opt/PostgreSQL/9.2/bin/pgpool -f /opt/PostgreSQL/9.2/etc/pgpool.conf -F /opt/PostgreSQL/9.2/etc/pcp.conf -n`

14. Conectar al cluster PostgreSQL utilizando pgpool

- `/opt/PostgreSQL/9.2/bin/psql -p 9999 postgres postgres`

15. Salir de la terminal psql y parar pgpool

- `/opt/PostgreSQL/9.2/bin/pgpool -mf stop`

Pgbouncer - "Bouncer is always right"

- Un pooler ligero de conexiones para PostgreSQL.
- Cualquier aplicación se puede conectar al Pgbouncer ya que éste se conecta a PostgreSQL.
- Pgbouncer ayuda a disminuir los impactos de conexiones en el servidor PostgreSQL.
- Pgbouncer proporciona un pooling de conexiones, reutilizando así conexiones ya existentes.

Tipos de conexiones

- pgbouncer soporta diversos tipos de pooling de conexiones:
 - **Session Pooling**

Una conexión de servidor se le asigna a la aplicación cliente para la vida de la conexión de cliente.
 - Transaction Pooling
 - Una conexión al servidor es asignada a la aplicación del cliente con la duración de una transacción
 - **Statement Pooling**
 - Una conexión de servidor es asignada a la aplicación del cliente para cada sentencia

Como se establecen las conexiones

- Una aplicación conecta a PgBouncer como si fuera una base de datos de PostgreSQL
- A partir de ahí, PgBouncer crea una conexión al servidor de la base de datos actual o reutiliza una existente del pool.
 - Paso 1: La aplicación del cliente intenta conectarse a PostgreSQL en el puerto donde se está ejecutando pgbouncer
 - Paso 2: El nombre de la base de datos suministrado por la aplicación del cliente debe coincidir con la lista en pgBouncer.ini
 - Paso 3: El nombre de usuario y contraseña suministrados, deben coincidir con la lista en users.txt
 - Paso 4: Si una conexión con los mismos ajustes está disponible en pool, será asignada al cliente, de otra manera una nueva conexión objeto será creada
 - Paso 5: Una vez el cliente salga del objeto de conexión volver al pool

Setup Pgouncer

- Crear pgbouncer.ini file.
 - `[databases] db1 = host=127.0.0.1 port=5432 dbname=db1`
 - `[pgbouncer] listen_port = 6543 listen_addr = 127.0.0.1 auth_type = md5 auth_file = users.txt logfile = pgbouncer.log pidfile = pgbouncer.pid admin_users = someuser`
- Crear users.txt file:
 - `"someuser" "same_password_as_in_server"`
- Lanzar pgbouncer:
 - `$ pgbouncer -d pgbouncer.ini`
- Tener tu aplicación (o el psql client) conectado a pgbouncer en lugar de tenerlo directamente con el servidor de PostgreSQL
 - `$ psql -p 6543 -U someuser template1`

Gestionar pgbouncer

- Muestra estadísticas, servidores, clientes, pools, listas, bases de datos, comandos fds que pueden ser utilizados.
- Gestiona pgbouncer mediante la conexión a la base de datos especial de administración y mostrar la ayuda
 - `$ psql -p 6543 -U someuser pgbouncer`
 - `pgbouncer=# show help;`
 - `NOTICE: Console usage`
 - `DETAIL: SHOW [HELP|CONFIG|DATABASES|FDS|POOLS|CLIENTS|SERVERS|SOCKETS|LISTS|VERSION]`

Pgbouncer Ejemplo

- En este ejemplo instalaremos y configuraremos pgbouncer para dar conexión de pooling para Postgres. El cluster se ejecutara en el puerto 5432
- Pasos:
 1. Abrir la aplicación Stack Builder en el menú de aplicaciones
 2. Seleccionar cluster por defecto ejecutándolo en el puerto 5432 y clic next
 3. Selecciona pgbouncer de complementos y haga click en siguiente
 4. Seleccion el puerto (por defecto 6432) para pgbouncer y click next para instalar
 5. Pgbouncer comenzará automáticamente y podrá ser parado y reiniciado utilizando pgbouncer service, que lo registrará automáticamente.
 - `pgbouncer -d /opt/PostgreSQL/9.2/pgbouncer/share/pgbouncer.ini`

Pgbouncer Ejemplo

- Vamos a configurar pgbouncer para proveer conexión pooling para la base de datos de hoplastore para usuarios postgres
 - Pasos:
7. Verificar que pgbouncer está ejecutándose:
 - `ps aux | grep pgbouncer`
 8. Abrir entrada del archivo de complemento pgbouncer.ini para hoplastore
 - `vi /opt/PostgreSQL/9.2/pgbouncer/share/pgbouncer.ini`
 - `hoplastore = host=127.0.0.1 port=5432 user=postgres password=postgres dbname=hoplastore`
 - Verify following entries:
 - `listen_addr = *`
 - `listen_port = 6432`
 - `auth_file = /opt/PostgreSQL/9.2/pgbouncer/etc/userlist.txt`
 - `pool_mode = session`

Pgbouncer Ejemplo

9. Añadir usuario de postgres en la fila de autenticación de usuario

- `vi /opt/PostgreSQL/9.2/pgbouncer/etc/userlist.txt`
- `"postgres" "postgres"`

10. Reiniciar pgbouncer para reflejar los cambios

- `/etc/init.d/pgbouncer restart`

11. Vamos a conectarnos a la base de datos de hoplastore utilizando pgbouncer

- `psql -p 6432 edbstore postgres`



Lab Ejercicio - 1

- Configurar pgbouncer para servir de pool de conexiones a su instancia

Resumen

- En este módulo usted ha aprendido
 - Pgpool-II
 - Pgpool-II Características
 - Instala y configura pgpool-II
 - Pgpool II Modos
 - iniciar/parar pgpool-II
 - Pgpool-II ejemplos
 - Pgbouncer
 - Pgbouncer Modos de conexión
 - Como se establecen las conexiones
 - Instalación y gestión de Pgbouncer
 - Pgbouncer Ejemplo
 - Lab



Módulo – 7

Monitorización

Objetivos

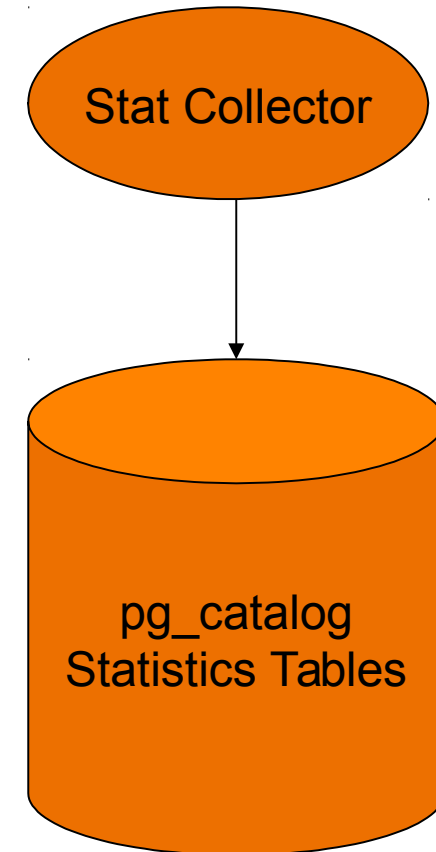
- En este módulo aprenderá:
 - Monitorización de la base de datos
 - Estadísticos de base de datos
 - Recolector estadístico
 - Tablas estadísticas de base de datos
 - Sistema operativo monitorización de procesos
 - Sesiones y bloqueos actuales
 - Carga de consultas de ejecución lenta
 - Uso del disco
 - Postgres Enterprise Manager
 - PEM - Características
 - PEM -Arquitectura
 - Lab

Monitorización de la base de datos

- La monitorización de la base de datos consiste en capturar y grabar los eventos de la base de datos
- Esta información ayuda a DBA a detectar, identificar y arreglar errores potenciales de la base de datos
- Las estadísticas de la monitorización de la base de datos hacen más fácil para DBAs la monitorización de la salud y rendimiento de las bases de datos PostgreSQL .
- Hay muchas herramientas disponibles para la monitorización de la actividad de la base de datos y analizar su rendimiento.

Estadísticos de base de datos

- Las tablas del catálogo de estadísticas de base de datos almacenan información sobre la actividad de la base de datos
 - Sesiones ejecutándose
 - Ejecución SQL
 - Bloqueos
 - DML conteos
 - Row conteos
 - Uso del índice



El recolector estadístico

- La recolección de estadísticas es un proceso que acumula y reporta información sobre la actividad de la base de datos
- Parámetro estadístico:
 - `track_counts` controla los estadísticos de tablas e índices
 - `track_activities` permite la supervisión del comando actual
- El recolector estadístico utiliza archivos temporales almacenados en el subdirectorio `pg_stat_tmp`
- Las estadísticas permanentes están almacenadas en el esquema `pg_catalog` en el subdirectorio global
- Performance Tip: El parámetro `stats_temp_directory` puede ser señalado como un archivo de sistema RAM-based

Tablas de bases de datos estadísticos

- El esquema `pg_catalog` contiene un conjunto de tablas, vistas y funciones que almacenan e informan los estadísticos de la base de datos
- `pg_class` and `pg_stats` tablas catalogadas que almacenan los estadísticos
- La vista `pg_stat_database` puede ser utilizada para observar información de la base de datos
- `pg_stat_bgwriter` muestra el background de los estadísticos de bgwriter
- `pg_stat_user_tables` muestra información sobre las actividades en la tabla como inserciones, actualizaciones, eliminaciones, vacuum, autovacuum etc.
- `pg_stat_user_indexes` muestra información acerca el uso del índice para todos los usuarios de la tabla

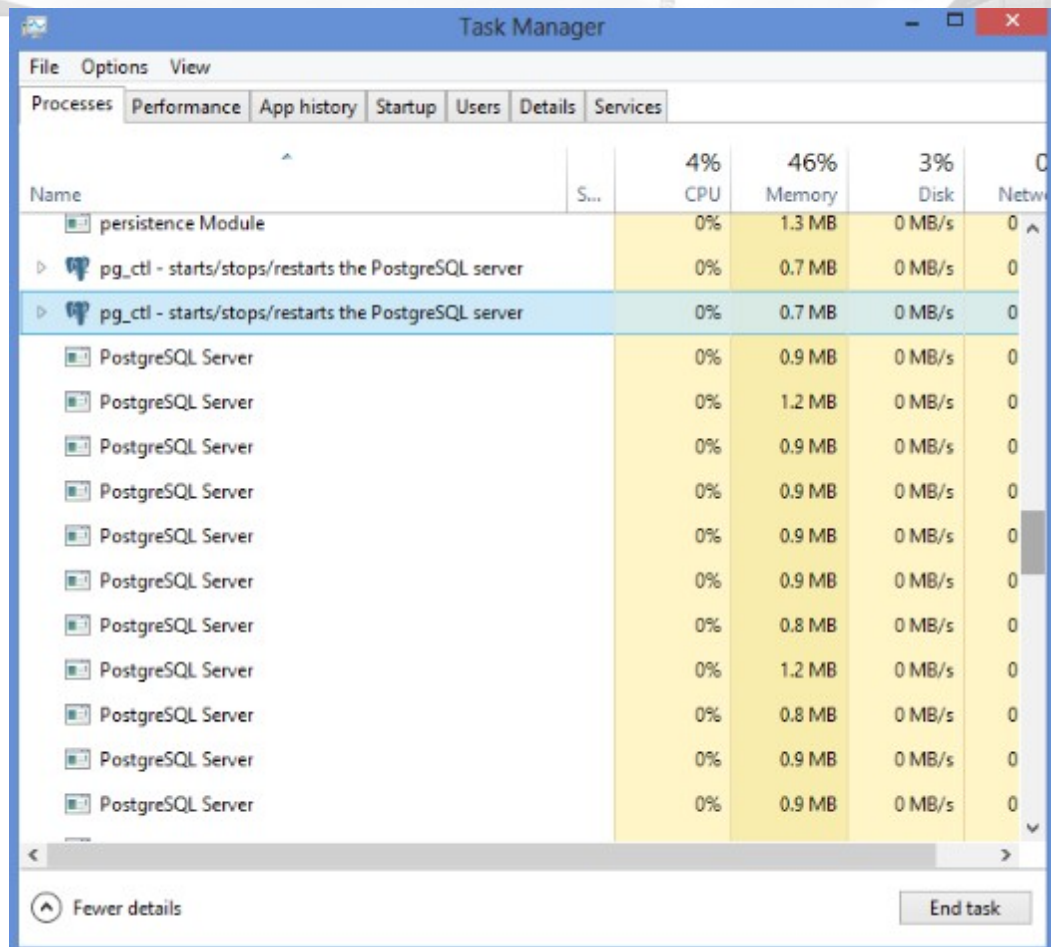
Sistema operativo proceso de monitorización (Unix)

- Unix
 - ps – Información acerca procesos actuales
 - `ps waux | grep post` (for Linux and OS/X)
 - `ps -ef | grep post` (other Unix)
 - netstat – Información acerca de las conexiones actuales en red
 - `netstat -an | grep LISTEN`

Procesos de servidor - Linux

```
- [PostgreSQL@localhost bin]$ ps -ef | grep post
- 502          27247          1  0 13:22 pts/3      00:00:06
  /opt/PostgreSQL/8.4//bin/postmaster -D ../data
- 502          27248 27247    0 13:22 pts/3      00:00:00 PostgreSQL: logger process
- 502          27250 27247    0 13:22 pts/3      00:00:01 PostgreSQL: writer process
- 502          27251 27247    0 13:22 pts/3      00:00:00 PostgreSQL: archiver process
- 502          27252 27247    0 13:22 pts/3      00:00:00 PostgreSQL: stats buffer
  process
- 502          27253 27252    0 13:22 pts/3      00:00:00 PostgreSQL: stats collector
  process
- 502          27275 27247    0 13:24 pts/3      00:00:00 PostgreSQL: fred edb [local]
  SELECT
- 502          27288 27247    0 13:27 pts/3      00:00:00 PostgreSQL: john test
  127.0.0.1(32818) idle
```

OS Proceso de monitorización (Windows)



The screenshot shows the Windows Task Manager window with the 'Processes' tab selected. The window title is 'Task Manager'. The menu bar includes 'File', 'Options', and 'View'. The tabs at the bottom are 'Processes', 'Performance', 'App history', 'Startup', 'Users', 'Details', and 'Services'. The 'Processes' tab is active, displaying a list of running processes. The columns are 'Name', 'Status', 'CPU', 'Memory', 'Disk', and 'Network'. The processes listed are:

Name	Status	CPU	Memory	Disk	Network
persistence Module		0%	1.3 MB	0 MB/s	0
pg_ctl - starts/stops/restarts the PostgreSQL server		0%	0.7 MB	0 MB/s	0
pg_ctl - starts/stops/restarts the PostgreSQL server		0%	0.7 MB	0 MB/s	0
PostgreSQL Server		0%	0.9 MB	0 MB/s	0
PostgreSQL Server		0%	1.2 MB	0 MB/s	0
PostgreSQL Server		0%	0.9 MB	0 MB/s	0
PostgreSQL Server		0%	0.9 MB	0 MB/s	0
PostgreSQL Server		0%	0.9 MB	0 MB/s	0
PostgreSQL Server		0%	0.9 MB	0 MB/s	0
PostgreSQL Server		0%	0.8 MB	0 MB/s	0
PostgreSQL Server		0%	1.2 MB	0 MB/s	0
PostgreSQL Server		0%	0.8 MB	0 MB/s	0
PostgreSQL Server		0%	0.9 MB	0 MB/s	0
PostgreSQL Server		0%	0.9 MB	0 MB/s	0

At the bottom of the window, there is a 'Fewer details' button on the left and an 'End task' button on the right.

Sesiones actuales y bloqueos

- La tabla `pg_locks` almacena información acerca de los bloqueos en la base de datos actual.
- `pg_stat_activity` muestra la id del proceso, base de datos, usuario, consultas y el tiempo inicial de la ejecución de la consulta actual.
 - Ejecutar el siguiente comando SQL :
 - `SELECT * FROM pg_stat_activity;`
- Terminar la sesión utilizando la función `pg_terminate_backend`.
- Deshaga una transacción utilizando la función `pg_cancel_backend`.

Carga de consultas de ejecución lenta

- `log_min_duration_statement`
 - Establezca una sentencia de tiempo mínimo de ejecución(en milisegundos) esto hará que la sentencia quede registrada. Todas las sentencias SQL que se ejecuten en el tiempo especificado o superior quedarán registradas con su duración.
 - Estableciendo esto a cero quedarán registradas todas las consultas y sus duraciones.
 - El valor -1 (predeterminado) deshabilita la característica.
 - Permitir esta opción puede ser útil para rastrear las consultas no optimizadas en tus aplicaciones.

Uso del disco

- Las siguientes funciones estadísticas pueden ser utilizadas para ver el tamaño de la base de datos:
 - `pg_database_size(name)`
 - `pg_tablespace_size(name)`
- EL uso del comando de disco 'du' es también útil para determinar el crecimiento del disco tiempo a tiempo.

Postgres Enterprise Manager

- Postgres Enterprise Manager (PEM) es un software de EDB para realizar multifunciones.
- Asiste a DBAs en la administración, monitorización, y puesta a punto de PostgreSQL. Y también a desarrolladores en sus tareas rutinarias.
- PEM está estructurado para gestionar y monitorizar múltiples servidores de una única consola

PEM - Características

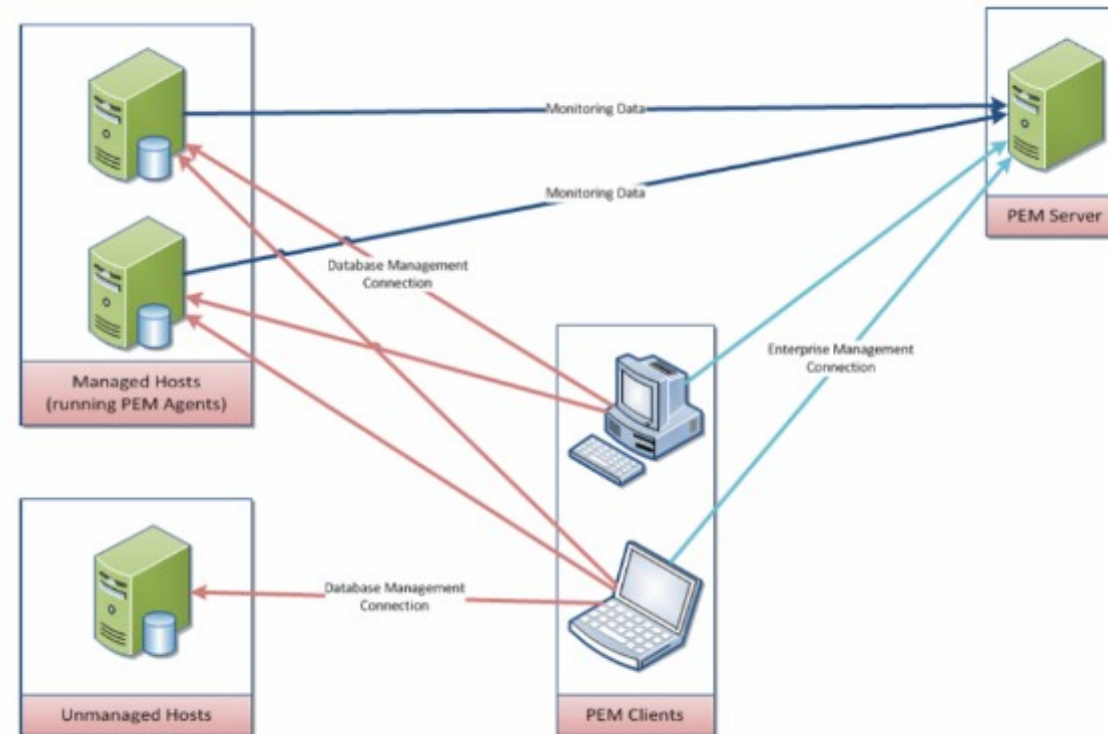
- Los dashboards globales te mantendrán al día en las subidas o bajadas del status de todos tus servidores.
- Gestiona las bases de datos independientemente del sistema operativo.
- La administración de la base de datos puede ser llevada a cabo mediante una interfaz gráfica.
- Provee SQL completo integrado en un ambiente de desarrollo (IDE).
- Agentes ligeros y eficientes supervisan todos los aspectos de las operaciones de cada servidor de base de datos

PEM - Características

- Habilidad para crear umbrales de rendimiento para cada clave métrica e.j. memoria, almacenamiento, etc.
- Cualquier resultado de violación del umbral dado en una alerta será enviado a un panel de control centralizado.
- Todas las estadísticas clave relacionadas con el rendimiento se recogen y se conservan de forma automática.
- Un capacity manager para realizar la previsión del crecimiento vegetativo.
- Identifica y perfecciona o afina malos funcionamientos de sentencias SQL.
- Amplio soporte de plataforma.

PEM - Arquitectura

- PEM está compuesto de tres componentes primarios
- The PEM Servidor
- The PEM Agente
- The PEM Cliente



Ejercicio Lab - 1

En este Lab aprenderemos a como utilizar PEM

1. Descargar e instalar Postgres Enterprise Manager Server
2. Descargar e instalar Postgres Enterprise Manager Client
3. Abrir PEM Client
4. El Cluster predeterminado enlazado se ejecuta en el puerto 5432 con PEM agent para coleccionar la monitorización de datos
5. Mirar panel de control global para PEM server
6. Mirar la actividad actual de la base de datos del panel de control para su cluster predeterminado
7. Ver el panel de control de la memoria para su cluster predeterminado
8. Mirar información OS para el servidor donde el cluster predeterminado se esté ejecutando

Ejercicio Lab - 2

1. Escriba una consulta para ver la sesión actual que está ejecutándose en tu cluster predeterminado.
2. Escriba una consulta para ver cuantos usuarios están en estado idle
3. Abra otro terminal y conectelo con la base de datos edbstore utilizando psql
 - Encuentra la id del proceso para los usuarios recientemente conectados utilizando la terminal previa psql
 - Escriba un comando para deshacer la ejecución de la consulta actual
 - Escriba un comando para terminar la sesión psql recién creada

Resumen

- En este módulo hemos aprendido:
 - Monitorización de la base de datos
 - Estadísticos base de datos
 - Recolector de estadísticos
 - Base de datos, tablas estadísticas
 - Sistema operativo y proceso de monitorización
 - Sesiones y bloqueos actuales
 - Registrar consultas de ejecución lenta
 - Uso del disco
 - Postgres Enterprise Manager
 - PEM - Características
 - PEM - Arquitectura
 - Lab



Módulo – 8

Lenguajes Procedurales

Objetivos

- En este módulo aprenderemos:
 - PostgreSQL Lenguajes Procedurales
 - Introducción a PL/PGSQL
 - Como funciona
 - PL/pgSQL Estructura bloque
 - Declaración de variables
 - Escribiendo sentencias ejecutables
 - Declarando funciones de parámetros
 - Estructuras de control
 - Manejo de excepciones
 - PL/pgSQL cursores
 - Triggers
 - Ejemplos y lab

PostgreSQL Lenguajes Procedurales

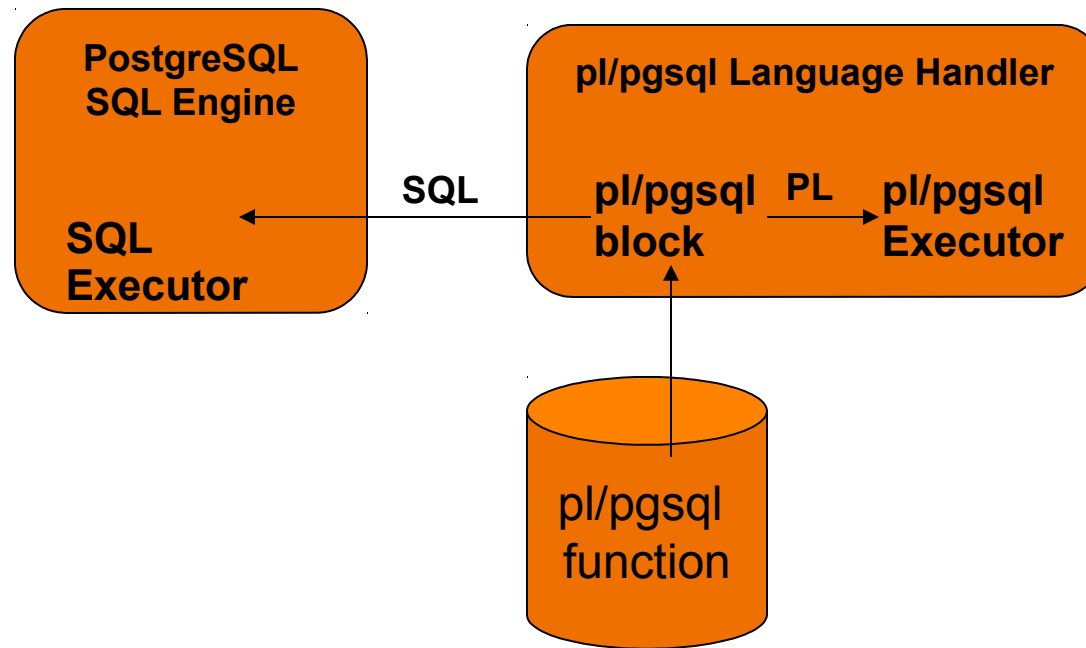
- PostgreSQL permite que las funciones definidas por el usuario sean escritas en una gran variedad de lenguajes procedurales
- PostgreSQL actualmente soporta varios tipos de lenguajes procedurales estándar
 - PL/pgSQL
 - PL/Tcl
 - PL/Perl
 - PL/Python
 - PL/Java
 - PL/Ruby
 - Otros lenguajes definidos por los usuarios

Introducción al PL/PGSQL

- PL/pgSQL es un lenguaje procedural para el sistema de base de datos PostgreSQL.
- Es el lenguaje procedural nativo de PostgreSQL es el PL/PgSQL.
- PL/pgSQL tiene muchas y diferentes características:
 - Puede ser utilizada para crear funciones y procesos de trigger,
 - Añade estructuras de control al lenguaje SQL ,
 - Puede realizar computaciones complejas,
 - Hereda todos los tipos de usuarios definidos, funciones y operadores,
 - Puede ser definida para ser de confianza para el servidor.

Como funciona

- PL/pgSQL define una estructura de bloque.
- Las funciones PL/pgSQL pueden ser compiladas y almacenadas en el servidor de la base de datos, ofreciendo un mejor rendimiento.



PL/pgSQL Estructura de bloques

- PL/pgSQL es un lenguaje de bloques estructurado.
- El texto completo de una definición de función tiene que ser un bloque
 - Declarar (opcional)
 - Variables, cursores, excepciones usuarios definidos
 - Comenzar (mandatorio)
 - SQL sentencias
 - PL/pgSQL sentencias
 - EXCEPCION (opcional)
 - Acciones a realizar cuando los errores ocurran
 - Fin; (mandatorio)
- Cada declaración y sentencia dentro de un bloque es terminada por un semicolon. Un bloque que aparece dentro de otro bloque tiene que tener un semicolon despues de END, como se muestra abajo, sin embargo, ese END final que concluye el cuerpo de la función no requiere un semicolon

PL/pgSQL Estructura

- Todas las palabras clave e identificadores pueden ser escritos de forma alterna utilizando mayúsculas y minúsculas
- Los identificadores son implícitamente convertidos a minúsculas, al menos que tengan comillas dobles
- Hay dos tipos de comentarios en PL/pgSQL.
 - -- Iniciar un comentario que se extiende hasta el final de la línea
 - /* comentarios multi-line */

Declaración de Variables

Variables

- Almacenamiento temporal de datos
- Pueden ser reutilizados en el bloque plpgsql
- Declarado e inicializado en la sección declarativa
- Utilizado y asignado nuevos valores en la sección ejecutables
- Pasan como parámetros a subprogramas PL / pgSQL
- Se utiliza para mantener la salida de un subprograma PL/pgSQL
- Pueden ser cualquier tipo de datos válidos SQL
- Pueden contener una cláusula DEFAULT o CONSTANT

Declaración de variables

- Ejemplos:

- `user_id integer;`
- `quantity numeric(5);`
- `url varchar;`
- `myrow tablename%ROWTYPE;`
- `myfield tablename.columnname%TYPE;`
- `arow RECORD;`
- `quantity integer DEFAULT 32;`

- La sintaxis general de una declaración de variable es

- `name [CONSTANT] type [NOT NULL] [{ DEFAULT | := } expression];`

Asignar valor a las variables

- Asignación

- `identifier := expression;`
- `user_id := 20;`
- `tax := subtotal * 0.06;`

- **SELECT INTO**

- `SELECT INTO target select_expressions FROM ...;`
- `SELECT INTO myrec * FROM emp WHERE empname = myname;`

Escribiendo sentencias ejecutables

- Las sentencias ejecutables son escritas dentro del bloque BEGIN .. END
- Los bloques se pueden anidar
- Las sentencias dentro de un bloque pueden continuar por muchas líneas
- Todas las sentencias deben terminar con ;

Declaración de los parámetros de la función

- Los parámetros pasados a funciones se nombran con los identificadores \$1 y \$2
- Se pueden utilizar alias de forma opcional
- Puede utilizarse alias o identificador numérico
- E.j
 - `CREATE FUNCTION sales_tax(subtotal real)`

Ejemplo 1

- La siguiente función suma dos numéricos que son pasados a la función llamada

```
create or replace function sum_val(i numeric, z numeric) returns numeric as $$  
begin  
return i+z;  
end;  
$$ language plpgsql;
```

- Invocar la función

```
Select sum_val(10,20);
```

Ejemplo 2

- La siguiente función mostrará la diferencia entre dos valores pasados:

```
create or replace function differ(x numeric, y numeric) returns void as $$  
declare  
  d numeric;  
begin  
  D := x - y;  
  raise notice 'Difference is: %',d;  
  return;  
end; $$ language plpgsql;
```

- Ejecutar la función

```
select differ(10,5);
```

Control de estructuras

- El control de estructuras cambia el flujo lógico de las sentencias con un bloque plpgsql
- Los tres principales tipos de estructuras de control se encuentran en plpgsql:
 - Sentencia IF
 - Expresiones case
 - Estructuras de control de bucle

Sentencia IF

- La estructura de sentencia IF de plpgsql es similar a otros lenguajes procedurales
- Permite a plpgsql realizar acciones de forma selectiva basándose en condiciones.

- Sintaxis:

```
IF boolean-expression THEN
statements
[ ELSIF boolean-expression THEN
statements]
[ ELSE
statements ]
END IF;
```

Ejemplo

- La siguiente función devolverá detalles de la cuenta de la tabla de clientes para un ID de cliente dado:

```
create or replace function dis_cus(c_id numeric) returns void as $$
declare
rec RECORD;
begin
select into rec * from customer where custid=c_id;
if found then
raise notice 'Customer Name: %', rec.custname;
raise notice 'Account No:      %', rec.accountid;
else
raise notice 'No Data';
end if;
return;
end;$$ language plpgsql;
```

- Ejecutar la función

```
Select dis_cus(130);
```

Expresión Case

- Una expresión CASE devuelve un resultado basado en una o más alternativas
- El valor de la selección determina que resultado se devuelve
- Sintaxis:

```
CASE search-expression  
WHEN expression [, expression [ ... ]] THEN  
statements  
[ WHEN expression [, expression [ ... ]] THEN  
statements  
... ]  
[ ELSE  
statements ]  
END CASE;
```

Ejemplo

- La siguiente función mostrará la cadena temperatura basada en la lectura del input temperatura utilizando buscar casos :

```
CREATE OR REPLACE FUNCTION dis_temp(tem numeric) RETURNS varchar AS $$ DECLARE
msg varchar;
BEGIN
CASE WHEN tem < 0 THEN
msg := 'ICY';
WHEN tem between 0 and 10 THEN
msg := 'COLD';
WHEN tem > 10 THEN
msg := 'NORMAL';
ELSE
msg := 'Cannot determine';
END CASE;
return msg;
END; $$ language plpgsql;
```

- Ejectuar la función:

```
select dis_temp(-10);
```

Sentencias LOOP

- Los bucles se utilizan principalmente para ejecutar sentencias varias veces hasta que se alcanza una condición de salida
- Hay tres tipos de bucles:
 - bucle básico
 - bucle WHILE
 - bucle FOR

Bucle básico

- Bucle simple que realiza acciones repetitivas sin condiciones generales
- Un bucle simple debe tener una EXIT

- sintaxis:

```
LOOP  
statements  
END LOOP;
```

- Sintaxis para EXIT

```
EXIT [ label ] [ WHEN boolean-expression ];
```

Ejemplo

- La siguiente función mostrará una serie de 0 a 10 utilizando un bucle básico:

```
create or replace function dis() returns void as $$  
declare  
  rec numeric;  
begin  
  rec:=0;  
  loop  
    raise notice '%',rec;  
    rec:=rec+1;  
    EXIT when rec>10;  
  end loop;  
  return;  
end; $$ language plpgsql;
```

- Ejecutar función:

```
Select dis();
```

Bucle WHILE

- Los bucles While realizan acciones iterativas basadas en una condición
- La condición se evalúa al inicio de cada iteración
- Si la condición genera un NULL el bucle se termina
- Sintaxis :

```
WHILE boolean-expression LOOP  
statements  
END LOOP;
```

Ejemplo

- La siguiente función mostrará una serie de 0-10 utilizando un bucle WHILE:

```
create or replace function dis() returns void as $$  
declare  
rec numeric;  
begin  
rec:=0;  
while rec<=10 loop  
raise notice '%',rec;  
rec:=rec+1;  
end loop;  
return;  
end; $$ language plpgsql;
```

- Ejecutar Función:

```
Select dis();
```

Bucle FOR

- Los Bucles FOR realizan funciones iterativas basadas en un recuento
- Los Bucles FOR tienen la misma estructura general que los Bucles Básicos
- Una sentencia de control se utiliza antes de la palabra clave LOOP para establecer el número de iteraciones

- Sintaxis:

```
FOR name IN [ REVERSE ] expression .. expression [ BY expression] LOOP  
statements  
END LOOP;
```

Ejemplo

- La siguiente función mostrará una secuencia numérica de 0 -10 con un incremento de 2

```
create or replace function dis1() returns void as
$$
declare
rec numeric;
begin
for rec in 0..10 by 2 loop
raise notice '%',rec;
end loop;
return;
end;$$ language plpgsql;
```

- Ejecutar función:

```
Select dis1();
```

Ejemplo

- En la siguiente función aparecerá una lista de nombres y números de cuenta para todos los clientes situados en la tabla clientes:

```
create or replace function dis_cus() returns void as
$$
declare
rec record;
begin
for rec in select*from customer loop
raise notice 'Customer Name: %', rec.custname;
raise notice 'Account No:      %', rec.accountid;
end loop;
return;
end;$$ language plpgsql;
```

- Ejecutar función:

```
Select dis_cus();
```

Capturando errores

- Los errores de sintaxis se manejan en tiempo de compilación
- El código plpgsql puede causar algunos errores inesperados en el momento de la ejecución.
- Para hacer frente a este tipo de errores plpgsql proporciona el Bloque de EXCEPTION
- Sintaxis:

```
[ DECLARE
declarations ]
BEGIN
statements
EXCEPTION
WHEN condition [ OR condition ... ] THEN
handler_statements
[ WHEN condition [ OR condition ... ] THEN
handler_statements
... ]
END;
```


Capturando errores

- La palabra clave EXCEPTION inicia la sección de manejo de excepciones
- Están permitidos varios controladores de excepciones
- Solo un controlador es procesado antes de abandonar el bloque
- WHEN OTHERS es la última cláusula y puede manejar todo tipo de excepciones.
- El bloque EXCEPTION debe estar al menos en un bloque BEGIN..END

Obteniendo información sobre un error

- Los controladores de excepciones necesitan identificar frecuentemente el error específico que ha ocurrido.
- Hay dos maneras de obtener información acerca de la excepción actual:
 - Variables especiales
 - GET STACKED DIAGNOSTICS comando.

Sintaxis: GET STACKED DIAGNOSTICS variable = item [, ...];

- Cada item es una palabra clave: RETURNED_SQLSTATE, MESSAGE_TEXT, PGPG_EXCEPTION_HINT, _EXCEPTION_DETAIL, PG_EXCEPTION_CONTEXT

Ejemplo

- La siguiente función dividirá el primer parámetro con el segundo y devolverá el resultado

```
create or replace function dis(a numeric, b numeric) returns numeric as $$
declare
result numeric;
begin
result=a/b;
EXCEPTION
When others then
Raise notice 'Wrong value for second parameter. Must be a non-zero value';
return result;
- end;$$ language plpgsql;
- Execute function:
Select dis();
```

Cursores PL/pgSQL

- Declarando cursores

- `DECLARE curs1 refcursor;`
- `DECLARE curs2 CURSOR FOR SELECT * FROM tenk1;`
- `DECLARE curs3 CURSOR (key integer) IS SELECT * FROM tenk1 WHERE unique1 = key;`

- Abriendo cursores

- `OPEN FOR query`
- `OPEN unbound_cursor FOR query;`
 - `OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;`
- `OPEN FOR EXECUTE`
- `OPEN unbound_cursor FOR EXECUTE query_string;`
 - `OPEN curs1 FOR EXECUTE 'SELECT * FROM ' || quote_ident($1);`

Cursores (cont) PL/pgSQL

- Abriendo un “Cursor Bound”

- `OPEN bound_cursor [(argument_values)];`
Opening Cursors
- `OPEN FOR query`
- DECLARE
 - `curs2 CURSOR FOR SELECT * FROM tenk1;`
 - `curs3 CURSOR (key integer) IS SELECT * FROM tenk1 WHERE unique1 = key;`
- `OPEN curs2;`
- `OPEN curs3(42);`

Cursores (cont) PL/pgSQL

- `FETCH`
 - `Cursor FETCH INTO target ;`
 - `FETCH` recupera la siguiente fila del cursor en un objetivo, que puede ser una variable de fila, una variable de registros o una lista sencilla de variables separadas por comas igual que `SELECT INTO`.
 - De la misma manera que con `SELECT INTO`, la variable especial `FOUND` se puede evaluar para ver si ha obtenido una fila o no
 - `FETCH curs1 INTO rowvar;`
 - `FETCH curs2 INTO foo, bar, baz;`
- `CLOSE`
 - `CLOSE` cierra el portal subyacente de un cursor abierto
 - Esto se puede usar para liberar recursos antes de que finalice la transacción, o para liberar la variable del cursor para que se abra de nuevo.
 - `CLOSE curs1;`

Ejemplo

- La siguiente función mostrará una lista de nombre y número de cuenta para todos los clientes en la “tabla clientes”
 - `create or replace function dis_cus() returns void as $$`
 - `declare`
 - `cur CURSOR for select*from customer;`
 - `rec RECORD;`
 - `begin`
 - `open cur;`
 - `fetch cur into rec;`
 - `while found loop`
 - `raise notice 'Customer Name: %', rec.custname;`
 - `raise notice 'Account No: %', rec.accountid;`
 - `fetch cur into rec;`
 - `end loop;`
 - `close cur;`
 - `return;`
 - `end; $$ language plpgsql;`
- Ejecutar función
 - `Select dis_cus();`

Triggers

- Creado con el comando CREATE FUNCTION
- Se crean varias variables especiales de forma automática en el nivel superior del bloque.
 - NEW
 - Tipo de dato RECORD; variable que tiene la nueva fila de base de datos para las operaciones INSERT/UPDATE de triggers a nivel de fila. Esta variable es NULL en triggers a nivel de sentencias.
 - OLD
 - Tipo de dato RECORD; variable que tiene la vieja fila de base de datos para las operaciones UPDATE/DELETE de triggers a nivel de fila. Esta variable es NULL en triggers a nivel de sentencias.
 - TG_NAME
 - Tipo de dato *name*; variable que contiene el nombre del trigger actual que ha saltado.

Triggers

- TG_WHEN
 - Tipo de dato *text*; un string de BEFORE o AFTER según la definición del trigger.
- TG_LEVEL
 - Tipo de dato *text*; un string de ROW o STATEMENT según la definición de trigger
- TG_OP
 - Tipo de dato *text*; un string de INSERT, UPDATE, o DELETE diciendo la operación para la cual ha saltado el TRIGGER.
- TG_RELNAME
 - Tipo de dato *name*; el nombre de la tabla que ha causado la invocación del trigger.
- TG_NARGS
 - Tipo de dato *integer*; el número de argumentos que se le da al trigger en la sentencia de CREATE TRIGGER.

Triggers

- Nota: Una función de trigger deberá devolver NULL o un registro/valor de fila que contenga la estructura exacta de la tabla para la que el trigger saltó.
- El valor de retorno de un trigger de nivel de sentencia de BEFORE o AFTER o un trigger de nivel de fila de AFTER se ignora siempre; bien podría ser null. Sin embargo, cualquiera de estos tipos de trigger puede abortar la operación completamente generando un error.

Ejemplo

- En este ejemplo la tabla “ciudad” almacena el total de la población de la ciudad y ésta crecerá en 1 cuando se realice n registro de nacimientos para dicha ciudad
- Crear tablas base:
 - `create table city(cityid numeric, population numeric);`
 - `create table birth(regid numeric, name varchar, cityid numeric);`
 - `insert into city values(1,0),(2,0);`
 - `select*from city;`
- Crear una función Trigger que será llamada por un trigger:
`create or replace function trg_ins() returns trigger as $$`
`begin`
`update city set population=population+1 where cityid=NEW.cityid;`
`return null;`
`end; $$ language plpgsql;`
- Crear Trigger en la tabla nacimientos:
`create trigger trg_ins_brth after insert on birth`
`for each row execute procedure trg_ins();`
- Probar si el trigger está funcionando:
`insert into birth values(101,'Raj',2);`
`select*from birth;`
`select*from city;`

Ejercicio Lab- 1

- Crear una función plpgsql
 - Declarar dos variables en la sección declarar
 - Asignar el valor 1 en la primera variable
 - Asignar el valor 2 en la segunda variable
 - Mostrar la diferencia entre numéricos en la pantalla utilizando mensajes de noticia

Resumen

- En este módulo hemos aprendido:
 - PostgreSQL lenguajes procedurales
 - Introducción a PL/PGSQL
 - Como funciona
 - PL/pgSQL Estructura de Bloques
 - Estableciendo las variables
 - Escritura de sentencias ejecutables
 - Estableciendo las funciones del parámetro
 - Estructuras de Control
 - Manejo de excepciones
 - PL/pgSQL Cursores
 - Triggers
 - Ejemplos y lab



Módulo – 9

Utilidades adicionales – Extensión

Objetivos

- Este módulo cubrirá las siguientes materias:
 - Qué son los módulos de extensión?
 - Construyendo módulos de extensión
 - Instalando módulos de extensión
 - Índice de módulos
 - Otros recursos

Qué son los módulos de extensión?

- Los módulos en el directorio extensión son características adicionales de PostgreSQL
- Generalmente, los módulos de extensión no están incluidos en el core de la base de datos, ya que, o bien no son muy demandados, o porque se encuentran en fase de desarrollo.
- Los módulos de extensión son , en ocasiones, aparcados en la distribución del core .
- Los módulos en extensión están ligados a esta versión particular de PostgreSQLThe modules in Extension are tied to that particular version of PostgreSQL, y los módulos que están disponibles cambian con el tiempo.

Instalando módulos de extensión

- Con el fin de utilizar un módulo de extensión en la base de datos, necesita ejecutar el comando `CREATE EXTENSION` para instalar las características del módulo en dicha base de datos.
- Por defecto, las filas de extensión son instaladas en `PREFIX/share/extension`, pero la mayoría de la administración de los paquetes de sistema cambian esto a `share/PostgreSQL/extension` o alguna variante.
- La documentación para los módulos de extensión aparecerá instalada en: `at share/Extension` o parecidas.

Instalando los módulos de extensión

- Necesita registrar los nuevos elementos u objetos en el sistema de la base de datos ejecutando el comando “create extension”
- Alternativamente, ejecutarlo en el template 1 de la base de datos de manera que el módulo quedará copiado en bases de datos creadas posteriormente por defecto.
- Sintaxis:

```
CREATE EXTENSION module_name;
```

-Este comando debe ser ejecutado por el superuser de la base de datos

Índice de módulos

- Adminpack - Rutinas de manipulación de log y ficheros, utilizado por pgAdmin
- btree_gist - Soporte para emular un BTREE indexando en GiST
- Chkpass - Un tipo de datos de contraseña auto-cifrada
- Cube -Tipo de datos de cubo multidimensional (ejemplo de GiST indexing)
- Dblink - Habilita la ejecución remota de consultas
- Earthdistance -Operador para computar la distancia terrestre entre dos puntos
- Fuzzystrmatch - Búsqueda difusa de cadenas tipo Levenshtein, metaphone y soundex
- Hstore - módulo para guardar pares (clave, valor)

Índice de módulos (cont'd)

- Intagg – Agregador de enteros
- Intarray – Índice de apoyo para las matrices de int4 , que utiliza GiST
- Isn - PostgreSQL extensiones tipo para los números de producto ISBN, ISSN, ISMN, EAN13
- Lo – Mantenimiento objeto grande
- Ltree – Añade datos del tipo ltree para el almacenamiento de estructuras tree-like
- pg_buffercache – Consultas en tiempo real para el buffer cache compartido
- pg_freespacemap – Pantallas de contenido en el espacio libre del mapa(FSM)

Índice de módulos (cont'd)

- pg_trgm – Funciona para determinar la similitud de texto basada en trigramas coincidentes .
- Pgcrypto – Funcionalidades criptográficas
- Pgrowlocks – Función para devolver información del bloqueo de filas
- Pgstattuple – Una función para devolver estadísticas sobre filas muertas y espacios libres en la tabla
- Seg – Intervalo de confianza (tipo dato) datatype (GiST indexing example)
- Spi – Diferentes funciones de trigger, ejemplos para el uso de SPI
- Sslinfo – Funciones para obtener información sobre el certificado SSL
- Tsearch2 – Dota de buscador compatible para búsqueda de textos atrasados.
- Uuid-ossdp – Provee funciones para generar UUIDs

Otros recursos

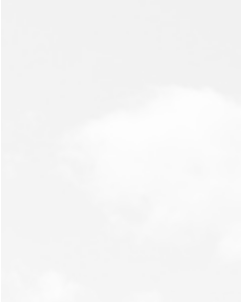
- Encontramos otros recursos para las extensiones de PostgreSQL
 - pgFoundry (<http://pgfoundry.org>)
 - Sourceforge (<http://sourceforge.org>)
- Errores de código PostgreSQL
 - <http://www.postgresql.org/docs/9.3/static/errcodes-appendix.html>

Resumen

- En este módulo hemos aprendido:
 - Qué son los módulos de extensión?
 - Construcción de módulos de extensión
 - Instalación de módulos de extensión
 - Índice de módulos
 - Otros recursos

Ejercicio lab - 1

- El equipo de base de datos quiere examinar que está pasando en el cache del buffer compartido a tiempo real. Podrás encontrar una extensión interesante de PostgreSQL llamada "pg_buffercache". Realizar la instalación y prueba la funcionalidad de este módulo de extensión.



THANK YOU

Words included in the word cloud:

- merci
- grazie
- spasiba
- kam ouen
- gratizias
- tak
- manana
- mahalo
- hvala
- cheers
- toda
- gracias
- grassie
- thank you
- danki
- kitos
- welalin
- mahalo
- danki
- thanks
- takk
- domo arrigato
- gratitude
- danke
- kitos
- dziekuje
- takk
- miigwetch
- talofa
- modupe
- mes
- dankon
- na gode
- thanks
- merci