PERCONA

PostgreSQL Vs. MySQL

Cuál es la diferencia?

Tengo dos amigos...

¿Quién soy?

Me llamo Pep Pla.

Soy un Consultor en Percona.

Tengo más de 30 años de experiencia. Lo que me hace sentir bastante viejo.

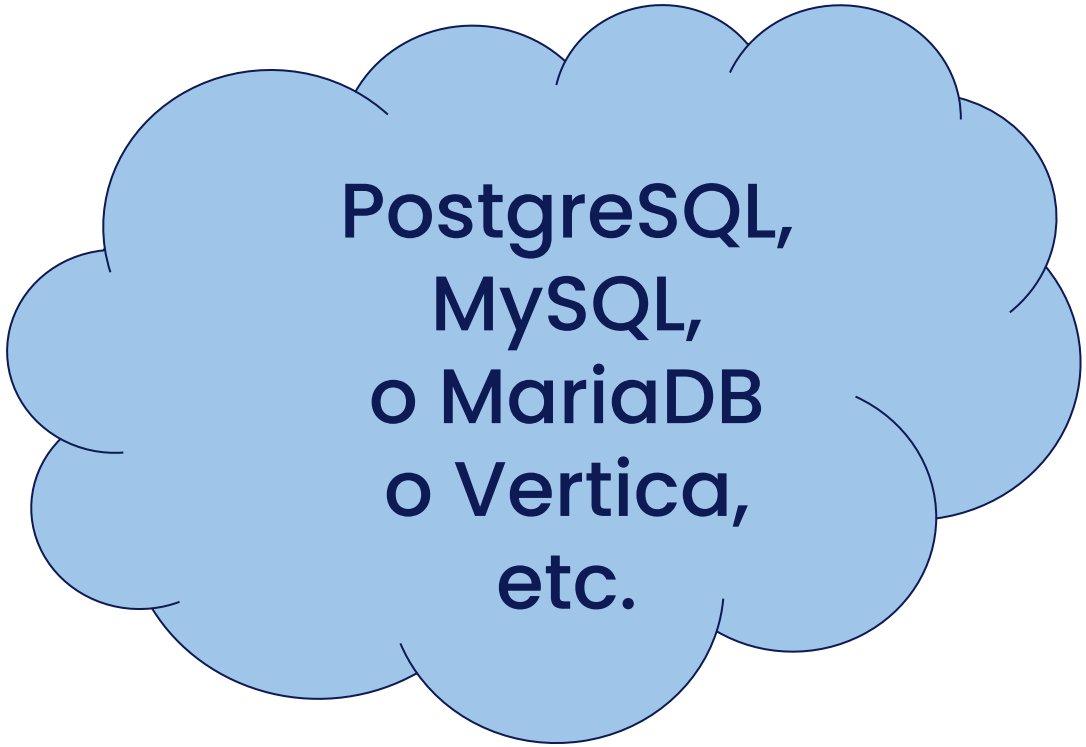
Actualmente vivo en Barcelona.

Adivinanza !!!!!!

Veo, veo! ¿Qué database es?

- Es una base de datos opensource muy popular.
- Creada por un tipo llamado Michael.
- Michael tiene un historial de decir cosas que quizás no debería.
- Michael ha creado varias empresas.

Y la respuesta es...

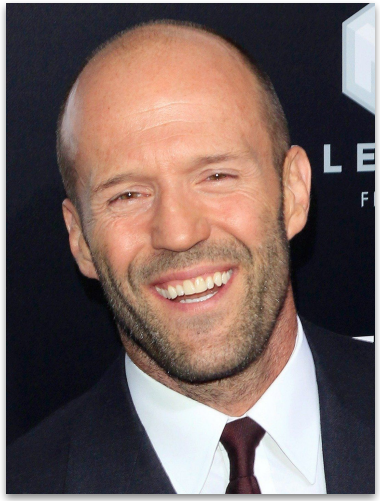


¿Cómo comparar base de datos?

Volviendo a mis dos amigos...

El propósito importa

Ferrari Roma
Potencia: 620 CV
Precio: $247,310



El propósito importa

John Deere 9R 640
Potencia: 640 HP
Precio: $732,231

Vamos a lo fácil: comparar funcionalidades!

- Prepara una lista de requerimientos.
- Compara los candidatos para ver cuál encaja mejor.
- No asumas que tienen las mismas funcionalidades.
  - MySQL no tiene secuencias.
  - MySQL no tiene vistas materializadas.
  - PostgreSQL no está tan avanzado en replicación.
  - PostgreSQL no tiene clustering nativo.
  - ...

Cualquiera puede hacerlo.

No es el tema de esta presentación.

Tampoco hablaremos de licencias.

El destornillador de Birmingham

Esto es un destornillador de Birmingham

La Ley del Instrumento

“El fenómeno conocido como Martillo de Maslow, efecto Einstellung o ley del instrumento, implica un sesgo cognitivo que se traduce en las personas en una tendencia excesiva a utilizar una herramienta familiar para resolver un problema desconocido.”

"Si la única herramienta que tienes es un martillo, entonces todos los problemas parecen un clavo."

Prejuicios de evaluación

- La mayoría de las comparaciones son tendenciosas.
- Pocas personas con conocimientos equivalentes de ambas bases de datos.
- Lo cual no es siempre negativo. A menudo es algo a tener en cuenta.
  - Si Dios te da limones, haz limonada.
  - Si Dios te da DBA’s, usa las bases de datos que conocen.

Bueno, al fin y al cabo, una base de datos relacional es igual que cualquier otra, no?

A pesar de que PostgreSQL y MySQL son los dos RDBMS, hay algunas diferencias significativas en funcionalidades, rendimiento y operación.

Hay ventajas y desventajas, que afectan a ambas, y que pueden afectar seriamente a tu negocio.

PostgreSQL contra MySQL

Las dos:

- Sistemas de Bases de Datos Relacionales.
- Open Source.
- Populares.
- Tienen edad suficiente para entrar en las discotecas.
  - (por lo que muchos ya no las consideran “cool”)

PostgreSQL contra MySQL: diferencias

PostgreSQL:
- Mejor soporte del estándar SQL.
- Gobernado por listas de correo y consenso.
- Comunidad Activa.

MySQL:
- ‘Más fácil’
- Gobernado (?) por Oracle
- Comunidad Activa.

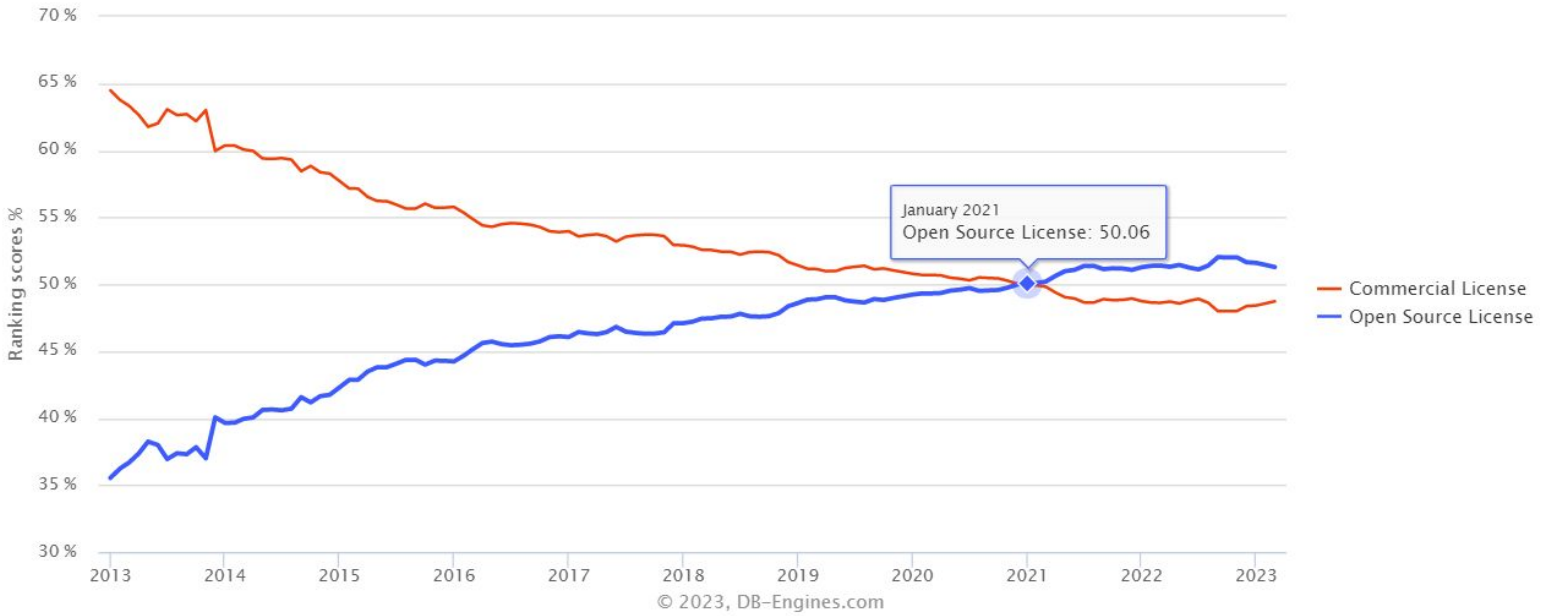
'El diablo está en los detalles'

Ludwig Mies Van Der Rohe.

¿Dónde va Vicente?

https://db-engines.com/en/ranking_osvsc

Popularity trend



The above chart shows the historical trend of the popularity of open source and commercial database management systems.

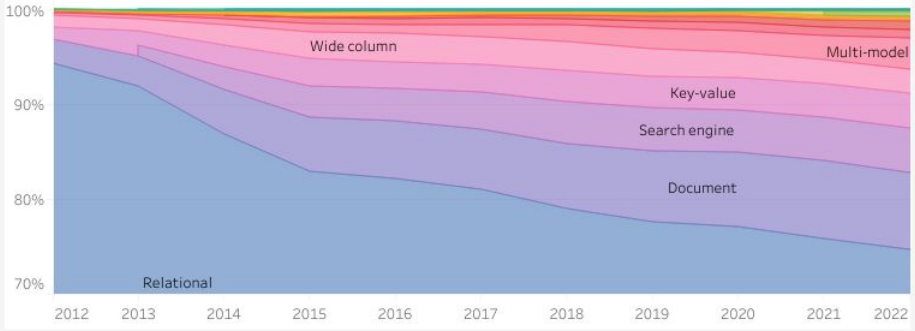
DB-Engines Trends

ARCHITECTURE & PERFORMANCE

DB-ENGINES

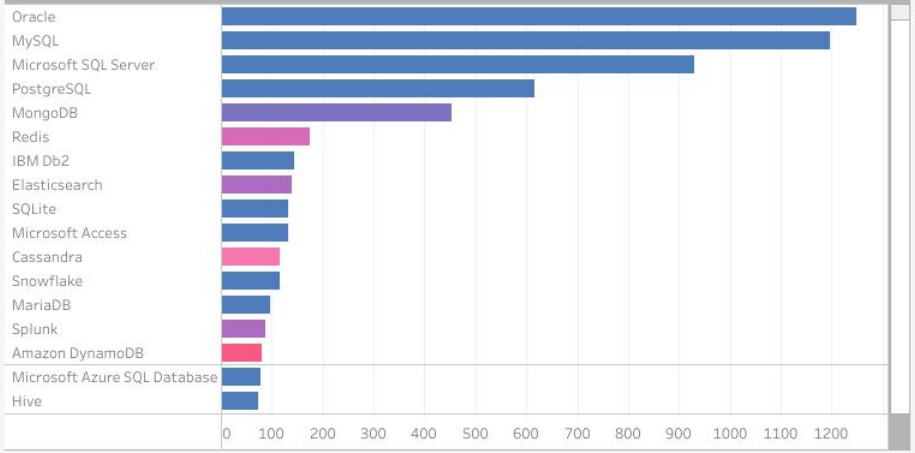
Relational | Document | Search engine | Key-value | Wide column | Multi-model | Graph | Time Series | Multivalue | Null | Object orient.. | Spatial DBMS | Content | RDF | Navigational | Native XML | Search | Event

Score Evolution Per Type (warning : Y-Axis root at 70%)



Database Engine Score for February 2023

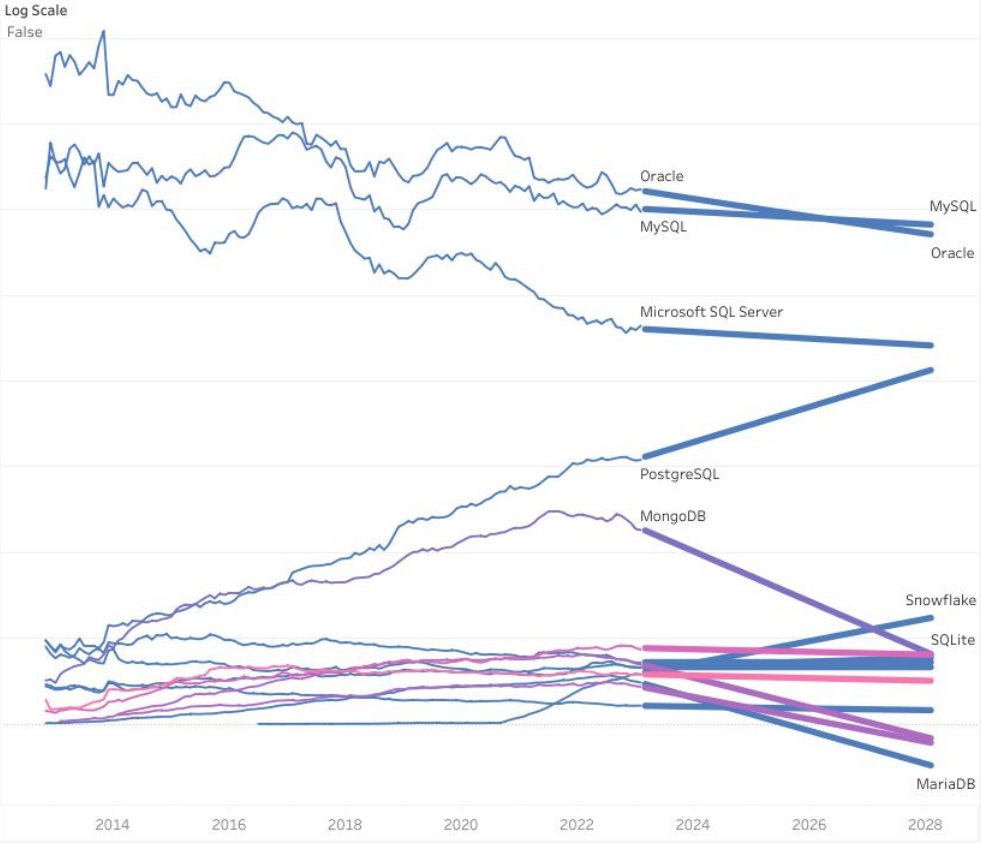
February 2023



Evolution Per Database and Forecast (Top 15 Db for All DBTypes)

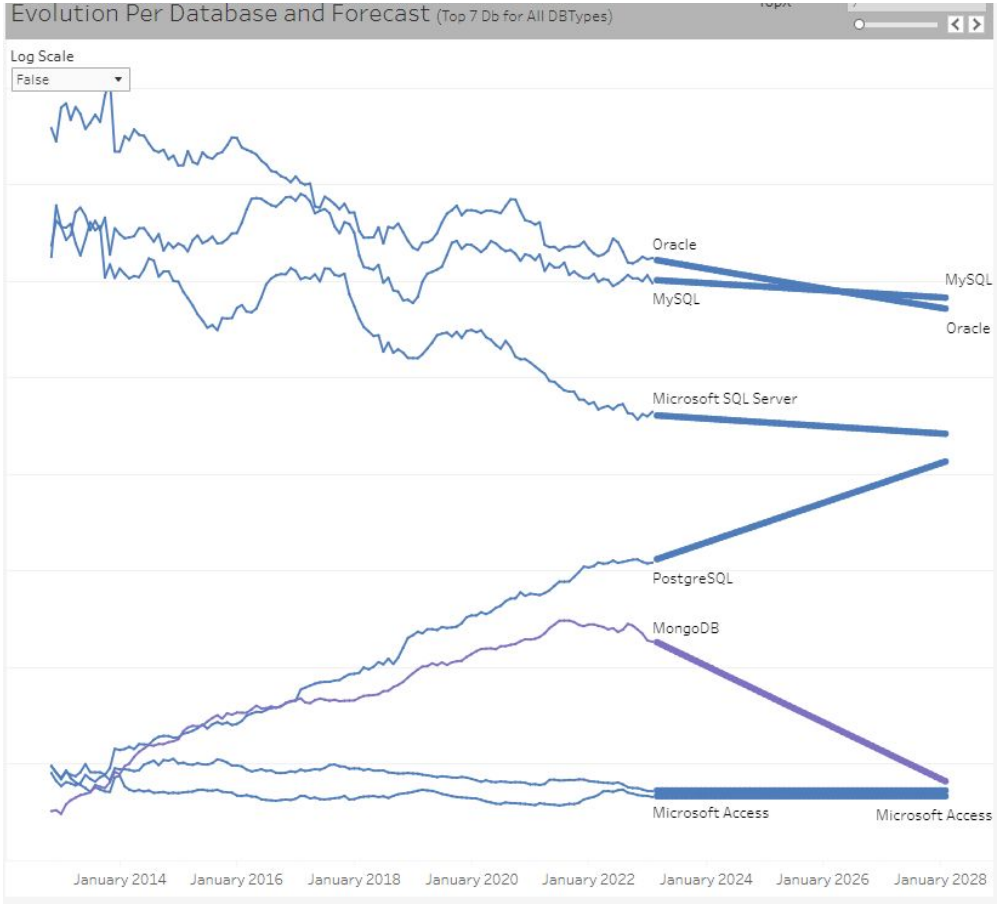
TopX
15

Log Scale
False



Tendencias proyectadas

Caramba!



PERCONA

Vamos a lío

Arquitectura

Hay un montón de diferencias entre PostgreSQL y MySQL.

Si tuviese que elegir una:

MySQL tiene una arquitectura de dos niveles, con un nivel sQL que procesa la consulta y un nivel de *engine* que procesa las operaciones de entrada/salida. Esto permite el uso *transparente* de múltiples motores con funcionalidades distintas.

En esta presentación hablaré del motor por omisión en MySQL8: InnoDB.

Ya está aquí el DBA de MySQL alabando MySQL!

No, no lo estoy haciendo. Diferente no significa mejor.

Es bueno saber que puedes utilizar diferentes motores en MySQL sin cambiar la aplicación.

Pero… necesitas esto?

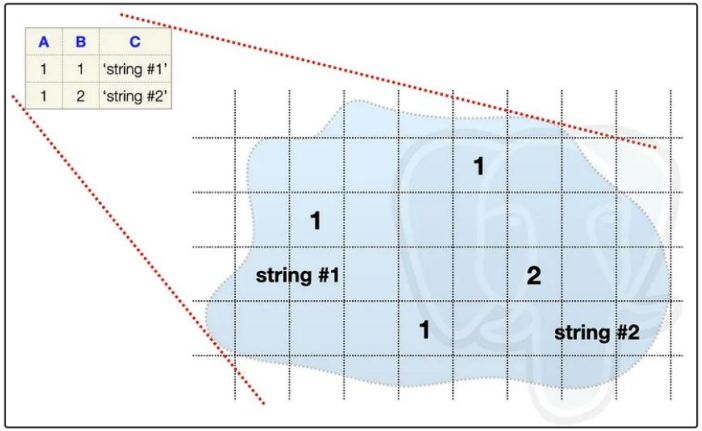
Seguramente no (como el 99.999% de los usuarios/developers/dba's)

Pero necesitas entender la arquitectura en niveles para comprender algunos conceptos de MySQL.

PostgreSQL's Heap

The Heap - https://medium.com/quadcode-life/structure-of-heap-table-in-postgresql-d44c94332052

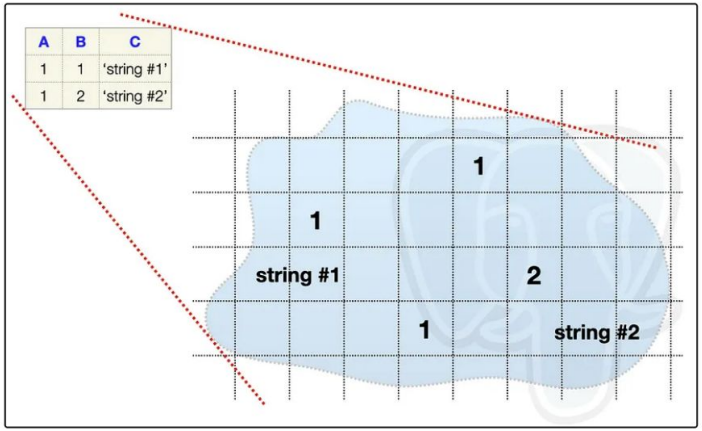
- Las tuplas (filas) se almacenan en un heap según el identificador de página e identificador de ubicación (ctid).
- Los datos no están ordenados (usar order by)
- Las filas actualizadas/borradas permanecen en el heap hasta que se hace vacuum (más después)
- Metadatos:
  - Xmin: transacción que insertó la fila
  - Xmax: transacción que borró la fila (o no)
  - t_ctid: el propio o tid de la fila que reemplaza
- El ide de transacción es un integer de 32 bits que puede repetirse en ciclo, liándola parda.



PERCONA

The Heap

- **PROS**
  - Los inserts son rápidos.
  - Los nuevos datos están juntos.
  - Más fácil de gestionar.
  - Las filas se localizan por ubicación(CTID).
- **CONTRAS**
  - Fragmentación.
  - Las filas se localizan por ubicación (actualizaciones y migraciones de fila).
  - Requiere reconstrucciones periódicas.
  - Espacio libre para actualizaciones.

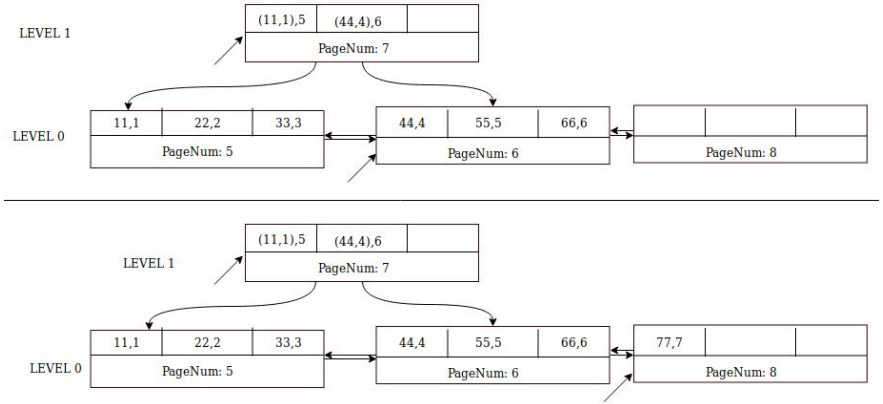


PERCONA

InnoDB Clustered Index

InnoDB Clustered Index

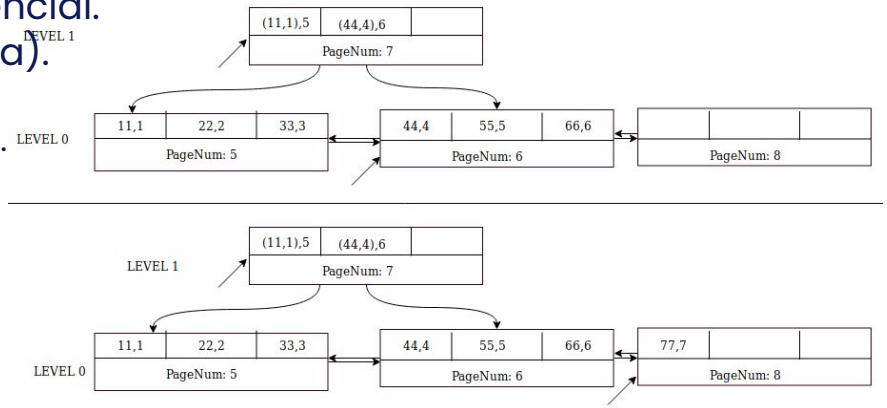
- Todas las tablas se almacenan en un índice.
- Necesita una clave primaria.
- InnoDB genera una PK en tu lugar si no hay una explícita o implícita. Esta será inutil en tus consultas (y en replicación).
- Las actualizaciones se graban a largo plazo y cualquier cambio se asume como definitivo.
- Metadatos:
  - Trx_id: Identificador de transacción que modificó la fila la última vez.
  - Roll_ptr: apuntador a versiones anteriores en el segmento de rollback.



PERCONA

InnoDB Clustered Index

- PROS
  - Los datos se almacenan según la clave primaria (ordenados).
  - No hay fragmentación.
  - Las filas se localizan por PK.
  - Muy eficiente en inserción secuencial.
- CONS
  - Muy ineficiente en inserción no secuencial.
  - Clave primaria necesaria (o generada).
  - Las filas se localizan por PK.
    - Índices secundarios: doble busqueda.
    - PK grandes: índices más grandes.



PERCONA

Índices secundarios

Índices secundarios (no clave primaria)

- **PostgreSQL**
  - Múltiples tipos de índices distintos.
  - Reorganización de la tabla -> Reorganización en el índice.
  - Migración/movimientos de fils -> Migración/movimiento en el índice.
  - Índices más pequeños.
- **InnoDB**
  - Sólo BTREE.
  - Índices más grandes: clave primaria en cada fila.
  - La reconstrucción de una tabla no afecta a los índices.
  - La migración de filas no afecta al los índices secundarios.
  - Sólo una versión de la fila (marcas de borrado).
  - El Change buffer retrasa la escritura en índices secundarios.

MVCC

Multiversion Concurrency Control

- **Acceso concurrente por múltiples transacciones.**
  - Lectura
  - Escritura
- **Aislamiento de los datos.**
  - Cada transacción tiene que ver una imagen consistente.
- **Sin bloqueos.**
  - Las escrituras no bloquean las lecturas.
  - Las escrituras bloquean otras escrituras mientras no se han confirmado(commit).

MVCC - PostgreSQL

- Las filas sólo se insertan o borran. Una actualización es un borrado y una inserción.
- Las filas borradas se conservan en la tabla junto a las filas actuales.
- CTID diferente para cada versión: las actualizaciones afecta a los índices incluso si la columna modificada no pertenece al índice (Las actualizaciones HOT tratan de mitigar este efecto).
- Xmin y Xmax:
  - Xmin: id de la transacción de la inserción.
  - Xmax: id de la transacción del borrado (o actualización).
    - Incluso si la transacción no se ha completado (rollback).
- Commit log (clog o xact)
  - Log de todas las transacciones con el id de transacción.
  - Estado:
    - In progress, committed, aborted o subcommitted.

MVCC - PostgreSQL

- **Como se si una fila tiene el valor que necesito?**
  - Si xmin es más grande que mi transaction ID no debo ver esa fila.
  - Si xmin es más pequeño que mi transaction ID y xmax está vacío, puedo ver la fila.
  - Si xmin más pequeño que mi transaction ID y xmax también, estoy viendo una versión antigua.
    - Verificar si xmax pertenece a una transacción en estado committed.
    - Verificar t_ctid para ver si hay una nueva versión válida.
  - Si xmin más pequeño que mi transaction ID y xmax es más grande, entonces estoy viendo una versión borrada (pero la que necesito)
- **En algunos casos tengo que validar el transaction log.**
- **Quién borra las versiones antiguas?**
  - Vacuum

InnoDB MVCC

- Basado en Rollback segments.
- Las versiones antiguas se almacenan en el rollback segment, se asume que la transacción va a completarse (commit).
- Metadatos de cada fila
  - Trx_id: identificador de transacción.
  - Roll_ptr: apunta a la versión de la fila anterior en el rollback segment.
- En el segmento de rollback hay varias versiones de una fila “encadenadas”:
  - Trx_id: identificador de transacción.
  - Roll_ptr: apunta a la versión de la fila anterior en el rollback segment.
- Transacciones de larga duración pueden necesitar un gran número de accesos al rollback segment.
  - Repeatable read es el nivel de aislamiento por omisión.
- History list:
  - Lista de transacciones activas y entradas en el rollback segment.

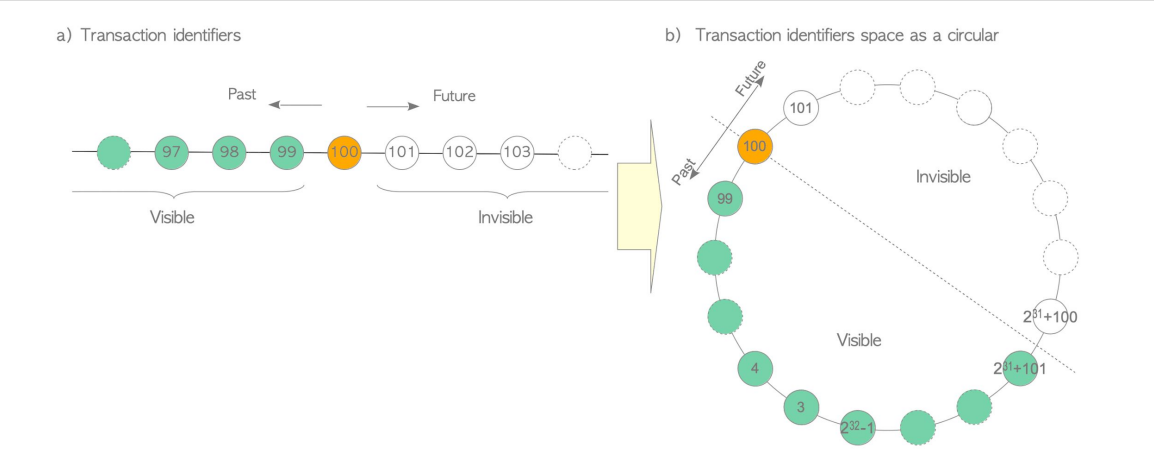
InnoDB MVCC

- **Cómo sé si una fila tiene el valor que necesito?**
  - Trx_id es menor que mi id y la transacción no está activa: Estoy viendo la versión correcta.
  - Trx_id es menor que mi id y la transacción está activa:
    - Ir al rollback segment (roll_ptr) a buscar la versión antigua.
    - Repetir el proceso hasta que la encuentres.
  - Trx_id es mayor que mi id y no hay roll_ptr: No hay versión antigua para mi, la fila no existe.
  - Trx_id es mayor que mi id y hay roll_ptr:
    - Ir al rollback segment (roll_ptr) a buscar la versión antigua.
    - Repetir el proceso hasta que la encuentres.
- **Hay un proceso en background que limpia las entradas en rollback antiguas.**

Vacuum

PostgreSQL Vacuum

- **Hay un trx_id especial que congela una fila.**
  Una fila puede ser congelada si fue modificada por una transacción que acabo con commit y no hay ninguna transacción previa activa.
  Vacuum le asigna el identificador de transacción especial que la congela.
- **Esto es necesario porque los transaction id son circulares:**



PERCONA

PostgreSQL Vacuum

- **Sin no congelamos filas, las podríamos acabar viendo como si hubiesen sucedido en el futuro:**
  - Después de 2100 millones de transacciones ejecutadas.
  - Si no congelamos, el servidor se para cuando se detecta un “transaction wraparound!.”
- **Vacuum también elimina las filas borradas.**
  - Auto-vacuum o vacuum no full, no devuelve el espacio al OS.
  - Full vacuum lo hace, pero necesita un bloqueo exclusivo de la tabla.
- **Las operaciones Vacuum requieren mucha io.**

Visibility Map

Vacuum requiere un mapa de visibilidad para cada tabla para hacer seguimiento de las páginas que sólo contienen tuplas congeladas, es decir visibles para todas las transacciones (y todas las transacciones futuras hasta que la página sea modificada de nuevo).

Esto tiene dos propósitos:

- Vacuum puede evitar estas páginas en siguientes ejecuciones ya que no hay nada que limpiar.
- Además, permite a PostgreSQL responder algunas consultas utilizando únicamente los índices, sin hacer referencia a la tabla subyacente.

Dado que los índices en PostgreSQL no contienen información sobre la visibilidad de las tuplas, un scan normal recupera la tupla para cada entrada coincidente en el índice, para verificar si debería ser visible para la transacción actual.

Un **index-only scan, por otra parte, accede al visibility map primero.** Si todas las filas son visibles, el acceso a la tabla puede ser evitado. Esto es, sobretodo, útil para conjuntos de datos grandes, ya que el visibility map es mucho menor que la tabla, por lo que puede permanecer en cache incluso si la tabla es grande.

PostgreSQL specifics

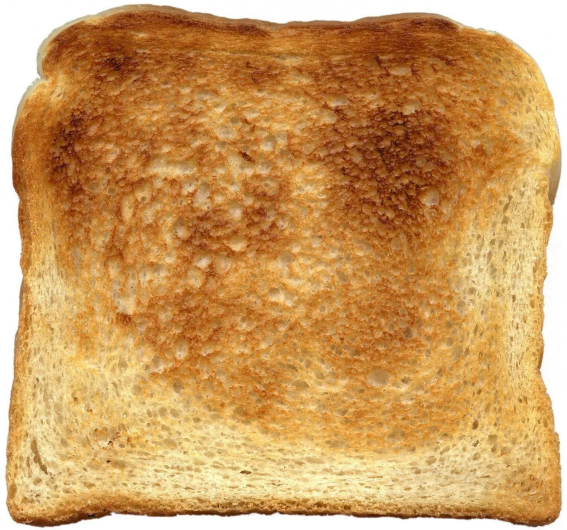
PostgreSQL tiene algunas cosas interesantes que MySQL no tiene

- Vistas materializadas
- MERGE() – permite mover determinados tipos de proceso de datos al servidor, reduciendo el tráfico de datos en la red.
- Dos tipos de datos JSON
- Varios tipos de índices.
  - Indexado parcial
- Mejor soporte del estándar SQL
  - Implementación de Window Functions más completa
  - Secuencias
    - Parecido al AUTO_INCREMENT de MySQL.
    - Excelente para generar datos de test.
- Hay multiples proyectos derivados
  - FerretDB - Protocolo MongoDB
- Más complicada de configurar y correr
  - Los upgrades pueden tener su miga.

TOAST

The Oversized-Attribute Storage Technique – similar a lo que hace InnoDB.

Almacenar datos grandes fuera del espacio de tabla.



Roles de Usuario

Sí, MySQL tiene roles pero no son tan populares.

PostgreSQL Basics: Roles y Privilegios

https://www.red-gate.com/simple-talk/databases/postgresql/postgresql-basics-roles-and-privileges/

PostgreSQL Basics: Propiedad de Objetos y Permisos por omisión

https://www.red-gate.com/simple-talk/uncategorized/postgresql-basics-object-ownership-and-default-privileges/

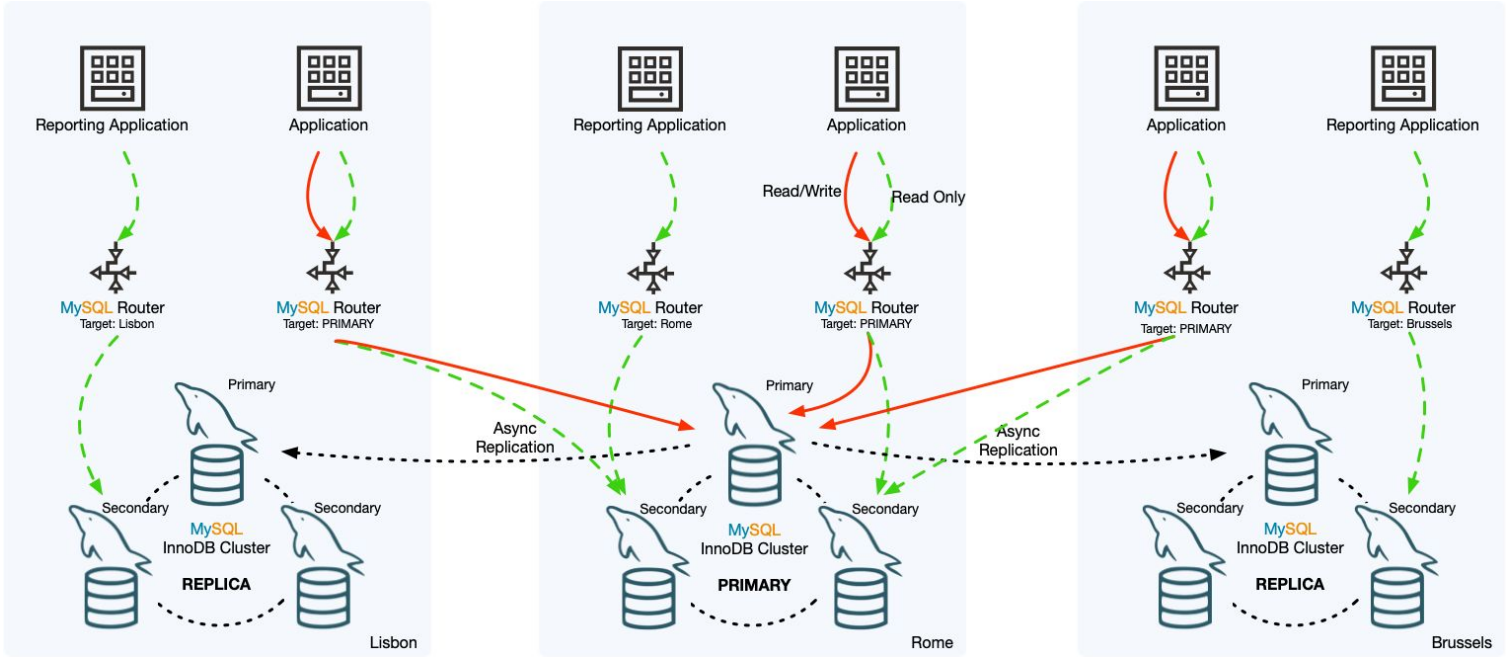
Vistas materializadas, Watch, múltiples tipos de índices y FILTER

```
SELECT
  fa.actor_id,
  SUM(length) FILTER (WHERE rating = 'R'),
  SUM(length) FILTER (WHERE rating = 'PG')
FROM film_actor AS fa
LEFT JOIN film AS f
  ON f.film_id = fa.film_id
GROUP BY fa.actor_id
```

Alta disponibilidad

Replicación y clustering

PostgreSQL no tiene un equivalente opensource a InnoDB Cluster o incluso Galera



PERCONA

Nos quedamos sin tiempo!

Procesos contra Hilos

- MySQL usa hilos
- PostgreSQL usa procesos

- What is better?
  - A quién quieres más? A tu papá o a tu mama?
  - Hay razones para preferir cualquiera de los dos:
    - Seguramente esto indica que no hay gran diferencia.

Uso de la memória

- MySQL gestiona la memória.
- PostgreSQL depende más del sistema operativo.

- Qué es mejor?
  - Depende de si se trata de un servidor dedicado o no.
    - El Sistema Operativo puede bolcar las cache y dar más memória a otras aplicaciones.
    - Chicos MySQL: Cuidado con el OOM killer.

“Son diferentes”

diferente != mejor

Thank You!

pep.pla@percona.com
@peppla