PERCONA

PostgreSQL Versus MySQL

I have two friends...

Who am I

My name is Pep Pla.

I’m a Consultant at Percona.

More than 30 years of experience. That makes me feel quite old.

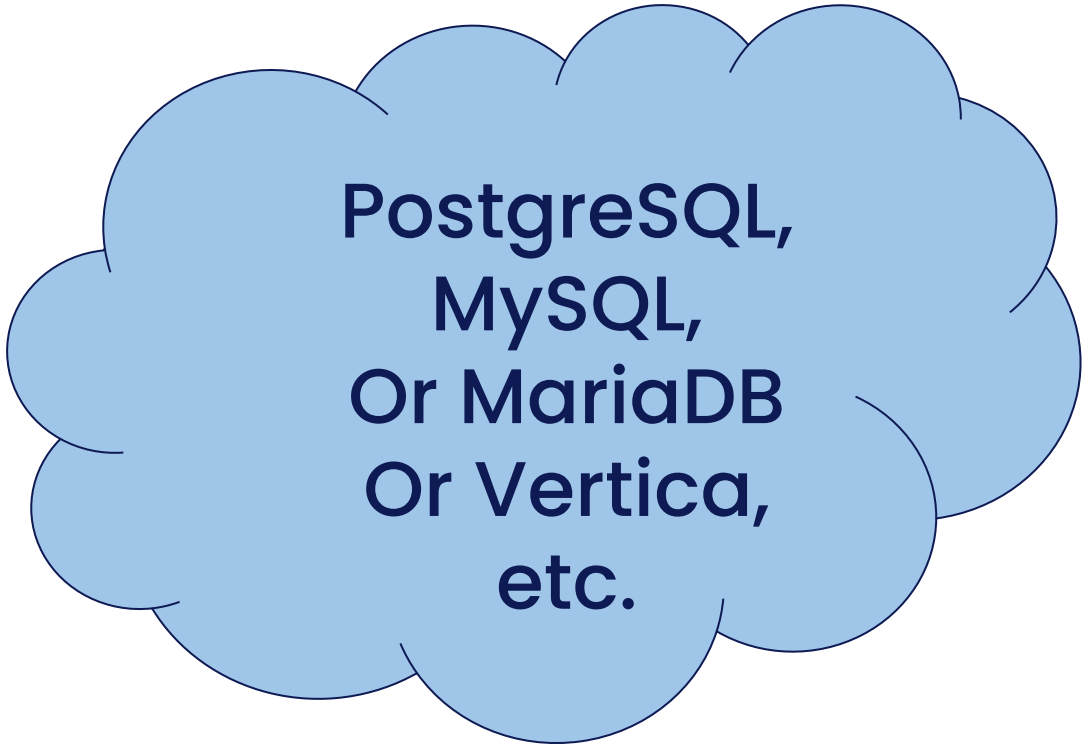
Currently located in Barcelona.

QUIZ TIME!!!!!!

Name that database!

- It is a Popular open-source database.
- Originated by a guy with the first name Michael.
- Michael has a history of saying some outrageous things.
- Michael has formed several companies.

The answer is

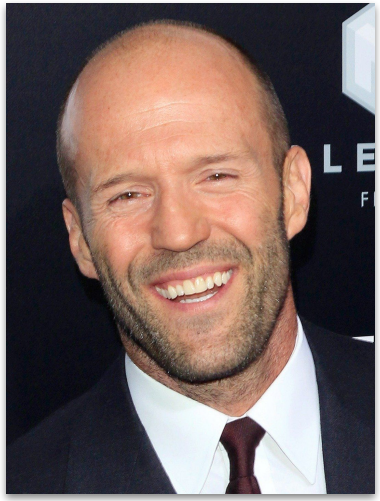


How to compare databases?

Back to my two friends...

Purpose matters

Ferrari Roma
Engine: 620 HP
List price: $247,310



Purpose matters

John Deere 9R 640
Engine: 640 HP
List price: $732,231

This is easy: check features!

- Build a list of your requirements.
- Compare candidates to see which one fits best.
- Don’t assume they have the same features:
  - MySQL lacks sequences
  - MySQL lacks materialized views
  - PostgreSQL is behind in replication
  - PostgreSQL lacks native clustering
  - …

You could do that.

This is not what this presentation is about.

Also, it is not about licensing.

The Birmingham Screwdriver

This is a Birmingham Screwdriver

The Law of the Instrument

“The law of the instrument is a cognitive bias that involves an over-reliance on a familiar tool.”

"If the only tool you have is a hammer, it is tempting to treat everything as if it were a nail."

Evaluation Bias

- Most comparisons out there are biased.
- Few people with comparable levels of knowledge of both databases.
- This is not always bad. Sometimes it is one of the things to consider.
  - If God gives you lemons, make lemonade.
  - If God gives you DBA's, use the databases they know.

- Disclaimer: I'm a mostly a MySQL DBA but I try to be as *agnostic* as possible.

One relational database is pretty much the same as another, right?

While both PostgreSQL and MySQL are both RDMSes, there are some rather dramatic differences in features, performance, and operation.

There are advantages, and disadvantaged, to both which could severely impact how you do business.

PostgreSQL versus MySQL

Both:

- Relational Database Management Systems
- Open Source
- Popular
- Old enough to be allowed to drink.
  - (therefore seen as ‘not cool’ by some)

PostgreSQL versus MySQL differences

PostgreSQL:
- Better SQL Standard Support
- Governed by mailing list, consensus
- Active community

MySQL:
- ‘Easier’
- Governed (?) by Oracle
- Active community

'The devil is in the details'

Ludwig Mies Van Der Rohe.

Where does Vincent go?*

* Spanish equivalent of “Monkey see, monkey do”

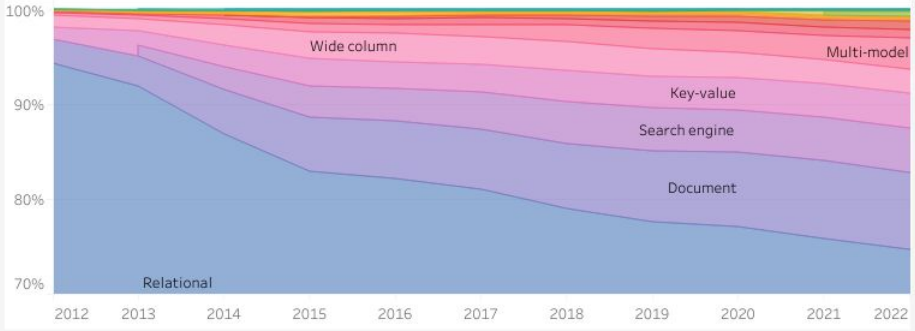
DB-Engines Trends

ARCHITECTURE & PERFORMANCE

DB-ENGINES

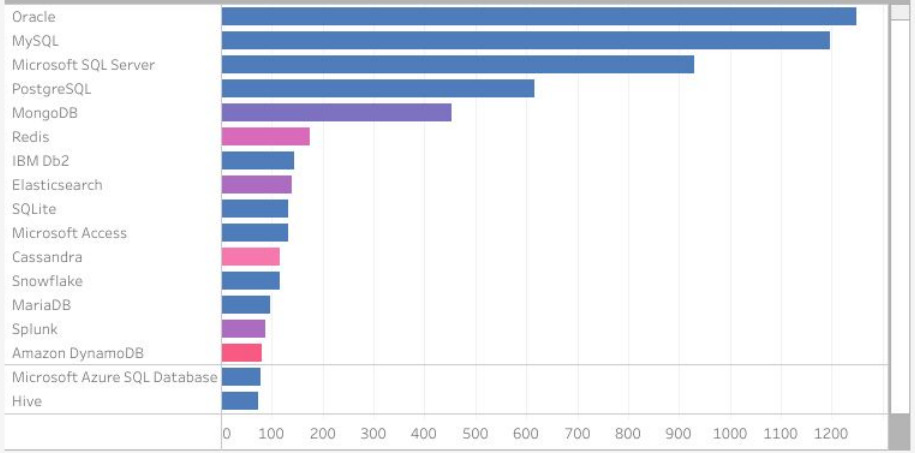
Relational | Document | Search engine | Key-value | Wide column | Multi-model | Graph | Time Series | Multivalue | Null | Object orient.. | Spatial DBMS | Content | RDF | Navigational | Native XML | Search | Event

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



Database Engine Score for February 2023

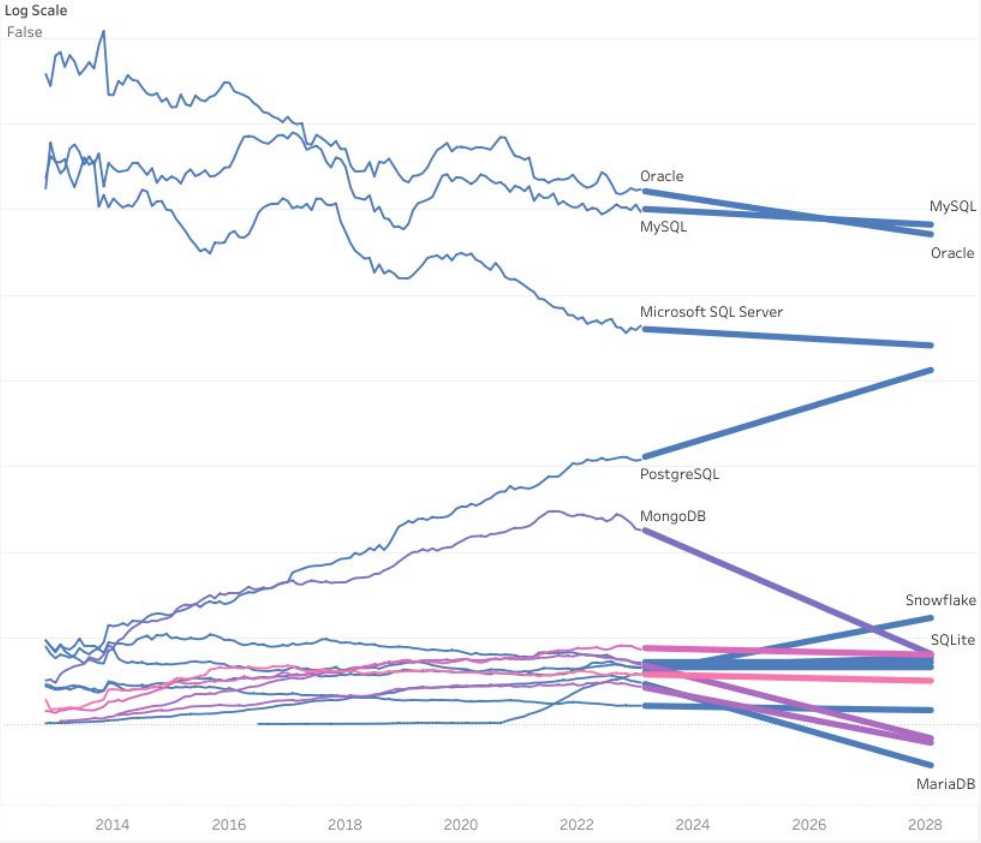
February 2023



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

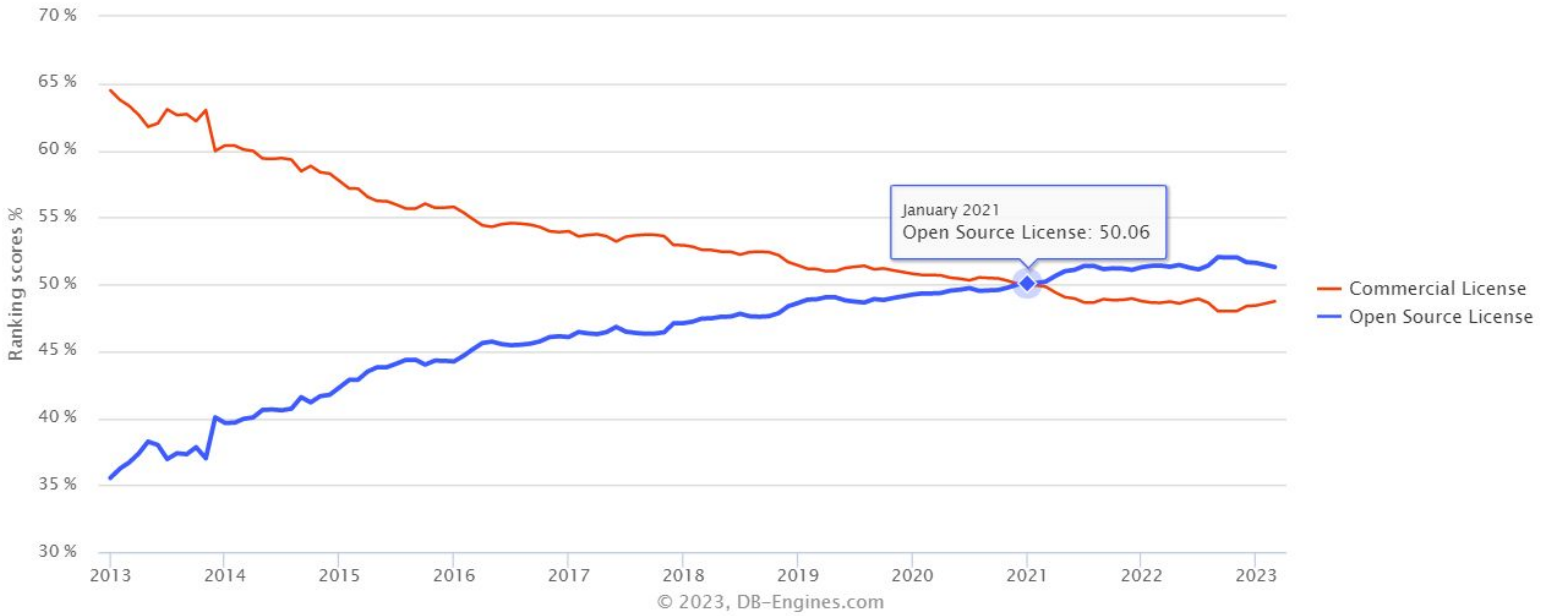
TopX
15

Log Scale
False



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

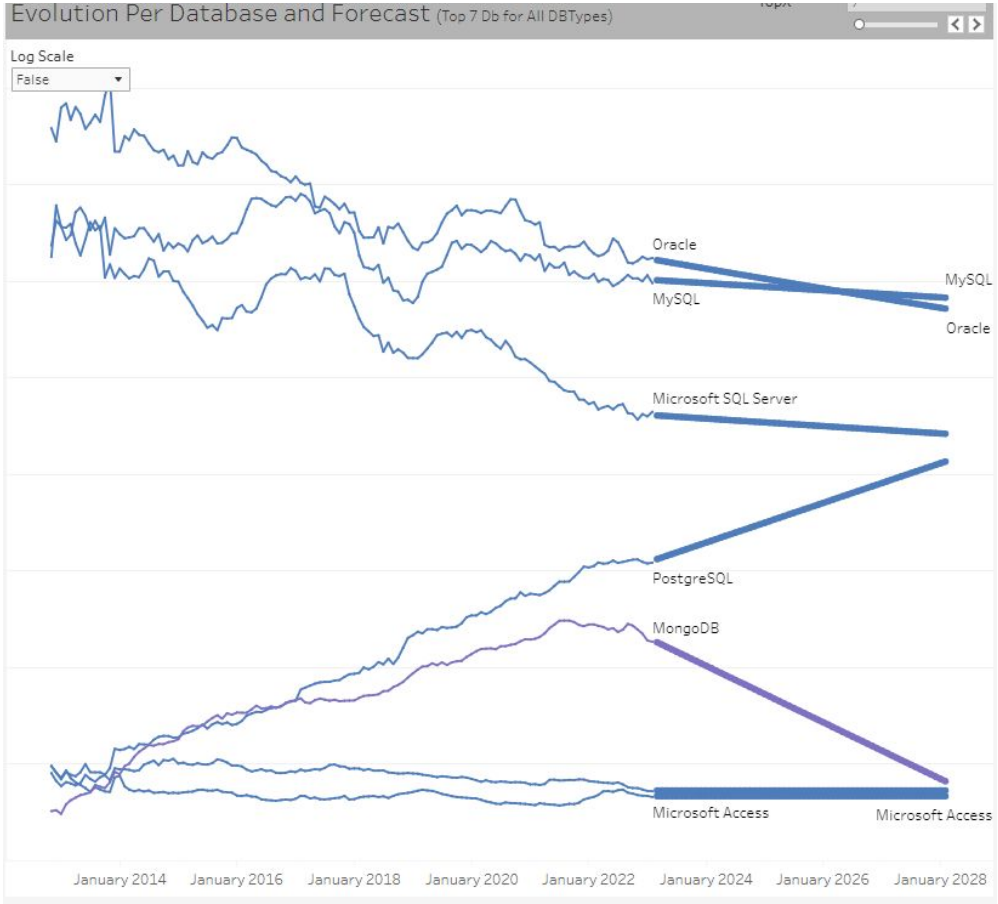
Popularity trend



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

Projected Trends

Yikes!



PERCONA

Let’s go to the meat

Architecture

There are plenty of differences between PostgreSQL and MySQL.

If I have to choose one:

MySQL has a two layer architecture with a SQL layer that processes the query and an engine layer that processes the read/write operations. This allows the *transparent* use of multiple engines with different features.

In this presentation we will be talking about the default engine in MySQL 8: InnoDB

Here is the MySQL DBA praising MySQL!

No, I'm not. Different doesn't mean better.

It is good to know that you can use different engines in MySQL without changing the application.

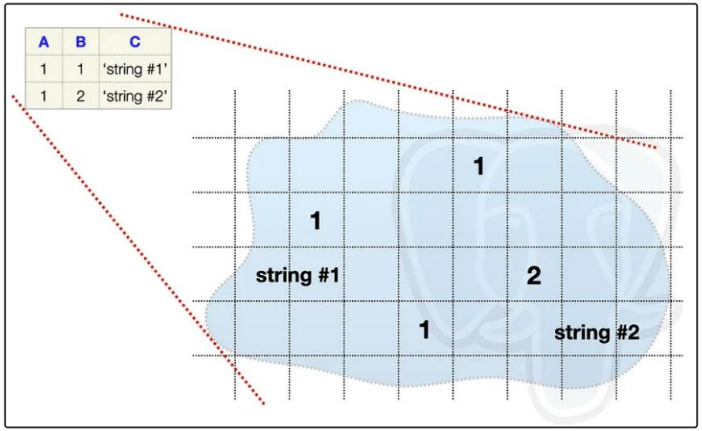
But... do you need that? Probably not (as 99.999% of database users/developers/dba's)

Need to know about the layer based architecture to understand some MySQL concepts.

PostgreSQL's Heap

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

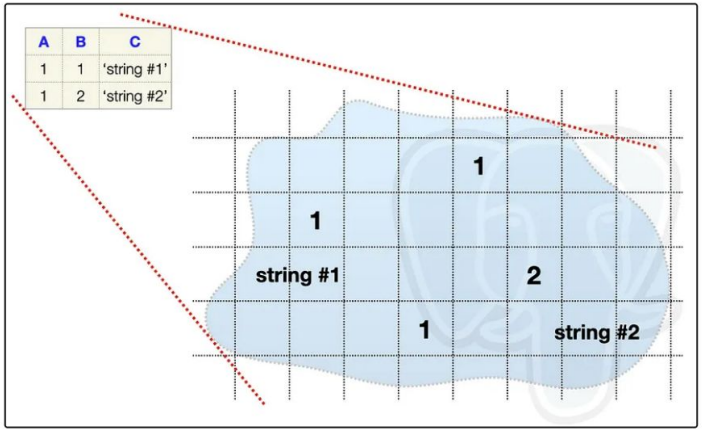
- Tuples (rows) are stored in a heap by page identifier and location identifier (ctid)
- Data is unordered (use order by)
- Updated/replaced rows are kept in the heap until vacuumed (more later)
- Metadata
  - Xmin: transaction that inserted the row
  - Xmax: transaction that removed the row (or not)
  - t_ctid: self or tid of replacing tuple
- Transaction id is a 32 bit INTEGER that can wrap around and make a mess.



PERCONA

The Heap

- PROS
  - Inserts are fast.
  - New data is placed together.
  - Simpler to manage.
  - Rows found by location (CTID).
- CONS
  - Fragmentation.
  - Rows found by location (updates and row migrations).
  - Requires periodic rebuilding.
  - Free space for updates.

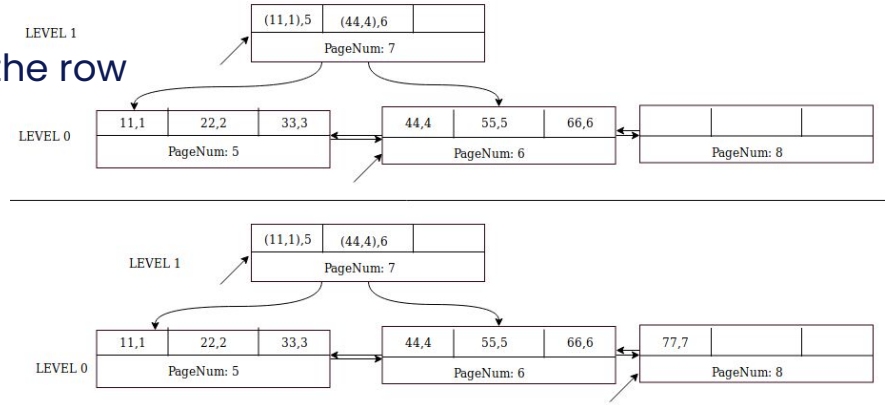


PERCONA

InnoDB Clustered Index

InnoDB Clustered Index

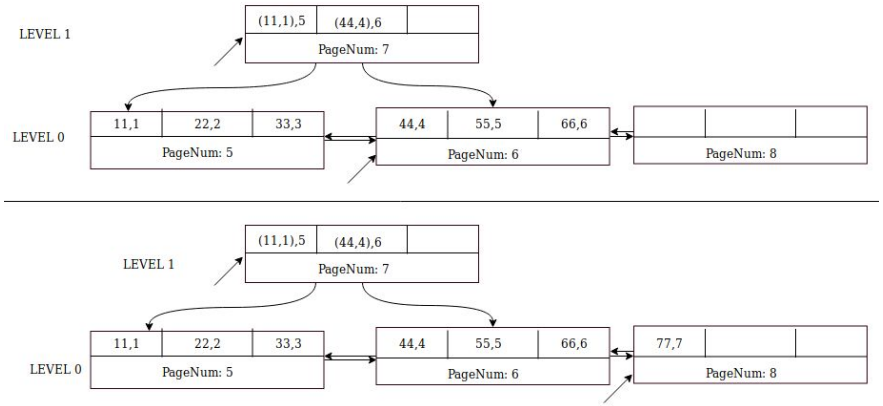
- Every table is stored as an index.
- Requires a primary key.
- InnoDB will pick a PK for you if you do not designate one and it will be useless for your queries (and replication).
- Updated rows are written to long and current version enthroned.
- Metadata:
  - Trx_id: transaction ID that last wrote the row
  - Roll_ptr: pointer to old entries in the rollback segment.



PERCONA

InnoDB Clustered Index

- PROS
  - Data is stored by Primary Key (sorted).
  - No fragmentation.
  - Rows found by PK.
  - Great at sequential insert.
- CONS
  - Very bad with non sequential inserts.
  - Primary key needed (or generated).
  - Rows found by PK.
    - Secondary indexes: double search.
    - Large PK: larger indexes.



PERCONA

Secondary Indexes

Secondary indexes (Non primary key indexes)

- PostgreSQL
  - Multiple different types of indexes.
  - Table rebuild -> Index rebuild.
  - Row migration/movement -> index migration/movement.
  - Smaller indexes.
- InnoDB
  - Just BTREE.
  - Larger indexes: primary key of each row.
  - Table rebuild does not affect indexes.
  - Row migration does not affect secondary.
  - Just one version of the row (delete mark).
  - Change buffer delays writes to secondary indexes.

MVCC

Multiversion Concurrency Control

- **Concurrent access by multiple transactions.**
  - Read
  - Write
- **Isolation of data.**
  - Each transaction has to see a consistent image.
- **Lock-less.**
  - Writes do not lock reads.
  - Writes lock writes while not committed.

MVCC - PostgreSQL

- Rows are inserted or deleted. An update is a delete + insert.
- Deleted rows are stored in the table with current rows.
- CTID is different for each version: updates affect indexes even if the modified column does not belong to the index (HOT updates mitigate this)
- Xmin and Xmax:
  - Xmin: transaction id of inserting transaction
  - Xmax: transaction id of removing (or updating) transaction
    - Even if transaction has been rolled back.
- Commit log (clog or xact)
  - Log of all transactions with transaction id
  - State:
    - In progress, committed, aborted or subcommitted.

MVCC - PostgreSQL

- **How do I know if I row has the value I need?**
  - If xmin is bigger than my transaction ID I should not see that row.
  - If xmin is smaller than my transaction ID and xmax is empty, I can see the row.
  - If xmin is smaller than my transaction ID and xmax is also, I'm seeing an old version.
    - Check if xmax was committed.
    - Check ctid to see if there is a newer version valid.
  - If xmin is smaller than my transaction ID and xmax is larger, then I'm seeing a removed row (but the version I need)
- **In some cases I have to check the transaction log.**
- **Who removes old versions?**
  - Vacuum

InnoDB MVCC

- Rollback segment based.
- Old versions are stored in the rollback segment, commit is assumed.
- Row metadata
  - Trx_id: transaction identified.
  - Roll_ptr: pointer to rollback segment previous version of row.
- Multiple versions of a row are chained in the rollback segment:
  - Trx_id: transaction identified.
  - Roll_ptr: pointer to rollback segment previous version of row.
- Long running transactions can required a lot of rollback segment accesses.
  - Repeatable read is the default isolation level.
- History list:
  - List of active transactions and rollback entries.

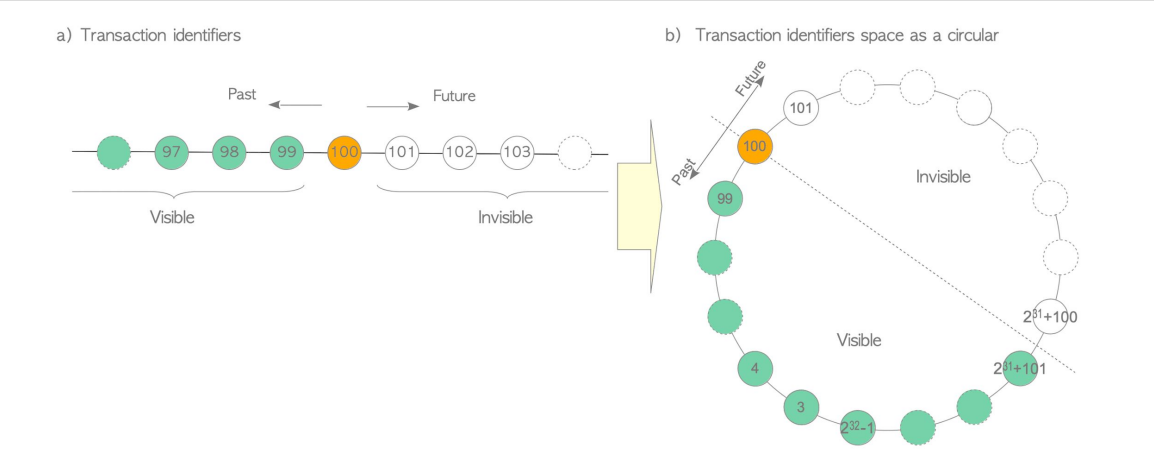
InnoDB MVCC

- **How do I know if I row has the value I need?**
  - Trx_id is smaller than my id and transaction is not active: I'm seeing the good version.
  - Trx_id is smaller than my id and transaction is active:
    - Go to the rollback segment (roll_ptr) to find the old version.
    - Repeat until you find it.
  - Trx_id is greater than my id and no roll_ptr: there is no old version.
  - Trx_id is greater than my id and there is a roll_ptr:
    - Go to the rollback segment to find the old version.
    - Repeat until you find it.
- **There is a background process that purges old rollback segment entries.**

Vacuum

PostgreSQL Vacuum

- **There is a special trx_id that freezes a row.**
  - A row can be frozen if it was modified by a transaction that committed, and there are no previous transactions active.
  - Vacuum assigns the special frozen transaction id.
- **We need that because transaction ids are circular:**



PERCONA

PostgreSQL Vacuum

- **If we do not freeze rows, we could see them as being in the future:**
  - After 2.1 billion transaction have been executed.
  - If we do not freeze, the server stops: transaction wraparound!
- **Vacuum also removes deleted rows.**
  - Auto-vacuum or non full vacuum does not return that space the the OS.
  - Full vacuum does, but requires an exclusive table lock.
- **Vacuum operations are io intensive.**

PostgreSQL specifics

PostgreSQL has some interesting stuff not in MySQL

- Materialized Views
- MERGE() – process transactions logs, like from cash registers, as a batch rather than multiple round trips between application and database
- TWO JSON data types
- Many different types of indexes
  - Ability to index only parts of an index
  - Can ‘roll your own’
- Better SQL standard compliance
  - More complete Window Functions
  - Sequences
    - Similar to MySQL AUTO_INCREMENT
    - Great for building test data
- Basis for many projects
  - FerretDB - MongoDB protocol
- Harder to setup and run
  - Upgrades can be tricky

Visibility Map

Vacuum maintains a visibility map for each table to keep track of which pages contain only tuples that are known to be visible to all active transactions (and all future transactions, until the page is again modified).

This has two purposes.

vacuum itself can skip such pages on the next run, since there is nothing to clean up.

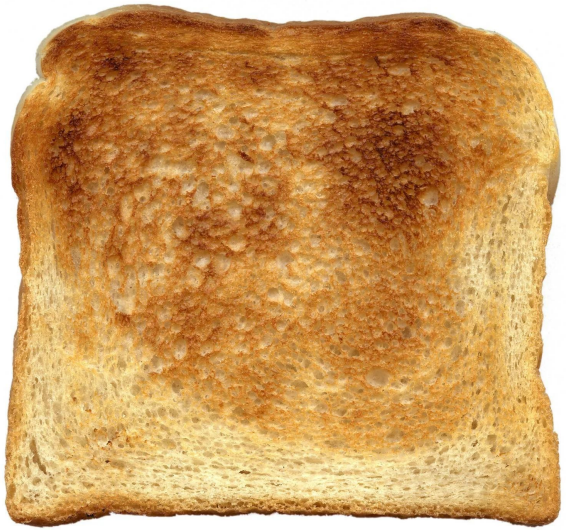
Second, it allows PostgreSQL to answer some queries using only the index, without reference to the underlying table.

Since PostgreSQL indexes don't contain tuple visibility information, a normal index scan fetches the heap tuple for each matching index entry, to check whether it should be seen by the current transaction. **An index-only scan, on the other hand, checks the visibility map first.** If it's known that all tuples on the page are visible, the heap fetch can be skipped. This is most useful on large data sets where the visibility map can prevent disk accesses.

The visibility map is vastly smaller than the heap, so it can easily be cached even when the heap is very large.

TOAST

The Oversized-Attribute Storage Technique – similar to what InnoDB does



Use Roles

Yes, MySQL has roles but they are not that popular.

PostgreSQL Basics: Roles and Privileges

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

PostgreSQL Basics: Object Ownership and Default Privileges

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

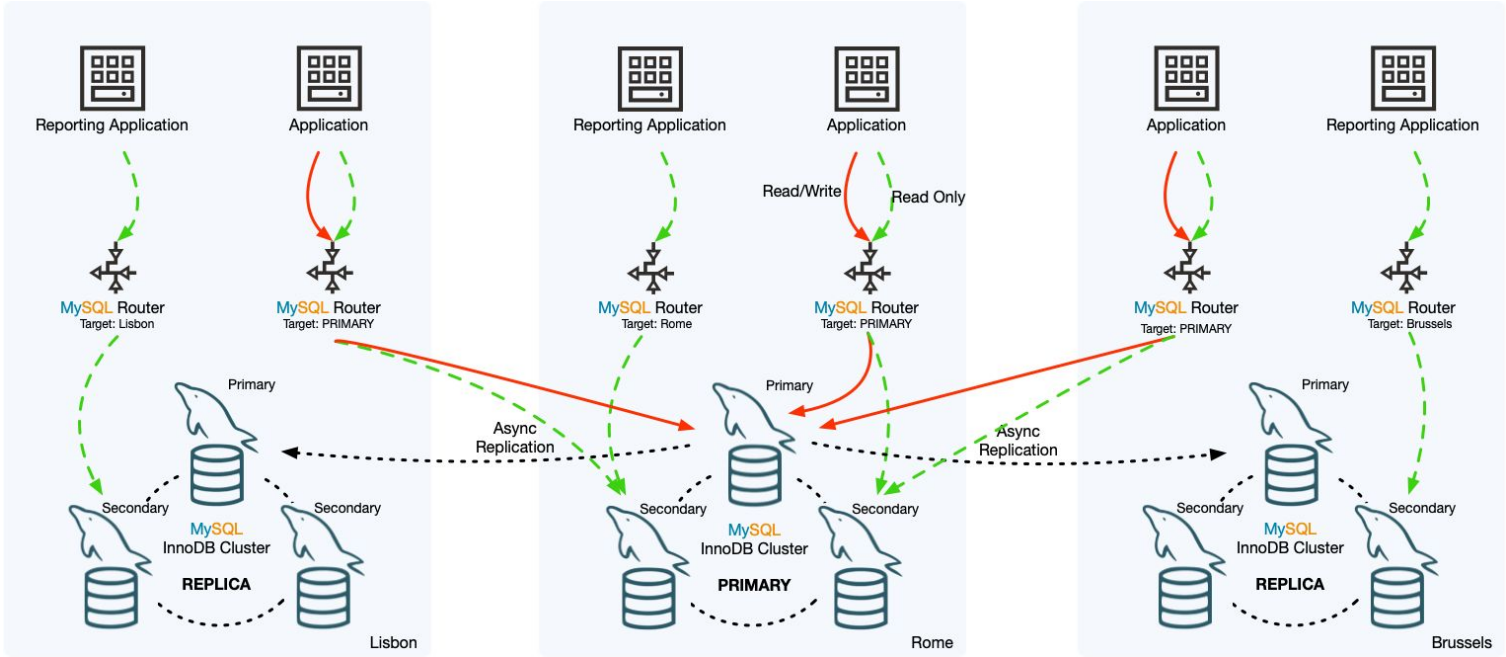
Materialized Views, Watch, Many Types of Indexes, & FILTER

```
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
```

High Availability

Replication

PostgreSQL has no open source equivalent to InnoDB Cluster or even Galera



PERCONA

We run out of time!

Processes vs. Threads

- MySQL uses threads
- PostgreSQL uses processes

- What is better?
  - Who do you love more, mom or dad?
  - There are good reasons for liking each:
    - Probably this means that it makes no difference.

Memory usage

- MySQL manages memory
- PostgreSQL relies more on the Operating System

- What is better?
  - I have an opinion. But I promised to be agnostic.
  - Point for MySQL… managing its memory makes it more reliable and stable.
  - It depends if it is a dedicated server or not.
    - The OS can flush caches and give them to other applications.
    - MySQL guys: Beware of OOM killer.

“It is different”

Different != Better

Thank You!

pep.pla@percona.com
@peppla