10 Reasons why you should prefer PostgreSQL to MySQL

Anand Chitipothu

Who am I?

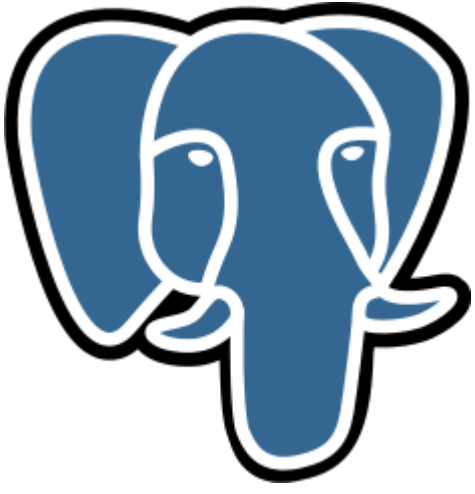
Anand Chitipothu

Independent Software Consultant & Trainer

Ex. Internet Archivist

MySQL or PostgreSQL?

The world's most **popular** open source database



The world's most **advanced** open source database

Quick Comparison

	MySQL MyISAM	**MySQL InnoDB**	**PostgreSQL**
Transactions	No	Yes	Yes
Foreign Key Constraints	No	Yes	Yes
Locking	Table	Row/MVCC	Row/MVCC

Who Uses MySQL?

- Facebook
- FriendFeed
- Quora
- Flickr
- Second Life

Who Uses PostgreSQL

- Skype
- Heroku
- Instagram
- Disqus
- Yahoo!
- NASA

MySQL ate my cake

MySQL ate my cake

```
mysql> CREATE TABLE cake (name VARCHAR(3));
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO cake (name) VALUES
('pancake');
Query OK, 1 row affected, 1 warning (0.03 sec)
```

MySQL ate my cake

```
mysql> SELECT * FROM cake;
+------+
| name |
+------+
| pan  |
+------+
1 row in set (0.03 sec)
```

OMG! Where is my “cake”?

PostgreSQL?

```
testdb=# CREATE TABLE cake (name VARCHAR(3));
CREATE TABLE

testdb=# INSERT INTO cake (name)
         VALUES ('pancake');
ERROR:  value too long for type character
varying(3)
```

Ever Seen This?

```
CREATE TABLE users (
    id integer auto_increment,
    username varchar(40),
    password varchar(8)
);
```

Data Conversion Errors - MySQL

```
mysql> CREATE TABLE test (x INTEGER);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO test (x)
       VALUES ('bad-number');
Query OK, 1 row affected, 1 warning (0.01 sec)
```

Data Conversion Errors - MySQL

```
mysql> SELECT * FROM foo;
+------+
| x    |
+------+
|    0 |
+------+
1 row in set (0.00 sec)
```

Data Conversion Errors - PostgreSQL

```
testdb=# CREATE TABLE test (x integer);
CREATE TABLE
testdb=# INSERT INTO test (x) VALUES ('bad-number');
ERROR:  invalid input syntax for integer: "bad-number"
LINE 1: INSERT INTO foo (x) VALUES ('bad-number');
                                    ^
```

Parallels - PHP vs. Python

```
$ php -r '$x = "bad-number";
          $y = (int)$x;
          echo $y."\n";'
0
```

Parallels - PHP vs. Python

```
$ python -c 'print int("bad-number")'
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: 'bad-number'
```

File Layout

File Layout - MySQL MyISAM

`/var/lib/mysql/dbname`

- `dbname.MYD` - data of all tables
- `dbname.MYI` - indexes

File Layout - MySQL InnoDB

`/var/lib/mysql/`

- **`ibdata1`** - data of all databases, including tables and indexes

It is possible to tell mysql, by changing a config flag, to make it use one file for table.

File Layout - PostgreSQL

`/var/lib/postgresql/9.3/main/base`

- **`131384/`** - directory per database
  - **`2654`** - one (or more) files for each table/index
  - **`2703`**
  - **`2683`**
  - **`2683.1`**
  - **`2683.2`**
  - **`...`**

Database Maintenance

CREATE INDEX - MySQL MyISAM

While `CREATE INDEX` is in progress:

- Entire table is locked for writes
- A new index file (dbname.MYI) need to created

CREATE INDEX - InnoDB

- Entire table rebuilt to create an index.
- Seems to have improved in recent versions

CREATE INDEX - PostgreSQL

- CREATE INDEX
  - locks the table for writes
- CREATE INDEX CONCURRENTLY
  - Doesn’t hold the lock for entire period
  - Slower than plain CREATE INDEX
- Each index is a new file

DROP INDEX - MySQL

Takes long time as it needs to rewrite:

- the index file (**`dbname.MYI`**) for MyISAM
- the **`ibdata1`** file for InnoDB

DROP INDEX - PostgreSQL

- Almost instantaneous
- Just need to delete the files corresponding to that index

ADDING NEW COLUMN - MySQL

Entire table data needs to be rewritten.

ADDING NEW COLUMN - PostgreSQL

Almost instantaneous if the new column has a default value.

Connection Model

Connection Model - MySQL

A thread for each connection

PROS

- Very easy to create a new conn

CONS

- Difficult to scale on multi-core systems
- difficult monitor threads

Connection Model - PostgreSQL

A process for each connection

PROS

- better concurrency
- complete isolation
- plays nicely with UNIX tools (ps, top, kill)

CONS

- lot of overhead for creating new conn

$ top

```
  PID  %CPU %MEM COMMAND
17618  82.7  6.8 postgres: anand voterdb [local] COPY    +
11894   0.0  7.1 postgres: [redacted] [local] idl+
16091   0.0  2.2 /usr/lib/postgresql/9.3/bin/postgres -D /va+
16093   0.0 17.3 postgres: checkpointer process          +
16094   0.0  0.4 postgres: writer process                +
16095   0.0  0.9 postgres: wal writer process           +
16096   0.0  0.1 postgres: autovacuum launcher process  +
16097   0.0  0.1 postgres: stats collector process      +
30283   0.0  6.7 postgres: [redacted] [local] idl+
```

$ top

```
  PID  %CPU %MEM COMMAND
17618  82.7  6.8 postgres: anand voterdb [local] COPY    +
11894   0.0  7.1 postgres: [redacted] [redacted] [local] idl+
16091   0.0  2.2 /usr/lib/postgresql/9.3/bin/postgres -D /va+
16093   0.0 17.3 postgres: checkpointer process          +
16094   0.0  0.4 postgres: writer process                +
16095   0.0  0.9 postgres: wal writer process           +
16096   0.0  0.1 postgres: autovacuum launcher process  +
16097   0.0  0.1 postgres: stats collector process      +
30283   0.0  6.7 postgres: [redacted] [redacted] [local] idl+
```

```
kill 17618
kill -STOP 17618
kill -CONT 17618
```

Query Planning

The Query

```
EXPLAIN
   SELECT name, total
   FROM names
   WHERE year=1995
   ORDER BY total DESC
   LIMIT 10;
```

MySQL - Query Plan

```
| id | select_type | table | type | possible_keys |
+----+-------------+-------+------+---------------+
|  1 | SIMPLE      | names | ALL  | NULL          |

key  |key_len|ref  |rows    |Extra          |
-----+-------+-----+--------+---------------+
NULL | 5     |const|12420176|Using where;   |
     |       |     |        |Using filesort |

1 row in set (0.00 sec)
```

MySQL - Add Index

```
CREATE INDEX names_total_idx
ON names(total)
```

MySQL - Query Plan With Index

```
| id | select_type | table | type | possible_keys |
+----+-------------+-------+------+---------------+
|  1 | SIMPLE      | names | ref  | NULL          |

key              |key_len|ref  |rows  |Extra         |
-----------------+-------+-----+------+--------------+
names_total_idx| 5      |const|10    |Using where;  |

1 row in set (0.00 sec)
```

PostgreSQL - QUERY PLAN

```
Limit  (cost=246987.39..246987.42 rows=10 width=13)
  ->  Sort  (cost=246987.39..247467.64 rows=192099 width=13)
        Sort Key: total
        ->  Seq Scan on names
              (cost=0.00..242836.20 rows=192099 width=13)
              Filter: (year = 1995)
(5 rows)
```

PostgreSQL - Add Index

```
CREATE INDEX names_total_idx
ON names(total)
```

PostgreSQL - Query Plan With Index

```
 Limit (cost=0.43..1891.80 rows=10 width=13)
   ->  Index Scan Backward using
       names_total_idx on names
      (cost=0.43..36332875.67 rows=192099 width=13)
         Filter: (year = 1995)
(3 rows)
```

PostgreSQL - Query Plan

```
names=# EXPLAIN ANALYZE SELECT ...

 Limit  (cost=0.43..1891.80 rows=10 width=13)
          (actual time=0.037..0.462 rows=10 loops=1)
   ->  Index Scan Backward using names_total_idx
on names  (cost=0.43..36332875.67 rows=192099 width=13)
          (actual time=0.036..0.458 rows=10 loops=1)
         Filter: (year = 1995)
         Rows Removed by Filter: 467
 Total runtime: 0.517 ms
(5 rows)
```

PostgreSQL - Complex Query

```
EXPLAIN
  SELECT name, sum(total) as count
  FROM names
  WHERE year > 1980
  GROUP BY name
  ORDER BY count
```

PostgreSQL - Query Plan

```
Sort  (cost=254889.31..255063.44 rows=69653 width=13)
  Sort Key: (sum(total))
  -> HashAggregate(cost=248589.93..249286.46 rows=69653 width=13)
        -> Bitmap Heap Scan on names
           (cost=83212.02..226363.10 rows=4445366 width=13)
             Recheck Cond: (year > 1980)
             -> Bitmap Index Scan on names_year_idx
                (cost=0.00..82100.68 rows=4445366 width=0)
                  Index Cond: (year > 1980)
(7 rows)
```

Replication

MySQL Replication

Replication Format

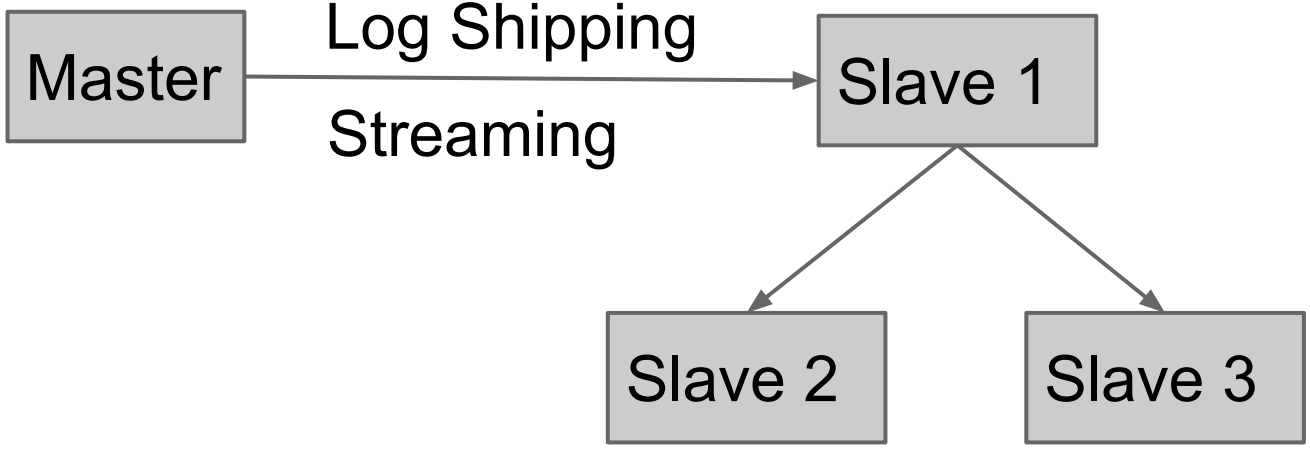
- Statement based
- Row based
- Mixed

Mode

- binlog
- GTID

PostgreSQL Replication

- Synchronous / Asynchronous
- Streaming / Log shipping



Data Recovery

PG - Point In Time Recovery

- Postgres maintains Write Ahead Log of all changes made to the database.
- The WAL files can be replayed up to any given timestamp.

Time machine of your database!

Other Interesting Features of PostgreSQL

Partial Indexes

```
SELECT * FROM comments
WHERE email LIKE '%@spam.com';


CREATE INDEX comments_spam_idx
ON comments
WHERE email LIKE '%@spam.com';
```

Functional Indexes

```
SELECT tag, count(*)FROM posts
GROUP BY lower(category);

CREATE INDEX post_category_idx
ON post (lower(category));
```

JSON datatype

```
CREATE TABLE book (
   id serial primary key,
   data JSON
);

INSERT INTO book (data) VALUES (
'{"title": "Tom Sawyer",
  "author": "Mark Twain"}')
```

JSON datatype

```
SELECT * FROM book
WHERE data->>'author' = 'Mark Twain'

id|                          data
--+-------------------------------------------
1 |{"title": "Tom Sawyer", "author": "Mark Twain"}
(1 row)
```

pg_stat_statements

```
SELECT total_time/calls AS t, calls, query
FROM pg_stat_statements
ORDER BY t DESC
```

```
8606.75|3|select name, sum(total) as count
       | |from names group by name order by count limit ?;
   4.92|8| select name, total from names
       | | where year=? order by total desc limit ?;
```

Summary

PostgreSQL is better than MySQL in

- Data Consistency
- Query Planning
- Stability
- Database Maintenance
- Data Recovery

Worth trying PostgreSQL for your next project!

Credits

PostgreSQL Logo - By Jeff MacDonald

http://pgfoundry.org/docman/?group_id=1000089

Thanks!

Anand Chitipothu
@anandology