“Drop-in Replacement”: Defining Compatibility for Postgres and MySQL Derivatives

Jimmy Angelakos, Daniël van Eeden

FOSDEM 2026 Database Devroom

Who are we?

- Jimmy Angelakos
- Based in Edinburgh, Scotland
- Staff Software Engineer, pgEdge
- Open Source user & contributor (25+ years)
- PostgreSQL Significant Contributor
- Author, PostgreSQL Mistakes and How to Avoid Them
- pg_statviz PostgreSQL extension

Who are we?

- Daniël van Eeden
- Working for PingCAP on TiDB
- Contributor to MySQL, go-mysql, Wireshark, ...
- Awarded MySQL Rockstar in 2023

The landscape

The success of open source databases like PostgreSQL and MySQL has created an ecosystem of derivatives claiming "drop-in compatibility."

- User confusion and brand dilution as derivatives diverge from upstream
- Compatibility is not an absolute Yes/No situation
- Even different versions of the *same* database are not 100% compatible (deprecated & added features)
- What does "PostgreSQL compatible" or "MySQL compatible" actually mean?

Defining compatibility

From a user perspective:

- Will my application work with this database?
- What database drivers and APIs can I use?
- Can I replicate from and to this database?
- Does a cloud DBaaS offering provide the same experience?
- Can I use an existing ecosystem of tools and drivers?

Two perspectives

The Standard: PostgreSQL Compatibility

The Implementation: MySQL Compatibility

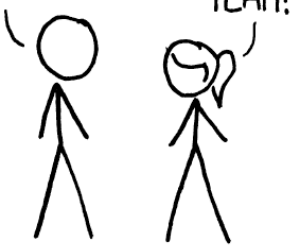
PGConf.EU 2025 working group

- "Establishing the PostgreSQL Standard: What's Postgres compatible?" working session at PGConf.EU in Riga, Latvia (Oct 2025)
- Goal: practical framework of criteria and tests for PostgreSQL compatibility
- With PostgreSQL becoming "the new Linux" for the enterprise, this is increasingly important

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

Why not just ISO SQL?

Why care about “PostgreSQL compatible” instead of simply ISO/IEC SQL:2023?

The SQL standard doesn't define:

- Implementation details like indexes and on-disk data formats
- Function names for features not yet standardized (vector search, UUIDv7)
- Administrative commands
- Network protocol

The Riga consensus

Work has pivoted from a binary "Pass/Fail" certification to a granular **compatibility matrix**.

- **Weighted checklist** (Core vs Optional), to avoid stifling innovation
- **"Managed" PG:** Specific exceptions for managed environments (restricted superuser, filesystem access)
- **No silent failures:** `CREATE INDEX` must actually build the index, not just return "success"

Core SQL & feature set

Everything covered by Postgres documentation in "Part II: SQL Commands" is required.

- Even rarely-used features must be supported
- **Implicit behaviours:** Undocumented behaviours users rely on (e.g. `INSERT ... ORDER BY`)
- **Data types:** `ARRAY`, `BYTEA`, `JSONB` required
- **Feature dependencies:** Triggers imply full PL/pgSQL language support

Protocol & behavioural compatibility

- **Transaction isolation:** Standard levels (Read Committed, Repeatable Read, Serializable) with identical behaviour
- **Error codes:** Identical `SQLSTATE` codes for common errors
- **System catalogs:** `pg_catalog` must be present and predictable for monitoring tools
- **Hard limits:** Must generally match Postgres limits (e.g. identifier length, max columns)

Connectivity & tools

- Server version string must satisfy standard tools (e.g. `psql`)
- Standard drivers (JDBC, Psycopg) must work without modification
- Ability to `pg_dump` import/export to/from PostgreSQL
- Execution plans need not be identical, but functional results must match (e.g. partitions are pruned)
- Standard tools compatibility (Patroni, `pg_basebackup`, RepMgr)

Replication compatibility

Logical Replication:

- Bi-directional (Product X $\leftrightarrow$ PostgreSQL)
- Observable via standard replication catalogs
- Vendor extensions must not break connection to vanilla Postgres

Physical Replication:

- Physical copying of binary files (or equivalent)
- Hybrid clusters: mixing vanilla and vendor nodes
- Point-In-Time Recovery (PITR) and WAL access required

The test harness

- Test suite outside the PostgreSQL codebase
- Test suite versions correspond numerically to PG releases
- Vendors provide compatible build target
- Testing feature compliance, not bugs—fixing a Postgres bug should not cause test failure

The Compatibility Illusion

```
mysql> INSERT INTO t1 VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)

mysql> PRAGMA table_list;
+--------+------+-------+------+----+
| schema | name | type  | ncol | wr |
+--------+------+-------+------+----+
| main   | t1   | table | 1    | 0  |
+--------+------+-------+------+----+
1 row in set (0.00 sec)

mysql> SELECT sqlite_version();
+------------------+
| sqlite_version() |
+------------------+
| 3.46.0           |
+------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM generate_series(
    '2025-02-01'::TIMESTAMP,
    '2025-02-02'::TIMESTAMP,
    '12 hour');
+----------------------+
| generate_series      |
+----------------------+
| 2025-02-01T00:00:00Z |
| 2025-02-01T12:00:00Z |
| 2025-02-02T00:00:00Z |
| 2025-02-02T12:00:00Z |
+----------------------+
4 rows in set (0.00 sec)

mysql> SELECT VERSION();
+---------------------------------+
| version                         |
+---------------------------------+
| PostgreSQL 17.2 (Debian 17.2-1.. |
+---------------------------------+
1 row in set (0.00 sec)
```

Compatible, no?

- Is this MySQL compatible? ... well the protocol is.
- Is this PostgreSQL compatible? ... well the syntax is.
- Examples created with `go-mysql` as MySQL protocol proxy with SQLite and PostgreSQL backends.
- What if a cloud provider did something similar?

TiDB: Architecture

TiDB is:

- A MySQL compatible database
- Not based on or descending from MySQL code like MariaDB
- Written in Go
- Using TiKV (RocksDB) as storage layer, not InnoDB
  - Accepts `ENGINE=InnoDB` but silently ignores it
- A distributed database

TiDB: Feature Compatibility

- Some features like Geospatial and `VECTOR` are seen as optional
- **XML:** TiDB never implemented it—JSON has become more popular
- **Vector:** MySQL has the datatype, but search only in Cloud
- TiDB adds useful MariaDB features (e.g. sequences)
- Should compatibility on its own be the target?

MariaDB

- Originally a true drop-in replacement for MySQL
  - Stop MySQL, swap binaries, start—no import/export needed
  - Over time, MySQL and MariaDB have diverged significantly
- MySQL `sql_mode`: historically offered compatible syntax modes for Oracle, PostgreSQL, and others

Wire protocol challenges

MySQL protocol v10 uses **capability flags** for feature negotiation (compression, auth methods, etc.).

Architectural friction:

- Clients also use the announced *version string* for feature detection
- MySQL Connector/J queries `information_schema.keywords` if version ≥ 8.0
  - When TiDB changed to `8.0.11-TiDB-v7.5.3`, we had to implement this table
- TiDB forked MySQL Connector/J to support TiDB-specific authentication

SQL Syntax & Type Friction

Reserved keywords: Vary significantly between database products and versions.

EXPLAIN formats: MySQL supports multiple formats; TiDB uses its own—distributed query plans look fundamentally different.

Type implementation divergence:

- MySQL = **`JSON`** type, MariaDB = `TEXT` + `JSON_VALID()`
- MySQL = `UUID_TO_BIN()`, MariaDB = **`UUID`** type

Explain formats

```
mysql-8.4.7> EXPLAIN SELECT user FROM mysql.user WHERE host='%' AND user='root';
+----+-------------+-------+------------+-------+---------------+---------+---------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len |
+----+-------------+-------+------------+-------+---------------+---------+---------+
|  1 | SIMPLE      | user  | NULL       | const | PRIMARY       | PRIMARY | 351     |
+----+-------------+-------+------------+-------+---------------+---------+---------+

TiDB-v8.5.4> EXPLAIN SELECT user FROM mysql.user WHERE host='%' AND user='root';
+-------------+---------+------+----------------------------------------+---------------+
| id          | estRows | task | access object                          | operator info |
+-------------+---------+------+----------------------------------------+---------------+
| Point_Get_1 | 1.00    | root | table:user, index:PRIMARY(Host, User) |               |
+-------------+---------+------+----------------------------------------+---------------+
```

Binlog replication compatibility

The Shift: Statement → Row-Based (RBR)

- Hard to guarantee same result on replica

The CDC Gap: RBR was built for MySQL, not generic CDC

- Missing type metadata (e.g. signedness)

Protocol Drift: New `mysql::serialization` format

- Introduced for GTID tags; custom format, not Protobuf
- Minimal documentation → `go-mysql` friction

TiDB Strategy: Bypasses binlog generation

- **Outbound:** TiCDC **Inbound:** Data Migration (DM)

Errors, limits, and bugs

Error Codes:

- MySQL errors have Code, SQLState, and message
- Should messages be identical or only codes?
- TiDB uses error codes >8000 to avoid conflicts

Limits: TiDB matches MySQL limits by default (e.g. 64-char object names).

Bug-for-bug compatibility? MySQL has a bug in `FORMAT()` for Bulgarian. Should TiDB replicate bugs to stay compatible?

Thank you!

Questions?