The Language of Databases

A Deep Dive into MySQL and PostgreSQL Wire Protocols

\+ a quick look at TDS

Sergey Olontsev

Software Engineer

ex-MVP (2013-2019), Microsoft Certified Master.

20 years of experience with databases and high load systems. Worked with OLTP databases up to 50 TB, relational DWH up to 200 TB. Each, not in total! :)

100+ billions of rows in tables (OLTP).

https://olontsev.io

https://linkedin.com/in/sergeyolontsev

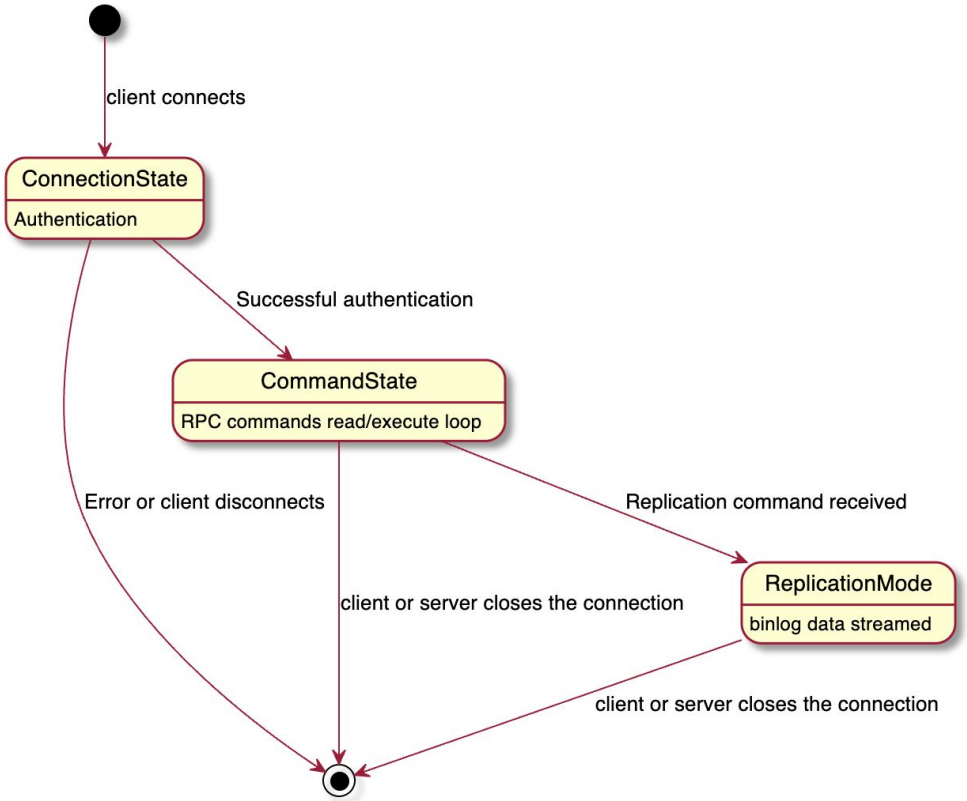
https://github.com/solontsev

Microsoft CERTIFIED Master SQL Server® 2008

Microsoft CERTIFIED Solutions Master Charter - Data Platform

What to expect?

MySQL Protocol



Connection Lifecycle

https://dev.mysql.com/doc/dev/mysql-server/9.4.0/page_protocol_connection_lifecycle.html

Packet

Data between client and server is exchanged in packets of max 16MB size.

Type	Name	Description
int<3>	payload_length	Length of the payload. The number of bytes in the packet beyond the initial 4 bytes that make up the packet header.
int<1>	sequence_id	**Sequence ID**
string<var>	payload	payload of the packet

`01 00 00 00 01`	• length: 1 • sequence_id: x00 • payload: 0x01

Sending More Than 16MB

If the payload is larger than or equal to 2^24-1 bytes the length is set to 2^24-1 (ff ff ff) and a additional packets are sent with the rest of the payload until the payload of a packet is less than 2^24-1 bytes.

Sending a payload of 16 777 215 (2^24-1) bytes looks like:

```
ff ff ff 00 ...
```

```
00 00 00 01
```

Compressed Packet Header

If the length of ***length of payload before compression*** is more than 0 the Compressed Packet Header is followed by the compressed payload.

Type	Name	Description
int<3>	length of compressed payload	raw packet length minus the size of the compressed packet header (7 bytes) itself.
int<1>	compressed sequence id	Sequence ID of the compressed packets, reset in the same way as the **Protocol::Packet**, but incremented independently
int<3>	length of uncompressed payload	Length of payload before compression

PostgreSQL Protocol

History

Current version: 3.0, introduced in server version 7.4

Released 21 years ago (17 Nov 2003), support ended 14 years ago (01 Oct 2010)

Version 2 support was dropped in server version 14 (30 Sep 2021)

Three Main Phases

Startup Phase: This phase establishes the connection between the client and server. The client sends a StartupMessage containing information such as the desired database, user name, and protocol version. The server responds with an AuthenticationRequest, which may prompt the client for password authentication or other authentication methods.

Query Phase: Once the connection is established, the client can send queries to the server. Queries are sent as Query messages, and the server executes the queries and returns the results in a series of messages, including RowDescription, DataRow, and CommandComplete.

Termination Phase: When the client is done with the connection, it sends a Terminate message to the server, which then closes the connection.

Startup Phase

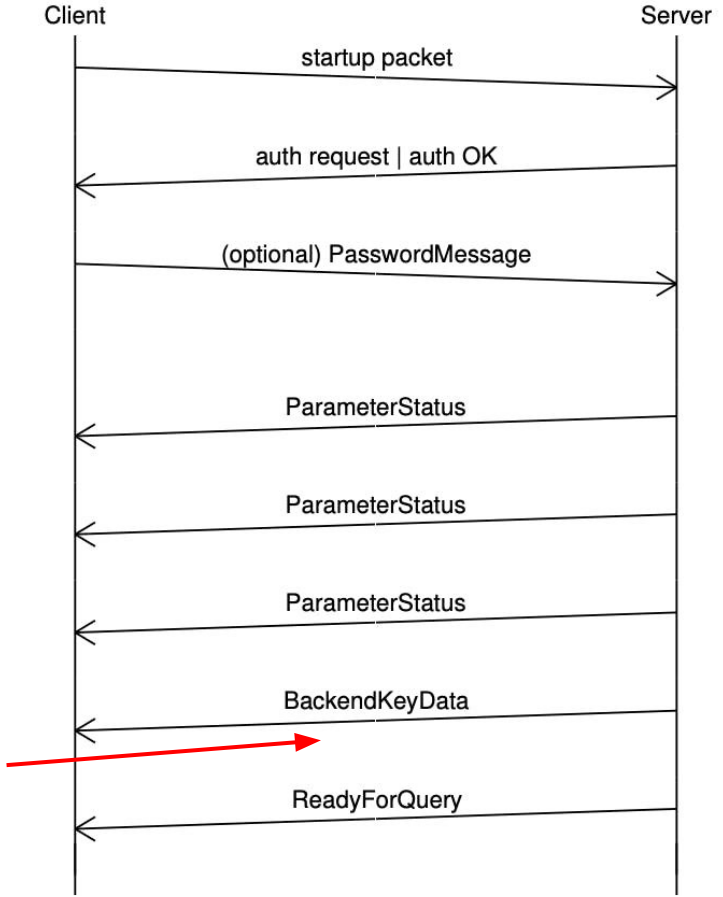
Startup packet

int32 len	int32 protocol	str name	\0	str value	...	\0

Regular packet

char tag	int32 len	payload

Contains the “query cancellation key” that will be needed to perform query cancellation later. The client saves it somewhere.

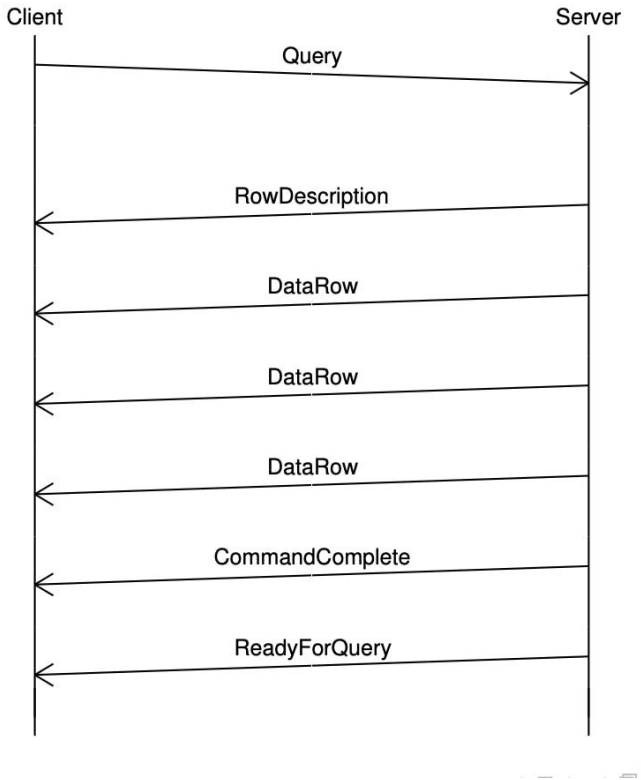


Simple Query Protocol

Single-step process: The client sends a single SQL statement to the server, and the server executes it and returns the result. This is done using the QUERY message.

No parameter binding: The query is sent as a plain text string, and there is no support for parameterized queries. This means that values must be included directly in the query string, which can increase the risk of SQL injection if not handled properly.

No prepared statements: The Simple Query Protocol does not support prepared statements, so the query is parsed and planned each time it is executed.



Extended Query Protocol

Multi-step process: The Extended Query Protocol involves several steps: parsing the query (PARSE message), binding parameters to the parsed query (BIND message), executing the bound statement (EXECUTE message), and optionally describing the statement (DESCRIBE message).

Parameter binding: It supports parameterised queries, where parameters are sent separately from the query text, reducing the risk of SQL injection.

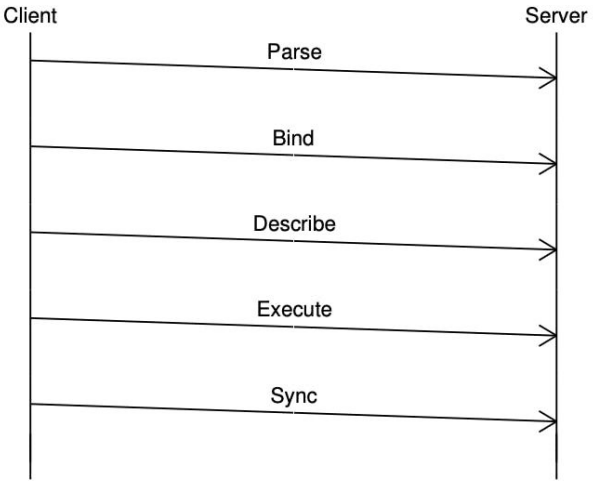
Prepared statements: The protocol supports prepared statements, which allows the server to parse, plan, and optimize a query once and then execute it multiple times with different parameters. This can improve performance for frequently executed queries.

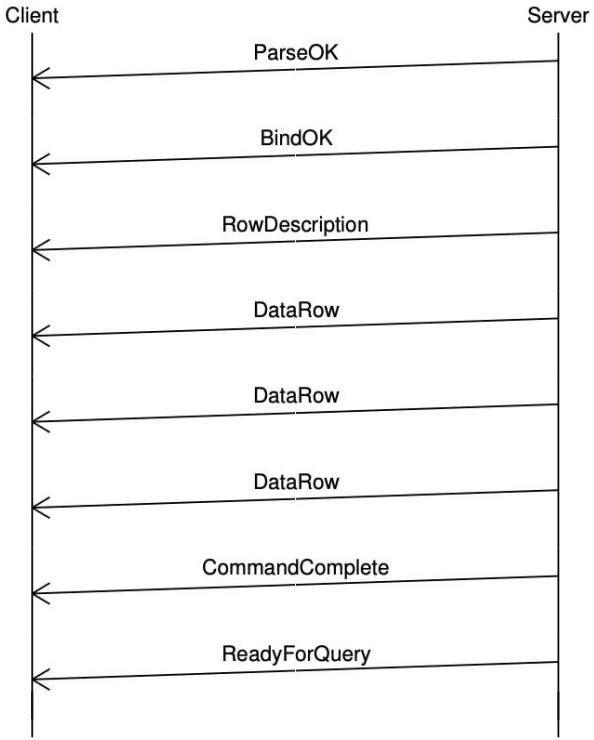
More control and flexibility: It provides more control over the execution of queries, such as specifying result formats, and is more suitable for complex applications that require higher performance and security.

Implementations by client: In PostgreSQL's wire protocol, clients use unique identifiers for prepared statements (Parse) like S_1, S_2, etc per connection. I mean S_1 can be mapped with a query like a COMMIT but the same S_1 Identifier can be for a different query like SELECT id, name FROM pets WHERE p1_0.owner_id = $1; . This shows how each connection maintains an isolated internal state between the client and server.

These statements are then bound with parameters (Bind) and executed (Execute) efficiently. This approach ensures secure and optimized SQL query execution between clients and servers, enhancing performance and flexibility in handling parameterized queries.

Extended Query Protocol





References

- MySQL
  - https://dev.mysql.com/doc/dev/mysql-server/9.4.0/PAGE_PROTOCOL.html
  - Book “Understanding MySQL Internals” by Sasha Pachev, chapter 4 “Client/Server Communication”
  - A Deep Dive Into the MySQL Client/Server Protocol: https://www.youtube.com/watch?v=t8AMrLblyI8
  - https://github.com/jonhoo/msql-srv
  - https://github.com/databendlabs/opensrv
- PostgreSQL
  - https://www.postgresql.org/docs/current/protocol.html
  - The PostgreSQL Protocol: The Good, the Bad and the Future (PGConf.dev 2024): https://www.youtube.com/watch?v=nh62VgNj6hY
  - Heikki Linnakangas - The Wire Protocol (PGConf.EU 2024): https://www.youtube.com/watch?v=FBPubrwGKhI
  - https://github.com/sunng87/pgwire
  - https://github.com/cube-js/cube/tree/master/rust/cubesql/pg-srv
- TDS
  - https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-tds/b46a581a-39de-4745-b076-ec4dbb7d13ec
- https://github.com/solontsev/db-protocols