PostgreSQL Capabilities

Replication

- Replication allows data from one database server to be replicated to another server.
- Replication is used mainly to achieve the following:
  - **High availability**: A second server can take over if the primary server fails.
  - **Load balancing**: Several servers can serve the same requests.
  - **Fast execution**: PostgreSQL is shipped with the tools to execute the queries efficiently, such as several kind of indexes and smart planner. In addition, the read load can be distributed on the replicas in streaming replication.

- PostgreSQL supports replication out of the box via streaming and logical replication.

- **Streaming replication** is a master-standby replication that uses file-based log shipping.
- It's based on taking a snapshot of the master node, and then shipping the changes—the WAL files—from the master node to the slave node and replaying them on the slave.
- The master can be used for read/write operations, and the slave can be used to serve read requests.
- Streaming replication is relatively easy to set up and configure; it can support synchronous and asynchronous replications as well as cascading replication.

- A synchronous replication is the default streaming-replication mode.
  - If the primary server crashes, transactions that were committed on the primary may not have been replicated to the standby server, leading to data inconsistency.
  - In synchronous replication, a data-modifying operation must be committed on one or more synchronous servers in order to be considered successful.
- In cascading replication, one could add a replica to a slave.
  - This allows PostgreSQL to scale horizontally for read operations.

- PostgreSQL also supports **logical streaming replication**; unlike streaming replication, which replicates the binary data bit by bit, logical replication translates the WAL files back to logical changes.
- This gives us the freedom to have more control over the replication such as applying filters.
  - For example, in logical replication, parts of data can be replicated, while in streaming replication the slave is a clone of the master.
- Another important aspect of logical replication is being able to write on the replicated server, such as extra indexes, as well as temporary tables.
- Finally, you can replicate data from several servers and combine it on a single server.

Security

- PostgreSQL supports several authentication methods, including password, LDAB, GSSAPI, SSPI, Kerberos, ident-based, RADUIS, certificate, and PAM authentication.
- All database vulnerabilities are listed in the PostgreSQL security information web page (http://www.postgresql.org/support/security/) with information about the affected version, the vulnerability class, and the affected component.

- The PostgreSQL security updates are made available as minor updates.
- Also, known security issues are always fixed with the next major releases.
- Publishing security updates in minor updates makes it easy for a PostgreSQL administrator to keep PostgreSQL secure and up to date with minimal downtime.

- PostgreSQL can control the database object access at several levels, including database, table, view, function, sequence, column, and row.
- This enables PostgreSQL to have great authorization control.

- PostgreSQL can use encryption to protect data by hardware encryption.
- Also, you can encrypt certain information by utilizing the `pgcrypto` extension.

Extensions

- PostgreSQL can be extended to support new data types and functionality.
- Extensions in PostgreSQL are a great way to add new functions, not being a core developer.
- PostgreSQL provides the `CREATE EXTENSION` command to load extensions to the current database.
- Also, PostgreSQL has a central distribution network (PGXN: www.pgxn.org) that allows users to explore and download extensions.
- When installing the PostgreSQL binaries, the `postgresql-contib` package contains many useful extensions, such as `tablefunc`, which allows table pivoting, and the `pgcrypto` extension; the `README` file in the installation directory contains the summary information.

NoSQL capabilities

- PostgreSQL is more than a relational database and a SQL language.
- PostgreSQL is now home to different NoSQL data types.
- The power of PostgreSQL and schema-less data stores enables developers to build reliable and flexible applications in an agile way.

- PostgreSQL supports the **JavaScript Simple Object Notation** (**JSON**) data type, which is often used to share data across different systems in modern RESTful web applications.
- In PostgreSQL release 9.4, PostgreSQL introduced another structured binary format to save JSON documents instead of using the JSON format in prior versions.
  - The new data type is called JSONB.
  - This data type eliminates the need to parse a JSON document before it's committed to the database.
  - In other words, PostgreSQL can ingest a JSON document at a speed comparable with document databases, while still maintaining compliance with ACID.
- In PostgreSQL version 9.5, several functions are added to make handling JSON documents much easier.
- In version 10, full text search is supported for JSON and JSONB documents.

- Key/value pairs are also supported by the PostgreSQL `hstore` extension.
- `hstore` is used to store semi-structured data, and it can be used in several scenarios to decrease the number of attributes that are rarely used and often contain null values.

- Finally, PostgreSQL supports the **Extensible Markup Language** (**XML**) data type.
- XML is very flexible and is often used to define document formats.
- XML is used in RSS, Atom, SOAP, and XHTML.
- PostgreSQL supports several XML functions to generate and create XML documents.
- Also, it supports XPath to find information in an XML document.

Foreign Data Wrappers

- In 2011, PostgreSQL 9.1 was released with a read-only support for SQL/**Management of External Data** (**MED**) ISO/IEC 9075-9:2003 standard.
- SQL/MED defines **foreign data wrappers** (**FDWs**) to allow the relational database to manage external data.
- FDW can be used to achieve data integration in a federated database-system environment.
- PostgreSQL supports RDBMS, NoSQL, and foreign data wrapper files, including Oracle, Redis, MongoDB, and delimited files.

- A simple use case for FDWs is to have one database server for analytical purposes, and then ship the result of this server to another server that works as a caching layer.

- Also, FDW can be used to test data changes.
- Imagine you have two databases, one with different data due to applying a certain development patch.
- One could use FDW to assess the effect of this patch by comparing the data from the two databases.

- PostgreSQL supports `postgres_fdw` starting from release 9.3.
- `postgres_fdw` is used to enable data sharing and access between different PostgreSQL databases.
- It supports the SELECT, INSERT, UPDATE, and DELETE operations on foreign tables.

- The following example shows you how to read **comma-separated value** (**CSV**) files using FDW; you can read CSV files to parse logs.
- Let's assume that we want to read the database logs generated by PostgreSQL.
- This is quite useful in a production environment as you can have statistics about executed queries; the table structure can be found in the documentation at https://www.postgresql.org/docs/current/static/runtime-config-logging.html.

- To enable CSV logging, you need to change the following values in `postgresql.conf`.
- For simplicity, all statements will be logged, but this isn't recommended in a production environment since it'll consume a lot of server resources:

```
log_destination = 'csvlog'
logging_collector = on
log_filename = 'postgresql.log'
log_statement = 'all'
```

- For the changes to take effect, you need to restart PostgreSQL from the Terminal as follows:

```
sudo service postgresql restart
```

- To install the FDW file, we need to run the following command:

```
postgres=# CREATE EXTENSION file_fdw ;
CREATE EXTENSION
```

- To access the file, we need to create the FDW server, as follows:

```
CREATE SERVER fileserver FOREIGN DATA WRAPPER file_fdw;
```

- Also, we need to create an FDW table and link it to the log file; in our case, it's located in the log folder in the PostgreSQL cluster directory:

```
CREATE FOREIGN TABLE postgres_log
( log_time timestamp(3) with time zone,
  user_name text,
  database_name text,
  process_id integer,
  connection_from text,
  session_id text,
  session_line_num bigint,
  command_tag text,
  session_start_time timestamp with time zone,
  virtual_transaction_id text,
  transaction_id bigint,
  error_severity text,
  sql_state_code text,
  message text,
  detail text,
  hint text,
  internal_query text,
  internal_query_pos integer,
  context text,
  query text,
  query_pos integer,
  location text,
  application_name text
) SERVER fileserver OPTIONS ( filename '/var/lib/postgresql/11/main/log/postgresql.csv', header 'true', format 'csv' );
SELECT row_to_json(postgres_log, true) FROM postgres_log limit 1;
```

- To test our example, let's get one log line in JSON format, as follows:

```
postgres=# SELECT row_to_json(postgres_log, true) FROM postgres_log limit 1;
                                row_to_json
-------------------------------------------------------------------------
 {"log_time":"2018-12-10T00:35:19.768+01:00", +
  "user_name":null, +
  "database_name":null, +
  "process_id":25847, +
  "connection_from":null, +
  "session_id":"5c0da6b7.64f7", +
  "session_line_num":1, +
  "command_tag":null, +
  "session_start_time":"2018-12-10T00:35:19+01:00", +
  "virtual_transaction_id":null, +
  "transaction_id":0, +
  "error_severity":"LOG", +
  "sql_state_code":"00000", +
  "message":"database system was shut down at 2018-12-10 00:35:19 CET",+
  "detail":null, +
  "hint":null, +
  "internal_query":null, +
  "internal_query_pos":null, +
  "context":null, +
  "query":null, +
  "query_pos":null, +
  "location":null, +
  "application_name":""}
(1 row)
```

- As we've seen, you can store the PostgreSQL logs in the PostgreSQL cluster.
- This allows the developer to search for certain user actions.
- Also, it allows the administrators to conduct statistical analysis on performance, such as finding the slowest queries to tune the PostgreSQL server configuration or rewriting slow queries.

Performance

- PostgreSQL has a proven performance.
- It employs several techniques to improve concurrency and scalability.

- **PostgreSQL locking system**: PostgreSQL provides several types of locks at the table and row levels. PostgreSQL is able to use more granular locks that prevent locking/blocking more than necessary; this increases concurrency and decreases the blocking time.

- **Indexes**: PostgreSQL provides six types of indexes: btree, hash, Generalized Inverted Index (**GIN**), the **Generalized Search Tree** (**GiST**) index, SP-GiST, and **Block Range Indexes** (**BRIN**). Each index type can be used for a certain scenario. For example, btree can be used for efficient equality and range queries. GIST can be used for text search and for geospatial data. PostgreSQL supports partial, unique, and multicolumn indexes. It also supports indexes on expressions and operator classes.

- **Explain, analyze, vacuum, and cluster**: PostgreSQL provides several commands to boost performance and provide transparency. The `explain` command shows the execution plan of a SQL statement. You can change some parameter settings, such as memory settings, and then compare the execution plan before and after the change. The `analyze` command is used to collect the statistics on tables and columns. The `vacuum` command is used for garbage collection to reclaim unused hard disk space. The `cluster` command is used to arrange data physically on the hard disk. All these commands can be configured based on the database workload.

- **Table inheritance and constraint exclusion**: Table inheritance allows us to easily create tables with the same structure. Those tables are used to store subsets of data based on certain criteria. This allows a very fast retrieval of information in certain scenarios, because only a subset of data is accessed when answering a query.

- **Very rich SQL constructs**: PostgreSQL supports very rich SQL constructs. It supports correlated and uncorrelated subqueries. It supports **common table expression** (**CTE**), window functions, and recursive queries. Once developers have learned these SQL constructs, they will be able to write a crisp SQL code very quickly. Moreover, they will be able to write complex queries with minimal effort. The PostgreSQL community keeps adding new SQL features in each release.