Assessment

Chapter 2: Understanding Transactions and Locking

- **What is the purpose of a transaction?**

The purpose of a transaction is to make sure that large operations can be made **atomic**. What does that mean? It means that you want everything or nothing. If you want to delete a million rows, for example, you want them all or none of them to be done. The purpose of a transaction is to ensure exactly this kind of behavior.

- **How long can a transaction in PostgreSQL be?**

Practically the length of a transaction is almost unlimited. However, there are of course theoretical limitations. In the case of PostgreSQL, we are talking about 4 billion statements. Note: It does not 4 billion lines - it really means 4 billion writing statements (each can modify as many rows as desired).

- **What is transaction isolation?**

Transaction isolation is a way to control the visibility of data within a transaction. This is especially important to handle reporting and long transactions.

- **Should we avoid table locks?**

Table locks should be avoided when possible. When designing a database one has to keep in mind that a user is usually never alone. This table lock has the potential to exclude others.

- **What do transactions have to do with VACUUM?**

VACUUM is in charge of cleaning up dead rows created by transactions. Dead rows can be caused by a deletion, update, or simply inserts that are facing a rollback. Without VACUUM dead rows will accumulate in storage and cause issues including but not limited to performance problems.

Chapter 3: Making Use of Indexes

- **Do indexes always improve performance?**

  No, an index scan speeds up some reads but it is definitely going to slow down writes. If an index would always make sense it would be there by default which it isn't. Thus one has to think cleverly about indexes and use them when they make sense. In general, too many indexes are a smaller evil than not having enough indexes.

- **Does an index use a lot of space?**

  It depends on the type of index. Brin indexes are really small while `btree`, `gist`, `gin`, and alike use a lot more space. However, space consumption is just one part of the equation. Indexes have to make sense. It is mostly trading space for selection speed.

- **How can I find missing indexes?**

  One has to take a look at `pg_stat_statements` and `pg_stat_user_tables`. Understanding missing indexes is not possible without gaining a reasonably good understanding of SQL in general.

- **Can indexes be built in parallel?**

  A more recent version can use more than one CPU core to build a single index. Two indexes on the same table cannot be built in parallel. However, one can always create indexes on different tables in parallel.

Chapter 4: Handling Advanced SQL

- **Why should one use windowing functions instead of subselects?**

  In general. it is ways more efficient to use windowing functions than complicated subselects. In addition to that, the SQL statements are going to be ways more simplistic and easier to read in general. The `WINDOW` clause will also offer a fair amount of abstraction in case code is used in more than one place.

- **What is the purpose of** `PARTITION BY`?

  `PARTITION BY` is basically a reverse `GROUP BY`. It splits data into various groups and makes sure that a certain operation is executed for the right subset of data (**window**). It allows to flexibly break up the data into the desired subsets.

- **Are analytics supported by every database?**

  All major database systems including but not limited to Oracle, DB2, MS SQL, etc support windowing functions, ordered sets, and other advanced SQL features. These features are required by the ANSI SQL standard and are therefore implemented almost identically in all major database engines used these days.

- **It is possible to write custom aggregates?**

  Yes, this is possible. PostgreSQL offers the `CREATE AGGREGATE` command to do that. All you need is a set of stored procedures to write your custom business logic.

- **What is the difference between ordered sets and normal aggregates?**

  The main difference is that an ordered set requires data to be processed in a certain order. A normal aggregate such as sum, count, avg, and so on does not care about the order. The result of a count does not change if data is fed in a different order. However, if you want to rank data using a hypothetical aggregate (= some kind of ordered set) order does matter. You want to rank according to some field and thus this order has to be specified.

Chapter 5: Log Files and System Statistics

- **Why do we need log files?**

  PostgreSQL allows you to send a lot of information to the log file which other databases store in tables. One example would be information about who connected and disconnected when and so on. Logfiles are easy to read and offer great insights into what is going on on your server.

- **What is the performance overhead of logging?**

  This depends on various factors. If statements are short and can be executed quickly logging will naturally create more overhead than in the case of long-running transactions which take minutes or sometimes even hours to process. If all logging is turned on we have seen performance going down up to 40%. However, this is not the standard case.

  It is not recommended to log all statements for performance reasons.

- **Can logs be stored in a database?**

  By default, logs are sent to the filesystem. However, nothing stops us from importing those logs back into PostgreSQL for further processing. However, this is not done too often. Many people use external tools to handle logging in a centralized way.

- **How expensive are system statistics?**

  Basically, system tables are not too expensive. We have not seen issues caused by system statistics performance.

Chapter 6: Optimizing Queries for Good Performance

- **What does the optimizer do in general?**

  The idea behind SQL is that the end-user writes a query and the database tries to figure out the best way to execute it. The job of the optimizer is to find the *best way* to run the query. To facilitate this the optimizer uses system statistics and countless mathematical transformations to get the best out of the query.

- **Does every database have an optimizer?**

  Most modern databases such as Oracle, DB2, and MS SQL use a cost-based optimizer to optimize queries. While every database engine is doing things differently the basic concepts are more or less similar. Every database tries to come up with some sort of cost to compare various operations to each other. At the end of the day, the goal is to find the best way to execute a query.

- **Can the database preserve execution plans?**

  Well, this is a bit more tricky than people might expect. In Oracle, plans can be stored globally. In PostgreSQL execution plans are stored in a database connection. So yes, plans can be stored and cached inside a connection but they are not global as in other databases. Therefore databases connections will have a copy of the plan for the same type of query locally which is not necessarily a bad thing as it makes a lot of things less complicated.

- **Can we tune the optimizer?**

  Yes, this is doable. The PostgreSQL optimizer offers various parameters to adjust costs, strategy, and a lot more. There is even a method to perform genetic query optimization. One can adjust all those settings. However, it is important to have a decent understanding of what is going on in the optimizer before touching those things. Changing those settings can backfire and create unexpected results.

Chapter 7: Writing Stored Procedures

- **Which languages are available to write stored procedures?**

  PostgreSQL provides various languages out of the box to write stored procedures. It is of course always possible to use plain SQL to write functions. However, to do more advanced stuff it is recommended to use PL/pgSQL. It provides more features such as flow control.

  However, some people want to use other languages. On Linux PL/Perl, PL/Python, and PL/TCL are also available out of the box to write procedures. If this is still not enough there are many more language bindings out there. One of the more noteworthy ones is PL/R which allows you to write mathematical models in R.

- **What is the difference between a function and a procedure?**

  A function is part of a transaction. It can be used as part of some kind of SQL statement. A procedure is different: It cannot be used in `SELECT`, `INSERT`, `UPDATE` or `DELETE` statements and might span more than one transaction. A procedure is more like a batch-job while a function is a piece of business logic used as part of a bigger operation.

  Functions cannot do advanced transaction processing while procedures got more options in this area.

- **Which language is "the best one"?**

  This is like asking *what is the best food in the world?*. There are most likely more opinions than people on this planet. In general, the best language is one that serves a purpose well. There is no best language that fits all. While PL/Perl might be nice to handle regular expressions PL/pgSQL might be more suitable to handle SQL.

- **Are there any other languages?**

  There are many language bindings for PostgreSQL out there. You can install what is best for you using `CREATE EXTENSION` and your favorite package manager. Of course, not all languages are created equal and not all language bindings work equally well. However, one can be sure that the built-in languages are rock solid and have been tested extensively.

Chapter 8: Managing PostgreSQL Security

- **Is PostgreSQL a secure database?**

  PostgreSQL offers many security-related features that can be used to make the system really secure. However, security needs constant attention. One has to think about it and make use of those features provided by the database to end up with the desired result.

- **Does PostgreSQL support in disk encryption?**

  Standard PostgreSQL is currently not able to encrypt data in disk. However, CYBERTEC provides a flavor of PostgreSQL called PostgreSQL TDE which is able to encrypt data on disk. The code, as well as binary packages, can be downloaded for free from the CYBERTEC website: `https://www.cybertec-postgresql.com/en/products/postgresql-transparent-data-encryption/`

- **Can PostgreSQL encrypt client connections?**

  Yes. It is possible to encrypt client/server connections using SSL. All you have to do is to set a couple of parameters and put certificates on the server. A detailed guideline is available in this book and in the official PostgreSQL documentation.

- **Does security impact performance?**

  It depends on what you are trying to achieve. Nothing is free. However, most things such as `GRANT`, `REVOKE`, and so on have no relevant impact. Other features such as Row-Level-Security might impact performance in case those security policies are not written carefully.

Chapter 9: Handling Backup and Recovery

- **What is the difference between a logical and a binary backup?**

A logical backup is done using `pg_dump` is simply a long-running repeatable read transaction reading all the data in a consistent way. This snapshot can then be stored in a file. The output is simply an SQL script that can be replayed anywhere easily.

A binary backup is platform-dependent and is usually used for backups and replication. It is as large as the original database. In a logical backup and index is simply a `CREATE INDEX` instruction. In the case of a binary backup, it is a full copy.

- **Which formats does `pg_dump` support?**

`pg_dump` supports various formats. *plain* is simple a text stream. *custom format* is basically a compressed text stream including a TOC (= table of contents). It supports parallel backup replay.

Finally, there is a *directory format* that breaks up the backup into multiple files. It supports parallel dumps as well as a parallel replay.

- **How long does it take to replay a backup?**

That depends on many factors. Hardware, network bandwidth (in case the backup has to be fetched from somewhere). In the case of a logical backup, the biggest limiting factor is often indexing. All indexes have to be rebuilt which takes a lot of time and usually also I/O capacity.

Chapter 10: Making Sense of Backups and Replication

- **What is the purpose of the WAL?**

  The purpose of the WAL is to protect your database in case of a crash. The WAL contains all changes happening on the binary level. In case of a crash, it can be used to repair your data files to ensure that data can never be lost.

- **What is a checkpoint?**

  At some point, the WAL has to be recycled. This is exactly what happens during a checkpoint. The idea is to ensure that data is actually stored in the data files so that the *repair instructions* can be removed from the disk. That is exactly what a checkpoint does.

  In PostgreSQL, many performance improvements have been made around checkpoints over the years which greatly benefit the end-user.

- **How does PostgreSQL stream data?**

  If you are using streaming replication data is streamed over a database connection. The way this is done is as follows: The PostgreSQL wire protocol supports a thing called *copy mode*. The transaction log streaming protocol is embedded within this *copy mode* to move data between servers. In other words: A streaming connection is using standard PostgreSQL means to communicate. No additional ports are needed.

- **Is synchronous replication a lot slower than asynchronous replication?**

  Yes. In case of short transactions, it definitely does add some performance penalty. However, there are various flavors of synchronous replication. While `synchronous_commit = remote_apply` might have a real impact a local `synchronous_commit = remote_write` setting for a long transaction might not be too bad. It really depends on network latency, bandwidth, transaction length, and a lot more.

- **Do I need a superuser to replicate data?**

  No. There is no need to be a superuser to stream and replicate data. It is enough if you create a normal user and assign `REPLICATION` permissions. This allows you to set up replicas without granting superuser access from outside.

Chapter 11: Deciding on Useful Extensions

- **Which extensions are the most popular ones?**

There are countless extensions for PostgreSQL out there. A good place to getting started is the PostgreSQL extension network: `https://pgxn.org/`. It contains a variety of modules for PostgreSQL. The most important extensions were covered in this chapter. However, it really depends on your needs what is most important for you.

- **How can I find out more about PostGIS?**

PostGIS is by far the biggest and most comprehensive extension for PostgreSQL. It is the gold standard to handle GIS data in a relational database in the Open Source world. Including extensive coverage of PostGIS in this book is impossible. PostGIS offers thousands of functions and features which cannot be explained in detail easily.

- **Can I write extensions myself?**

PostgreSQL has an easy to use interface to write your own extensions. Code can be written in many different languages including but not limited to SQL, PL/pgSQL, C, and a lot more. If you want to figure out how a server-side extension can be written check out the following website: `https://www.postgresql.org/docs/current/extend-extensions.html`

The official documentation contains a detailed guide to how this works.

Chapter 12: Troubleshooting PostgreSQL

- **What is the most common performance problem?**

  After 20 years of PostgreSQL consulting and support, I can say with confidence that the most common performance problems are caused by missing indexes. In case an important index is missing performance will go down the drain almost instantly assuming the table is large enough.

- **Does PostgreSQL crash often?**

  No, we have rarely seen issues in the field. What happens once in a while is that hardware fails but this is not a PostgreSQL related issue but a more general one. You can perfectly rely on PostgreSQL.

- **How can I find slow queries?**

  There are various ways to find slow queries. The method I like most is to take a closer look at what `pg_stat_statement` has to offer. It keeps track of how often queries have been executed and how long it took to run them. Aggregated and easy to use information is offered which pinpoints expensive queries.

- **How can I monitor my database efficiently?**

  Ideally, you decide on some ready-made tooling to monitor and track PostgreSQL. One of my most beloved tools is pgwatch2 which offers a ready to use Grafana dashboard including countless metrics and ready-made checks. If you are looking for a demo you can check out the following website: `https://demo.pgwatch.com/`.

Chapter 13: Migrating to PostgreSQL

- **Which tools are available to move from Oracle to PostgreSQL?**

  There are various tools out there to move from Oracle to PostgreSQL.

- **What are the main pitfalls when moving to PostgreSQL?**

  It depends on which database engine you have used before. If you are moving from Oracle to PostgreSQL the most tricky thing is certainly NULL handling. Oracle handles NULLs differently than most other databases.

  If you are moving from MS SQL to PostgreSQL the biggest challenge might be related to stored procedures which are quite different from what we know in PostgreSQL.

  Other database engines might pose different challenges in different areas. However, in general, it is totally feasible to move to PostgreSQL in the vast majority of cases.

- **How does performance compare between databases?**

  Database performance is not a *global thing*. It really depends on your workload and your queries. One database engine might be able to execute query X faster while some other database engine might be best for query Y. It really depends on what you are doing. There is no such thing as *always faster*.