PostgreSQL when it's not your job.

Christophe Pettus
PostgreSQL Experts, Inc.
PGCon 2017

Welcome!

- Christophe Pettus
- CEO of PostgreSQL Experts, Inc.
- Based in sunny Alameda, California.
- Technical blog: **thebuild.com**
- Twitter: **@xof**
- **christophe.pettus@pgexperts.com**

PGX
POSTGRESQL
EXPERTS, INC.

What is this?

- "Just enough" PostgreSQL for a developer.
- PostgreSQL is a rich environment.
- Far too much to learn in a single tutorial.
- But enough to be dangerous!

The DevOps World

- "Integration between development and operations."
- "Cross-functional skill sharing."
- "Maximum automation of development and deployment processes."
- "We're way too cheap to hire real operations staff. Anyway: **Cloud!**"

PGX
POSTGRESQL
EXPERTS, INC.

This means...

- No experienced DBA on staff.
  - Have you seen how much those people *cost*, anyway?
- Development staff pressed into duty as database administrators.
- But it's OK… it's **PostgreSQL**!

Everyone Loves PostgreSQL!

- Fully ACID-compliant relational database management system.
- Richest set of features of any modern production RDMS.
- Relentless focus on quality, security, and spec compliance.
- Capable of very high performance.

PGX
POSTGRESQL
EXPERTS, INC.

PostgreSQL Can Do It.

- Tens of thousands of transactions per second.
- Enormous databases (into the petabyte range).
- Supported by pretty much any application stack you can imagine.

Cross-Platform.

- Operates natively on all modern operating systems.
  - Plus Windows.
- Scales from development laptops to huge enterprise clusters.

ASK QUESTIONS.

PGX
POSTGRESQL
EXPERTS, INC.

Installation

PGX
POSTGRESQL
EXPERTS, INC.

If you have packages...

- … use them!
  - Provides platform-specific scripting, etc.
- RedHat-flavor and Debian-flavor have their own repositories.
- Other OSes have a variety of packaging systems.

If you use packages...

- … get them from the community-maintained repos.
- Distros sometimes have older versions.
- apt.postgresql.org for Debian-derived.
- yum.postgresql.org for RedHat-derived.

Or you can build from source.

- Works on any platform.
- Maximum control.
- Requires development tools.
- Does not come with platform-specific utility scripts (/etc/init.d, etc.).
- A few (very rare) config options require rebuilding.

Other OSes.

- Windows: One-click installer available.
- OS X: One-click installer, MacPorts, Fink and Postgres.app from Heroku.
- For other OSes, check postgresql.org.

Creating a database cluster.

- A single PostgreSQL server can manage multiple databases.
- The whole group on a single server is called a “cluster”.
- This is very confusing, yes. We’ll use the term “server” here.

initdb

- The command to create a new database is called initdb.
- It creates the files that will hold the database.
- It doesn't automatically start the server.
- Many packaging systems automatically create and start the server for you.

Note on Debian/Ubuntu

- Debian-style packaging has a sophisticated cluster management system.
- Use it! It will make your life much easier.
- pg_createcluster instead of initdb

Just Do This.

- Always create databases as UTF-8.
  - Once created, cannot be changed.
  - Converting from "SQL ASCII" to a real encoding is a total nightmare.
- Use your favorite locale, but not "C locale."
- UTF-8 and system locale are the defaults.

Checksums.

- Introduced in 9.3.
- Maintains a checksum for data pages.
- Very small performance hit. Use it.
- initdb option.
- Can add in /etc/postgresql-common/createcluster.conf for Debian packaging.

Examples

- Using initdb:
  - `initdb -D /data/9.5/ -k -E UTF8 \ --locale=en_US.UTF-8`
- Using pg_createcluster:
  - `pg_createcluster 9.5 main -D /data/9.5/main \ -E UTF8 --locale=en_US.UTF-8 -- -k`

Other Important Things.

- Create a separate database volume / partition for data.
- Do not put the version number in the mountpoint (/data, not /data/9.5).
- EXT4 or XFS for the filesystem (ZFS is extra for experts).

PGX
POSTGRESQL
EXPERTS, INC.

pg_ctl

- Built-in command to start and stop PostgreSQL.
- Frequently called by init.d, upstart or other scripts.
- Use the package-provided scripts if they exist; they do the right thing.

Stopping PostgreSQL.

- Three “shutdown modes”: smart, fast, immediate. -m option on pg_ctl
- Don’t use smart. It’s not really that smart.
- Use fast (cancels queries, does shutdown).
- Use immediate if required.
  - immediate crashes PostgreSQL!

PGX
POSTGRESQL
EXPERTS, INC.

psql

- Command-line interface to PostgreSQL.
- Run queries, examine the schema, look at PostgreSQL's various views.
- Get friendly with it! It's very useful for doing quick checks.

PostgreSQL directories

- All of the data lives under a top-level directory.
- Let's call it $PGDATA.
- Find it on your system, and do a ls.
- The data lives in "base".
- The transaction logs live in pg_xlog.

NEVER EVER TOUCH THESE THINGS!

- The contents of subdirectories and special files in $PGDATA should never, ever be modified directly. Ever.
- Exceptions: pg_log (if you put the log files there), and the configuration files.
- pg_xlog and pg_clog are off-limits!

Tablespaces

- A quick note on tablespaces.
- Don’t use them.
- Extra for experts: Use them if you have unusual storage configuration, but they will make your life more complex.
  - NEVER put the tablespace storage inside $PGDATA.

Configuration files.

- On most installations, the configuration files live in $PGDATA.
- On Debian-derived systems, they live in /etc/postgresql/9.6/main/...
- Find them. You should see:
  - postgresql.conf
  - pg_hba.conf

Configuration

PGX
POSTGRESQL
EXPERTS, INC.

Configuration files.

- Only two really matter:
  - postgresql.conf — most server settings.
  - pg_hba.conf — who gets to log in to what databases?

postgresql.conf

- Holds all of the configuration parameters for the server.
- Find it and open it up on your system.

We're All Going To Die.

PGX
POSTGRESQL
EXPERTS, INC.

PGX
POSTGRESQL
EXPERTS, INC.

Important parameters.

- Logging.
- Memory.
- Checkpoints.
- Planner.
- You're done.
- No, really, you're done!

Logging.

- Be generous with logging; it's very low-impact on the system.
- It's your best source of information for finding performance problems.

Where to log?

- syslog — If you have a syslog infrastructure you like already.
- Otherwise, CSV format to files.
- "Standard format" or "stderr" is obsolete. There is no good reason to use it anymore.

What to log?

```
log_destination = 'csvlog'
log_directory = 'pg_log'
logging_collector = on
log_filename = 'postgres-%Y-%m-%d_%H%M%S'
log_rotation_age = 1d
log_rotation_size = 1GB
log_min_duration_statement = 250ms
log_checkpoints = on
log_connections = on
log_disconnections = on
log_lock_waits = on
log_temp_files = 0
```

Memory configuration

- shared_buffers
- work_mem
- maintenance_work_mem

shared_buffers

- Below 2GB (?), set to 20% of total system memory.
- Below 64GB, set to 25% of total system memory.
- Above 64GB (lucky you!), set to 16GB.
- Done.

work_mem

- Start low: 32-64MB.
- Look for ‘temporary file’ lines in logs.
- Set to 2-3x the largest temp file you see.
- Can cause a **huge** speed-up if set properly!
- But be careful: It can use that amount of memory per planner node.

maintenance_work_mem

- 10% of system memory, up to 1 GB.
- Maybe even higher if you are having VACUUM problems.
- (We'll talk about VACUUM later.)

effective_cache_size

- Set to the amount of file system cache available.
- If you don't know, set it to 75% of total system memory.
- And you're done with memory settings.

Checkpoints.

- A complete flush of dirty buffers to disk.
- Potentially a lot of I/O.
- Done when the first of two thresholds are hit:
  - A particular number of WAL segments have been written.
  - A timeout occurs.

Checkpoint settings, 9.4 and earlier.

```
wal_buffers = 16MB

checkpoint_completion_target = 0.9

checkpoint_timeout = 10m-30m # Depends on restart time

checkpoint_segments = 32 # To start.
```

Checkpoint settings, 9.5 and later.

```
wal_buffers = 16MB

checkpoint_completion_target = 0.9

checkpoint_timeout = 10m-30m # Depends on restart time

min_wal_size = 512MB

max_wal_size = 2GB
```

Checkpoint settings, 9.4 and earlier.

- Look for checkpoint entries in the logs.
- Happening more often than checkpoint_timeout?
  - Adjust checkpoint_segments so that checkpoints happen due to timeouts rather filling segments.
- And you're done with checkpoint settings.

PGX
POSTGRESQL
EXPERTS, INC.

Checkpoint settings, 9.5 and later

- Look for checkpoint entries in the logs.
- Happening more often than checkpoint_timeout?
- Step 1: Adjust min_wal_size so that checkpoints happen due to timeouts rather filling segments.
- More will improve performance.

Checkpoint settings, 9.5 and later

- Step 2: Adjust max_wal_size to be about three times min_wal_size.
  - More will improve performance.
- And you're done with checkpoint settings.

Checkpoint settings notes.

- Pre-9.5, the WAL can take up to 3 x 16MB x checkpoint_segments on disk.
- 9.5+, the WAL varies between min_wal_size and max_wal_size.
- Restarting PostgreSQL *from a crash* can take up to checkpoint_timeout (but usually much less).

Planner settings.

- effective_io_concurrency — Set to the number of I/O channels; otherwise, ignore it.
- random_page_cost — 3.0 for a typical RAID10 array, 2.0 for a SAN, 1.1 for Amazon EBS.
- And you're done with planner settings.

Do not touch.

- fsync = on
  - Never change this.
- synchronous_commit = on
  - Change this, but only if you understand the data loss potential.

Changing settings.

- Most settings just require a server reload to take effect.
- Some require a full server restart (such as shared_buffers).
- Many can be set on a per-session basis!

pg_hba.conf

PGX
POSTGRESQL
EXPERTS, INC.

Users and roles.

- A "role" is a database object that can own other objects (tables, etc.), and that has privileges (able to write to a table).
- A "user" is just a role that can log into the system; otherwise, they're synonyms.
- PostgreSQL's security system is based around users.

Basic user management.

- Don't use the "postgres" superuser for anything application-related.
- Sadly, you probably will have to grant schema-modifications privileges to your application user, if you use migrations.
- If you don't have to, don't.

User security.

- By default, database traffic is not encrypted.
- Turn on ssl if you are running in a cloud provider.
- For pre-9.4, set ssl_renegotiation_limit = 0.

pg_hba.conf basics.

- Don't ever expose port 5432 to the public internet.
- Don't ever use trust authentication.
- On a cloud hosting environment, use SSL always.

The WAL.

PGX
POSTGRESQL
EXPERTS, INC.

Why are we talking about this now?

- The Write-Ahead Log is key to many PostgreSQL operations.
- Replication, crash recovery, etc., etc.
- Don’t worry (too much!) about the internals.

The Basics.

- When each transaction is committed, it is logged to the write-ahead log.
- The changes in that transaction are flushed to disk.
- If the system crashes, the WAL is "replayed" to bring the database to a consistent state.

A continuous record of changes.

- The WAL is a continuous record of changes since the last checkpoint.
- Thus, if you have the disk image of the database, and every WAL record since that was created…
- … you can recreate the database to the end of the WAL.

pg_xlog

- The WAL is stored in 16MB segments in the pg_xlog directory.
- Don't mess with it! Never delete anything out of it!
- Records are automatically recycled when they are no longer required.

On a crash...

- When PostgreSQL restarts, it replays the WAL log to bring itself back to a consistent state.
- The WAL segments are essential to proper crash recovery.
- The longer since the last checkpoint, the more WAL it has to process.

sychronous_commit

- When "on", COMMIT does not return until the WAL flush is *done*.
- When "off", COMMIT returns when the WAL flush is *queued*.
- Thus, you might lose transactions on a crash.
- No danger of database corruption.

Backup and Recovery

PGX
POSTGRESQL
EXPERTS, INC.

pg_dump

- Built-in dump/restore tool.
- Takes a logical snapshot of the database.
- Does not lock the database or prevent writes to disk.
- Low (but not zero) load on the database.

pg_restore

- Restores database from a pg_dump.
- Is not a fast operation.
- Great for simple backups, not suitable for fast recovery from major failures.

pg_dump / pg_restore advice

- Back up globals with pg_dumpall --globals-only.
- Back up each database with pg_dump using --format=custom.
- This allows for a parallel restore using pg_restore.

pg_restore

- Restore using --jobs=<# of cores + 1>.
- Most of the time in a restore is spent rebuilding indexes; this will parallelize that operation.
- Restores are not fast.

PITR backup / recovery

- Remember the WAL?
- If you take a snapshot of the data directory…
- … it won't be consistent, but if we add the WAL records…
- … we can bring it back to consistency.

WAL archiving.

- archive_command
- Runs a command each time a WAL segment is complete.
- This command can do whatever you want.
- What you want is to move the WAL segment to someplace safe…
  - … on a different system.

Getting started with PITR.

- Decide where the WAL segments and the backups will live.
- Configure archive_command properly to do the copying.
- Always use rsync (or a dedicated tool) to do the copy, not scp.

Creating a PITR backup.

- SELECT pg_start_backup(...);
- Copy the disk image and any WAL files that are created.
- SELECT pg_stop_backup();
- Make sure you have all the WAL segments.
- The disk image + WAL segments are your backup.

PGX
POSTGRESQL
EXPERTS, INC.

WAL-E

- http://github.com/wal-e/wal-e
- Provides a full set of appropriate scripting.
- Automates create PITR backups into AWS S3.
- Highly recommended!

PITR Restore

- Copy the disk image back to where you need it.
- Set up recovery.conf to point to where the WAL files are.
- Start up PostgreSQL, and let it recover.

How long will this take?

- The more WAL files, the longer it will take.
- Generally takes 10-20% of the time it took to create the WAL files in the first place.
- More frequent snapshots = faster recovery time.

“PITR”?

- Point-in-time recovery.
- You don’t have to replay the entire WAL stream.
- It can be stopped at a particular timestamp, or transaction ID.
- Very handy for application-level problems!

PGX
POSTGRESQL
EXPERTS, INC.

Disaster recovery.

- Always have a disaster recovery strategy.
- What if you data center / AWS region goes down?
- Have a plan for recovery from a remote site.
- WAL archiving is a great way to handle this.

pg_basebackup

- Utility for doing a snapshot of a running server.
- Easiest way to take a snapshot to start a new secondary.
- Can also be used as an archival backup.

Backup Notes.

- Always test your backups. Always, always, always.
  - Give them to developers to prime their dev systems.
- Do not backup to mounted network (NFS, etc.) shares.

Packaged Solutions

- WAL-E
- repmgr
- barman
- backrest
- Use a packaged solution; don't roll your own unless you must.

Replication!

PGX
POSTGRESQL
EXPERTS, INC.

Replication.

- If you are serious about your data, you need a replica.
- In general, you want binary replication, at least to start.
- But there are other kinds of PostgreSQL replication.

Hmm... what if we...

- … transmitted the WAL changes directly to the secondary without having to ship the file?
- Great idea!
- Such a great idea, PostgreSQL implements it!
- That's what Binary Replication is.

PGX
POSTGRESQL
EXPERTS, INC.

Binary Replication Basics.

- The secondary connects via a standard PostgreSQL connection to the primary.
- As changes happen on the primary, they are sent down to the secondary.
- The secondary applies them to its local copy of the database.

recovery.conf

- All replication is orchestrated through the recovery.conf file.
- Always lives in your $PGDATA directory.
- Controls how to connect to the primary, how far to recover (for PITR), etc., etc.
- Also used if you are bringing the server up as a PITR recovery instead of replication.

Binary Replication, the good.

- Easy to set up.
- Schema changes are automatically replicated.
- Secondary can be used to handle read-only queries for load balancing.
- Very few gotchas; it either works or it doesn't, and it is vocal about not working.

Binary Replication, the bad.

- Entire database or none of it.
- No writes of any kind to the secondary.
  - This includes temporary tables.
- Some things aren't replicated.
  - Temporary tables, unlogged tables, hash indexes.

Advice?

- Start with WAL-E.
  - The README tells you everything you need to know.
- Handles a very large number of complex replication problems easily.
- As you scale out of it, you'll have the relevant experience.

PGX
POSTGRESQL
EXPERTS, INC.

Trigger-based replication

- Installs triggers on tables on master.
- A daemon process picks up the changes and applies them to the secondaries.
- Third-party add-ons to PostgreSQL.

Trigger-based rep: Good.

- Highly configurable.
- Can push part or all of the tables; don't have to replicate everything.
- Multi-master setups possible (Bucardo).

Trigger-based rep: The bad.

- Fiddly and complex to set up.
- Schema changes must be pushed out manually.
- Imposes overhead on the master.

Built-In Logical Replication

- Brand new as of 9.4.
- Decodes the WAL stream back into SQL level changes.
- Plug-ins decode the changes.
- Ships with (and RDS supports) a toy demo plug-in.

pg_logical

- Builds PostgreSQL-to-PostgreSQL logical replication on top of logical decoding.
- Functional, flexible.
- Not part of the core distribution, and does impose some limitations on operations.

PGX
POSTGRESQL
EXPERTS, INC.

Coming in Version 10!

- In-core logical replication!
- Roughly equivalent to pg_logical, but some significant differences and limitations.

Transactions, MVCC and VACUUM

PGX
POSTGRESQL
EXPERTS, INC.

“Transaction”

- A unit of which which must be:
  - Applied atomically to the database.
  - Invisible to other database clients until it is committed.

The Classic Example.

```
BEGIN;
INSERT INTO transactions(account_id, value, offset_id)
   VALUES (11, 120.00, 14);
INSERT INTO transactions(account_id, value, offset_id)
   VALUES (14, -120.00, 11);
COMMIT;
```

Transaction Properties.

- Once the COMMIT completes, the data has been written to permanent storage.
- If a database crash occurs, any transactions will be COMMITed or not; no half-done transactions.
- No transaction can (directly) see another transaction in progress.

In PostgreSQL...

- Everything runs inside of a transaction.
- If no explicit transaction, each statement is wrapped in one for you.
  - This has certain consequences for database-modifying functions.
- Everything that modifies the database is transactional, even schema changes.

A brief warning...

- Many resources are held until the end of a transaction.
  - Temporary tables, working memory, locks, etc.
- Keep transactions brief and to the point.
- Be aware of IDLE IN TRANSACTION sessions.

Transaction would be easy...

- … if databases were single user.
- They're not.
  - Thank goodness.
- So, how do we handle concurrency control when two sessions are trying to use the same data?

The Problem.

- Process 1 begins a transaction.
- Process 2 begins a transaction.
- Process 1 updates a tuple.
- Process 2 tries to read that tuple.
- What happens?

Bad Things.

- Process 2 can't get the new version of the tuple (ACID [generally] prohibits dirty reads).
- But where does it get the old version of the tuple from?
  - Memory? Disk? Special roll-back area?
  - What if we touch 250,000,000 rows?

Some Approaches.

- Lock the whole database.
- Lock the whole table.
- Lock that particular tuple.
- Reconstruct the old state from a rollback area.
- None of these are particularly satisfactory.

Multi-Version Concurrency Control.

- Create multiple "versions" of the database.
- Each transaction sees its own "version."
  - We call these "snapshots" in PostgreSQL.
- Each snapshot is a first-class member of the database.
  - There is no privileged "real" snapshot.

The Implications.

- Readers do not block readers.
- Readers do not block writers.
- Writers do not block readers.
- Writers only block writers to the same tuple.

Snapshots.

- Each transaction maintains its own snapshot of the database.
- This snapshot is created when a statement or transaction starts (depending on the transaction isolation mode).
- The client only sees the changes in its own snapshot.

Nothing's Perfect.

- PostgreSQL will not allow two snapshots to "fork" the database.
- If this happens, it resolves the conflict with locking or with an error, depending on the isolation mode.
- Example: Two separate clients attempt to update the same tuple.

Isolation Modes.

- PostgreSQL supports:
  - READ COMMITTED — The default.
  - REPEATABLE READ
  - SERIALIZABLE
- It does not support:
  - READ UNCOMMITTED (“dirty read”)

Higher isolation modes.

- REPEATABLE READ and SERIALIZABLE take the snapshot when the transaction begins.
- Snapshot lasts until the end.
- An attempt to modify a tuple another transaction has changed blocks…
  - … and returns an error if that transaction commits.

MVCC consequences.

- Deleted tuples are not (usually) immediately freed.
  - Tuples on disk might not be available to be readily checked.
- This results in dead tuples in the database.
- Which means: VACUUM!

PGX
POSTGRESQL
EXPERTS, INC.

VACUUM

- VACUUM's primary job is to scavenge tuples that are no longer visible to any transaction.
- They are returned to the free space for reuse.
- autovacuum generally handles this problem for you without intervention.

ANALYZE

- The planner requires statistics on each table to make good guesses for how to execute queries.
- ANALYZE collects these statistics.
- Done as part of VACUUM.
- Always do it after major database changes — especially a restore from a backup.

“Vacuum’s not working.”

- It probably is.
- The database generally stabilize at 20% to 50% bloat. That’s acceptable.
- If you see autovacuum workers running, that’s generally not a problem.

“No, really, VACUUMs not working!”

- Long-running transactions, or “idle-in-transaction” sessions?
- Manual table locking?
- Very high write-rate tables?
- Many, many tables (10,000+)?

PGX
POSTGRESQL
EXPERTS, INC.

Unclogging the VACUUM.

- Reduce the autovacuum sleep time.
- Increase the number of autovacuum workers.
- Do low period manual VACUUMs.
- Fix IIT sessions, long transactions, manual locking.

Excessive VACUUM Load.

- "It's never twins, it's never lupus, and it's never autovacuum."
- Autovacuum is rarely the culprit.
- Diagnosis: Turn off autovacuum (temporarily! never permanently!) to see if that unloads the I/O subsystem.

Adjusting Vacuum.

- The first and safest way to "lighten" autovacuum is to reduce autovacuum_vacuum_cost_delay.
- Default 20ms, start by turning down to 100ms.

VACUUM FREEZE

- Details are tedious, but:
- A periodic "major" vacuum that PostgreSQL must perform to prevent transaction ID wraparound.
- Generally, not a problem, but for high-update rate, large databases, can be a I/O issue.

Avoiding VACUUM FREEZE problems.

- Do a manual VACUUM FREEZE at low-load periods.
- Every 1-4 months depending on transaction load.
- Can use the built-in vacuumdb tool:
  - `vacuumdb --all --freeze --analyze`

Or, upgrade to 9.6!

- Significant improvements in VACUUM FREEZE in 9.6.
- Upgrade if you possibly can.

PGX
POSTGRESQL
EXPERTS, INC.

Schema Design Notes.

PGX
POSTGRESQL
EXPERTS, INC.

A grab-bag of notes.

- Schema design is a deep topic.
- This is just a quick set of random important things.

NULL

- NULL is a total pain in the neck.
- Sometimes, you have to deal with NULL, but:
- Only use it to mean “missing value.”
- Never, ever have it as a meaningful value in a key field.
- WHERE NOT IN (SELECT ...)

JSON.

- It's a core type.
  - Not a contrib/ or extension module.
- Introduced in 9.2.
- Enhanced in 9.3.
- And really enhanced in 9.4.

We liked JSON so much...

- … we created two types.
  - json
  - jsonb
- json is a pure text representation.
- jsonb is a parsed binary representation.
- Each can be cast to the other, of course.

json type.

- Stores the actual json text.
- Whitespace included.
- What you get out is what you put in.
- Checked for correctness, but not otherwise processed.

Why use json?

- You are storing the json and never processing it.
- You need to support two JSON "features":
  - Order-preserved fields in objects.
  - Duplicate keys in objects.
- For some reason, you need the *exact* JSON text back out.

Oh, and...

- jsonb wasn't introduced until 9.4.
- So, if you are on 9.2-9.3, json is what you've got.
- Otherwise, you want to use jsonb.

jsonb

- Parsed and encoded on the way in.
- Stored in a compact, parsed format.
- Considerably more operator and function support.
- Has indexing support.

Very Large Objects

- Let's say 1MB or more.
- Store them in files, store metadata in the database.
- The database API is not designed for passing large objects around.

Many-to-Many Tables

- These can get extremely large.
- Consider replacing with array fields.
  - Either one way, or both directions.
- Can use a trigger to maintain integrity.
- Much smaller and more efficient.
- Depends, of course, on usage model.

Character Encoding.

- Use UTF-8.
- Just. Do. It.
- There is no compelling reason to use any other character encoding.
  - One edge case: the bottleneck is sorting text strings. This is very, very rare.

Time Representation.

- Always use TIMESTAMPTZ.
  - TIMESTAMP is a bad idea.
  - TIMESTAMPTZ is “timestamp, converted to UTC.”
  - TIMESTAMP is “timestamp, at some time zone but we don’t know which one, hope you do.”

Indexing

PGX
POSTGRESQL
EXPERTS, INC.

Test your database knowledge!

What does the SQL standard require for indexes?

Trick Question!

PGX
POSTGRESQL
EXPERTS, INC.

It doesn't.

- The database should work identically whether or not you have indexes.
- Of course, "identically" in this case does not mean "just as fast."
- No real-life database can work properly without indexes.

PostgreSQL Index Types.

- B-Tree.
- ~~Hash~~.
- GiST.
- ~~SP-GiST~~.
- GIN.

PGX
POSTGRESQL
EXPERTS, INC.

B-Tree Indexes.

- The standard PostgreSQL index is a B-tree.
- Provides O(log N) access to leaf notes.
- Provides total ordering.
- Operates on scalar values that implement standard comparison operators.

B-Tree Index Types.

- Single column.
- Multiple column (composite).
- Expression (“functional”) indexes.

Single Column B-Trees

- The simplest index type.
- Can be used to optimize searches on <, <=, =, >=, >.
- Can be used to retrieve rows in sorted order on that column.

When to create?

- If a query uses that column, and…
  - … uses one of the comparison operators.
    - … and selects <10-15% of the rows.
    - … and is run frequently.
- … the index will likely be helpful.

Indexes and JOINs

- Indexes can accelerate JOINs considerably.
- But the usual rules apply.
- Generally, they help the most when indexing the key on the larger table and…
- … that results in high selectivity against the smaller table.

Indexes and Aggregates.

- Some GROUP BY and related operations can benefit from an index.
- Often only in the presence of a HAVING clause, though.
- If it has to scan the whole index, it might as well scan the whole table.

Mandatory indexes.

- Constraints must have indexes to enforce them.
- Just accept those.

Ascending vs Descending?

- By default, B-trees index in ascending order.
- Descending indexes are much faster in retrieving tuples in descending order.
- So, if the primary function is descending sortation, use that.
- Otherwise, just use ascending order.

Composite Indexes.

- A single index can have multiple columns.
- The columns must be used left-to-right.
- An index on (A, B, C) does not help a query on just C.
- But it does on (A, B).

Expression Indexes.

- Indexes on an expression.
- PostgreSQL can recognize when you are querying on that expression and use the index.
- Can be expensive to create, but very fast to execute.
- Make sure PostgreSQL is really using it!

Partial Indexes.

- An index does not have to contain all of the rows of the table.
- The WHEN clause's boolean predicate limits the size of the index.
- This can be a huge performance improvement for queries that match the predicate, all or in part.

Indexes and MVCC

- The full key value is copied into the index.
- Every version of the tuple on the disk appears in the index.
- Thus, PostgreSQL needs to check whether a retrieved tuple is live.
- This means indexes can bloat as dead tuples pile up.

GiST Indexes.

- GiST is not a single index type, but an index framework.
- It can be used to create B-tree-style indexes.
- It can also be used to create other index types, like bounding-box and geometric queries.

GiST Index Usage.

- Non-total-ordered types generally require a GIST index.
- Each type's index implementation decides what operators to support.
  - Inclusion, membership, intersection…
- Some GiST indexes do provide ordering.
  - KNN indexes, for example.

GIN

- Generalized Inverted Index.
- Maps index items (words, dict keys) to rows whose field contains those.
- Core PostgreSQL use: Full text search indexes.
  - Maps tokenized words to the rows containing those words.

GIN implementation

- A B-tree of B-trees.
- Tokens organized into B-trees.
- Row pointers also organized into B-trees.
- On-disk footprint can be quite large.
  - Recent versions have major optimizations here.

“Why isn’t it using my indexes?”

- The most common complaint.
- First, get the EXPLAIN ANALYZE output of the query.
- Sometimes, it is using the index, and it’s just slow anyway!

Bad Selectivity.

- If PostgreSQL thinks that the index scan will return a large percentage of the table, it will do a seq scan instead.
- Generally, it's right to think this.
- If it's wrong, and the query is very selective, try re-running ANALYZE.

ANALYZE didn't help.

- Try running the query with:
  - SET enable_seqscan = 'off';
  - See how long it takes to use the index then.
    - PostgreSQL might be right.
- Hey, it didn't use the index even then!

Index Prohibitorum

- This means PostgreSQL thinks that index doesn't apply to this query.
- Query mis-written? Index invalid? Confusing expression index?
- Try doing a very simple query on just that field, and build up.

PostgreSQL is right, but wrong.

- In fact, using the index is faster even though PostgreSQL thinks it is not.
- Try lowering random_page_cost.
- Consider changing the default statistics target for that field.

Index Creation.

- Two ways of creating an index:
  - CREATE INDEX
  - CREATE INDEX CONCURRENTLY

CREATE INDEX

- Does a single scan of the table, building the index.
- Uses maintenance_work_mem to do the creation.
- Keeps an exclusive lock on the table while the index build is going on.

CREATE INDEX CONCURRENTLY

- Does two passes over the table:
  - Builds the index.
  - Validates the index.
- If the validation fails, the index is marked as invalid and won't be used.
- Drop it, run again.

REINDEX

- Rebuilds an existing index from scratch.
- Takes an exclusive lock on the table.
- Generally no need to do this unless an index has gotten badly bloated.

Index Bloat.

- Over time, B-tree indexes can become bloated.
- Sparse deletions from within the index range are the usual cause.
  - http://pgsql.tapoueh.org/site/html/news/20080131.bloat.html
- Generally, don't worry about it.

Index Usage.

- pg_stat_user_indexes
- Reports the number of times an index is used.
- If non-constraint indexes are not being used, drop them.
- Indexes are very expensive to maintain.

And finally...

- … don't create indexes on columns prospectively.
- Only create an index in response to a particular query problem.
- It's easy to over-index a database!

Special Situations.

PGX
POSTGRESQL
EXPERTS, INC.

Minor version upgrade.

- Do this promptly!
- Only requires installing new binaries.
- If using packages, often as easy as just an apt-get / yum upgrade.
- Very small amount of downtime.

Major version upgrade.

- Requires a bit more planning.
- pg_upgrade is now reliable.
- Trigger-based replication is another option for zero downtime.
- A full pg_dump / pg_restore is always safest, if practical.
- Always read the release notes!

Don’t get caught!

- Major versions are EOLd after 5 years.
  - 9.2 support ends September 2017.
- Always have a plan for how you are going to move between major versions.
- All parts of a replication set must be upgraded at once (for major versions).

Bulk loading data.

- Use COPY, not INSERT.
- COPY does full integrity checking and trigger processing.
- Do a VACUUM ANALYZE afterwards.

Very high insert rates.

- Reduce shared buffers by 25%-75%.
- Reduce checkpoint timeouts to 3min or less.
- Make sure to do enough ANALYZEs to keep the statistics up to date, manual if required.

PGX
POSTGRESQL
EXPERTS, INC.

AWS

- Generally, works like any other system.
- Remember that instances can disappear and come back up without instance storage.
- Always have a good backup / replication implementation on AWS!
- PIOPS are useful (but pricey) if you are using EBS.

Larger-Scale AWS Deployments

- Script everything: Instance creation, PostgreSQL setup, etc.
- Put everything inside a VPC.
- Scale up and down as required to meet load.
  - AWS is a very expensive equipment rental service.

PostgreSQL RDS

- Overall, not a bad product.
- BIG plus: Automatic failover.
- BIG minus: Bad performance relative to bare EC2, often mysterious.
- Other minuses: Expensive, fixed (although large) set of extensions.
- Not a bad place to start with PostgreSQL.

Tools

PGX
POSTGRESQL
EXPERTS, INC.

Log Analysis

- pgbadger
  - The only choice now for monitoring text logs.
- pg_stat_statements
  - Maintains a buffer of data on statements executed, within PostgreSQL.

Monitor, monitor, monitor.

- Use Nagios / Ganglia to monitor:
  - Disk space — at minimum.
  - CPU usage
  - Memory usage
  - Replication lag.
- check_postgres.pl (bucardo.org)

Lots of Tools Out There.

- DataDog, VividCortex, New Relic.
- CloudWatch (if you're on AWS).
- pganalyze
- A profusion of vendor-based tools.

Check out...

- https://wiki.postgresql.org/images/6/6a/Dba_toolbelt_2017.pdf

Thank you!

thebuild.com / @xof / pgexperts.com