Essential Extensions for DBAs

Performance Optimization and Tuning

- **HypoPG**: Test the performance impact of potential indexes **without actually creating them**. This helps evaluate improvements before consuming disk space.
- **pg_hint_plan**: Influence the PostgreSQL query planner by adding **optimizer hints** directly in SQL comments.
- **pg_repack & pg_squeeze**: **Manage table bloat** and reorganize data without long-term locks, recovering space and improving query efficiency.
- **pg_partman**: **Automate partitioning** for large datasets, whether based on time or serial numbers. It simplifies the maintenance of partitions.

Installation: Make sure you install the contrib packages. After that, most of the packages are already included in the contrib. Below is an example for installing one, and you can install others according to your needs.

```
Example
-------
dnf install -y postgresql17-contrib

dnf install ddlx_17

CREATE EXTENSION ddlx;
```

Monitoring, Logging, and Diagnostics

- **pg_stat_kcache & pg_profile**: Capture **kernel-level I/O statistics** and generate detailed performance reports to identify bottlenecks.
- **Installation**: `CREATE EXTENSION pg_stat_kcache;`

- **pg_qualstats**: **Track query predicates** and optimize indexing strategies based on predicate usage.
- **Installation**: `CREATE EXTENSION pg_qualstats;`
- **pg_wait_sampling**: Identify **real-time wait events** (locks, I/O issues) causing delays and pinpoint the bottlenecks in your system.
- **Installation**: `CREATE EXTENSION pg_wait_sampling;`
- **logerrors**: **Aggregate and analyze log file messages** to troubleshoot proactively and detect issues early.
- **Installation**: `CREATE EXTENSION logerrors;`

Replication and High Availability

- **pg_ddl_deploy**: **Streamline DDL replication** across replicas, ensuring schema changes propagate seamlessly in logical replication setups.
- **Installation**: `CREATE EXTENSION pg_ddl_deploy;`
- **pg_failover_slots**: Safeguard **logical replication slots** during physical failovers, preventing data loss in case of replication failures.
- **Installation**: `CREATE EXTENSION pg_failover_slots;`
- **pg_dirtyread**: **Recover data from dead rows**, useful for recovering from accidental deletions or corruption.
- **Installation**: `CREATE EXTENSION pg_dirtyread;`

Security and Access Control

- **login_hook**: Execute **custom security code on user login**, similar to Oracle's `AFTER LOGON` triggers. This is ideal for auditing and enforcing security policies.
- **Installation**: `CREATE EXTENSION login_hook;`

- **pg_readonly**: Lock the database into **read-only mode** to prevent accidental writes, perfect for maintenance or when enforcing strict access controls.
- **Installation**: `CREATE EXTENSION pg_readonly;`

Job Scheduling and Automation

- **pg_cron**: Schedule periodic tasks (e.g., **vacuuming**, **reporting**, **backups**) directly within PostgreSQL using **cron-style syntax**.
- **Installation**: `CREATE EXTENSION pg_cron;`
- **pgagent**: A more advanced job scheduler, managed via **pgAdmin**, that supports multi-step workflows, error handling, and logging.
- **Installation**: `CREATE EXTENSION pgagent;`

System Insights and Utilities

- **pgmeminfo & system_stats**: Monitor **system-level metrics** such as memory usage, CPU, disk, and network stats directly from SQL.
- **Installation**: `CREATE EXTENSION pgmeminfo;`
- **sslutils**: Generate **SSL certificates on the fly** to secure PostgreSQL connections.
- **Installation**: `CREATE EXTENSION sslutils;`
- **pgsql_tweaks**: A **Swiss army knife** of helper functions for daily DBA tasks, making routine operations easier.
- **Installation**: `CREATE EXTENSION pgsql_tweaks;`

Documentation and Metadata

- **ddlx**: Automatically generate **DDL scripts** for database objects to ensure schema consistency and easy sharing of database structures.

- **Installation**: `CREATE EXTENSION ddlx;`
- **pg_readme**: Automatically generate **README.md documentation** for schemas or extensions, streamlining documentation-driven workflows.
- **Installation**: `CREATE EXTENSION pg_readme;`

Why Extensions Matter

Extensions enable PostgreSQL to evolve from a simple relational database into a full-fledged platform. By **integrating specialized tools directly within the database**, you eliminate the need for external applications or workarounds. For example, **pg_cron** replaces the need for external cron jobs, while **pg_partman** automates partitioning without relying on external scripts.

1. **Installation**: Extensions are installed using the `CREATE EXTENSION <extension_name>;` command. Make sure to add the extension to `shared_preload_libraries` in your `postgresql.conf` file.
2. **Compatibility**: Always **check compatibility** with your PostgreSQL version and thoroughly **test in a staging environment** before deploying to production.