PostgreSQL Database Maintenance Operations

1. VACUUM Operations (vacuum, vacuum full)

Regularly performed maintenance operation in PostgreSQL. This operation:

- Reclaims disk space occupied by updated and deleted rows.
- Updates statistics used by the query planner.
- Updates Visibility Map files, allowing faster data access.

Autovacuum

The vacuum operation can be automated with autovacuum. Autovacuum regularly examines statistics and detects tables with a large number of dead rows. It creates workers within the limits of the `autovacuum_max_workers` parameter to perform maintenance on these tables. These workers apply VACUUM and ANALYZE operations to the necessary tables. If the number of workers is insufficient for all tasks, the tables are queued.

VACUUM FULL

- Uses a more effective cleaning algorithm compared to normal VACUUM.
- Rebuilds tables in a new copy, using more disk space temporarily. Once the operation is complete, this space is released.
- Locks the table and prevents access until the operation is finished.
- Useful for tables experiencing significant data changes.

VACUUM Examples

```
VACUUM; -- no locks, removes dead rows and marks for reuse
VACUUM FULL; -- locks, compacts, more time
VACUUM FULL VERBOSE emp;

-- \h vacuum (check detail vacum options)
VACUUM (index cleanup true, verbose true, analyze true) customers;
VACUUM (full true, index_cleanup true, verbose true, analyze true) customers;
VACUUM (index cleanup true, verbose true, analyze true, parallel 4) customers; -- with full
no parallel
```

Monitoring Vacuum Status

```
SELECT * FROM pg_stat_progress_vacuum;
```

2. CLUSTER Operations

The CLUSTER command physically reorganizes a table according to an index. Subsequent data entries may not follow this order, and the operation locks the table, preventing access until complete.

CLUSTER Examples

```
create table testcluster (id int, name text);
CREATE TABLE

insert into testcluster values(2,'B'),(1,'A'),(3,'A'),(5,'C'),(4,'B');
INSERT 0 5

select * from testcluster;
 id | name
----+------
  2 | B
  1 | A
  3 | A
  5 | C
  4 | B
(5 rows)

create index idx_id on testcluster(id);
CREATE INDEX

postgres=# cluster testcluster using idx_id;
CLUSTER

postgres=# select * from testcluster;
 id | name
----+------
  1 | A
```

```
  2 | B
  3 | A
  4 | B
  5 | C
(5 rows)
```

Monitoring CLUSTER Status

```
SELECT * FROM pg_stat_progress_cluster;
```

3. REINDEX Operations

REINDEX rebuilds and updates old indexes. It is useful in the following situations:

- If index data is corrupted or damaged.
- To clean up indexes with too many invalid and empty entries.
- To apply parameter changes related to the index.
- To correct a previously failed index operation.

REINDEX Examples

```
REINDEX INDEX index_name;
REINDEX TABLE table_name;
REINDEX SCHEMA schema name;
REINDEX DATABASE db_name;
REINDEX TABLE CONCURRENTLY cs agreement;
REINDEX (verbose, CONCURRENTLY, TABLESPACE pg_default) DATABASE db_name;
REINDEX (verbose) SYSTEM;
```

4. Updating Statistics (ANALYZE)

As data in tables changes, these statistics can become outdated. The ANALYZE command is used to update these statistics.

ANALYZE Examples

```
ANALYZE table_name;
ANALYZE;
ANALYZE VERBOSE;
```

Statistical Data

Gathers information about database objects, such as access frequency, how often the data block is found in the cache, and the frequency of disk reads. This data is stored in relevant statistics tables.

- `pg_stat_activity`
- `pg_stat_replication`
- `pg_stat_user_tables`
- `pg_stat_user_indexes`
- `pg_statio_user_tables`
- `pg_statio_user_indexes`

Table Statistics

Table statistics are used by the query planner to generate the most efficient plan. Database content statistics are stored in the `pg_statistic` catalog. The ANALYZE command updates these statistics.

Lock Mechanism

The lock mechanism ensures data integrity during concurrent read/write operations in the database. PostgreSQL uses page-level locking to control access to pages in the `shared_buffer`.

Starting a Transaction

```
postgres=# create table emp (salary int , number int);
CREATE TABLE

postgres=# insert into emp (salary, number) values (1000, 12);
INSERT 0 1

postgres=# Begin;
BEGIN

postgres=*# update emp set salary=10 where number=12;
UPDATE 1

postgres=*# commit;
COMMIT
```

Monitoring Locks

```
SELECT * FROM pg_locks pl LEFT JOIN pg_stat_activity psa ON pl.pid = psa.pid;
SELECT a.datname, c.relname, l.transactionid, l.mode, l.GRANTED, a.usename, a.query,
a.query start, age(now(), a.query start) AS "age"
FROM pg_stat_activity a
JOIN pg locks l ON l.pid = a.pid
JOIN pg_class c ON c.oid = l.relation
ORDER BY a.query start;

-- Also, you can see idle in transaction
watch -n 2 "psql -c \"SELECT pid, datname, usename, state, query, state_change FROM
pg_stat_activity WHERE state = 'idle in transaction';\""
```

Session Kill

```
SELECT pg terminate backend(pid)
FROM pg stat activity
WHERE pid <> pg_backend_pid() -- don't kill your connection!
AND datname = 'exampledb'; -- don't kill the connections to other databases
```

These maintenance operations will make your PostgreSQL database more efficient and secure. For more detailed and technical articles like this, keep following our blog on Medium. If you have any questions or need further assistance, feel free to reach out in the comments below and directly.