COMPARING THE ORACLE AND POSTGRESQL TRANSACTION SYSTEMS

BY LAURENZ ALBE
TALKING HEAD

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

PROUD CONTRIBUTOR TO
PostgreSQL
the world's most advanced open source database


AUSTRIA (HQ)
CYBERTEC POSTGRESQL
INTERNATIONAL (HQ)
ESTONIA
CYBERTEC POSTGRESQL
NORDIC
SWITZERLAND
CYBERTEC POSTGRESQL SWITZERLAND
POLAND
CYBERTEC POSTGRESQL
POLAND
URUGUAY
CYBERTEC POSTGRESQL
SOUTH AMERICA
SOUTH AFRICA
CYBERTEC POSTGRESQL
SOUTH AFRICA

DATABASE PRODUCTS & TOOLS

CYBERTEC MIGRATOR

CYPEX

CYBERTEC PGEE

scalefield

YAIM

WAL BOUNCER

DATA MASKING

pg_ TIMETABLE

PL/pgSQL_sec CYBERTEC

PG WATCH

PG SQUEEZE

POSTGRESQL CONFIGURATOR

PG SHOW PLANS

PATRONI ENVIRONMENT SETUP

ISO 27001 CERTIFIED

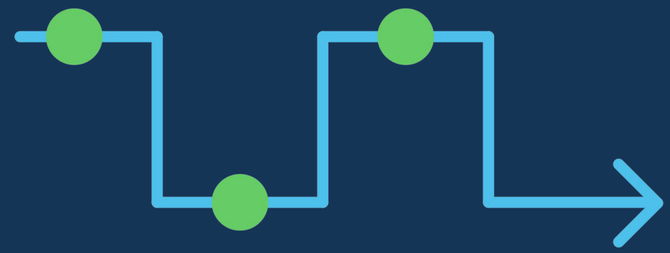
INTRODUCTION

WHY COMPARE ORACLE AND POSTGRESQL?

- Both are relational databases governed (to some extent) by the SQL standard
- The difference is often in the details
- Many people want to move from Oracle to PostgreSQL
  - Lower total cost of ownership
  - More and better support options
  - More insight into and control over the software
- Migration projects often get caught up in minor differences
- It is important to know about the differences!

SCOPE OF THE TALK

- It is impossible to compare all aspects of Oracle and PostgreSQL
- Comparing feature lists is boring
- I will limit myself to database transactions
  - Important for development and porting applications
  - Differences are often not visible at first glance
  - Differences may not show up during testing, but under high concurrency
  - It is an area I know well in both databases



Special thanks to Piotr Vatulik for his help with Oracle internals!

GENERAL OBSERVATIONS

ACID: THE SERVICES A DATABASE TRANSACTION PROVIDES

- **A**tomicity: all statements in a single database transaction form a unit
  - Either all statements succeed or none of them take effect
- **C**onsistency: no database transaction will ever violate a database constraint
- **I**solation: concurrent transactions won't cause certain *anomalies*
  - An anomaly is a visible state of the database that no serial execution of the transactions could ever cause
- **D**urability: a committed (completed) database transaction can never be undone
  - Even by a system crash or localized hardware failure

We will look for differences in each category.

SIMILARITIES BETWEEN ORACLE AND POSTGRESQL

- Use of multiversioning: readers and writers don’t block each other
  - The reader is served the latest committed data
- Locks are held until the end of the transaction
- Row locks are stored in the table, not in the shared memory lock table
  - Row locks cause writes, but there is no need for lock escalation, even though the lock table is of limited size
- Support for `SELECT ... FOR UPDATE` for explicit concurrency control
- Default transaction isolation level is `READ COMMITTED`
  - Behaves pretty similarly on both systems

COMPARING TRANSACTION ATOMICITY

AUTOCOMMIT

- In Oracle, DML automatically starts transaction
  - Explicit COMMIT needed to end a transaction
- PostgreSQL uses autocommit
  - Automatically commits at the end of each statement
  - Start multi-statement transactions with an explicit **`BEGIN`** or **`START TRANSACTION`**
- Many database APIs allow turning autocommit on or off
  - In Oracle, autocommit is simulated by automatically sending **`COMMIT`** as needed
  - In PostgreSQL, disabled autocommit is simulated by automatically sending **`BEGIN`** as needed
  - Using such an API, you don’t need to worry about the difference

STATEMENT-LEVEL ROLLBACK

- In Oracle, a statement that causes an error is rolled back, but the transaction continues
- In PostgreSQL, an error on a statement aborts the transaction:
  `ERROR: current transaction is aborted, commands ignored until end of transaction block`
  - Set a savepoint before the statement that you can rollback to
  - **Warning:** use savepoints sparingly ⇒ performance impact!
- Most well-written applications will roll back after an error
  - In that case, you won't notice the difference

TRANSACTIONAL DDL

- In Oracle, DDL statements automatically commit the transaction
  - Savepoints created before the DDL statement are not lost, but get moved to after the DDL statement!
- In PostgreSQL, DDL statements can be rolled back like all other statements
  - Useful for schema migration (simply roll back on error)
  - Attention: DDL statements normally take heavy locks
  - A few statements must run in their own transaction:
    `CREATE DATABASE, VACUUM, CREATE INDEX CONCURRENTLY, ...`

COMPARING TRANSACTION CONSISTENCY

CONSISTENCY IN ORACLE AND POSTGRESQL

- Both systems maintain database constraints properly
- Oracle can enable and disable constraints with `ALTER TABLE`
  - Constraints can be re-enabled with `ENABLE NOVALIDATE`, which may break consistency (primary and unique keys must be `DEFERRABLE` for that)
- PostgreSQL enables foreign keys and `DEFERRABLE` primary and unique keys with system triggers
  - Such triggers can be disabled by a superuser
  - Another way to break trigger-based consistency as a superuser is setting `session_replication_role` to `replica`

DIFFERENCES IN CONSTRAINT VALIDATION

This script runs fine in Oracle:

```
CREATE TABLE tab (id integer PRIMARY KEY);

INSERT INTO tab (id) VALUES (1);
INSERT INTO tab (id) VALUES (2);

COMMIT;

UPDATE tab SET id = id + 1;

COMMIT;
```

In PostgreSQL, the **UPDATE** fails with

```
ERROR: duplicate key value violates unique constraint "tab_pkey"
DETAIL: Key (id)=(2) already exists.
```

DIFFERENCES IN CONSTRAINT VALIDATION (EXPLANATION & REMEDY)

- PostgreSQL checks primary key and unique key constraints *after the modification of each row*
- This violates the SQL standard:

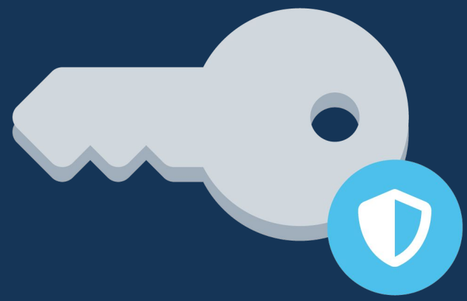
  Whenever an SQL-statement is executed, every enforced constraint whose mode is immediate is checked, at a certain point after any changes to SQL-data and schemas resulting from that execution have been effected, to see if it is satisfied.
- To get unique and primary key constraints checked at the end of the statement, define them as `DEFERRABLE INITIALLY IMMEDIATE`

FOREIGN KEYS AND LOCKS

```
CREATE TABLE parent (pid bigint PRIMARY KEY, val text);
INSERT INTO parent VALUES (1, NULL);
CREATE TABLE child (cid bigint PRIMARY KEY, pid bigint REFERENCES parent);

START TRANSACTION;
INSERT INTO child VALUES (100, 1);

                /* second concurrent session */
                DELETE FROM parent WHERE pid = 1;
```



The **DELETE** must block to maintain integrity.

- In Oracle, the **DELETE** locks the referencing rows in **child**
  ⇒ locks the whole table if there is no index on the foreign key
- In PostgreSQL, the **INSERT** locks referenced rows in **parent**
  ⇒ contention on the lock manager with many **INSERT**s

COMPARING TRANSACTION ISOLATION

SUPPORT FOR TRANSACTION ISOLATION LEVELS

- PostgreSQL supports all isolation levels: **`READ UNCOMMITTED`**, **`READ COMMITTED`**, **`REPEATABLE READ`** and **`SERIALIZABLE`**
  - However, **`READ UNCOMMITTED`** is silently upgraded to **`READ COMMITTED`**
    ⇒ no “dirty data”
  - **`SERIALIZABLE`** is implemented using non-blocking predicate locks (SIRead locks)
- Oracle offers only **`READ COMMITTED`** and **`SERIALIZABLE`**
  - **`SERIALIZABLE`** is a lie ⇒ really, it is **`REPEATABLE READ`**
  - Oracle’s implementation of “**`SERIALIZABLE`**” is so horrible that it is mostly unusable; it gives serialization errors on any pretext:
    - The first insert into a table created with **`SEGMENT CREATION DEFERRED`**
    - Any index page split during an **`INSERT`**
    - . . .

CONCURRENCY AND LOCKING WITH READ COMMITTED IN POSTGRESQL

PostgreSQL example (a bill and positions on the bill):

```
SELECT * FROM bill;

 bill_id | total
---------+-------
       1 | 60.00

SELECT * FROM item;

 item_id | bill_id | amount
---------+---------+--------
     101 |       1 | 10.00
     102 |       1 | 20.00
     103 |       1 | 30.00
```

CONCURRENCY AND LOCKING WITH READ COMMITTED IN POSTGRESQL

Inserting a new position and updating the redundant sum in **`bill`**:

```
BEGIN;

INSERT INTO item
   (item_id, bill_id, amount)
VALUES
   (104, 1, 40.0);

-- really, this should be in a trigger
UPDATE bill SET total = total + 40.0
WHERE bill_id = 1;

COMMIT;
```

CONCURRENCY AND LOCKING WITH READ COMMITTED IN POSTGRESQL

A concurrent report:

```
SELECT bill_id, item.amount, bill.total
FROM bill
   LEFT JOIN item USING (bill_id)
FOR UPDATE OF bill;
```

When run after the `UPDATE` on `bill` and before the `COMMIT`, the statement blocks. After the `COMMIT`, we get the following inconsistent result:

```
 bill_id | amount | total
---------+--------+--------
       1 |  10.00 | 100.00
       1 |  20.00 | 100.00
       1 |  30.00 | 100.00
```

CONCURRENCY AND LOCKING WITH READ COMMITTED IN POSTGRESQL

- After it gets the lock, PostgreSQL fetches the most recent committed version **of the locked rows** and re-evaluates the query conditions for them
- We see different versions for different rows:
  - Rows that were not locked (`item`) appear like at query start
  - Rows where locking blocked (`bill`) appear in their most recent version
- Oracle would simply start the query from the beginning and get consistent results
- PostgreSQL performs better, at the price of a potential (allowed) anomaly

COMPARISON OF PERSISTENCE AND DURABILITY

DIFFERENCES CAUSED BY THE DIFFERENT IMPLEMENTATION

Both systems provide durability. But the implementation is different:

- Oracle uses an UNDO tablespace to keep old versions of the data
  - Long queries/transactions can hit “snapshot too old”
  - (You may need to increase **`UNDO_RETENTION`** or the UNDO tablespace)
  - Rollback can take a long time
  - Big deletes are slow (move data to UNDO)

DIFFERENCES CAUSED BY THE DIFFERENT IMPLEMENTATION

- PostgreSQL keeps old row versions in the table
  - Both commit and rollback are very fast
  - Regular garbage connection necessary (autovacuum)
  - Big updates can make the table grow (“bloat”)
- UPDATE tends to perform worse in PostgreSQL
  - Usually more index maintenance and more dirty pages
  - To mitigate these problems, use “HOT updates”

TRANSACTION ID WRAPAROUND IN POSTGRESQL

- PostgreSQL implements multiversioning by storing transaction IDs in table rows
- Transaction IDs are generated from a 4-byte unsigned integer counter
  - The counter will eventually “wrap around”
- Before transaction IDs can safely wrap around, extra maintenance has to “freeze” old table rows
- On highly transactional systems, this can require special tuning

QUESTIONS?

LAURENZ ALBE

TALKING HEAD

EMAIL

laurenz.albe@cybertec.at

PHONE

+43 670 605 6265

www.cybertec-postgresql.com

@cybertec-postgresql

www.youtube.com/@cybertecpostgresql

PROUD CONTRIBUTOR TO

PostgreSQL
the world's most advanced open source database

→ Tickets now live!

→ Call for papers

→ Call for sponsors

pgday.at

Open Alliance

For PostgreSQL Education

- Independent industry certification for PostgreSQL
- First launch at PGConf.DE 2025
- For more information visit booths of CYBERTEC, Data Egret and Proventa

Scan for Updates

oapg-edu.org

MEET OUR PARTNERS AT PGCONF.DE

re_forms 21

redgate

PROVENTA

Microsoft

readyset

X-cellent technologies

straDATA