PostgreSQL for Oracle DBAs

Thuymy Tran
Database Solutions Architect

Wanda He
Principal RDS PostgreSQL Solutions Architect
Agenda

• PostgreSQL Introduction
• Oracle vs. PostgreSQL
  • Architecture Comparison
  • MVCC
  • Indexes
  • PostgreSQL Extensions
• Common Mistakes for Oracle to PostgreSQL Migration
• Summary – Key Takeaways
PostgreSQL Introduction
History of PostgreSQL

• First version was released in 1997
• Initiated as Ingres project at UC Berkeley (Michael Stonebraker)
• Written in C
• Flexible across all the UNIX platforms, Windows, MacOS and others
• Standard Postgres Sources and Knowledge base
  • www.postgresql.org – (documentation, release notes and community)
  • PostgreSQL Wiki page
Features

• Full network client-server architecture
• ACID compliant
• Transactional (uses WAL / REDO)
• Partitioning
• Tiered storage via tablespaces
• Multiversion Concurrency Control (readers don’t block writers)
• On-line maintenance operations
• Hot (readonly) and Warm (quick-promote) standby
• Log-based and trigger based replication
• SSL
• Full-text search
• Procedural languages
## General Database Maximum

<table>
<thead>
<tr>
<th>Limit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Database Size</td>
<td>64 ZB</td>
</tr>
<tr>
<td>Maximum Table Size</td>
<td>32 TB</td>
</tr>
<tr>
<td>Maximum Row Size</td>
<td>1.6 TB</td>
</tr>
<tr>
<td>Maximum Field Size</td>
<td>1 GB</td>
</tr>
<tr>
<td>Maximum Rows / Table</td>
<td>Unlimited</td>
</tr>
<tr>
<td>Maximum Columns / Table</td>
<td>250-1600</td>
</tr>
<tr>
<td>Maximum Indexes / Table</td>
<td>Unlimited</td>
</tr>
</tbody>
</table>
Oracle vs. PostgreSQL
## Terminology

<table>
<thead>
<tr>
<th>Oracle</th>
<th>PostgreSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>rowid</td>
<td>ctid</td>
</tr>
<tr>
<td>row</td>
<td>tuple</td>
</tr>
<tr>
<td>table</td>
<td>relation</td>
</tr>
<tr>
<td>block</td>
<td>page</td>
</tr>
<tr>
<td>redo</td>
<td>WAL</td>
</tr>
<tr>
<td>undo</td>
<td>MVCC</td>
</tr>
<tr>
<td>SCN</td>
<td>LSN</td>
</tr>
</tbody>
</table>
Architecture Comparison
Process/Memory Architecture

**PostgreSQL**

- **Instance**
  - System Memory
  - Shared Memory
  - Physical Files
  - OS Cache

- **Utility Processes**
  - BG Writer
  - WAL writer
  - Checkpointer
  - AutoVacuum
  - Stats Collector
  - Archiver
  - Logging Collector

- **Postmaster (server process)**

- **Client Process**

**Oracle**

- **Instance**
  - System Global Area (SGA)
    - Shared Pool
    - Large Pool
    - Library Cache
    - Shared SQL Area
    - Fixed SGA
    - Java Pool
    - Streams Pool
    - Pools (Service, Library, Large, Memory, and Credits)

- **Database Buffer Cache**

- **PGA**

- **Server Process**
  - DBWn
  - CKPT
  - LGWR
  - ARCn
  - RVWR

- **Free Memory**
  - V0 Buffer Area
  - UGA

© 2024, Amazon Web Services, Inc. or its affiliates.
PostgreSQL Processes

- PostgreSQL utilizes a multi-process architecture
- Similar to Oracle’s ‘Dedicated Server’ mode
- Types of processes
  - Primary (postmaster)
  - Per-connection backend process
    - Dedicated, per-connection server process
    - Known as a ‘worker’ process
    - Responsible for fetching data from disk and communicating with the client
  - Utility (e.g. checkpoint, wal-writer, autovacuum, etc.)
Connection

Connect process flow

1. A client connection is sent to the postmaster
2. Authentication is performed
3. The postmaster spawns a user-backend process
4. The user-backend calls back to the client to continue operation

Each process has its own:

- Backend
- Private memory – catalog cache, prepare stmt, query execution …
Scale with connection pooling

- Connection is expensive
  - Connection local cache (catalog cache, prepare statement, and etc.)
  - High CPU context switches when ratio of CPU : active connections is high
- Enhance scalability with connection pooling solution
  - PgBouncer, Pgpool-II, Amazon RDS Proxy
MVCC
What is MVCC?

• Multiversion Concurrency Control

• Offers high concurrency even during significant database read/write activity

• Readers never block writers, and writers never block readers

• Reduces locking requirements, but does not eliminate locking
MVCC (Oracle vs. PostgreSQL)

- MVCC store
  - Oracle: rollback segment (undo)
  - PostgreSQL: in data table

Update operation:
- Oracle: update row in-place
  - Store old version of row in undo
  - Update row in-place
- PostgreSQL: “copy-on-write”
  - The new tuple is inserted
  - The old tuple is marked “dead”

Update/Delete: Space is not reclaim immediately
MVCC Behavior

- Visibility is driven by transaction IDs (XID)
- Tuples have an XMIN and XMAX
  - XMIN is the XID that created the tuple
  - XMAX is the XID that removed the tuple

Visibility rule:
\[
xmin \leq \text{pg\_current\_xact\_id}() \quad \text{AND} \quad (xmax = 0 \text{ OR } \text{pg\_current\_xact\_id}() < xmax)
\]
Table or Index Bloat

- Side-effect of MVCC leaves “dead” space in table and indexes after UPDATE and DELETE ➔ BLOAT

- BLOAT - space occupied by dead tuples
  - Increase physical IOs
  - Reduce efficiency in memory usage

- Reclaim space used by “dead” tuples
  - Autovacuum / Vacuum
  - Space is reclaim for subsequent inserts
  - Storage not turn back to OS until re-org or rebuild

PostgreSQL BLOAT & page storage
How VACUUM does it

Vacuum phases:
1. **Scan** heap, remembering tuples (ctid) to remove in memory
2. **Vacuum** indexes and heap
3. **Cleanup**, remove tuples from heap

*Repeat steps 1-3 if vacuum cannot complete in a single pass*
Vacuum

- Vacuum cleans up dead tuples
- Periodic vacuuming is required to:
  - Recover or reuse disk space by update or deleted operation
  - Update data statistics
  - Update visibility map, free space map
  - Protect against transaction ID wraparound
    - XIDs are limited in size (32 bits)
    - More than 2 billion transactions would suffer transaction ID wraparound
Autovacuum

• # of autovacuum workers: autovacuum_max_workers (default to 3)
• Memory per worker: maintenance_work_mem (or autovacuum_work_mem)

• Triggering autovacuum
  • autovacuum_vacuum_scale_factor
  • autovacuum_vacuum_threshold
  • autovacuum_vacuum_insert_scale_factor
  • autovacuum_vacuum_insert_threshold

• Control cost
  • autovacuum_vacuum_cost_limit (shared by all workers)
  • autovacuum_vacuum_cost_delay (sleep time to reduce IO impact)

• Tuning at table level (recommended for large tables):
  • ALTER TABLE myablename SET autovacuum_scale_factor = 0
  • ALTER TABLE myablename SET autovacuum_vacuum_threshold = 10000
Minimize bloat

• Best practices to control bloat
  ▪ Create process for ongoing monitoring of bloated table / index
    • [https://wiki.postgresql.org/wiki/Show_database_bloat](https://wiki.postgresql.org/wiki/Show_database_bloat) or pgstattuple extension
  ▪ Tune autovacuum/manual vacuum to minimize bloat
    • Default setting may not be sufficient
    • Use table level tuning for large tables
  ▪ Rebuild to release storage back to OS (Shrink)
    • Rebuild index (online option)
    • Online rebuild with pg_repack extension (online)
    • Rebuild with vacuum full (offline operation, generally not recommended)
Indexes
## Index Compatibility or Equivalent

<table>
<thead>
<tr>
<th>Postgres</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-Tree</td>
<td>B-Tree</td>
</tr>
<tr>
<td>Multicolumn Indexes</td>
<td>Composite Indexes</td>
</tr>
<tr>
<td>Expression Indexes</td>
<td>Function-based Indexes</td>
</tr>
<tr>
<td>HypoPG extension</td>
<td>Invisible Indexes</td>
</tr>
<tr>
<td>Cluster Index</td>
<td>Indexed-Organized Tables (IOT)</td>
</tr>
<tr>
<td>Consider BRIN index</td>
<td>BITMAP index / Bitmap join</td>
</tr>
</tbody>
</table>
## More PostgreSQL Index Types

**ADDITIONAL INDEXES THAT ORACLE DOESN’T HAVE**

<table>
<thead>
<tr>
<th>Index</th>
<th>Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generalized Inverted Index (GIN)</td>
<td>• Map a large amount of values to one row</td>
</tr>
<tr>
<td></td>
<td>• Optimal for fulltext search and indexing array values</td>
</tr>
<tr>
<td>Generalized Search Tree (GiST)</td>
<td>• Optimal for more complex comparisons (geometric data types)</td>
</tr>
<tr>
<td>Space Partitioned GiST (SP-Gist)</td>
<td>• Optimal for partitioned search trees</td>
</tr>
<tr>
<td>Block Range Index (BRIN)</td>
<td>• Stores min and max values contained in a group of database pages</td>
</tr>
<tr>
<td></td>
<td>• Optimal for time series data</td>
</tr>
<tr>
<td></td>
<td>• Rule out certain records and therefore reduce query run time</td>
</tr>
<tr>
<td>BLOOM</td>
<td>• Test whether an element is a member of a set</td>
</tr>
<tr>
<td></td>
<td>• Optimal when a table has many attributes and queries test arbitrary combinations on them</td>
</tr>
</tbody>
</table>
GIN Index

CREATE INDEX idx_users_lname
ON users USING gin (lname gin_trgm_ops);

EXPLAIN SELECT * FROM users WHERE lname LIKE '%ing%';

QUERY PLAN

Bitmap Heap Scan on users (cost=8.00..12.02 rows=1 width=654)
  Recheck Cond: ((lname)::text ~~ '%ing '::text)
  -> Bitmap Index Scan on idx_users_lname
     (cost=0.00..8.00 rows=1 width=0)
     Index Cond: ((lname)::text ~~ '%ing '::text)
PostgreSQL Extensions
PostgreSQL Extensions for added functionality

- PostgreSQL is designed to be extensible
- Large community support, 1000+ extensions to add functionality on top of core PostgreSQL
- Extensions loaded into the database can function just like features that are built in

Sample popular extensions:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Postgres (Extensions or 3rd Party)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auditing</td>
<td>pgAudit</td>
</tr>
<tr>
<td>Partition Management</td>
<td>pg_partman</td>
</tr>
<tr>
<td>Query optimization</td>
<td>pg_hint_plan</td>
</tr>
<tr>
<td>Cron</td>
<td>pg_cron</td>
</tr>
<tr>
<td>Monitoring</td>
<td>pg_stat_statements</td>
</tr>
<tr>
<td>Vector Search</td>
<td>pg_vector</td>
</tr>
<tr>
<td>Spatial Database</td>
<td>PostGIS</td>
</tr>
<tr>
<td>Database Link</td>
<td>Foreign Data Wrapper (FDW)</td>
</tr>
<tr>
<td>Invisible Index</td>
<td>HypoPG</td>
</tr>
</tbody>
</table>
Common Mistakes for Oracle to PostgreSQL Migration
Synonyms

Oracle

- Synonyms are used commonly to avoid fully qualifying objects

```sql
CREATE SYNONYM [schema .] synonym_name FOR [schema .] object_name ;
```

PostgreSQL

- No synonyms in Postgres
- Use schema search path instead, SEARCH_PATH
- To view current search path:
  ```sql
  # SHOW search_path;
  ```

  Default set up returns:
  ```text
  search_path
  ------------
  "$user", public
  ```

- To add new schema in path:
  ```sql
  # SET search_path to schema1, schema2;
  ```
NULLs

• PostgreSQL and Oracle handle NULLs differently
  • Oracle: Empty string is considered NULL
  • PostgreSQL: NULL is treated as none value

• Affecting:
  • String concatenation
  • NULL comparisons
  • Unique constraints
NULLs – String Concatenation

Oracle:

```
SQL> SELECT fname || ' ' || mname || ' ' || lname FROM people;

FNAME||MNAME||LNAME
Marilyn Monroe
Nelson Mandela
John F. Kennedy
Martin Luther King
Winston Churchill
Michael Jordan
Mahatma Gandhi
Margaret Thatcher
Elvis Presley
Albert Einstein
10 rows selected.
```

Postgres:

```
test=# SELECT fname || ' ' || mname || ' ' || lname FROM people;
?column?

John F. Kennedy
Martin Luther King

(10 rows)
```
NULLs – String Concatenation

PostgreSQL uses coalesce() or built-in functions to handle nulls.

```sql
test=# SELECT COALESCE(fname, '') || ' ' || COALESCE(mname, '') || ' ' || COALESCE lname, '') FROM people;
?column?
------------------------
Marilyn Monroe
Nelson Mandela
John F. Kennedy
Martin Luther King
Winston Churchill
Michael Jordan
Mahatma Gandhi
Margaret Thatcher
Elvis Presley
Albert Einstein
(10 rows)
```
```sql
test=# SELECT concat_ws(' ', fname, mname, lname) FROM people;
concat_ws
------------------------
Marilyn Monroe
Nelson Mandela
John F. Kennedy
Martin Luther King
Winston Churchill
Michael Jordan
Mahatma Gandhi
Margaret Thatcher
Elvis Presley
Albert Einstein
(10 rows)
```
NULLs – Unique Constraints

Oracle
• Unique constraint violation if attempt to inserting rows with NULL values

PostgreSQL
• NULL is not equal to NULL => NO Unique Constraint Violation
• Started w/ PostgreSQL v15

```sql
CREATE UNIQUE INDEX null_test_idx ON null_test (c1, c2) NULLS NOT DISTINCT;
```
Oracle Number vs. PostgreSQL Numeric

Most migration tools translate an Oracle Number to a PostgreSQL Numeric

• Oracle NUMBER:
  - NUMBER(precision, scale)
  - Up to 38 digits before the decimal point
  - Up to 127 digits after the decimal point

• PostgreSQL NUMERIC:
  - NUMERIC(precision, scale]
  - Up to 131072 digits before the decimal point
  - Up to 16383 digits after the decimal point
Migrating Oracle Number Data Type

• Consider storage and performance impacts

• Choose the right PostgreSQL number data type

<table>
<thead>
<tr>
<th>Precision(m)</th>
<th>Scale(n)</th>
<th>Oracle</th>
<th>PostgreSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 9</td>
<td>0</td>
<td>NUMBER(m,n)</td>
<td>INT</td>
</tr>
<tr>
<td>9 &gt; m &lt;=18</td>
<td>0</td>
<td>NUMBER(m,n)</td>
<td>BIGINT</td>
</tr>
<tr>
<td>m+n &lt;= 15</td>
<td>n&gt;0</td>
<td>NUMBER(m,n)</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>m+n &gt; 15</td>
<td>n&gt;0</td>
<td>NUMBER(m,n)</td>
<td>NUMERIC</td>
</tr>
</tbody>
</table>

⚠️ Never use NUMERIC for PKs or FKs, use BIGINT
PostgreSQL TEXT Data Type

- TEXT and VARCHAR are equivalent and behave the same
- TEXT is VARCHAR without specific length
- PostgreSQL TEXT is not a “CLOB”
  - Managing CLOB in Oracle requires special operations
    - Get Length: DBMS_LOB.GETLENGTH(x)
Exceptions
Exceptions

Oracle

```sql
CREATE FUNCTION get_first_name(p_lname varchar2) 
  RETURN varchar2 
IS 
  l_fname varchar2(100);
BEGIN 
  SELECT fname
  INTO l_fname
  FROM people
  WHERE lname = p_lname;

  RETURN l_fname;
EXCEPTION 
  WHEN no_data_found THEN 
    l_fname := null;
  RETURN l_fname;
END get_first_name;
```

PostgreSQL

```sql
CREATE FUNCTION get_first_name(p_lname varchar) 
  RETURNS varchar AS $$
DECLARE 
  l_fname varchar;
BEGIN 
  SELECT fname
  INTO l_fname
  FROM people
  WHERE lname = p_lname;

  RETURN l_fname;
EXCEPTION 
  WHEN no_data_found THEN 
    l_fname := null;
  RETURN l_fname;
END$$ LANGUAGE plpgsql;
```
Exceptions

- PostgreSQL uses subtransactions (SAVEPOINT) to handle Exceptions
- Subtransactions are heavy lift

```sql
SAVEPOINT hidden_savepoint;

SELECT fname
    INTO l_fname
    FROM people
WHERE l_name = p_l_name;

if exception
    ROLLBACK TO SAVEPOINT hidden_savepoint;
    l_fname := null;

otherwise
    RELEASE SAVEPOINT hidden_savepoint;
```
Most exceptions are not necessary

```sql
CREATE OR REPLACE FUNCTION get_first_name(p_lname varchar)
    RETURNS varchar
AS $$
DECLARE
    l_fname varchar := null;
BEGIN
    SELECT fname
    INTO l_fname
    FROM people
    WHERE lname = p_lname;

    RETURN l_fname;
END
$$ LANGUAGE plpgsql;
```

test=> SELECT get_first_name('Jordan');
get_first_name
----------------------
Michael
(1 row)

test=> SELECT get_first_name('jordan');
get_first_name
----------------------
(1 row)
Exceptions

NO_DATA_FOUND AND TOO_MANY_ROWS ARE NOT EXCEPTIONS RAISED FOR A SELECT INTO STATEMENT

CREATE FUNCTION get_first_name(p_lname varchar) RETURNS varchar
AS $$
DECLARE
l_fname varchar;
BEGIN
SELECT fname
INTO l_fname
FROM people
WHERE lname = p_lname;

RETURN l_fname;
EXCEPTION
WHEN no_data_found THEN
l_fname := 'NOT_FOUND';
RETURN l_fname;
END$$ LANGUAGE plpgsql;

test=> SELECT get_first_name('jordan');
get_first_name
-------------

(1 row)
Exceptions

**USE STRICT TO GET ORACLE-LIKE BEHAVIOR**

```
CREATE FUNCTION get_first_name(p_lname varchar) RETURNS varchar
AS $$
DECLARE
  l_fname varchar;
BEGIN
  SELECT fname
    INTO STRICT l_fname
    FROM people
    WHERE lname = p_lname;
  RETURN l_fname;
EXCEPTION
  WHEN no_data_found THEN
    l_fname := 'NOT_FOUND';
  RETURN l_fname;
END$$ LANGUAGE plpgsql;
```

test=> SELECT get_first_name('jordan');
get_first_name
----------
NOT_FOUND
(1 row)
```
Key Takeaways

- Enhance PostgreSQL scalability with connection pooling
- Effective vacuuming is important to PostgreSQL performance
- Take advantages of PostgreSQL native features such as functions, index types
- Utilize the rich set of extensions to enhance functionalities beyond core PostgreSQL
- Be aware of the common mistakes in migration
  - Synonyms
  - NULLS
  - Data Types: Numeric, TEXT
  - Exceptions
Q & A
Thank you!

Thuymy Tran
thuymyt@amazon.com

Wanda He
wanhe@amazon.com