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

Limit	Value
Maximum Database Size	64 ZB
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows / Table	Unlimited
Maximum Columns / Table	250-1600
Maximum Indexes / Table	Unlimited

Oracle vs. PostgreSQL

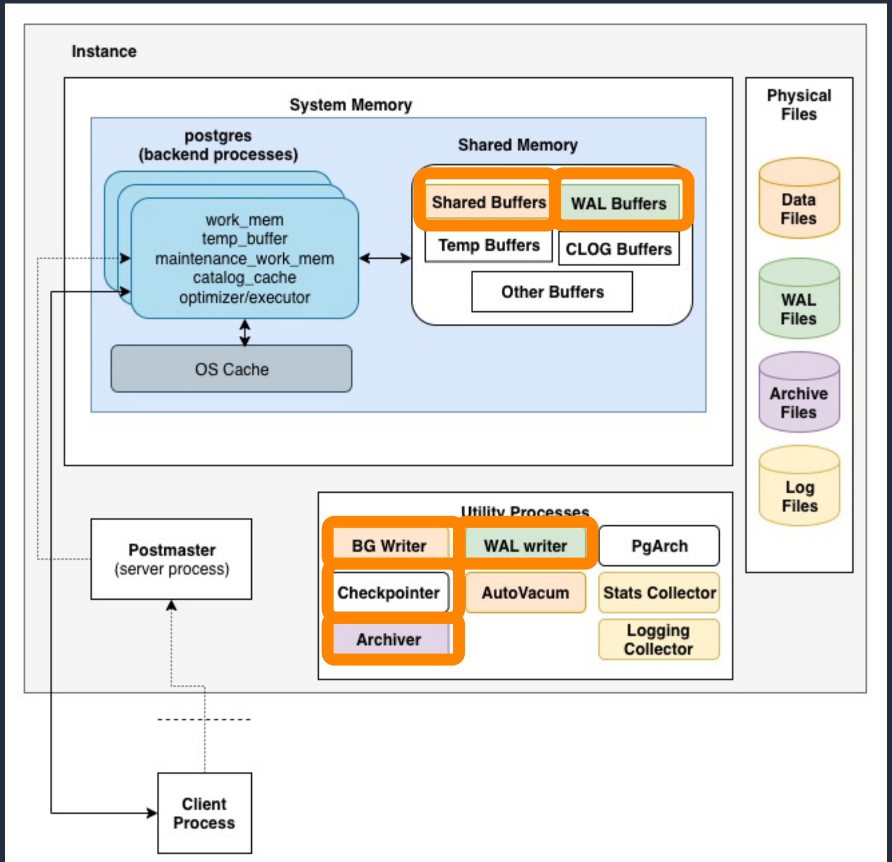
Terminology

Oracle	PostgreSQL
rowid	ctid
row	tuple
table	relation
block	page
redo	WAL
undo	MVCC
SCN	LSN

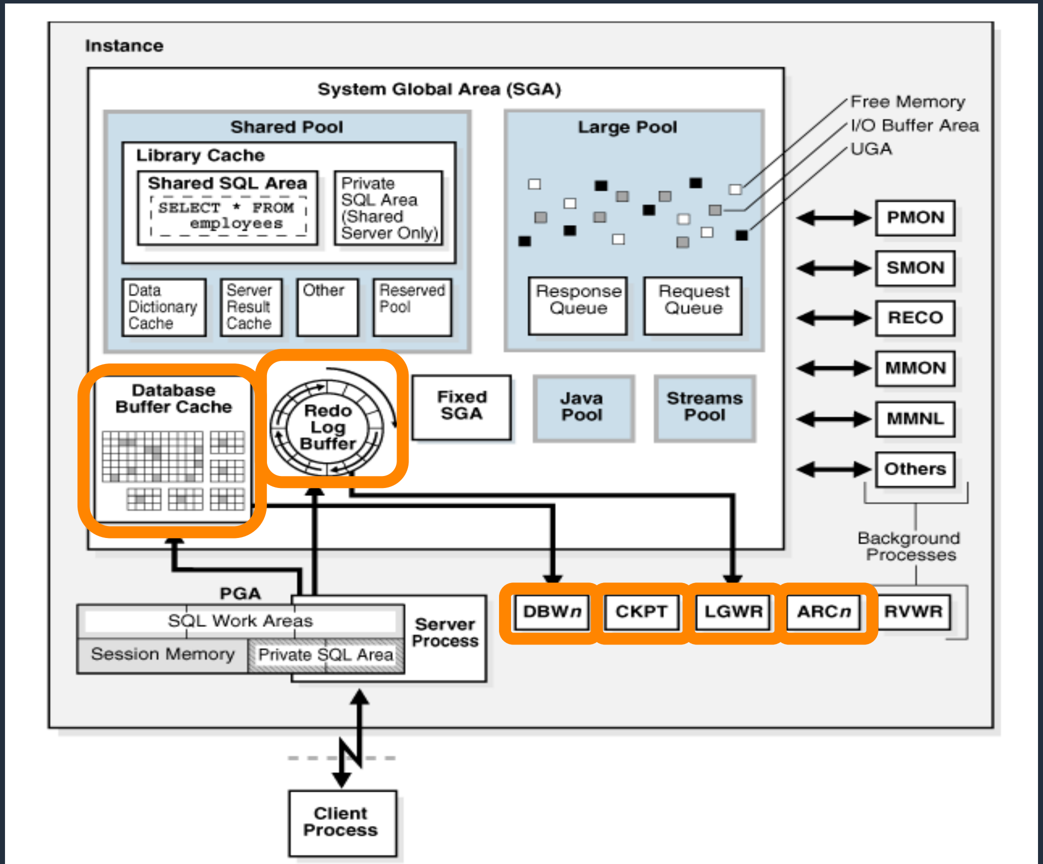
Architecture Comparison

Process/Memory Architecture

PostgreSQL

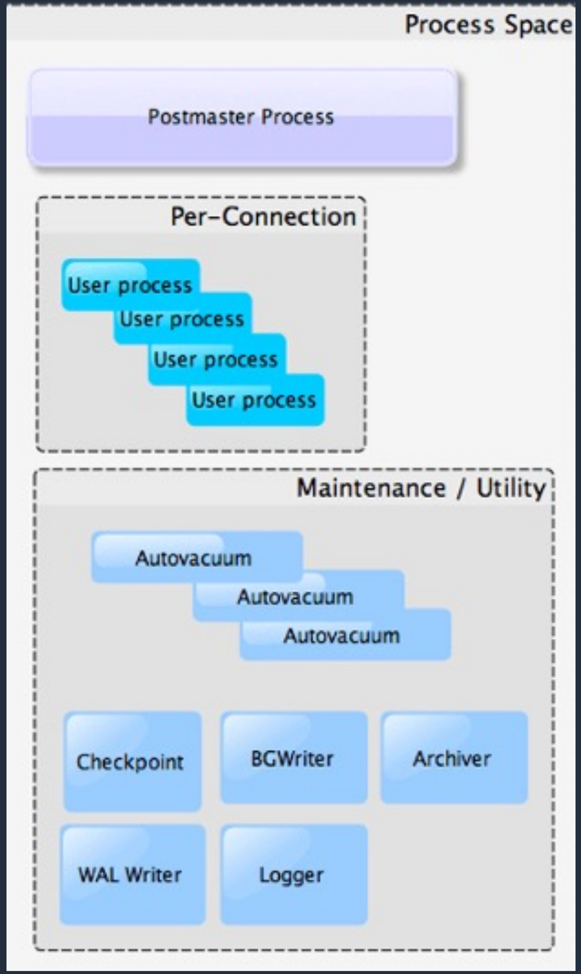


Oracle



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. checkpointer, wal-writer, autovacuum, etc.)



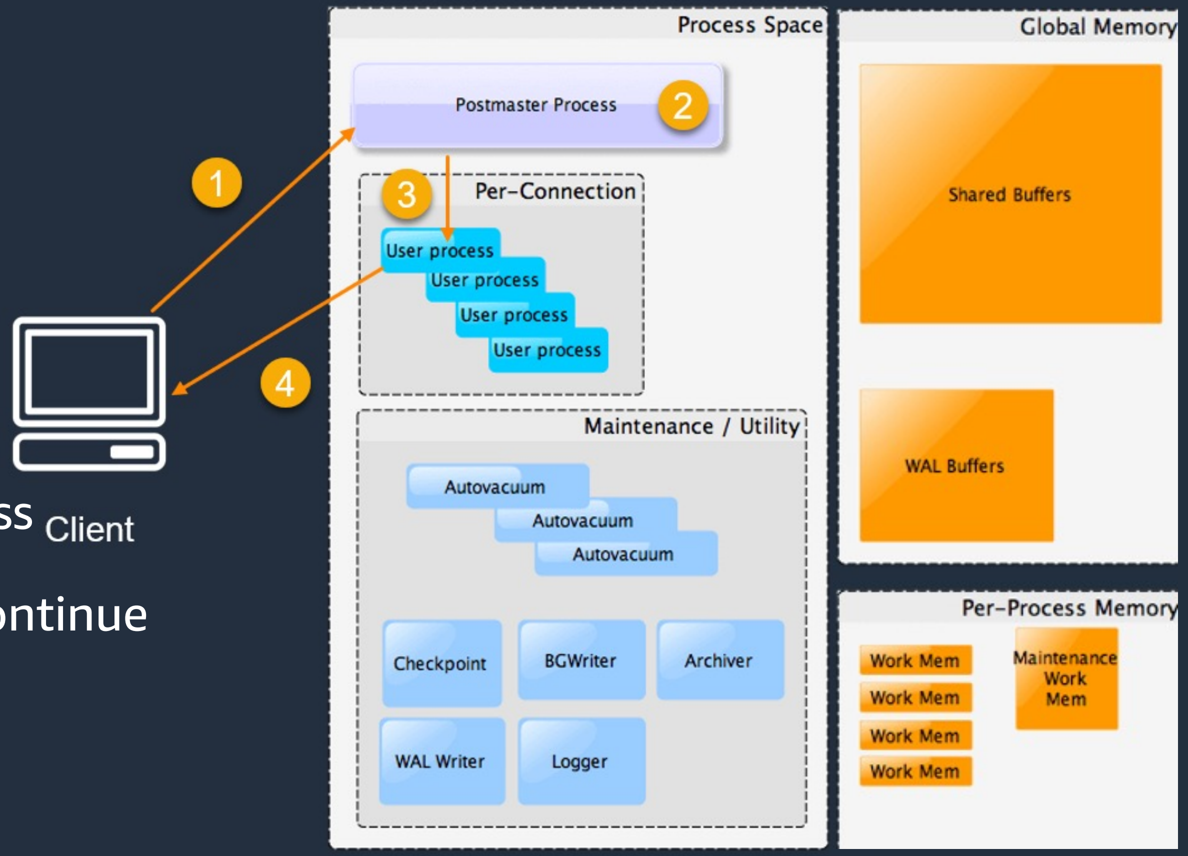
Connection

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)

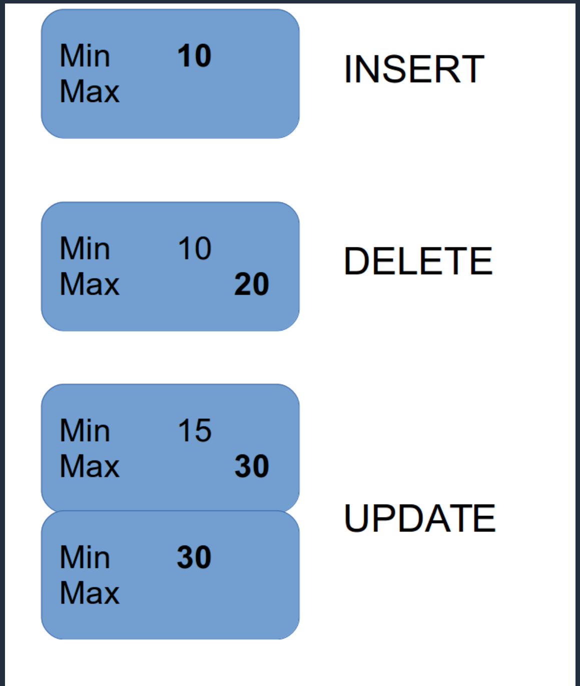
- MVCC store
  - Oracle: rollback segment (undo)
  - PostgreSQL: in data table

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



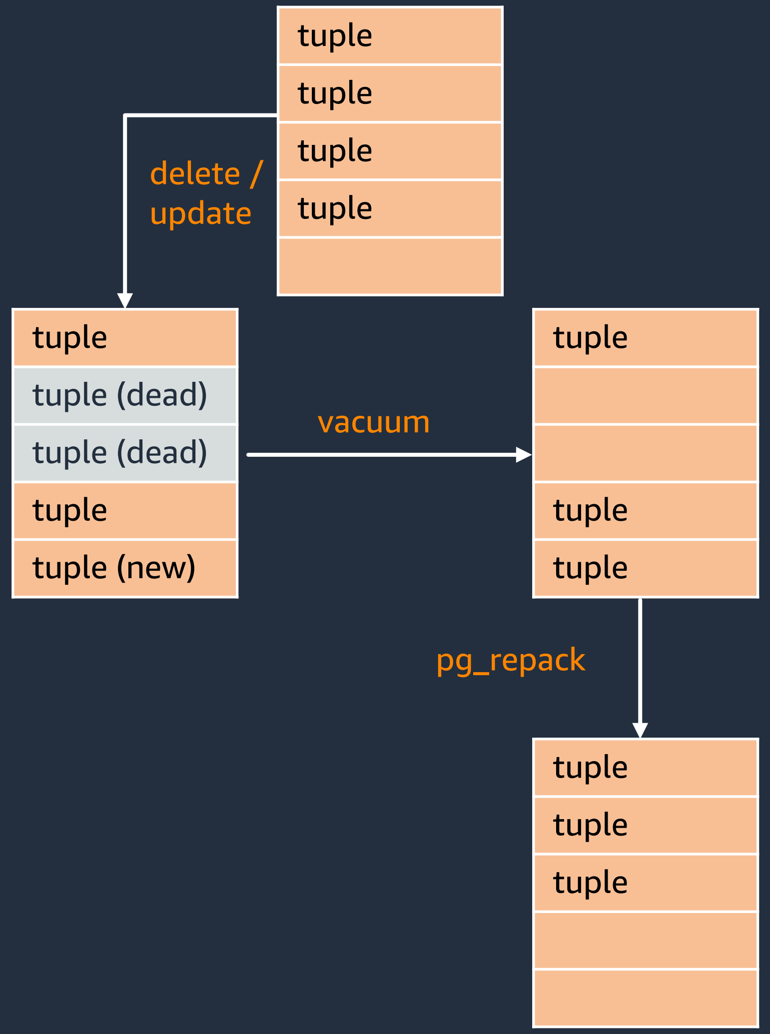
- Visibility is driven by transaction IDs (XID)
- Tuples have an XMIN and XMAX
  - XMIN is the XID that created the tuple
  - XMAX is the XID that removed the tuple

Visibility rule:

xmin <= pg_current_xact_id ()
AND (xmax = 0 OR 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

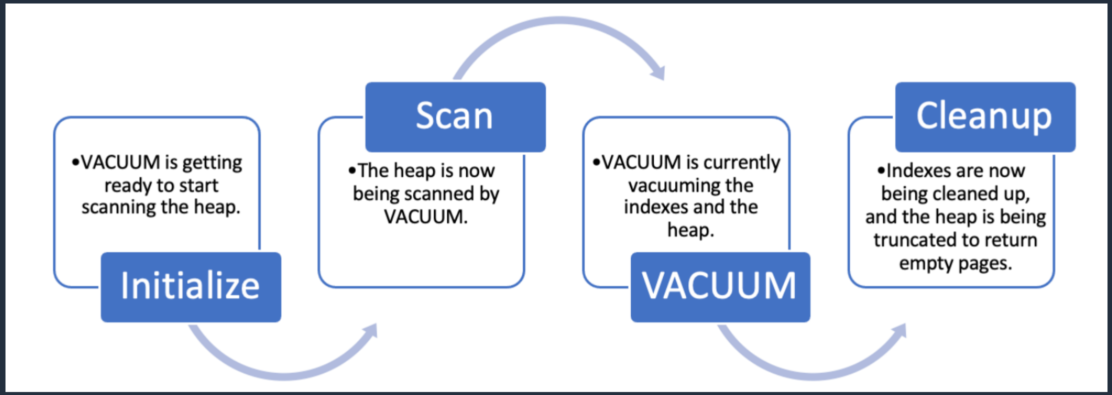


PostgreSQL BLOAT & page storage

How VACUUM does it

Vacuum phases:

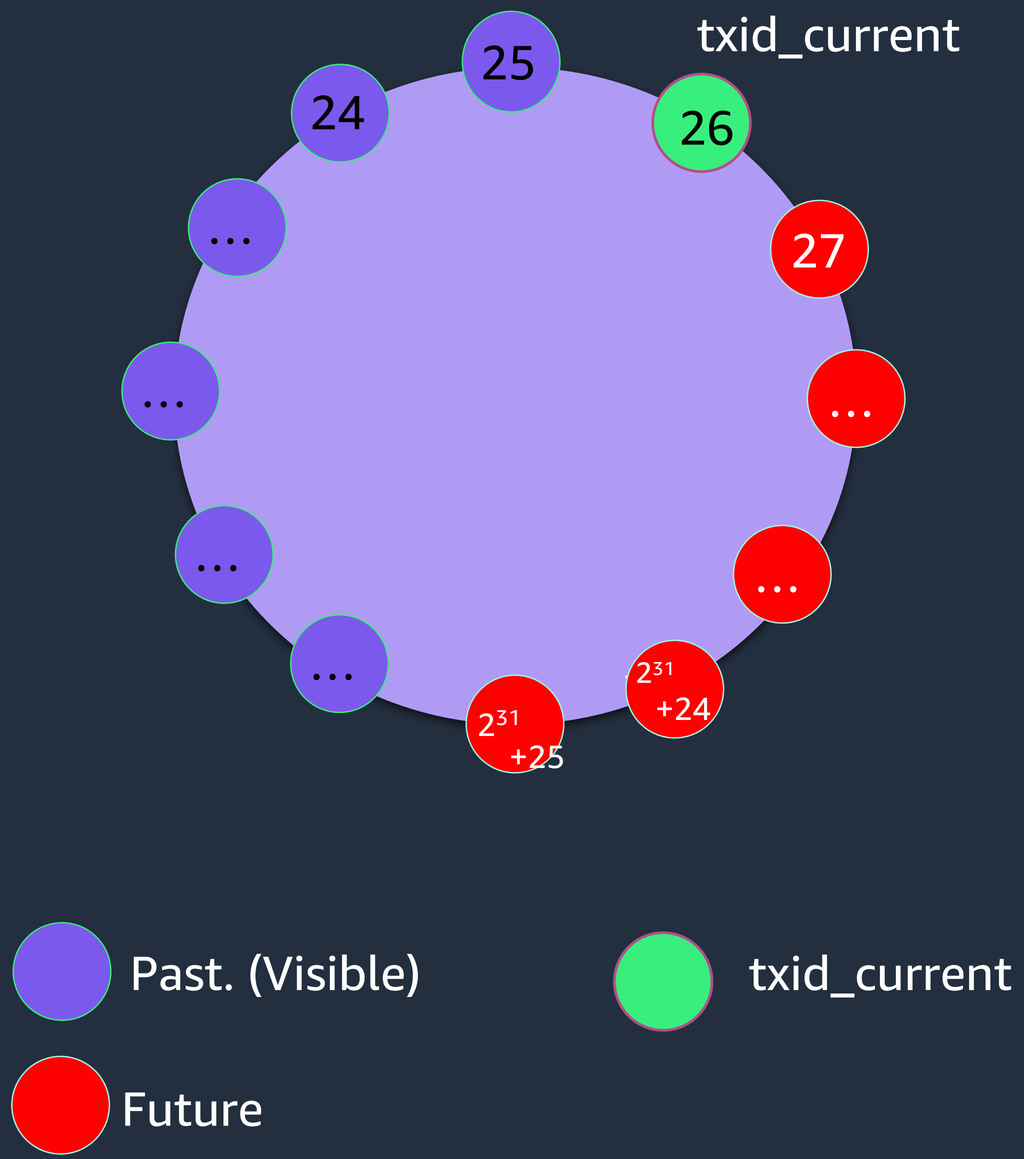
1. **Scan** heap, remembering tuples (ctid) to remove in memory
2. **Vacuum** indexes and heap
3. **Cleanup,** remove tuples from heap



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 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

Postgres	Oracle
B-Tree	B-Tree
Multicolumn Indexes	Composite Indexes
Expression Indexes	Function-based Indexes
HypoPG extension	Invisible Indexes
Cluster Index	Indexed-Oraganized Tables (IOT)
Consider BRIN index	BITMAP index / Bitmap join

More PostgreSQL Index Types

ADDITIONAL INDEXES THAT ORACLE DOESN'T HAVE

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

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:

Feature	**Postgres (Extensions or 3rd Party)**
Auditing	pgAudit
Partition Management	pg_partman
Query optimization	pg_hint_plan
Cron	pg_cron
Monitoring	pg_stat_statements
Vector Search	pg_vector
Spatial Database	PostGIS
Database Link	Foreign Data Wrapper (FDW)
Invisible Index	HypoPG

Common Mistakes for Oracle to PostgreSQL Migration

Synonyms

Oracle

- Synonyms are used commonly to avoid fully qualifying objects

```
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:

```
# SHOW search_path;
```

Default set up returns:

```
search_path
--------------
"$user", public
```

- To add new schema in path:

```
# 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 use coalesce() or built-in functions to handle nulls

```
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)
```

```
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

```
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

Precision(m)	Scale(n)	Oracle	PostgreSQL
<= 9	0	NUMBER(m,n)	INT
9 > m <=18	0	NUMBER(m,n)	BIGINT
m+n <= 15	n>0	NUMBER(m,n)	DOUBLE PRECISION
m+n > 15	n>0	NUMBER(m,n)	NUMERIC

⚠ 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

```
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

```
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

```
SAVEPOINT hidden_savepoint;

SELECT fname
  INTO l_fname
  FROM people
 WHERE lname = p_lname;

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

 otherwise
     RELEASE SAVEPOINT hidden_savepoint;
```

Most exceptions are not necessary

```
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