Overcoming Migration Challenges: From Oracle to PostgreSQL

Baji Shaik
(He/Him)
Sr Consultant, AWS

Sameer Malik
(He/Him)
Specialist Principal SA, AWS

Agenda

- Why migrate?
- Why PostgreSQL?
- Migration Life Cycle
- Top Migration Challenges
- Bonus Tips

Why migrate?

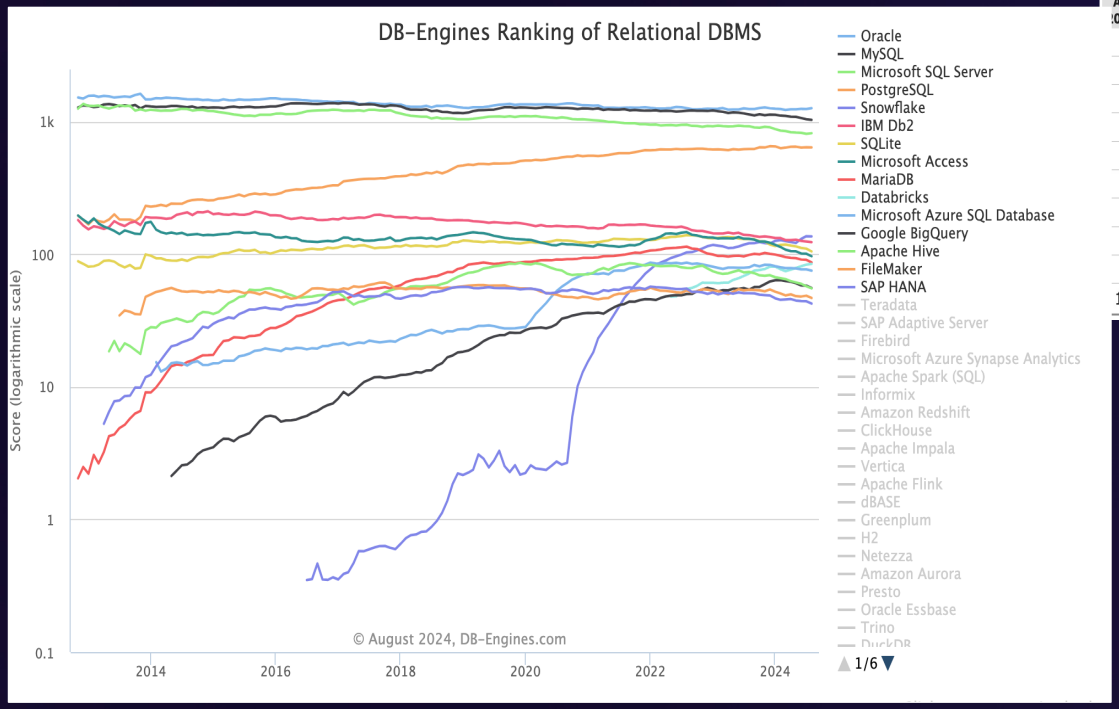
Revisiting Oracle Database licensing cost

Database Edition/Options	Processor License (list price USD)	Software Update & support (USD)
Enterprise Edition	47,500	10,450
Multitenant	17,500	3,850
Real Application Clusters	23,000	5,060
Active Data Guard	11,500	2,530
Partitioning	11,500	2,530
Diagnostics pack	7,500	1,650
Tuning pack	5,000	1,100
Total	1,23,500	27,170

Ref -https://www.oracle.com/a/ocom/docs/corporate/pricing/technology-price-list-070617.pdf

Why PostgreSQL?

PostgreSQL popularity



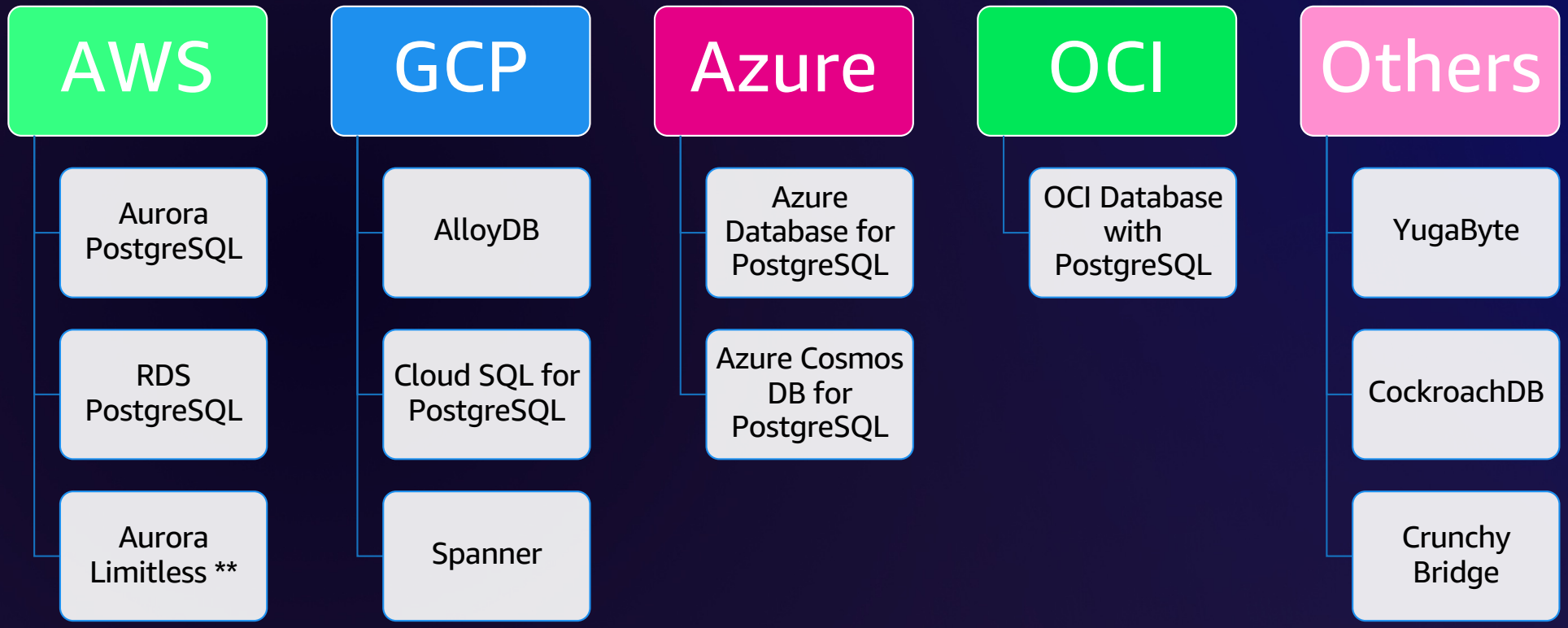
Rank Aug 2024	Rank Jul 2024	Rank Aug 2023	DBMS	Database Model	Score Aug 2024	Score Jul 2024	Score Aug 2023
1.	1.	1.	Oracle	Relational, Multi-model	1258.48	+18.12	+16.39
2.	2.	2.	MySQL	Relational, Multi-model	1026.86	-12.60	-103.59
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	815.18	+7.52	-105.64
4.	4.	4.	PostgreSQL	Relational, Multi-model	637.39	-1.52	+17.01
5.	5.	5.	MongoDB	Document, Multi-model	420.98	-8.85	-13.51
6.	6.	6.	Redis	Key-value, Multi-model	152.71	-4.06	-10.26
7.	7.	↑ 11.	Snowflake	Relational	135.97	-0.56	+15.34
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	129.83	-0.99	-10.09
9.	9.	↓ 8.	IBM Db2	Relational, Multi-model	123.00	-1.40	-16.23
10.	10.	10.	SQLite	Relational	104.79	-5.16	-25.13

Source - https://db-engines.com/en/ranking

Why PG is most lucrative option?

- Cost (No Licensing and Ecosystem)
- No vendor lock-in (True Open source)
- Modernization to cloud native Database technology (DBaaS support by major cloud vendors)
- Support for relational ,JSON (non-relational) and very strong geospatial Data Source for GIS. Supports most leading programming languages and protocols
- Compatibility with Oracle and SQL Server (PL/SQL vs PL/pgSQL)
- Vector Data Store to support Gen AI Retrieval Augmented Generation(RAG) use cases

DBaaS support by major cloud vendors



** in public preview

Pgvector Data store for Gen-AI RAG use cases

Adds support for storage, indexing, searching, metadata with choice of distance

github.com/pgvector/pgvector

Pgvector enables variety of Gen-AI use-cases

Semantic Search

Build intuitive search bar or smarter product catalogue, which understands intent

Recommendation Engine

Boost sales and user satisfaction with precise product recommendations

Sentiment/Emotion Analysis

Detect customer sentiment/emotion based on positive, negative, neutral or mixed feedback

RAG/Chatbots

Enhance customer support with AI chatbots that provide instant, relevant responses

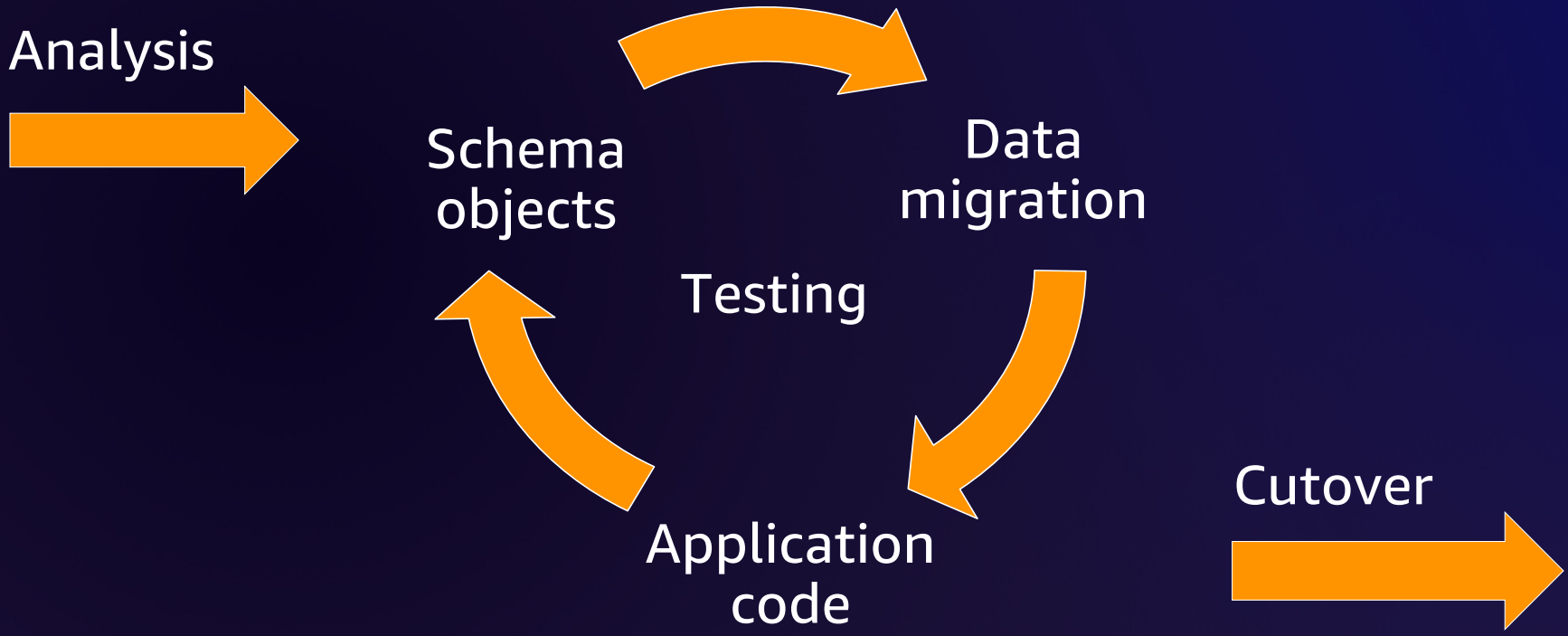
Migration Assessment and Life Cycle

Workload qualification

One way to classify the workload can be:

Simple	Complex
Single schema & few hundred objects	Multiple schemas & few thousand objects
Up to few hundred GBs of data	TBs of data
ORM	Embedded SQL
Business logic is in application	Business logic is in database
Commercial BI/ETL tools	Hand-coded Perl/C
Active dev & test team	Legacy code minimal support

A typical database migration lifecycle



Top Migration Challenges

1. Number vs Numeric

Number vs Numeric

- ln Oracle, the largest number data type can hold 38 digits
- A PostgreSQL NUMERIC can hold 131072 before and 16383 after the decimal point.
- By default, many DBAs pick the numeric data type in PostgreSQL for converting all number columns in Oracle. It is not the same as NUMBER.
- PostgreSQL offers several other number data types: smallint (2 bytes), integer (4 bytes) and bigint 8 bytes. Instead of choosing numeric datatype as default, consider using the right datatype based on the range and storage.

Numeric vs INT - PostgreSQL

```
postgres=# SELECT pg_column_size('999999999'::NUMERIC) AS "NUMERIC length", pg_column_size('999999999'::INT) AS
 "INT length";
 NUMERIC length | INT length
----------------+------------
             12 |          4
(1 row)
```

INT and BIGINT have a fixed length: INT has 4 bytes, and BIGINT has 8 bytes. However, the NUMERIC type is of variable length and stores 0–255 bytes as needed. So NUMERIC needs more space to store the data.

Numeric vs INT - PostgreSQL

```
postgres=# select count(1) from test_numeric1 t1, test_numeric2 t2 where t1.id=t2.id;
  count
---------
 1000000
(1 row)

Time: 510.642 ms
postgres=#
postgres=#
postgres=#
postgres=# select count(1) from test_int1 t1, test_int2 t2 where t1.id=t2.id;
  count
---------
 1000000
(1 row)

Time: 389.999 ms
```

Recommendations

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

https://aws.amazon.com/blogs/database/convert-the-number-data-type-from-oracle-to-postgresql-part-1/

2. Null vs Empty string

Null vs Empty string

- Oracle treats nulls and empty string the same by automatically converting the empty strings to null. PostgreSQL on the other hand treats nulls and empty strings differently.
- If your application is inserting empty strings they are automatically treated as null and your application will see different behavior in PostgreSQL.

Oracle(NULL= '')

```
SQL> CREATE TABLE test_null (id INT, name VARCHAR(20));

Table created.

SQL> INSERT INTO test_null VALUES (1, 'test');
INSERT INTO test_null VALUES (2, null);
INSERT INTO test_null VALUES (3, '');

1 row created.

SQL>
1 row created.

SQL>
1 row created.

SQL> SELECT * FROM test_null WHERE name IS null;

        ID NAME
---------- --------------------
         2
         3

SQL>
```

PostgreSQL (NULL != '')

```
postgres=# CREATE TABLE test_null (id INT, name VARCHAR(20));
CREATE TABLE
postgres=# INSERT INTO test_null VALUES (1, 'test');
INSERT 0 1
postgres=# INSERT INTO test_null VALUES (2, null);
INSERT 0 1
postgres=# INSERT INTO test_null VALUES (3, '');
INSERT 0 1
postgres=# SELECT * FROM test_null WHERE name IS null;
 id | name
----+------
  2 |
(1 row)

postgres=# SELECT * FROM test_null WHERE name='';
 id | name
----+------
  3 |
(1 row)
```

Business Logic

```
IF column_name IS NULL THEN
        something_here;
    END IF;

IF column1!=column2 THEN
        something_here;
    END IF;
```

Input Expression	Oracle Condition Evaluation	PostgreSQL Condition Evaluation
'' IS NULL	True	False
'' IS NOT NULL	False	True
NVL('', 'A')	A	Not available in PostgreSQL. You can use the Orafce extension NVL function. You must use NULLIF with NVL function to parse empty string as NULL.
COALESCE('', 'A')	A	'' (empty string)
DECODE('', NULL, 'A', 'B')	A	Not available in PostgreSQL. You can use the Orafce extension DECODE function. You must use NULLIF with the DECODE function to parse empty strings as NULL. Another approach is to convert DECODE to CASE.
CASE WHEN '' IS NULL	True	False
CASE '' WHEN NULL	False	False

Recommendations

```
postgres=# CREATE TABLE test_null (id INT, name VARCHAR(20));
CREATE TABLE
postgres=# INSERT INTO test_null VALUES (1, 'test');
INSERT 0 1
postgres=# INSERT INTO test_null VALUES (2, null);
INSERT 0 1
postgres=# INSERT INTO test_null VALUES (3, '');
INSERT 0 1
postgres=# SELECT * FROM test_null WHERE name IS null;
 id | name
----+------
  2 |
(1 row)

postgres=# SELECT * FROM test_null WHERE name='';
 id | name
----+------
  3 |
(1 row)

postgres=# SELECT * FROM test_null WHERE nullif(name,'') is null;
 id | name
----+------
  2 |
  3 |
(2 rows)
```

Recommendations

```
postgres=# SELECT coalesce('', 'A') res;
 res
-----

(1 row)

Time: 44.032 ms
postgres=# SELECT coalesce(nullif('', ''), 'a') res;
 res
-----
 a
(1 row)
```

```
postgres=# CREATE OR REPLACE FUNCTION nvl(text, text)
postgres-# RETURNS text
postgres-# LANGUAGE sql
postgres-# AS $$
postgres$#     select coalesce(NULLIF($1,''), NULLIF($2,''))
postgres$# $$;
CREATE FUNCTION
Time: 42.499 ms
postgres=# SELECT nvl(NULL, 'A') res;
 res
-----
 A
(1 row)
```

3. Null with composite index

Null behavior composite Index

- Many applications using Oracle Database are designed to handle nulls according to how Oracle treats nulls. In Oracle 2 nulls are treated as equal as both the nulls values are “unknown” to Oracle.

- However, in PostgreSQL two nulls are not equal so it allows duplicate null values considering other column is different. In PostgreSQL, the null value represents an unknown value, and it cannot decide whether two unknown values are equal.

Null behavior composite Index

Oracle

```
SQL> create table testnull(ID int, name varchar2(15));

Table created.

SQL> alter table testnull add constraint uniq_testnull unique (id,name);

Table altered.

SQL> insert into testnull values(1,'A');

1 row created.

SQL> insert into testnull(id) values (2);

1 row created.

SQL> insert into testnull(id) values (2);
insert into testnull(id) values (2)
*
ERROR at line 1:
ORA-00001: unique constraint (CO.UNIQ_TESTNULL) violated


SQL> select * from testnull;

        ID NAME
---------- ---------------
         1 A
         2
```

PostgreSQL

```
postgres=> create table testnull(ID int, name varchar);
CREATE TABLE
postgres=> alter table testnull add constraint uniq_testnull unique (id,name);
ALTER TABLE
postgres=> insert into testnull values(1,'A');
INSERT 0 1
postgres=> insert into testnull(id) values (2);
INSERT 0 1
postgres=> insert into testnull(id) values (2);
INSERT 0 1
postgres=> select * from testnull;
 id | name
----+------
  1 | A
  2 |
  2 |
(3 rows)
```

Recommendations

Partial Index

```
postgres=> delete from testnull;
DELETE 3
postgres=> create unique index partial_unq_testnull on testnull(id) where name is null;
CREATE INDEX
postgres=> insert into testnull values(1,'A');
INSERT 0 1
postgres=> insert into testnull(id) values (2);
INSERT 0 1
postgres=> insert into testnull(id) values (2);
ERROR:  duplicate key value violates unique constraint "partial_unq_testnull"
DETAIL:  Key (id)=(2) already exists.
```

PostgreSQL15 - NULLS NOT DISTINCT

The NULLS NOT DISTINCT option modifies this and causes the index to treat nulls as equal

```
postgres=# DROP TABLE test_null;
DROP TABLE
postgres=# CREATE TABLE test_null (id INT, name VARCHAR(20));
CREATE TABLE
postgres=# CREATE UNIQUE INDEX N_distinct ON test_null(id, name);
CREATE INDEX
postgres=#
postgres=#
postgres=# INSERT INTO test_null VALUES (1, 'test');
INSERT 0 1
postgres=# INSERT INTO test_null VALUES (1, null);
INSERT 0 1
postgres=# INSERT INTO test_null VALUES (1, null);
INSERT 0 1
postgres=#
postgres=# DELETE FROM test_null;
DELETE 3
postgres=#
postgres=# CREATE UNIQUE INDEX N_N_distinct ON test_null(id, name) NULLS NOT DISTINCT;
CREATE INDEX
postgres=#
postgres=# INSERT INTO test_null VALUES (1, 'test');
INSERT 0 1
postgres=# INSERT INTO test_null VALUES (1, null);
INSERT 0 1
postgres=# INSERT INTO test_null VALUES (1, null);
ERROR:  duplicate key value violates unique constraint "n_n_distinct"
DETAIL:  Key (id, name)=(1, null) already exists.
```

Recommendations – “expr = NULL”

```
postgres=# SELECT * FROM test_null WHERE name=null;
 id | name
----+------
(0 rows)

postgres=# SET transform_null_equals TO on;
SET
postgres=# SELECT * FROM test_null WHERE name=null;
 id | name
----+------
  2 |
(1 row)

postgres=# SELECT * FROM test_null WHERE nullif(name,'')=null;
 id | name
----+------
  2 |
  3 |
(2 rows)
```

4. Date/Timestamp

Date/Timestamp issues

- Oracle’s date keeps timestamp information at a resolution of seconds.
- PostgreSQL’s date keeps the date without a time. Oracle’s date and timestamp types should both be converted to PostgreSQL timestamp to preserve the same level of information.
- The PostgreSQL type Timestamptz(Timestamp with time zone) is different from the Oracle Timestamp with time zone. It is equivalent to Oracle’s Timestamp with local time zone

Date/Timestamp issues

Oracle

```
SQL> SET SERVEROUTPUT ON ;
BEGIN
DBMS_OUTPUT.PUT_LINE('Start : ' || to_char(SYSDATE, 'YYYY-MM-DD HH24:MI:SS'));
dbms_lock.sleep(30);
DBMS_OUTPUT.PUT_LINE('End : ' || to_char(SYSDATE, 'YYYY-MM-DD HH24:MI:SS'));
END;
/SQL>   2    3    4    5    6
Start : 2024-09-16 16:01:27
End : 2024-09-16 16:01:57

PL/SQL procedure successfully completed.
```

PostgreSQL

```
postgres=# DO
postgres-# $BODY$
postgres$# BEGIN
postgres$# RAISE NOTICE 'clock_timestamp()       : %', clock_timestamp();
postgres$# RAISE NOTICE 'statement_timestamp()   : %', statement_timestamp();
postgres$# RAISE NOTICE 'now()                   : %', now();
postgres$# RAISE NOTICE 'current_timestamp       : %', current_timestamp;
postgres$# RAISE NOTICE 'transaction_timestamp() : %', transaction_timestamp();
postgres$# RAISE NOTICE '';
postgres$# RAISE NOTICE 'sleep for 15 secs and see the difference below: %', pg_sleep(15);
postgres$# RAISE NOTICE '';
postgres$# RAISE NOTICE 'clock_timestamp()       : %', clock_timestamp();
postgres$# RAISE NOTICE 'statement_timestamp()   : %', statement_timestamp();
postgres$# RAISE NOTICE 'now()                   : %', now();
postgres$# RAISE NOTICE 'current_timestamp       : %', current_timestamp;
postgres$# RAISE NOTICE 'transaction_timestamp() : %', transaction_timestamp();
postgres$# END;
postgres$# $BODY$;
NOTICE:  clock_timestamp()       : 2024-09-16 11:03:19.217318-05
NOTICE:  statement_timestamp()   : 2024-09-16 11:03:19.06683-05
NOTICE:  now()                   : 2024-09-16 11:03:19.06683-05
NOTICE:  current_timestamp       : 2024-09-16 11:03:19.06683-05
NOTICE:  transaction_timestamp() : 2024-09-16 11:03:19.06683-05
NOTICE:
NOTICE:  sleep for 15 secs and see the difference below:
NOTICE:
NOTICE:  clock_timestamp()       : 2024-09-16 11:03:34.240107-05
NOTICE:  statement_timestamp()   : 2024-09-16 11:03:19.06683-05
NOTICE:  now()                   : 2024-09-16 11:03:19.06683-05
NOTICE:  current_timestamp       : 2024-09-16 11:03:19.06683-05
NOTICE:  transaction_timestamp() : 2024-09-16 11:03:19.06683-05
DO
```

https://aws.amazon.com/blogs/database/converting-the-sysdate-function-from-oracle-to-postgresql/

Date/Timestamp issues

Oracle

```
SQL> select to_date(sysdate) from dual;

TO_DATE(S
---------
16-SEP-24
```

date_col1 – date_col2 = **number**

PostgreSQL

```
postgres=# select to_date(now());
ERROR:  function to_date(timestamp with time zone) does not exist
LINE 1: select to_date(now());
               ^
```

```
postgres=# SELECT to_date(now()::text, 'yyyy-mm-dd');
  to_date
------------
 2024-09-29
(1 row)
```

date_col1 – date_col2 = **integer**

tstamp_col1 – tstamp_col2 = **interval**

date_col1 – tstamp_col2 = **interval**

tstamp_col1 – date_col1 = **interval**

Date/Timestamp issues

```
postgres=> CREATE EXTENSION orafce;
ERROR:  extension "orafce" already exists
postgres=> set search_path TO oracle,"$user", public, pg_catalog;
SET
postgres=> create table oracle_date(col1 date);
ERROR:  relation "oracle_date" already exists
postgres=> drop table oracle_date;
DROP TABLE
postgres=> create table oracle_date(col1 date);
CREATE TABLE
postgres=> insert into oracle_date values('2022-08-25 11:11:11'::date);
INSERT 0 1
postgres=> select * from oracle_date;
        col1
---------------------
 2022-08-25 11:11:11
(1 row)
```

With orafce

```
mydb=> set search_path TO oracle,"$user", public, pg_catalog;
SET
mydb=> create table oracle_date(col1 date);
CREATE TABLE
mydb=> insert into oracle_date values('2022-08-25 11:11:11'::date);
INSERT 0 1
mydb=> select * from oracle_date;
    col1
------------
 2022-08-25
(1 row)
```

Without orafce

Date/Timestamp issues

- TRUNC(timestamp) returns a DATE…
  - ..but only a date by truncating time portion to a date

- DATE_TRUNC() works more like Oracle trunc()
  - …but the parameters are "backward" and
  - …there are a whole new set of datepart abbreviations

    date_trunc('mon', sysdate) = 2024-09-16 00:00:00
  - …and….

Oracle

```
trunc(t, 'day') = Sunday
```

PostgreSQL

```
date_trunc('week', d) = Monday
date_trunc('week', d+1)-1 = Sunday
date_trunc('week', d+1)-1+6 = Saturday
```

5. Synonyms

Synonyms Challenges

- In Oracle Database you can create Synonyms (public or private) which enables
  - referencing of objects easier i.e. to not fully qualify cross schema objects
  - eliminates the hard coding
  - hides object details like object name, owner, database link
- Enter PostgreSQL – **search_path**
  - Search_path can satisfies the synonyms use cases. It is list of schemas to look. The first matching object in the search_path will be returned and if there is no match an error is reported.

```
postgres=# \dt private.tab
        List of relations
 Schema  | Name | Type  |  Owner
---------+------+-------+----------
 private | tab  | table | postgres
(1 row)
```

```
postgres=# grant usage on SCHEMA private to test;
GRANT
postgres=# grant select on private.tab to test;
GRANT
```

Synonyms Challenges

```
postgres=> select current_user;
 current_user
--------------
 test
(1 row)

postgres=> show search_path ;
   search_path
-----------------
 "$user", public
(1 row)

postgres=> select * from tab ;
ERROR:  relation "tab" does not exist
LINE 1: select * from tab ;
                      ^
```

```
postgres=> set search_path to "$user", public, private;
SET
postgres=> show search_path ;
       search_path
-------------------------
 "$user", public, private
(1 row)

postgres=> select * from tab ;
 id
----
(0 rows)
```

6. Query Hints

Hints in my application query ?

- PostgreSQL treats Oracle hints as comments.
- The execution plan with the lowest cost is executed. The planner does its best to select the best execution plan, in some cases doesn't generate the best execution plan
- One approach to fixing this problem is to put query hints into your application code, but this approach is widely discouraged because it makes applications more brittle and harder to maintain
- For this purpose, PostgreSQL users can use the "**pg_hint_plan"** extension to provide directives such as "scan method," "join method," "join order,", or "row number correction," to the optimizer.
- SET pg_hint_plan.enable_hint = true;

https://aws.amazon.com/blogs/database/monitor-query-plans-for-amazon-aurora-postgresql/

Hints continue...

```
postgres=> EXPLAIN SELECT * FROM pgbench_branches b JOIN pgbench_accounts a ON b.bid = a.bid ORDER BY a.aid;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.57..60745.80 rows=801909 width=461)
   ->  Index Scan using pgbench_accounts_pkey on pgbench_accounts a  (cost=0.42..41866.20 rows=801909 width=97)
   ->  Memoize  (cost=0.14..0.16 rows=1 width=364)
         Cache Key: a.bid
         Cache Mode: logical
         ->  Index Scan using pgbench_branches_pkey on pgbench_branches b  (cost=0.13..0.15 rows=1 width=364)
               Index Cond: (bid = a.bid)
(7 rows)
```

```
postgres=> SET pg_hint_plan.enable_hint = true;
SET
```

```
postgres=> /*+ HashJoin(a b) */ EXPLAIN SELECT * FROM pgbench_branches b JOIN pgbench_accounts a ON b.bid = a.bid ORDER BY a.aid;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Gather Merge  (cost=190003.48..267972.35 rows=668258 width=461)
   Workers Planned: 2
   ->  Sort  (cost=189003.45..189838.77 rows=334129 width=461)
         Sort Key: a.aid
         ->  Hash Join  (cost=45.02..19012.04 rows=334129 width=461)
               Hash Cond: (a.bid = b.bid)
               ->  Parallel Seq Scan on pgbench_accounts a  (cost=0.00..17625.29 rows=334129 width=97)
               ->  Hash  (cost=44.92..44.92 rows=8 width=364)
                     ->  Index Scan using pgbench_branches_pkey on pgbench_branches b  (cost=0.13..44.92 rows=8 width=364)
(9 rows)
```

7. Autonomous transaction

Autonomous transaction

- Autonomous Transactions is used for auditing access to stores and commits this information irrespective of whether the original transaction completes successfully on not.
- PostgreSQL does not explicitly support Autonomous Transactions.
- One of the workaround is to use the PostgreSQL dblink or dblink_fdw extensions.

Recommendations

Oracle

```
CREATE OR REPLACE PROCEDURE insert_critical_info(v_critical_info varchar2)
IS PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO critical_info_table (critical_info) VALUES (v_critical_info);
commit;
END; /
```

PostgreSQL

```
CREATE OR REPLACE FUNCTION insert_critical_info(v_critical_info TEXT)
RETURNS void AS $BODY$
DECLARE v_sql text;
BEGIN
PERFORM dblink_connect('myconn', 'loopback_dblink’);
v_sql := format('INSERT INTO critical_info_table (critical_info) VALUES
(%L)', v_critical_info);
PERFORM dblink_exec('myconn', v_sql);
PERFORM dblink_disconnect('myconn’);
END;
$BODY$
LANGUAGE plpgsql;
```

https://aws.amazon.com/blogs/database/migrating-oracle-autonomous-transactions-to-postgresql/

But…

- The PostgreSQL dblink is session-specific. Any dblink opened in one session cannot be used in a different session.
- This means that each session will have to open a new DB connection and this may impacts application performance and you may need to increase max_connections parameter value.

8. Sequence Caching

Sequence caching problem

- In Oracle Database sequence caching happens at the system level, whereas PostgreSQL sequence caching happens at the session level.
- In PostgreSQL if multiple sessions are trying get sequence values with sequence has caching enabled then the sequence values will not be in the order. This behavior will cause application issue if the application has dependency on the order of the values generated by sequence.

Session A- T1

```
SQL> create sequence oraseq start with 100 increment by 1 cache 10;

Sequence created.

SQL> select oraseq.nextval from dual;

   NEXTVAL
----------
       100
```

Session B -T2

```
SQL> select oraseq.nextval from dual;

   NEXTVAL
----------
       101
```

Back to Session A- T3

```
SQL> select oraseq.nextval from dual;

   NEXTVAL
----------
       102
```

Sequence caching problem

Session A -T1

```
mydb=> Create sequence pgseq start with 100 increment by 1 cache 10;
CREATE SEQUENCE
Time: 5.429 ms
mydb=> Select nextval('public. Pgseq');
 nextval
---------
     100
(1 row)
```

Session B –T2

```
mydb=> Select nextval('public. Pgseq');
 nextval
---------
     110
(1 row)
```

Back to Session A - T3

```
mydb=> Select nextval('public. Pgseq');
 nextval
---------
     101
(1 row)
```

If both sessions are inserting to same table concurrently, value 111 from session B can be inserted before value 102 from session A

Sequence caching problem

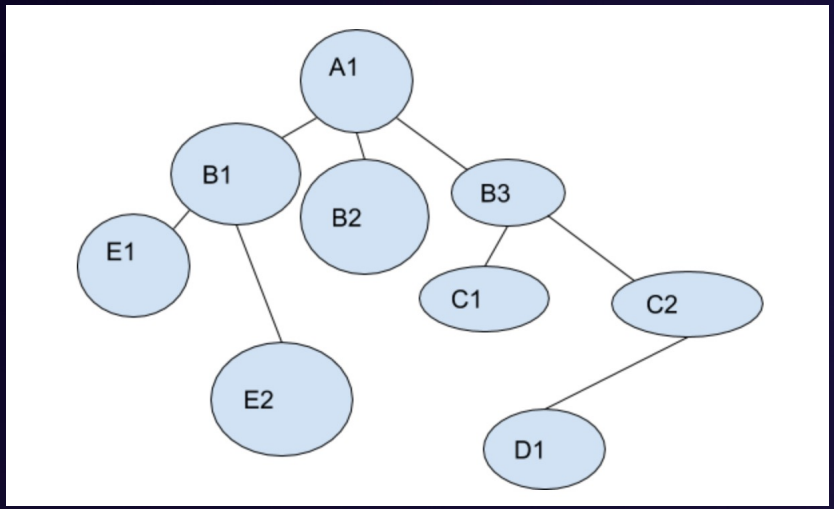
```
CREATE TABLE seq (id bigint NOT NULL);

INSERT INTO seq (id) VALUES (0);

CREATE FUNCTION next_val() RETURNS bigint
   LANGUAGE sql AS
'UPDATE seq SET id = id + 1 RETURNING id';
```

Session B- T2

9. Hierarchical Queries

Hierarchical Queries

- In Oracle, hierarchical queries are used to query data that has a parent-child relationship where each child can have only one parent, whereas a parent can have multiple children.
- Although PostgreSQL doesn't have functions or predefined keywords to handle the hierarchical queries directly, you can define custom solutions with the help of the tablefunc extension and CTEs.



Hierarchical Queries

Oracle

```
SELECT   emp_no,ename,job,level
FROM hier_test
  CONNECT BY PRIOR emp_no = manager_no
  START WITH manager_no IS NULL
  order by level ;
```

Script Output | Query Result

SQL | All Rows Fetched: 9 in 0.25 seconds

	EMP_NO	ENAME	JOB	LEVEL
1	10	A1	CEO	1
2	11	B1	VP HARDWARE	2
3	12	B2	VP ADMIN	2
4	13	B3	VP DEVELOPMENT	2
5	15	C2	DIRECTOR DEVELOPMENT	3
6	14	C1	DIRECTOR DEVELOPMENT	3
7	18	E2	ENGINEER HARDWARE	3
8	17	E1	ENGINEER HARDWARE	3
9	16	D1	MANAGER DEVELOPMENT	4

PostgreSQL

```
WITH RECURSIVE cte AS (
  SELECT emp_no, ename,job, manager_no, 1 AS level
  FROM   hier_test
  where manager_no is null
  UNION  ALL
  SELECT e.emp_no, e.ename, e.job,e.manager_no, c.level + 1
  FROM   cte c
  JOIN   hier_test e ON e.manager_no = c.emp_no
  )
  SELECT emp_no,ename,job,level
  FROM   cte
  order by level;
```

Explain | Query Editor | Query History | Notifications | Messages | Data Output

	emp_no integer	ename character varying (5)	job character varying (20)	level integer
1	10	A1	CEO	1
2	11	B1	VP HARDWARE	2
3	12	B2	VP ADMIN	2
4	13	B3	VP DEVELOPMENT	2
5	14	C1	DIRECTOR DEVELOPMENT	3
6	15	C2	DIRECTOR DEVELOPMENT	3
7	17	E1	ENGINEER HARDWARE	3
8	18	E2	ENGINEER HARDWARE	3
9	16	D1	MANAGER DEVELOPMENT	4

https://aws.amazon.com/blogs/database/migrate-oracle-hierarchical-queries-to-amazon-aurora-postgresql/

Bonus tips

Oracle	PostgreSQL
Global temporary tables (GTT)	PostgreSQL has the concept of Local Temporary Table but not a Global temporary table. **PGTT** extension emulates the Oracle implementation of Global Temporary Table using underlying regular local temporary tables
Bulk collect	Array_Agg()
External tables	Copy command, file_fdw

https://aws.amazon.com/blogs/database/key-considerations-while-migrating-bulk-operations-from-oracle-to-postgresql/

https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/migrate-oracle-external-tables-to-amazon-aurora-postgresql-compatible.html

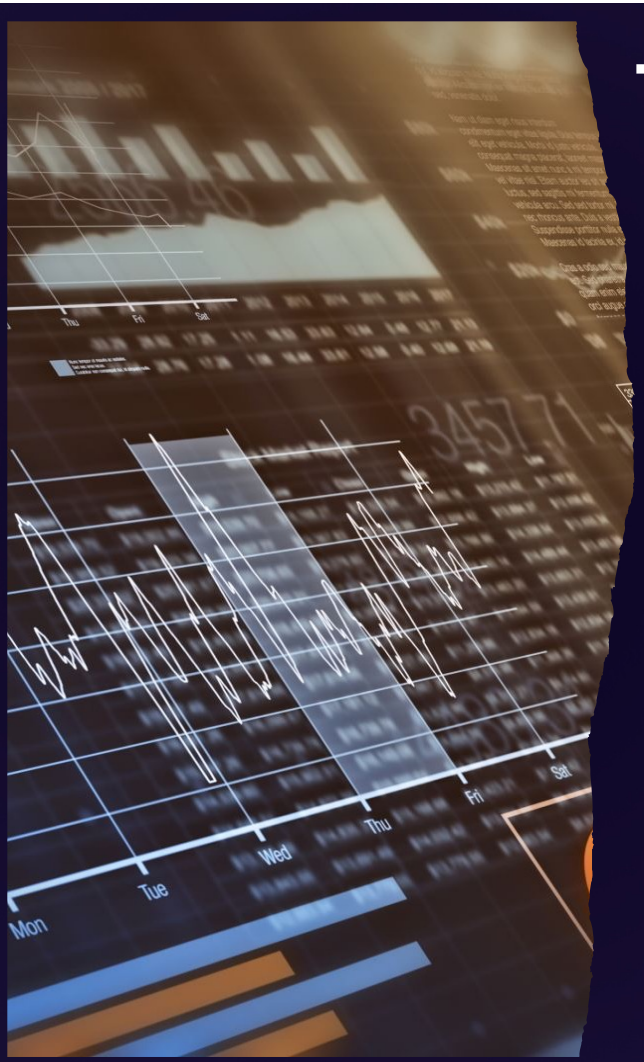
Thank you

Baji Shaik
https://www.linkedin.com/in/baji-shaik/

Sameer Malik
https://www.linkedin.com/in/sameer malik/

Appendix

Time series use case and data



Time series use case and data

- Applications designed to store and query large amounts of time series data such as collecting metrics from a fleet of internet of things (IoT) devices. Others may have a single table of time series data such as a transaction history table in an OLTP database.
- Interval Partitioning in Oracle
  - New partitions will be created automatically based on interval criteria when the data is inserted to the table. We don't need to create the future partitions e.g.
  - PARTITION BY RANGE (EVENT_TIMESTAMP) INTERVAL(NUMTODSINTERVAL(1,'DAY'))
- No support for Interval partition in PostgreSQL
  - extensions **pg_partman** and **pg_cron** for maintaining highly partitioned time series data
- Partitioning enhancements in PostgreSQL
    - ability to add partitions without the need to take an exclusive lock on the parent table and changed the planning algorithm and improvement in partition pruning.

Time series use case and data?

Configuring partitions using the create_parent function

```
SELECT partman.create_parent( p_parent_table => ‘event’,
p_control => ‘event_timestamp',
p_type =>'native',
p_interval=> 'daily’,
p_premake => 30);
```

Configuring partition maintenance using the run_maintenance_proc function

```
UPDATE partman.part_config
SET infinite_time_partitions = true,
retention = ‘3 months’,
retention_keep_table=true WHERE parent_table =‘events’;
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

Column & row level security/encryption?

Column & row level security/encryption?

- Oracle provides column and row level security natively with the Database options such as Virtual Private Database(VPD) and Oracle label Security (OLS)
- Table-level security can be implemented in PostgreSQL at
  - Column-level security (using views)
  - Column level encryption (using pgcrytpo extension)
  - Row-level security (creating policy and enabling ROW LEVEL SECURITY)
- Row level Security example
  - ALTER TABLE employee ENABLE ROW LEVEL SECURITY;
  - CREATE POLICY emp_rls_policy ON employee FOR ALL TO PUBLIC
    USING (ename=current_user);
  - Superusers sees all rows due to the BYPASSRLS attribute on the superuser role by default
- Another alternate to implement VPD is with set_config()/current_setting() functions: You can set_config() in the application and modify the view definition to fetch the config

Bonus tips – Case folding

Oracle Data Dictionary folds and stores all meta-data in Uppercase. In PostgreSQL all the meta-date is folds to lowercase

Some Oracle to PostgreSQL database migration tools carry the uppercase from Oracle to PostgreSQL. PostgreSQL will explicitly require to double quote the column.

```
postgres-# \d emp
                    Table "public.emp"
 Column |       Type        | Collation | Nullable | Default
--------+-------------------+-----------+----------+---------
 ID     | character varying |           |          |
 name   | character varying |           |          |
```

```
postgres=# select id, name from emp;
ERROR:  column "id" does not exist
LINE 1: select id, name from emp;
               ^
```

```
postgres=# select "ID", name from emp;
 ID | name
----+------
(0 rows)
```

https://aws.amazon.com/blogs/database/manage-case-insensitive-data-in-postgresql/