Oracle System Table Comparision with Postgres System Table

Comparing Oracle system tables with PostgreSQL system tables involves understanding the equivalent tables and their usage in each database. Below is a step-by-step comparison with examples.

1. Overview of System Tables

Oracle:

- **ALL_TABLES**: Information about all tables accessible to the user.
- **ALL_INDEXES**: Information about all indexes accessible to the user.
- **ALL_USERS**: Information about all users in the database.
- **ALL_TAB_COLUMNS**: Information about all columns of tables accessible to the user.
- **V$ views**: Dynamic performance views (e.g., V$SESSION, V$SQL).

PostgreSQL:

- **pg_tables**: Information about tables in the database.
- **pg_indexes**: Information about indexes in the database.
- **pg_user**: Information about users in the database.
- **information_schema.columns**: Information about columns of tables in the database.
- **pg_stat_activity**, **pg_stat_user_tables**: Dynamic performance views.

2. Accessing Table Information

Oracle:

```
SELECT table_name, tablespace_name
FROM all_tables
WHERE owner = 'HR';
```

PostgreSQL:

```
SELECT tablename, schemaname
FROM pg_tables
WHERE schemaname = 'public';
```

3. Accessing Index Information

Oracle:

```
SELECT index_name, table_name
FROM all_indexes
WHERE table_owner = 'HR';
```

PostgreSQL:

```
SELECT indexname, tablename
FROM pg_indexes
```

```
WHERE schemaname = 'public';
```

4. Accessing User Information

Oracle:

```
SELECT username, user_id
FROM all_users;
```

PostgreSQL:

```
SELECT usename, usesysid
FROM pg_user;
```

5. Accessing Column Information

Oracle:

```
SELECT table_name, column_name, data_type
FROM all_tab_columns
WHERE owner = 'HR';
```

PostgreSQL:

```
SELECT table_name, column_name, data_type
FROM information_schema.columns
WHERE table_schema = 'public';
```

6. Dynamic Performance Views

Oracle: V$SESSION

```
SELECT sid, serial#, status, username, osuser
FROM v$session;
```

PostgreSQL: pg_stat_activity

```
SELECT pid, usename, application_name, state
FROM pg_stat_activity;
```

7. System Catalog Comparison

Oracle: DBA_TABLES

```
SELECT table_name, owner, num_rows
FROM dba_tables
WHERE owner = 'HR';
```

PostgreSQL: pg_class, pg_namespace

```
SELECT c.relname AS table_name, n.nspname AS schema_name, c.reltuples AS num_rows
FROM pg_class c
JOIN pg_namespace n ON c.relnamespace = n.oid
WHERE n.nspname = 'public';
```

8. Constraints Information

Oracle:

```
SELECT constraint_name, constraint_type, table_name
FROM all_constraints
WHERE owner = 'HR';
```

PostgreSQL:

```
SELECT conname AS constraint_name, contype AS constraint_type, conrelid::regclass AS table_name
FROM pg_constraint
WHERE connamespace = 'public'::regnamespace;
```

9. Viewing Stored Procedures and Functions

Oracle:

```
SELECT object_name, object_type
FROM all_objects
WHERE object_type IN ('PROCEDURE', 'FUNCTION') AND owner = 'HR';
```

PostgreSQL:

```
SELECT routine_name, routine_type
FROM information_schema.routines
WHERE routine_schema = 'public';
```

10. Viewing Sequences

Oracle:

```
SELECT sequence_name
FROM all_sequences
WHERE sequence_owner = 'HR';
```

PostgreSQL:

```
SELECT sequence_name
FROM information_schema.sequences
WHERE sequence_schema = 'public';
```

Example Comparisons in Detail

Example 1: Table Information

Oracle:

```
SELECT table_name, tablespace_name
FROM all_tables
WHERE owner = 'HR';
```

Explanation:

- table_name: Name of the table.
- tablespace_name: Name of the tablespace where the table resides.

PostgreSQL:

```
SELECT tablename, schemaname
FROM pg_tables
WHERE schemaname = 'public';
```

Explanation:

- tablename: Name of the table.
- schemaname: Name of the schema where the table resides.

Example 2: Index Information

Oracle:

```
SELECT index_name, table_name
FROM all_indexes
WHERE table_owner = 'HR';
```

Explanation:

- index_name: Name of the index.
- table_name: Name of the table on which the index is created.

PostgreSQL:

```
SELECT indexname, tablename
FROM pg_indexes
WHERE schemaname = 'public';
```

Explanation:

- indexname: Name of the index.

- tablename: Name of the table on which the index is created.

Conclusion

The comparison shows that while both Oracle and PostgreSQL provide comprehensive system tables and views for database management, there are differences in naming conventions and the structure of these tables. Migrating from Oracle to PostgreSQL involves mapping these differences and adapting queries and tools accordingly. The use of PostgreSQL-specific tools and views can help manage and optimize the database post-migration.

Comparison of Monitoring and Maintenance in Oracle vs PostgreSQL

Both Oracle and PostgreSQL provide robust tools and features for monitoring and maintenance. Below is a detailed comparison between the two databases.

Monitoring

Oracle:

- **Dynamic Performance Views (V$ views)**: Oracle uses dynamic performance views to monitor the database.
  - **V$SESSION**: Provides information about active sessions.
  - **V$SQL**: Shows statistics on SQL statements.
  - **V$SYSTEM_EVENT**: Provides information about system events.
  - **OEM (Oracle Enterprise Manager)**: A comprehensive tool for database monitoring and management.

PostgreSQL:

- **Statistics Collector Views**: PostgreSQL uses various system views to monitor the database.
  - **pg_stat_activity**: Displays information about the current database connections.
  - **pg_stat_statements**: Provides query performance statistics.
  - **pg_stat_database**: Shows database-wide statistics.
  - **pgAdmin**: A powerful, open-source management tool for PostgreSQL.

Example: Monitoring Active Sessions

Oracle:

```
SELECT sid, serial#, status, username, osuser, machine
FROM v$session
WHERE status = 'ACTIVE';
```

PostgreSQL:

```
SELECT pid, usename, application_name, state, query
```

```
FROM pg_stat_activity
WHERE state = 'active';
```

Maintenance

Oracle:

- **Automatic Workload Repository (AWR)**: Collects, processes, and maintains performance statistics.
- **Automatic Database Diagnostic Monitor (ADDM)**: Analyzes AWR data to identify performance issues.
- **RMAN (Recovery Manager)**: Provides comprehensive backup and recovery capabilities.
- **DBMS_STATS**: Used to gather and manage statistics for cost-based optimization.

PostgreSQL:

- **Autovacuum**: Automatically handles routine vacuuming and analyzing.
- **pg_dump/pg_restore**: Utilities for backup and restore.
- **VACUUM**: Reclaims storage occupied by dead tuples.
- **ANALYZE**: Collects statistics about the contents of tables.

Example: Gathering Table Statistics

Oracle:

```
EXEC DBMS_STATS.GATHER_TABLE_STATS('HR', 'EMPLOYEES');
```

PostgreSQL:

```
ANALYZE employees;
```

Backup and Recovery

Oracle:

- **RMAN (Recovery Manager)**: Manages backups, restores, and recovery.
- **Data Pump**: Provides high-speed data export and import.

Example: Backup using RMAN:

```
RMAN> BACKUP DATABASE;
```

PostgreSQL:

- **pg_dump**: Utility for backing up a single database.
- **pg_basebackup**: Used for streaming backups.

Example: Backup using pg_dump:

pg_dump -U username -F c -b -v -f backup_file.sql database_name

Index Maintenance

Oracle:

- **Rebuilding Indexes**: Oracle allows index rebuilding for maintenance.

  ALTER INDEX index_name REBUILD;

PostgreSQL:

- **Reindexing**: PostgreSQL provides REINDEX for index maintenance.

  REINDEX INDEX index_name;

Monitoring Tools Comparison

Feature	**Oracle**	**PostgreSQL**
Active Sessions	V$SESSION	pg_stat_activity
Query Performance	V$SQL, OEM	pg_stat_statements, pgAdmin
Database Statistics	V$SYSTEM_EVENT, AWR, ADDM	pg_stat_database, pg_stat_user_tables
Backup Utility	RMAN, Data Pump	pg_dump, pg_basebackup
Vacuuming/Analyzing	Automatic Maintenance Tasks, DBMS_STATS	Autovacuum, VACUUM, ANALYZE
GUI Tools	Oracle Enterprise Manager, SQL Developer	pgAdmin, DBeaver

Maintenance Tools Comparison

Task	**Oracle Commands/Tools**	**PostgreSQL Commands/Tools**
Gather Statistics	DBMS_STATS.GATHER_TABLE_STATS	ANALYZE
Backup Database	RMAN: BACKUP DATABASE	pg_dump: pg_dump -U username -F c ...
Rebuild Indexes	ALTER INDEX index_name REBUILD	REINDEX INDEX index_name
Manage Partitions	ALTER TABLE table_name ADD PARTITION	CREATE TABLE ... PARTITION BY ...

Detailed Examples

Example 1: Monitoring Active Sessions

Oracle:

```
SELECT sid, serial#, status, username, osuser, machine, program
FROM v$session
WHERE status = 'ACTIVE';
```

Explanation:

- sid: Session identifier.
- serial#: Session serial number.
- status: Status of the session.
- username: Username of the session.
- osuser: Operating system user.
- machine: Machine from which the session is connected.
- program: Program being run by the session.

PostgreSQL:

```
SELECT pid, usename, application_name, state, query
FROM pg_stat_activity
WHERE state = 'active';
```

Explanation:

- pid: Process ID.
- usename: Username of the session.
- application_name: Application name.
- state: State of the connection.
- query: Query being executed.

Example 2: Backup and Restore

Oracle:

- **Backup**:

  rman
  Copy code
  RMAN> BACKUP DATABASE PLUS ARCHIVELOG;

- **Restore**:

  rman
  Copy code
  RMAN> RESTORE DATABASE;

PostgreSQL:

- **Backup**:

  pg_dump -U postgres -F c -b -v -f mydb.backup mydb

- **Restore**:

  pg_restore -U postgres -d mydb -v mydb.backup

Conclusion

Both Oracle and PostgreSQL offer comprehensive monitoring and maintenance capabilities. Oracle's tools are more integrated and enterprise-focused, while PostgreSQL provides powerful open-source options that are highly configurable. The choice between the two often depends on specific organizational needs, budget, and existing infrastructure. For migration projects, understanding these tools and their equivalents in both systems is crucial for a smooth transition.

Handling Deadlocks in Oracle vs PostgreSQL

Deadlocks occur when two or more transactions block each other by holding locks on resources that the other transactions need. Both Oracle and PostgreSQL have mechanisms to detect and handle deadlocks.

Oracle

Deadlock Detection

Oracle automatically detects deadlocks and resolves them by rolling back one of the transactions involved in the deadlock. The database chooses the transaction to roll back based on the cost of the transactions.

Handling Deadlocks

1. **Automatic Detection and Resolution**: Oracle detects deadlocks and resolves them by rolling back the transaction with the least cost. An ORA-00060 error is generated, and one of the sessions involved in the deadlock is chosen as the victim and rolled back.
2. **Error Logging**: Oracle logs deadlock errors in the alert log and trace files for further investigation.

Example Scenario:

- **Transaction 1**:

  SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
  UPDATE employees SET salary = salary + 1000 WHERE emp_id = 1;

- **Transaction 2**:

  SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
  UPDATE employees SET salary = salary + 1000 WHERE emp_id = 2;

  If Transaction 1 then tries to update emp_id = 2 and Transaction 2 tries to update emp_id = 1, a deadlock occurs. Oracle will detect this and roll back one of the transactions.

  PostgreSQL

  Deadlock Detection

  PostgreSQL also automatically detects deadlocks and resolves them by rolling back one of the transactions involved. PostgreSQL uses a deadlock detection algorithm that periodically checks for cycles in the lock graph.

  Handling Deadlocks

1. **Automatic Detection and Resolution**: PostgreSQL detects deadlocks and resolves them by rolling back one of the transactions. A DeadlockDetected error is raised.
2. **Error Logging**: Deadlock errors are logged in the PostgreSQL server logs, providing details about the transactions involved.

  Example Scenario:

- **Transaction 1**:

  BEGIN;
  UPDATE employees SET salary = salary + 1000 WHERE emp_id = 1;

- **Transaction 2**:

  BEGIN;
  UPDATE employees SET salary = salary + 1000 WHERE emp_id = 2;

  If Transaction 1 then tries to update emp_id = 2 and Transaction 2 tries to update emp_id = 1, a deadlock occurs. PostgreSQL will detect this and roll back one of the transactions.

  Steps to Handle Deadlocks

  Oracle:

1. **Monitor and Log**: Regularly check the alert log and trace files for ORA-00060 errors.
2. **Transaction Design**: Ensure that transactions acquire locks in a consistent order to prevent deadlocks.
3. **Timeouts**: Use lock timeouts to avoid long waits for locks.
4. **Application Logic**: Implement retry logic in the application to handle deadlocks.

PostgreSQL:

1. **Monitor and Log**: Check the PostgreSQL logs for DeadlockDetected errors.
2. **Transaction Design**: Ensure that transactions acquire locks in a consistent order.
3. **Timeouts**: Set lock_timeout and statement_timeout to prevent long waits.
4. **Application Logic**: Implement retry logic in the application to handle deadlocks.

Example: Checking PostgreSQL Logs for Deadlocks

To check the logs in PostgreSQL for deadlocks, you can use the following SQL query to view log entries:

```
SELECT log_time, user_name, database_name, process_id, command_tag, error_severity, message
FROM pg_log
WHERE message LIKE '%deadlock detected%';
```

Summary

- **Oracle**:
  - Automatically detects and resolves deadlocks.
  - Logs errors in the alert log and trace files.
  - Recommends consistent lock ordering and application-level retry logic.
- **PostgreSQL**:
  - Automatically detects and resolves deadlocks.
  - Logs errors in the server logs.
  - Recommends consistent lock ordering, timeouts, and application-level retry logic.

Both databases provide mechanisms to detect and handle deadlocks effectively. The key to minimizing deadlocks lies in good transaction design and proper error handling at the application level.

Performance Tunning in Oracle vs Postgres

Performance tuning is essential for ensuring that databases run efficiently. Both Oracle and PostgreSQL provide a range of tools and techniques for optimizing database performance. Here’s a detailed comparison of performance tuning in Oracle versus PostgreSQL:

Performance Tuning in Oracle

Key Techniques and Tools

1. **Optimizer Hints**:
   Oracle allows the use of hints to influence the optimizer's decisions.
   Example: SELECT /*+ FULL(emp) */ emp_id, emp_name FROM employees;

2. **Indexing**:
   Various types of indexes (B-tree, bitmap, etc.).
   Creating an index:
   CREATE INDEX emp_name_idx ON employees(emp_name);

3. **Partitioning:**
   Supports range, list, and hash partitioning.
   Example:

```
CREATE TABLE employees (
    emp_id NUMBER,
    emp_name VARCHAR2(100),
    emp_dept VARCHAR2(100)
)
PARTITION BY RANGE(emp_id) (
    PARTITION p1 VALUES LESS THAN (100),
    PARTITION p2 VALUES LESS THAN (200)
);
```

4. **Materialized Views:**

Used for query optimization by storing the results of complex queries.

Example:

CREATE MATERIALIZED VIEW emp_summary AS

SELECT emp_dept, COUNT(*) FROM employees GROUP BY emp_dept;

5. **Automatic Workload Repository (AWR):**

Collects, processes, and maintains performance statistics.

Example of generating AWR report:

@?/rdbms/admin/awrrpt.sql

6. **SQL Tuning Advisor:**

Provides recommendations for SQL query optimizations.

Example of running SQL Tuning Advisor:

```
EXEC DBMS_SQLTUNE.create_tuning_task(sql_id => 'YOUR_SQL_ID');
```

7. **Automatic Database Diagnostic Monitor (ADDM):**

Analyzes AWR data and provides recommendations for performance improvement.

Example of running ADDM report:

```
@?/rdbms/admin/addmrpt.sql
```

Performance Monitoring

- **Dynamic Performance Views (V$ views)**:
  - VSQL, VSESSION, V$SYSTEM_EVENT, etc.
  - Example:

```
SELECT sql_text, elapsed_time, cpu_time FROM v$sql WHERE elapsed_time > 1000000;
```

Performance Tuning in PostgreSQL

Key Techniques and Tools

1. **Query Planner and Optimizer**:
   - PostgreSQL uses a cost-based optimizer.
   - Example of viewing the execution plan:

```
EXPLAIN ANALYZE SELECT emp_id, emp_name FROM employees WHERE emp_dept = 'Sales';
```

2. **Indexing**:
   - Supports B-tree, hash, GIN, GiST, and BRIN indexes.
   - Creating an index:

```
CREATE INDEX emp_name_idx ON employees(emp_name);
```

3. **Partitioning**:
   - Supports range, list, and hash partitioning.
   - Example:

```
sql
Copy code
CREATE TABLE employees (
    emp_id INT,
    emp_name TEXT,
    emp_dept TEXT
```

) PARTITION BY RANGE (emp_id);

CREATE TABLE employees_p1 PARTITION OF employees FOR VALUES FROM (1) TO (100);
CREATE TABLE employees_p2 PARTITION OF employees FOR VALUES FROM (101) TO (200);

4. **Materialized Views**:
- Used for query optimization by storing the results of complex queries.
- Example:

sql
Copy code
CREATE MATERIALIZED VIEW emp_summary AS
SELECT emp_dept, COUNT(*) FROM employees GROUP BY emp_dept;
REFRESH MATERIALIZED VIEW emp_summary;

5. **pg_stat_statements**:
- Extension that provides a way to track execution statistics of all SQL statements executed.
- Example of enabling and using:

sql
Copy code
CREATE EXTENSION pg_stat_statements;
SELECT query, total_time, calls, mean_time FROM pg_stat_statements ORDER BY total_time DESC LIMIT 10;

6. **AutoVacuum**:
- Automatically reclaims storage occupied by dead tuples.
- Configuration:

sql
Copy code
SET autovacuum = on;

7. **VACUUM and ANALYZE**:
- VACUUM cleans up dead tuples; ANALYZE collects statistics.
- Example:

VACUUM (VERBOSE, ANALYZE) employees;

Performance Monitoring

- **pg_stat_activity**:
  - View currently running queries and their states.
  - Example:

sql
Copy code

SELECT pid, usename, state, query FROM pg_stat_activity;

- **pg_stat_user_tables**:
  - Shows statistics for user tables.
  - Example:

SELECT relname, seq_scan, idx_scan, n_tup_ins, n_tup_upd, n_tup_del FROM pg_stat_user_tables;

Comparison Summary

Feature/Tool	Oracle	PostgreSQL
Query Optimizer	Cost-based, supports hints	Cost-based, no hints
Indexing	B-tree, bitmap, function-based, etc.	B-tree, hash, GIN, GiST, BRIN
Partitioning	Range, list, hash	Range, list, hash
Materialized Views	Supported	Supported
Monitoring Tools	AWR, ADDM, SQL Tuning Advisor	pg_stat_statements, pg_stat_activity
Performance Views	V$ views	pg_stat views
Automatic Maintenance	Automatic Workload Repository, Auto Space Management	AutoVacuum, regular VACUUM and ANALYZE
Error Handling	Detailed error messages, logs in trace files	Detailed error messages, logs in server logs
Query Execution Plan	EXPLAIN PLAN	EXPLAIN, EXPLAIN ANALYZE

Both Oracle and PostgreSQL provide robust performance tuning capabilities, but their approaches and tools differ. Oracle offers more advanced built-in tools for tuning and diagnostics, while PostgreSQL provides extensive configuration options and open-source extensions to achieve similar outcomes. Understanding the specific tools and techniques available in each can help database administrators effectively tune and optimize their systems.

Partitioning and Replication in Oracle vs Postgres

Partitioning and replication are essential features for managing large databases and ensuring high availability and scalability. Both Oracle and PostgreSQL provide robust mechanisms for partitioning and replication, but they differ in implementation and features. Here's a detailed comparison:

Partitioning

Oracle

Concepts and Types

1. **Range Partitioning**:
   - Divides data based on a range of values.
   - Example:

```
CREATE TABLE employees (
    emp_id NUMBER,
    emp_name VARCHAR2(100),
    emp_dept VARCHAR2(100)
)
PARTITION BY RANGE(emp_id) (
    PARTITION p1 VALUES LESS THAN (100),
    PARTITION p2 VALUES LESS THAN (200)
);
```

2. **List Partitioning**:
   - Divides data based on a list of values.
   - Example:

```
sql
Copy code
CREATE TABLE employees (
    emp_id NUMBER,
    emp_name VARCHAR2(100),
    emp_dept VARCHAR2(100)
)
PARTITION BY LIST(emp_dept) (
    PARTITION p_sales VALUES ('Sales'),
    PARTITION p_hr VALUES ('HR')
);
```

3. **Hash Partitioning**:
   - Distributes data evenly using a hash function.
   - Example:

```
sql
Copy code
CREATE TABLE employees (
```

```
    emp_id NUMBER,
    emp_name VARCHAR2(100),
    emp_dept VARCHAR2(100)
)
PARTITION BY HASH(emp_id) PARTITIONS 4;
```

4. **Composite Partitioning**:
   - Combines range, list, and hash partitioning.
   - Example:

```
sql
Copy code
CREATE TABLE employees (
    emp_id NUMBER,
    emp_name VARCHAR2(100),
    emp_dept VARCHAR2(100)
)
PARTITION BY RANGE(emp_id)
SUBPARTITION BY LIST(emp_dept) (
    PARTITION p1 VALUES LESS THAN (100) (
        SUBPARTITION sp_sales VALUES ('Sales'),
        SUBPARTITION sp_hr VALUES ('HR')
    ),
    PARTITION p2 VALUES LESS THAN (200) (
        SUBPARTITION sp_sales VALUES ('Sales'),
        SUBPARTITION sp_hr VALUES ('HR')
    )
);
```

Management and Maintenance

- **Partition Pruning**: Automatically excludes irrelevant partitions during queries.
- **Partition Maintenance**: Operations like adding, dropping, splitting, and merging partitions.

PostgreSQL

Concepts and Types

1. **Range Partitioning**:
   - Example:

```
CREATE TABLE employees (
    emp_id INT,
    emp_name TEXT,
    emp_dept TEXT
) PARTITION BY RANGE (emp_id);

CREATE TABLE employees_p1 PARTITION OF employees FOR VALUES FROM (1)
TO (100);
```

```
CREATE TABLE employees_p2 PARTITION OF employees FOR VALUES FROM (101) TO (200);
```

2. **List Partitioning**:
   - Example:

```
CREATE TABLE employees (
    emp_id INT,
    emp_name TEXT,
    emp_dept TEXT
) PARTITION BY LIST (emp_dept);

CREATE TABLE employees_sales PARTITION OF employees FOR VALUES IN ('Sales');
CREATE TABLE employees_hr PARTITION OF employees FOR VALUES IN ('HR');
```

3. **Hash Partitioning**:
   - Example:

```
CREATE TABLE employees (
    emp_id INT,
    emp_name TEXT,
    emp_dept TEXT
) PARTITION BY HASH (emp_id);

CREATE TABLE employees_p1 PARTITION OF employees FOR VALUES WITH (MODULUS 4, REMAINDER 0);
CREATE TABLE employees_p2 PARTITION OF employees FOR VALUES WITH (MODULUS 4, REMAINDER 1);
```

Management and Maintenance

- **Partition Pruning**: Automatically excludes irrelevant partitions during queries.
- **Partition Maintenance**: Operations like adding, dropping, splitting, and merging partitions.

Replication

Oracle

Concepts and Types

1. **Oracle Data Guard**:
   - Provides high availability, data protection, and disaster recovery.
   - Example:

```
-- Setting up a Data Guard environment involves configuring primary and standby databases.
ALTER DATABASE ADD STANDBY LOGFILE GROUP 4 ('/u01/app/oracle/oradata/standby_redo4.log') SIZE 500M;
ALTER DATABASE ADD STANDBY LOGFILE GROUP 5 ('/u01/app/oracle/oradata/standby_redo5.log') SIZE 500M;
```

2. **Oracle GoldenGate**:
   - Supports real-time data integration and replication.
   - Example:

```
-- Setting up Oracle GoldenGate involves configuring Extract, Pump, and Replicat processes.
ADD EXTRACT ext1, TRANLOG, BEGIN NOW
ADD EXTTRAIL ./dirdat/aa, EXTRACT ext1
```

3. **Oracle Streams**:
   - Allows data sharing between databases.
   - Example:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES (
    table_name    => 'hr.employees',
    streams_type  => 'capture',
    streams_name  => 'capture_process',
    queue_name    => 'streams_queue'
  );
END;
```

PostgreSQL

Concepts and Types

1. **Streaming Replication**:
   - Provides real-time replication from primary to standby servers.
   - Example:

```
-- On the primary server:
wal_level = replica
max_wal_senders = 3
wal_keep_segments = 64

-- On the standby server:
primary_conninfo = 'host=primary_host port=5432 user=replication password=replica_pass'
```

2. **Logical Replication**:
   - Allows selective replication of database objects.
   - Example:

```
-- On the publisher:
CREATE PUBLICATION my_publication FOR TABLE employees;

-- On the subscriber:
CREATE SUBSCRIPTION my_subscription CONNECTION 'host=primary_host port=5432 dbname=mydb user=myuser password=mypass' PUBLICATION my_publication;
```

3. **pglogical**:
   - An advanced logical replication system.
   - Example:

```
sql
Copy code
-- Setting up pglogical involves creating nodes and replicating data between them.
SELECT pglogical.create_node(
  node_name := 'provider',
  dsn := 'host=provider_host dbname=provider_db'
);

SELECT pglogical.create_node(
  node_name := 'subscriber',
  dsn := 'host=subscriber_host dbname=subscriber_db'
);

SELECT pglogical.create_subscription(
  subscription_name := 'subscription',
  provider_dsn := 'host=provider_host dbname=provider_db'
);
```

Comparison Summary

Feature/Tool	**Oracle**	**PostgreSQL**
Partitioning	Range, list, hash, composite	Range, list, hash
Partition Maintenance	Comprehensive support for partition operations	Comprehensive support for partition operations
Replication	Data Guard, GoldenGate, Streams	Streaming Replication, Logical Replication, pglogical
High Availability	Data Guard	Streaming Replication
Disaster Recovery	Data Guard	Streaming Replication
Real-time Replication	GoldenGate	Streaming Replication, pglogical
Selective Replication	Streams, GoldenGate	Logical Replication, pglogical

Both Oracle and PostgreSQL offer robust partitioning and replication capabilities. Oracle provides more advanced built-in tools for replication and high availability, while PostgreSQL offers flexible, open-source options for achieving similar outcomes. The choice between them will depend on the specific needs and constraints of the database environment.

Error Handling in Oracle vs Postgres with examples

Error handling is a crucial aspect of database management, ensuring that any issues that arise during database operations are correctly managed and do not lead to data inconsistencies or application crashes. Both Oracle and PostgreSQL provide robust mechanisms for handling errors, though their approaches and syntax differ. Below is a detailed comparison of error handling in Oracle and PostgreSQL with examples.

Error Handling in Oracle

Using EXCEPTION Block

Oracle uses the EXCEPTION block to handle errors within PL/SQL blocks. Here's an example:

```
BEGIN
   -- Attempt to insert a record into the employees table
   INSERT INTO employees (emp_id, emp_name, emp_dept) VALUES (1, 'John Doe',
'Sales');
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      -- Handle unique constraint violation
      DBMS_OUTPUT.PUT_LINE('Duplicate employee ID detected.');
   WHEN OTHERS THEN
      -- Handle other exceptions
      DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```

Using SQLCODE and SQLERRM

Oracle provides SQLCODE and SQLERRM functions to get the error code and message of the last exception:

```
BEGIN
   -- Attempt to insert a record into the employees table
   INSERT INTO employees (emp_id, emp_name, emp_dept) VALUES (1, 'John Doe',
'Sales');
EXCEPTION
   WHEN OTHERS THEN
      -- Handle any exception
      DBMS_OUTPUT.PUT_LINE('Error code: ' || SQLCODE);
      DBMS_OUTPUT.PUT_LINE('Error message: ' || SQLERRM);
END;
```

Error Handling in PostgreSQL

Using EXCEPTION Block

PostgreSQL uses the EXCEPTION block in PL/pgSQL functions to handle errors. Here's an example:

```
DO $$
BEGIN
   -- Attempt to insert a record into the employees table
   INSERT INTO employees (emp_id, emp_name, emp_dept) VALUES (1, 'John Doe',
'Sales');
EXCEPTION
   WHEN unique_violation THEN
      -- Handle unique constraint violation
      RAISE NOTICE 'Duplicate employee ID detected.';
   WHEN OTHERS THEN
      -- Handle other exceptions
      RAISE NOTICE 'An error occurred: %', SQLERRM;
END;
$$;
```

Using SQLSTATE and SQLERRM

PostgreSQL provides SQLSTATE and SQLERRM functions to get the error code and message of the last exception:

```
DO $$
BEGIN
   -- Attempt to insert a record into the employees table
   INSERT INTO employees (emp_id, emp_name, emp_dept) VALUES (1, 'John Doe',
'Sales');
EXCEPTION
   WHEN OTHERS THEN
      -- Handle any exception
      RAISE NOTICE 'Error code: %', SQLSTATE;
      RAISE NOTICE 'Error message: %', SQLERRM;
END;
$$;
```

Comparison Summary

Feature	**Oracle**	**PostgreSQL**
Basic Error Handling	EXCEPTION block within PL/SQL blocks	EXCEPTION block within PL/pgSQL blocks
Error Information Functions	SQLCODE, SQLERRM	SQLSTATE, SQLERRM
Custom Error Messages	Use DBMS_OUTPUT.PUT_LINE for output	Use RAISE NOTICE for output
Handling Specific Errors	Use specific error codes like DUP_VAL_ON_INDEX	Use specific error conditions like unique_violation
General Error Handling	Use WHEN OTHERS THEN	Use WHEN OTHERS THEN

Examples in Healthcare Domain

Oracle Example: Handling Data Insertion Errors

Consider a scenario where we need to insert patient records into a patients table and handle potential errors:

```
sql
Copy code
BEGIN
   -- Attempt to insert a patient record
   INSERT INTO patients (patient_id, patient_name, patient_age) VALUES (1, 'Jane
Doe', 30);
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      -- Handle unique constraint violation
      DBMS_OUTPUT.PUT_LINE('Duplicate patient ID detected.');
   WHEN OTHERS THEN
      -- Handle other exceptions
      DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```

PostgreSQL Example: Handling Data Insertion Errors

The equivalent error handling in PostgreSQL would look like this:

```
sql
Copy code
DO $$
BEGIN
   -- Attempt to insert a patient record
   INSERT INTO patients (patient_id, patient_name, patient_age) VALUES (1, 'Jane
Doe', 30);
EXCEPTION
   WHEN unique_violation THEN
      -- Handle unique constraint violation
      RAISE NOTICE 'Duplicate patient ID detected.';
   WHEN OTHERS THEN
      -- Handle other exceptions
      RAISE NOTICE 'An error occurred: %', SQLERRM;
END;
$$;
```

Conclusion

Both Oracle and PostgreSQL offer powerful mechanisms for error handling within their PL/SQL and PL/pgSQL blocks, respectively. While their syntax and functions differ, the core concepts of trapping errors, retrieving error information, and handling specific and general errors are consistent across both platforms. Understanding these mechanisms is crucial for writing robust and reliable database applications, especially in critical domains like healthcare.

Functions in Oracle vs Postgres healthcare examples

Functions in Oracle vs. PostgreSQL with Healthcare Examples

Overview

Both Oracle and PostgreSQL support user-defined functions, which can be used to encapsulate complex logic and reuse it across multiple queries. Here, we'll compare how to create and use functions in both databases with examples relevant to the healthcare domain.

Oracle Functions

Example: Calculate Patient Age

In Oracle, you can create a function to calculate a patient's age based on their date of birth.

```
CREATE OR REPLACE FUNCTION calculate_age (dob DATE) RETURN NUMBER
IS
   v_age NUMBER;
BEGIN
   SELECT FLOOR((SYSDATE - dob) / 365) INTO v_age FROM dual;
   RETURN v_age;
END calculate_age;
/

-- Using the function
SELECT patient_id, calculate_age(date_of_birth) AS age FROM patients;
```

PostgreSQL Functions

Example: Calculate Patient Age

In PostgreSQL, you can achieve the same by creating a function in a similar way.

```
CREATE OR REPLACE FUNCTION calculate_age(dob DATE) RETURNS INTEGER
AS $$
BEGIN
   RETURN FLOOR(EXTRACT(YEAR FROM AGE(dob)));
END;
$$ LANGUAGE plpgsql;

-- Using the function
SELECT patient_id, calculate_age(date_of_birth) AS age FROM patients;
```

Key Differences and Similarities

1. **Syntax Differences**:
   - Oracle uses PL/SQL as its procedural language, while PostgreSQL uses PL/pgSQL.
   - Oracle requires the CREATE OR REPLACE FUNCTION syntax, while PostgreSQL uses a similar syntax but with different function delimiters ($$).
2. **Return Types**:
   - In both databases, you must define the return type of the function. Oracle uses RETURN keyword while PostgreSQL specifies it in the function definition.
3. **Calling Functions**:
   - Both Oracle and PostgreSQL allow you to call functions in a SELECT statement in a similar manner.

More Complex Example: Calculate BMI

Oracle

```
CREATE OR REPLACE FUNCTION calculate_bmi (weight NUMBER, height NUMBER) RETURN NUMBER IS
    bmi NUMBER;
BEGIN
    bmi := weight / (height * height);
    RETURN bmi;
END calculate_bmi;
/

-- Using the function
SELECT patient_id, calculate_bmi(weight, height) AS bmi FROM patients;
```

PostgreSQL

```
CREATE OR REPLACE FUNCTION calculate_bmi(weight NUMERIC, height NUMERIC) RETURNS NUMERIC AS $$
BEGIN
    RETURN weight / (height * height);
END;
$$ LANGUAGE plpgsql;

-- Using the function
SELECT patient_id, calculate_bmi(weight, height) AS bmi FROM patients;
```

Error Handling in Functions

Oracle

Oracle functions can include exception handling.

```
CREATE OR REPLACE FUNCTION safe_divide (num1 NUMBER, num2 NUMBER) RETURN NUMBER IS
    result NUMBER;
```

```
BEGIN
   result := num1 / num2;
   RETURN result;
EXCEPTION
   WHEN ZERO_DIVIDE THEN
      RETURN NULL;
END safe_divide;
/
```

PostgreSQL

PostgreSQL functions also support exception handling.

```
CREATE OR REPLACE FUNCTION safe_divide(num1 NUMERIC, num2 NUMERIC)
RETURNS NUMERIC AS $$
BEGIN
   RETURN num1 / num2;
EXCEPTION
   WHEN division_by_zero THEN
      RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Conclusion

Both Oracle and PostgreSQL provide robust support for user-defined functions, allowing complex logic to be encapsulated and reused across queries. The examples above show how similar tasks can be accomplished in both databases with slight syntactic differences.

In a healthcare context, such functions can be instrumental in calculating patient metrics, handling data transformations, and encapsulating business logic for better code maintainability and reusability.

Oracle Architecture vs PostgreSQL Architecture: Differences and Explanation

Oracle Architecture

Oracle Database is a multi-model database management system, known for its scalability, robustness, and extensive feature set. Here is an overview of its architecture:

1. **Instance and Database**:
   - **Instance**: Comprises the memory structures (SGA - System Global Area) and background processes (PMON, SMON, DBWR, LGWR, etc.) that manage the database files.
   - **Database**: A collection of physical files, including datafiles, control files, redo log files, and other supporting files.
2. **Memory Structures**:

- **SGA (System Global Area)**: Shared by all server and background processes. Contains:
  - **Database Buffer Cache**: Stores copies of data blocks read from datafiles.
  - **Redo Log Buffer**: Holds log entries waiting to be written to the redo log files.
  - **Shared Pool**: Caches various constructs that can be shared among users.
  - **Large Pool, Java Pool, Streams Pool**: Optional memory areas for specific tasks.
- **PGA (Program Global Area)**: Memory region that contains data and control information for server processes.

3. **Processes**:
   - **Server Processes**: Handle user processes' requests.
   - **Background Processes**: Include essential processes like DBWR (Database Writer), LGWR (Log Writer), CKPT (Checkpoint), SMON (System Monitor), PMON (Process Monitor), etc.
4. **Storage Structures**:
   - **Datafiles**: Store the database data.
   - **Control Files**: Metadata about the database.
   - **Redo Log Files**: Record all changes made to the data.
   - **Archived Log Files**: Copies of redo log files for recovery.
   - **Temporary Files**: Store temporary data for sorting and other operations.
5. **Additional Components**:
   - **Tablespaces**: Logical storage units.
   - **Segments, Extents, and Blocks**: Hierarchical storage organization within tablespaces.
   - **Oracle Net**: Network interface for Oracle Database.

PostgreSQL Architecture

PostgreSQL is an open-source object-relational database system known for its extensibility and standards compliance. Its architecture consists of the following components:

1. **Instance and Database**:
   - **Instance**: Also known as a PostgreSQL server, it includes memory structures and background processes.
   - **Database**: A collection of data managed by a single PostgreSQL server instance.
2. **Memory Structures**:
   - **Shared Memory**: Shared across all server processes. Includes:
     - **Shared Buffers**: Caches frequently accessed data.
     - **WAL Buffers**: Buffers for Write-Ahead Logging (WAL) data.
     - **Work_mem**: Used for sorting and hashing operations.
     - **Maintenance_work_mem**: Used for maintenance tasks like vacuuming.
   - **Local Memory**: Specific to each backend process.
3. **Processes**:
   - **Postmaster**: Main server process.
   - **Backend Processes**: Handle individual client connections.
   - **Auxiliary Processes**: Include background writer, WAL writer, autovacuum daemon, stats collector, etc.
4. **Storage Structures**:

- **Data Directory**: Contains all the database files.
- **Datafiles**: Store the database data.
- **WAL Files**: Record changes made to the data.
- **Configuration Files**: Include postgresql.conf, pg_hba.conf, etc.

5. **Additional Components**:
- **Tablespaces**: Logical storage units to manage where the data is stored.
- **Schemas**: Logical separation within a database.
- **Extensions**: Additional modules to extend PostgreSQL functionality.
- **Indexes**: Enhance query performance.

Key Differences

1. **Instance and Database Management**:
- Oracle separates the instance (memory structures and processes) and the database (physical files), allowing multiple instances to access a single database in Real Application Clusters (RAC).
- PostgreSQL has a simpler architecture where an instance (PostgreSQL server) manages multiple databases.

2. **Memory Management**:
- Oracle uses SGA and PGA for memory management, with a variety of specific pools within the SGA.
- PostgreSQL has shared memory for common data and local memory for individual processes, with adjustable parameters like shared_buffers, work_mem, etc.

3. **Processes**:
- Oracle has a rich set of background processes (DBWR, LGWR, SMON, PMON, etc.) handling different tasks.
- PostgreSQL has fewer background processes, with key ones like the background writer, WAL writer, and autovacuum daemon.

4. **Storage**:
- Oracle's storage structure is more complex with control files, redo log files, archived log files, etc.
- PostgreSQL uses a simpler storage model with a data directory, WAL files, and configuration files.

5. **Extensibility and Customization**:
- PostgreSQL is highly extensible with a rich ecosystem of extensions.
- Oracle is also extensible but has a more closed ecosystem compared to PostgreSQL.

6. **Licensing**:
- Oracle is a commercial database with licensing costs.
- PostgreSQL is open-source and free to use, with commercial support available if needed.

Conclusion

While both Oracle and PostgreSQL are powerful and robust database systems, they have different architectures suited to different types of workloads and use cases. Oracle is known for its advanced features and scalability in enterprise environments, while PostgreSQL is celebrated for its extensibility, standards compliance, and open-source nature. Understanding the architectural differences helps in making informed decisions based on specific requirements and constraints.

Architecture Comparison table

Detailed comparison table of Oracle and PostgreSQL architectures:

Feature / Component	Oracle Architecture	PostgreSQL Architecture
Instance vs Database	Separate instance and database	Combined instance and database
Memory Management	SGA (System Global Area) and PGA (Program Global Area)	Shared Memory and Local Memory
Memory Structures	- Database Buffer Cache - Shared Pool - Redo Log Buffer	- Shared Buffers - WAL Buffers - Work_mem - Maintenance_work_mem
Background Processes	Rich set including DBWR, LGWR, SMON, PMON, etc.	Fewer processes including background writer, WAL writer, autovacuum daemon
Primary Process	DBWR, LGWR, CKPT, SMON, PMON, ARCn	Postmaster
Storage Structures	- Datafiles - Control Files - Redo Log Files - Archived Log Files - Temporary Files	- Data Directory - Datafiles - WAL Files - Configuration Files
Logical Storage Units	Tablespaces	Tablespaces
Physical Storage Management	Segments, Extents, Blocks	Pages, Segments, Blocks
Concurrency Control	Multi-version Concurrency Control (MVCC)	Multi-version Concurrency Control (MVCC)
Indexes	B-Tree, Bitmap, Reverse Key, Function-Based, etc.	B-Tree, Hash, GiST, SP-GiST, GIN, BRIN, etc.
Networking Interface	Oracle Net	libpq
Extensibility	Limited to Oracle-specific modules and tools	Highly extensible with numerous third-party extensions
Replication	Data Guard, GoldenGate	Streaming Replication, Logical Replication, Slony, Bucardo
Partitioning	Range, List, Hash, Composite	Range, List, Hash
Backup and Recovery	RMAN (Recovery Manager)	pg_basebackup, WAL Archiving, pgBackRest
Logging and Monitoring	Extensive logging and monitoring tools	Integrated logging, pg_stat_activity, pg_stat_statements

Feature / Component	Oracle Architecture	PostgreSQL Architecture
Configuration Files	init.ora, spfile.ora	postgresql.conf, pg_hba.conf, pg_ident.conf
Licensing	Commercial (costly)	Open-source (free), with commercial support options available
Community Support	Extensive, but primarily enterprise-driven	Large and active open-source community
Release Cycle	Regular, with major updates every few years	Regular, with major updates approximately every year

Key Differences Explained:

1. **Instance vs Database Management**:
   - **Oracle**: Allows multiple instances to access a single database, especially useful in Real Application Clusters (RAC).
   - **PostgreSQL**: One instance (server) manages multiple databases.
2. **Memory Management**:
   - **Oracle**: SGA is shared among all server and background processes, while PGA is specific to each process.
   - **PostgreSQL**: Shared memory for common data and separate memory for each backend process.
3. **Processes**:
   - **Oracle**: A comprehensive set of background processes ensures high performance and reliability.
   - **PostgreSQL**: Focuses on simplicity with essential background processes.
4. **Storage Management**:
   - **Oracle**: Uses a complex storage structure for high reliability and performance.
   - **PostgreSQL**: Simpler structure focusing on ease of use and maintenance.
5. **Extensibility**:
   - **PostgreSQL**: Offers extensive extensibility with numerous third-party extensions.
   - **Oracle**: Extensible but within the Oracle ecosystem.
6. **Replication and Partitioning**:
   - **Both**: Support advanced replication and partitioning techniques but through different mechanisms and tools.
7. **Backup and Recovery**:
   - **Oracle**: RMAN provides comprehensive backup and recovery solutions.
   - **PostgreSQL**: Uses tools like pg_basebackup and pgBackRest for reliable backups.
8. **Licensing**:
   - **Oracle**: Commercial licensing can be costly.
   - **PostgreSQL**: Open-source, free to use, with optional commercial support.

This table provides a clear comparison of Oracle and PostgreSQL architectures, highlighting the strengths and differences of each system.

Oracle to PostgreSQL Migration

Migrating from Oracle to PostgreSQL involves understanding the differences between the two databases in terms of SQL syntax, tools, and features. Here is a detailed comparison and guidance on tools that facilitate this migration process.

1. SQL Syntax and Query Differences

Data Types

- **Oracle**: Uses NUMBER, VARCHAR2, CLOB, BLOB, etc.
- **PostgreSQL**: Uses NUMERIC, VARCHAR, TEXT, BYTEA, etc.

Example:

- Oracle: NUMBER(p,s)
- PostgreSQL: NUMERIC(p,s)

String Functions

- **Oracle**: SUBSTR(), INSTR()
- **PostgreSQL**: SUBSTRING(), POSITION()

Example:

- Oracle: SUBSTR('Hello World', 1, 5)
- PostgreSQL: SUBSTRING('Hello World' FROM 1 FOR 5)

Date Functions

- **Oracle**: SYSDATE, ADD_MONTHS(), TRUNC()
- **PostgreSQL**: CURRENT_DATE, DATE_TRUNC(), AGE()

Example:

- Oracle: SYSDATE
- PostgreSQL: CURRENT_DATE

Sequence and Auto-Increment

- **Oracle**: Uses SEQUENCE for generating unique values.
- **PostgreSQL**: Uses SERIAL and BIGSERIAL.

Example:

- Oracle:

  CREATE SEQUENCE my_seq START WITH 1;

- PostgreSQL:

  CREATE SEQUENCE my_seq START 1;

PL/SQL vs PL/pgSQL

- **Oracle**: PL/SQL
- **PostgreSQL**: PL/pgSQL

Example:

- Oracle:

```
DECLARE
  v_name VARCHAR2(50);
BEGIN
  SELECT name INTO v_name FROM employees WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE(v_name);
END;
```

- PostgreSQL:

```
DO $$
DECLARE
  v_name VARCHAR;
BEGIN
  SELECT name INTO v_name FROM employees WHERE employee_id = 100;
  RAISE NOTICE '%', v_name;
END $$;
```

2. PostgreSQL Tools vs Oracle Tools

Database Management Tools

- **Oracle**: Oracle SQL Developer, Oracle Enterprise Manager
- **PostgreSQL**: pgAdmin, DBeaver

Migration Tools

- **Oracle**:
  - Oracle SQL Developer Migration Workbench
  - Oracle GoldenGate
- **PostgreSQL**:
  - **pgloader**: A tool for loading data into PostgreSQL from different sources.

- **ora2pg**: A tool to migrate an Oracle database to PostgreSQL.
- **AWS Schema Conversion Tool (SCT)**: Converts Oracle schema to PostgreSQL schema.

Backup and Restore Tools

- **Oracle**: RMAN (Recovery Manager)
- **PostgreSQL**: pg_dump, pg_restore, pg_basebackup

Performance Tuning and Monitoring

- **Oracle**: AWR (Automatic Workload Repository), ADDM (Automatic Database Diagnostic Monitor)
- **PostgreSQL**: pg_stat_statements, EXPLAIN, pgBadger

3. Migration Process

Step-by-Step Migration

1. **Assessment and Planning**:
   - Assess database size, complexity, and dependencies.
   - Plan the migration strategy and timeline.
2. **Schema Conversion**:
   - Use tools like ora2pg or AWS SCT to convert Oracle schema to PostgreSQL.
   - Manually adjust complex PL/SQL code to PL/pgSQL if necessary.
3. **Data Migration**:
   - Export data from Oracle using tools like Data Pump.
   - Load data into PostgreSQL using tools like pgloader or custom scripts.
4. **Application Migration**:
   - Update application code to interact with PostgreSQL.
   - Adjust any SQL queries or database calls in the application code.
5. **Testing**:
   - Perform extensive testing to ensure data integrity and application functionality.
   - Compare performance and optimize queries as needed.
6. **Go Live**:
   - Plan for a cutover strategy.
   - Perform the final data sync and switch to the new PostgreSQL database.
7. **Post-Migration**:
   - Monitor the new system.
   - Train staff on PostgreSQL tools and features.

Example: Migrating a Table from Oracle to PostgreSQL

Oracle Schema

```
CREATE TABLE employees (
  employee_id NUMBER PRIMARY KEY,
  first_name VARCHAR2(50),
  last_name VARCHAR2(50),
  hire_date DATE
);

INSERT INTO employees (employee_id, first_name, last_name, hire_date)
VALUES (1, 'John', 'Doe', SYSDATE);
```

Using ora2pg to Convert Schema

```
ora2pg -c config/ora2pg.conf -t TABLE -o employees.sql
```

PostgreSQL Schema

```
CREATE TABLE employees (
  employee_id SERIAL PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  hire_date DATE
);

INSERT INTO employees (employee_id, first_name, last_name, hire_date)
VALUES (1, 'John', 'Doe', CURRENT_DATE);
```

Conclusion

Migrating from Oracle to PostgreSQL involves understanding the differences in SQL syntax, leveraging appropriate tools for schema and data migration, and updating application code accordingly. Using tools like ora2pg, pgloader, and AWS SCT can simplify the process and help ensure a successful migration.