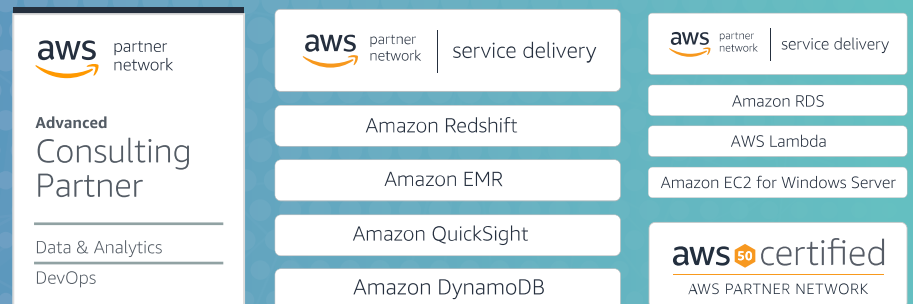


A Complete Guide to Oracle to Redshift Migration



The image displays several AWS Partner Network (APN) logos and service delivery boxes. On the left is the 'Advanced Consulting Partner' logo, which includes the text 'Data & Analytics' and 'DevOps'. To its right are two columns of 'service delivery' logos. The first column lists 'Amazon Redshift', 'Amazon EMR', 'Amazon QuickSight', and 'Amazon DynamoDB'. The second column lists 'Amazon RDS', 'AWS Lambda', and 'Amazon EC2 for Windows Server'. At the bottom right is the 'aws 50 certified' logo, indicating membership in the AWS Partner Network.

Contents

- Introduction 3
- The four-phases of a predictable migration 3
- Assess 4
 - Migration of Data and Objects 6
 - Migration of ETL Process 8
 - Migration of Users & Reporting Applications 9
- Planning 9
 - Migration Planning 10
 - Resources Planning 11
- Execute the Migration 11
 - One-Step Migration 11
 - Phased Approach 12
- Ensure Migration Success 12
- Benefits of Redshift Migration 14
- Redshift gotchas 15
- Case Study: Modernizing Legacy Oracle Datawarehouse Using Redshift 20
- References 23

Introduction

In the past few years, several enterprises have undertaken large scale digital transformation projects and chosen to re-platform their legacy infrastructure onto modern cloud-based technologies to lay a strong foundation for business change. Many businesses chose to re-platform on one of the top three cloud platform providers – Amazon Web Services (AWS), Microsoft’s Azure, and Google Cloud Platform. One of the benefits of the re-platforming is the ability to mine speed-of-thought insights on data through a modern data warehouse.

Oracle and AWS Redshift are leaders in traditional and cloud DW space, respectively. Given AWS has the largest market share, there is a high likelihood of a business migrating from its legacy Oracle DW to AWS Redshift. Redshift’s extreme cost-effectiveness, exemplary performance, and deep integration with AWS services make it the DW of choice for enterprises. Oracle is an RDBMS built on coupled monolithic architecture, and Redshift is a columnar database with MPP architecture - these differences make migration to the cloud DW a complicated project.

By taking a systematic approach and breaking down the process of migration into four distinct phases, thorough data discovery and validation of migration, we can incrementally reduce risks at each stage. Therefore, in this blog, we explore how to achieve predictable migration from an on-prem Oracle DW to AWS Redshift.

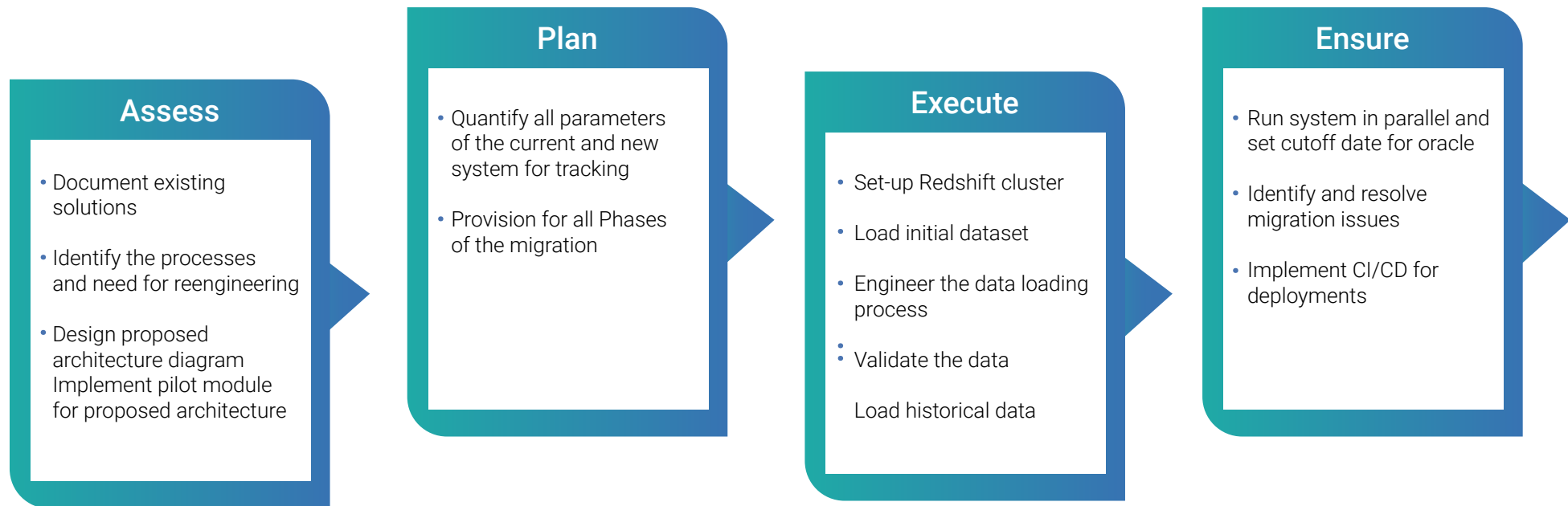
The four-phases of a predictable migration

01 **Assess** – Cloud Fit & Detailed DW component Assessment

02 **Plan** – Migration & resources

03 **Execute** – Actual Migration

04 **Ensure** – Post Migration Quality Checks



Assess

The first phase is to conduct a thorough assessment of the Oracle ecosystem and perform a high-level **Cloud-Fit assessment** of migration to AWS. The more questions asked, and the more clarity achieved leads to an increase in predictability and identifies difficulties that might arise. Here are some of the crucial questions to ask:

- Is the chosen cloud platform a good fit for the business use-cases?
- Is the migration feasible within the given budget and timeline?
- What are the business and technical risks associated with the migration?
- What are the success criteria?

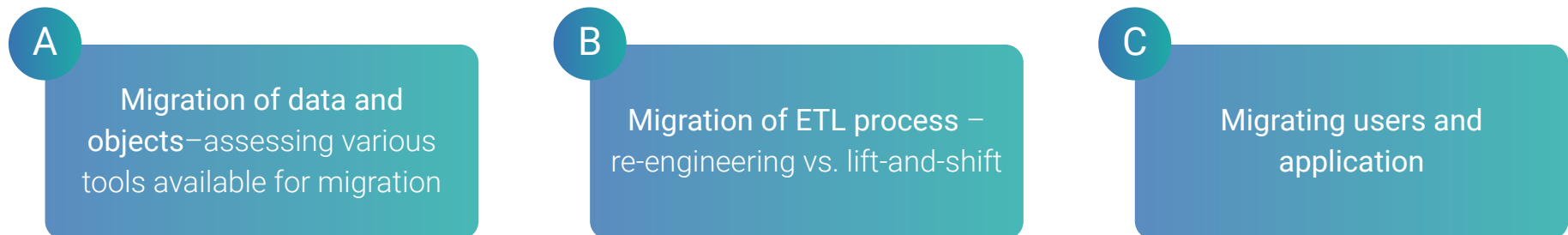
Once the answers to the above questions are clear and agreed upon by all stakeholders, start collecting information that

- Captures a comprehensive inventory of the current system process and change requirements along with the pain points.
- Evaluates alternative tools or AWS services for re-engineering the ETL, reports, and data science workloads as the application might involve complex processes and applications.
- Evaluates the performance and cost benefits with the migration of applications or databases or ML models.

This above information is the base on which to perform the next stage in the assessment – **a detailed assessment of DW components.**

Migration to AWS cloud is very platform-specific and requires AWS cloud ecosystem specific expertise. We need to understand the details of networking, permissions, roles, accounts, and all things associated with AWS. Let us explore some of the AWS services available for migrating to AWS Redshift from Oracle and the benefits and limitations of the same.

We shall consider the services available under three sections - the main components in a Datawarehouse:



Migration of Data and Objects

AWS Data Migration Service

Database Migration Service (DMS) is a managed service that runs on an Amazon Elastic Compute Cloud (EC2) instance. DMS helps in migration on homogenous and heterogeneous databases with virtually no downtime. The data changes to the source database that occur during the migration are continuously replicated to the target; thereby, the source database is fully operational during the migration. AWS DMS is a low-cost service where the consumers pay only for the compute resources used during the migration process and any log storage.

Limitations: DMS service does not support the migration of the complex processes involved, like stored procedures, packages, and triggers. Support is unavailable for the migration of data warehouses like Oracle, Netezza, Teradata. We might have to use SCT to migrate the schema and databases.

AWS Schema Conversion Tool (AWS SCT)

AWS Schema Conversion Tool makes heterogeneous database migrations predictable by automatically converting the source database schema and most of the database code objects, including views, stored procedures, and functions, to a format compatible with the target database. SCT also generates the assessment report and provides a high-level summary of how much of the schema can be converted automatically from Oracle to AWS Redshift. It also provides a fair estimation of how much needs to be manually migrated.

AWS Data migration agents are locally installed agents designed to extract data from data warehouses. The extracted data is optimized by agent and uploaded to either AWS Redshift or an S3 bucket. The migration agents work entirely independently from SCT and are designed to extract data in parallel.

Limitations: Since the agents are locally installed, choosing the proper instance type with compute, storage, and network features is a challenge and requires expertise in the area.

AWS Direct Connect & Snow Family

Moving large volumes of data out of legacy systems is difficult. A 1 Gbps network connection theoretically moves 1 PB of data in about 100 days, and in real-time, it is likely to take longer and cost more. Migrating data-intensive analytics environments, moving enterprise data centers to the cloud, and digital archives require bulk data transport methods that are simple, cost-efficient, and secure.

AWS Direct Connect is another option when we want the data transferred through a private physical network connection. Direct connection is not redundant, so unless there is a second line (enabling bi-direction forwarding detection) for failovers. If there is a need for transfer of data to AWS on an ongoing basis, then direct connect is ideal.

Devices in the AWS Snow family are simple to provision and extremely cost-effective. They temporarily and securely store data while it is shipped between the client location and an AWS region.

AWS Snowball

Petabyte-scale (50-80 TB) data transfer service in which a secure suitcase-sized device moves data in and out of the AWS Cloud quickly and securely. Typically, it takes around a week to get the data into AWS Cloud S3 Buckets.

AWS Snowball Edge

Devices contain slightly larger capacity (100 TB) and an embedded computing platform that helps you perform simple processing tasks like on-the-fly media transcoding, image compression, metrics aggregation ideally useful in IoT and Manufacturing Industries

AWS Snowball Mobile

The AWS Snowmobile moves up to 100PB of data, and is ideal for multi-petabyte or Exabyte-scale digital media migrations and data center shutdowns.

Migration of ETL Process

Moving data may very well turn out to be the easy part when compared to migrating ETL processes. In general, the choices are - lift-and-shift the existing process vs. re-engineering the process entirely. Before arriving at a conclusion, we may need to try addressing the below questions and evaluate the approach

- Which applications/modules are business-critical?
- Which applications have been designed to work as is in the cloud platform?
- Which applications need to be sunset and rewritten?
- Which applications fit the Lift and Shift, and which ones do not fit?

Lift & Shift: As the name indicates, lift and shift consists of moving an existing on-premise application and infrastructure with no change. Organizations typically choose this path because it's considered, without enough data, that re-engineering for the cloud is time-consuming and more expensive. Another reason why this method is favored is because of the assumption that the applications are moved faster to the cloud with less investment, and that it is less disruptive for business. Legacy applications are not architected for cloud services and in most cases, it requires partial or complete re-architecting just to meet the previous operating status. A legacy data integration tool, like Informatica, needs to be re-evaluated with the needs of modernizing the data warehouse to reap the benefits of moving to the cloud. Below we have listed some of the common challenges during lift and shift approach with legacy data integration tools like Informatica:

- Informatica is built for on-premise applications with pre-built transformations and might not be favorable for fast processing of data with Integration of Workday, Anaplan, SQL Server, REST APIs, Amazon Redshift, SFTP, Amazon S3, Flat Files
- Support for loading unstructured data like JSON, XML is limited
- Compliance with standards like GDPR is limited
- High License cost

By using the Lift and Shift approach, customers would be giving up an opportunity to modernize the data warehouse and leverage revolutionary capabilities that cloud provides, for example, scaling to meet the demands of application consumption, the guarantee of 99.99% application availability to customers and low administrative overheads.

Therefore, one may need to do one or both of the following while assessing ETL flows in the current ecosystem:

- a. Change the codebase to optimize the platform performance and change data transformations to sync with data restructuring.
- b. Determine if dataflows should remain intact or be reorganized.

Migration of Users & Reporting Applications

The last step in the migration process is migrating users and applications to the new cloud DW, with little or no interruption to business continuity. Security and access authorizations may need to be created or changed, BI and analytics tools should be connected to Redshift and their compatibility, and performance tested. It is recommended that the reporting application is also moved into the cloud platform for seamless integration.

AWS's BI service Quicksight provides cost-effective and swift BI capabilities for enterprise with inbuilt insights and ML capability.

Planning

Once the detailed assessment is complete, and clarity on best-fit AWS services for migration to the Redshift DW is determined, we move on to phase two - the planning process for the actual execution of migration. There are two parts to this planning process – planning the migration and planning for resources. The two are co-dependent on each other and shall be undertaken simultaneously.

Migration Planning:

- List all the databases and volume of data that needs to be migrated, which helps us in deciding the number of terabytes or petabytes of on-premises data to be loaded into AWS Redshift. This list will help narrow down the tools needed to move data efficiently. The options could be AWS Snowball services or an alternative approach like AWS SCT/AWS DMS, ETL Process.
- Analyze the database objects like schemas, tables, views, stored procedures, and functions for tuning and rewrite.
- Identify the processes and tools that populate the data and pull data from the oracle databases like BI, ETL, and other upstream and downstream applications.
- Identify data governance requirements like
 - security and access control requirements like users, roles, users, and related permissions.
 - Row-level and column level security
 - Data encryption and masking
- Outline the approach on migration plan by categorizing the modules which can be migrated as groups and which are least dependent on other modules by start creating the setup in Redshift with cluster launched to handle the storage and compute as required and proposed data growth rate. Migration plan includes

<ul style="list-style-type: none"> ◦ Define Migration scope ◦ List of database and process migrated as-is and with re-engineering ◦ The priority of datasets/modules to be loaded ◦ Future state architecture diagram ◦ ETL/ Reporting Tools introduced with the migration and the ones which are deprecated post-migration. 	<ul style="list-style-type: none"> ◦ Infrastructure for development and deployment activities ◦ Test plan document ◦ Operations and Maintenance activities ◦ Performance benchmarking of current systems ◦ Assumptions and risks involved in the migration
---	---

Resources Planning:

- **Identify the Migration Team:** It makes perfect sense to inform the relevant data stakeholders and technical teams of their forthcoming commitments before the migration kicks off. Prepare an outline of what roles are required on data migration and plan for the availability of the roles well before migration is started. The roles that are required are based mainly on the complexity of application and data warehouse that needs to be migrated; we would need Business owner, SME, Data Architect, Project Manager, Business Analyst, DevOps, Developers (ETL & Reporting), Database administrator. Also note, not every role is required for all the phases and needs to be factored in for specific timelines.
- **Budget Allocation:** A key consideration for budget planning is the underlying data warehouse sizing, such as the number of compute clusters required to support the migration and the post-migration data warehouse. Compare the amount of migration work and the associated costs to the available budget to ensure the availability of funds till completion of the project.

Execute the Migration

While planning the migration, the migration strategy for the actual execution of the migration is also being solidified. The strategy depends on a host of factors – the size of the data, number of data objects, their type and complexity, network bandwidth, data change rate, transformations involved, and ETL tools used, etc. Depending on the complexity and risks involved, migration can be done either via a one-step or a phased approach.

One-Step Migration:

A one-step migration approach is ideally suited for small data warehouses. There are no complex ETL processes or report migrations involved in such cases. Clients can migrate their existing database by any of the approach listed below-

- Use AWS Schema Conversion Tool and AWS Data Migration Service to convert the schema and load the data into AWS Redshift
- Extract existing database as files into a data lake or S3 and load the file data into Amazon Redshift via ETL tools or custom scripts.

Clients then test the destination Amazon Redshift database for data consistency with the source data, repoint the BI and ETL processes to AWS Redshift. Once all validations have passed, the database is switched over to AWS Redshift for release.

Note: In the above approach, we assume that the ETL and Reporting tools have proper connectors for Redshift. As an example, ETL tools like Informatica PowerCenter, Talend, and BI tools like Tableau and MicroStrategy fits into this category.

Phased Approach:

For any large data warehouse ecosystem with complex processes, the most commonly used approach is a phased approach where the following steps are performed iteratively until all data, ETL, and BI workloads are moved into a target system.

- Identify the module to be migrated and use AWS Schema Conversion Tool to convert the schema for the relevant data objects. AWS SCT tool helps with the identification of the areas requiring re-engineering (views, SPs, functions).
- Load the initial/historical dataset and validate objects, data types, and actual data.
- Build the ETL process to ingest incremental change or change data capture simultaneously into Redshift.
- Validate the data post-migration.
- Conduct performance benchmarking to ensure ETL & Analytical queries are executed faster within the required level of SLA.
- Run both Oracle and Redshift systems in parallel and ensure that data is in sync in both environments before redirecting the future process to Redshift.
- Cutover the ETL processes into Redshift and retire those processes in the source system.

Ensure Migration Success

Once the execution is complete, validating the migration is crucial for project success. Using the assessment details gathered while preparing for the migration, identify areas that could present issues based on the current usage of Oracle and/or other tools used in the migration. Make stakeholders aware of these known risks with available mitigation strategies and the proposed approach to meet their requirements. This is another method through which predictability in migration can be achieved. Here is a list of best practices that can ensure that the migration is a success:

Validate the new environment for connectivity and security

It's good practice to check networking, proxy, and firewall configurations while putting together the migration strategy. It usually helps to have details on what ports and URLs are needed to access new systems. Also, factors like setting up of account parameters such as IP whitelisting and role-based access control before opening the environment up to larger groups helps in increasing the confidence in the new applications.

Validate and Fix any data inconsistencies between Oracle and Redshift

To make sure the migration has completed successfully, compare data between the Oracle and Redshift environments throughout the migration. Investigate differences to determine the cause and resolve any issues. Compare the performance of the processes that load and consume data to ensure Redshift is performing as expected.

Fine-tune the Redshift for the performance & concurrency scaling

It is a best practice to test the process from end to end for both functionality and performance. Validating test cases and data sets are critical to early success. Before going live, it is good to re-run all performance tests to ensure the system is configured for individual query performance and concurrent users and ensure that we are getting the same calculated results from the old to the new system.

Bigger and more complex implementations should be relooked at to see if they can utilize cloud-native services like EMR, Glue, or Kinesis from cost, SLA, security, and operational overhead perspectives.

Once the data in the new system matches the new system, and any issues that pop up can be resolved by the in-house team, we can deem that the migration to AWS Redshift from Oracle is stable.

Benefits of Redshift Migration

Redshift is the most adopted cloud data warehouse as it offers a host of features and benefits covering a whole range of business requirements, from better performance and cost reduction to automation and operational excellence. Here are the top five benefits clients gain from this migration.

- ✓ The fully managed nature of Redshift makes quick work of the setup and deployment of a data warehouse with petabytes of data. Redshift also automates most of the repetitive administrative tasks involved in running a DW, reducing the operational and DBA overhead.
- ✓ Migrating to Redshift from an on-premise DW can reduce the Total Cost of Ownership (TCO) by over 90% with fast query performance, IO throughput, and lower operational challenges.
- ✓ Elastic resizing enables quick scaling of compute and storage, while the concurrency scaling feature can efficiently handle unpredictable analytical demand making Redshift perfect for handling loads of varied complexity and velocity.
- ✓ Redshift's inbuilt Redshift Spectrum feature means that businesses no longer need to load all their fine-grained data into the data warehouse or need an ODS. We can query a large amount of data directly from Amazon S3 data lake with zero transformation.
- ✓ Redshift has introduced an array of features like Auto Vacuum, Auto Data distribution, Dynamic WLM, Federated access, AQUA to help customers to address the challenges faced by on-prem data warehouses and provides automated maintenance. AWS continuously adds new features and improves the performance of Redshift. These updates are transparent to the customer with no additional costs or licenses.

Redshift gotchas:

Here are some of the pitfalls one needs to be aware before migration:

Empty string vs. null

Oracle automatically converts empty strings to null values, but Redshift does not! If the application is inserting empty strings and treating it as a null value in Oracle, we need to modify the load process to handle the inserts or extracts the data before moving to Redshift.

```
01. create table employee (name character varying (20), dept int)
02. insert into employee(name,dept) values (null, 1);
03. insert into employee(name,dept) values ('', 2);
04. select * from employee;
```

Oracle Query & Result: **select * from employee where name is null;**

The screenshot shows a database client interface with a SQL editor and a data grid. The SQL editor contains the following queries:

```
1 • create table employee(name varchar(20), dept int);
2 • insert into employee(name,dept) values (null, 1);
3 • select * from employee;
4 • insert into employee(name,dept) values ('', 2);
5 • select * from employee where name is null;
```

The data grid shows the results of the queries:

NAME	DEPT
	1
	2

Redshift Query & Result: `select * from employee where name is null;`

```
create table employee (name character varying(20), dept int)
insert into employee(name,dept) values (null, 1);
insert into employee(name,dept) values ('', 2);
|
select * from employee where name is null;
```

Output Result 1

Drag a column header here to group by that column.

name	dept
NULL	1

`select * from employee where name = '';`

```
select * from employee where name = '';
```

Output Result 1 Result 2

Drag a column header here to group by that column.

name	dept
	2

As shown above, null is treated as same by both databases. When inserting an empty string, Oracle stores it as null, and we can query just by using where the name is null, and both values are displayed. Whereas in Redshift we need to explicitly use the condition ='' to display the row with the empty strings

Choosing BIGINT over numeric data type in Redshift

When we migrate from the Oracle database, we tend to map the numeric data-type as is in Redshift, and with this, we have noticed that the performance of queries running with columns on these joins is executed much longer than the ones with SMALLINT, INTEGER or BIGINT data-types. In Oracle, NUMBER or NUMERIC types allow for arbitrary scale, which means Oracle accepts values with decimal components even if the data type was not defined with precision or scale, and accommodates the way data comes in. In Redshift, columns designed for values with decimal components must be defined with a scale to preserve the decimal portion which can define as NUMERIC (38,18)

Logical Evaluation when NULL

Oracle automatically concatenates the values of column ignoring the null values, but Redshift does not, and it returns null for the complete result. We had some scenarios where the concatenation of multiple columns formed the logical key, and the results differed when compared to Oracle, as shown below.

```
01. create table emp(empid int,region varchar(10),deptname varchar(10))
02. insert into emp(empid, region, deptname) values (1,'US','HR')
03. insert into emp(empid, region, deptname) values (2,'US','R&D')
04. insert into emp(empid, region, deptname) values (3,null,'HR')
05. insert into emp(empid, region, deptname) values (4,' ','HR')
```

Oracle Query & Result: `select empid || region || deptname from emp`

The screenshot shows a query tool interface with a query editor at the top containing the SQL query: `select empid || region || deptname from emp`. Below the editor is a toolbar with various icons. At the bottom, a data grid displays the results of the query as a table with four rows and three columns: EMPID, REGION, and DEPTNAME.

EMPID	REGION	DEPTNAME
1	US	HR
2	US	R&D
3		HR
4		HR

Redshift Query & Result: `select empid || region || deptname from emp`

The screenshot shows a Redshift query editor with the following SQL query:

```
select empid || region || deptname from rdw.emp
```

The output window displays a table with the following data:

empid	region	deptname
1	USHR	
2	USR&D	
4	HR	

Check Constraints before including

AWS Redshift does not enforce primary key, foreign key, or unique constraints. It still allows us to add them, though, and it assumes they are intact. Here is a demonstration with an example. Here we are trying to create a table which is then populated with some duplicates.

The screenshot shows a Redshift query editor with the following SQL queries:

```
CREATE TABLE table1 (column1 integer not null);
INSERT INTO table1 VALUES (1);
INSERT INTO table1 VALUES (2);
INSERT INTO table1 VALUES (3);
INSERT INTO table1 VALUES (1);

select * from table1
```

The output window displays a table with the following data:

column1
1
2
3
1

With the same table above, if we try selecting distinct values, the Redshift query planner includes the UNIQUE step post sequential scan, and we get the expected results.

The screenshot shows a Redshift query editor with the following SQL query:

```
EXPLAIN SELECT DISTINCT column1 from table1;
```

The output window displays the following query plan:

```
QUERY PLAN
XN Unique (cost=0.00..0.05 rows=3 width=4)
-> XN Seq Scan on table1 (cost=0.00..0.04 rows=4 width=4)
```

Let us add a primary key constraint on the column. Redshift does not complain when adding the constraint, even though there are duplicates in the column, as shown below.

```
ALTER TABLE rdw.table1 ADD PRIMARY KEY(column1);
```

Output

Standard | Text | Grid

T: 17:28
Current path set to \$user, public
T: 17:28 D: 1.0424624s. S: ALTER TABLE rdw.table1 ADD PRIMARY KEY(column1)
The command completed successfully

If we recheck for the query planner for selecting distinct, the step to eliminate the duplicates (UNIQUE) is removed, and Redshift assumes that the constraints are taken care of by the external process.

```
EXPLAIN SELECT DISTINCT column1 from table1;
```

Output

Result 1

Drag a column header here to group by that column.

QUERY PLAN

XN Seq Scan on table1 (cost=0.00..0.04 rows=4 width=4)

Be Cautious while performing a full-text search using LIKE clause

Redshift does not have indexes available in Oracle, which can be added on text columns and thereby improves the performance of queries. Redshift has an alternative to Indexes, distribution key based on which the data is distributed across the nodes and slices. The query planner relies on this distribution key and sort key to generate an optimal query execution plan. When we perform a search on the text column, which is neither a distribution key nor a sort key, the performance of the query is not very efficient. The better practice is to use more of Numeric or data fields for filters and more optimal when distribution or sort keys are used as part of filters in the query.

Be Aware:

Here are some things that need to be considered before migrating and factor the engineering effort:

- Redshift does not support Synonyms as available in Oracle
- Enforcing of Primary and foreign keys – Oracle enforces the keys, whereas in Redshift we need to engineer the load process that depends on these constraints to prevent duplicate entries
- Partitioning of the table is not available in Redshift; this can be accomplished indirectly using Redshift spectrum
- Updating data through view – Oracle includes insert, update, and delete from view, which gets updated in the underlying table, eventually. Whereas, in Redshift, this needs to be done solely at the table level.

Case Study: Modernizing Legacy Oracle Datawarehouse Using Redshift

The Problem:

The legacy warehouse had structural inefficiencies in debugging that was complex, time-consuming, and costly. Technical infrastructure teams administrated each Oracle environment manually without the benefit of standardized monitoring and management. The biggest issue was its inflexible scaling and long-running reports. The technical teams and the business wanted to overcome

- The limitations of a legacy warehouse
- Improved performance at a lower cost
- Maximizing operational efficiency and customer ROI

Another difficulty arising from the current setup was that of high latency. Even crucial daily reports took more than 6 hours to generate. It took over a week for the DW management team to finish the monthly data loading and reconciliation process. Also, the data quality of the loaded data was compromised.

They needed a scalable and cost-effective solution to process and analyze all this data.

The Solution/Approach

Agilisium's ongoing relationship with the customer ensured that they were aware of our expertise in Redshift. The customer was already on the AWS cloud, and hence it was an easy decision for them to decide on moving to Redshift. So, a team of 8 members (consisting of 2 architects, 1 DevOps engineers, 1 project managers, 2 developers, 1 Business Analyst, 1 Project Stakeholder) executed the migration.

However, before team selection, Agilisium worked through an extensive data discovery phase and assessment of the tools currently used in the RDW module. The data discovery ensured that both Agilisium and UMG project stakeholders understood the current systems, its loopholes, and interdependencies.

From the high-level architecture, we can see that the current process was relying on the FTP server for picking or pushing files, and the Control-m and Shell scripts triggered the load process as and when a file arrived. Business Objects generated most of the reports.

In this account, there was a need for evaluating the ETL tool and Reporting tool and its migration or re-engineering of the ETL process. After the cloud fit assessment, the AWS S3 service was used to source data files, processing, and archival. Snaplogic was determined to be the best-fit tool for performing ETL load and sequencing of loading the data files. BO reports remained as is; it was more of lift and shift from on-premise to cloud.

Once the tools to move forward were identified based on dependency, the entire RDW data warehouse was broken down into multiple modules and prioritized in moving to Redshift. After executing a sequential migration of these modules, the migration team made sure that each module was intact (no process breaks and proper failure management) before starting the next one.

Using the AWS Schema converter, building the schema and generating DDL scripts was accomplished. The data loading was also done in 2 parts, history data load and incremental Data loading.

- History data loading: Custom Snaplogic pipelines dumped the data from the on-prem data warehouse to Redshift equivalent tables. The data was cross-validated to match the contents and counts of rows.
- Incremental loading: The business logic in Oracle packages and procedures were re-engineered using Snaplogic pipelines and orchestrated using Snaplogic tasks. The pipelines were designed and scheduled to load files from S3 to the Redshift cluster. After the migration of BO reports and servers to AWS, the connection details were changed.

Significant time was spent testing the content of the data and validation of the process in terms of i) preprocessing or processing of files ii) loading of data, and iii) archival process. Multiple data incremental loads were performed, and it ensured that the values matched with their counterpart.

Post-migration, the performance of the queries, and reports were 3-5x improved in comparison to Oracle data warehouse. Even so, the team invested their time in a lot of performance fine-tuning of multiple distribution keys and sort keys for bigger tables. The tuning ensured that optimized WLM configuration could handle the concurrent users. During the migration process, we solved multiple complexities due to the differences in the way Oracle and Redshift approach, as listed above. We could create reusable components that can be shared across modules and also useful in the data loading process.

The Results:

- 80% reduction in time spent on the creation and maintenance of each environment
- Significant Reduction in Administration Overhead Costs
- Increased confidence in analytical results due to multiple "runs" per day to compare
- 3-5x improvement in both analytical response timeframes and iterations of analytical workloads
- Timely update and analysis on target audience information that increased confidence in audience attributes and preferences

References:

- [AWS Redshift](#)
- [Datawarehouse Migration](#)
- [AWS Redshift User Guide](#)
- [Amazon Redshift Migration Best Practices](#)
- [Amazon Enterprise Datawarehouse](#)



Agilisiium is a Big Data and Analytics company with clear focus on helping organizations take the "Data-to-Insights-Leap". As a AWS Advanced Consulting Partner with Redshift, EMR, DevOps & Quicksight competencies, Agilisiium has invested in all stages of data journey: Data Architecture Consulting, Data Integration, Data Storage, Data Governance and Data Analytics. With advanced Design Thinking capabilities, a thriving Partner Eco-system and top-notch industry certifications, Agilisiium is inherently vested in our clients' success

