# Lab 3 - Optimizing Database Schemas

run queries more efficiently in Amazon Redshift by correctly assigning distribution keys and sort keys

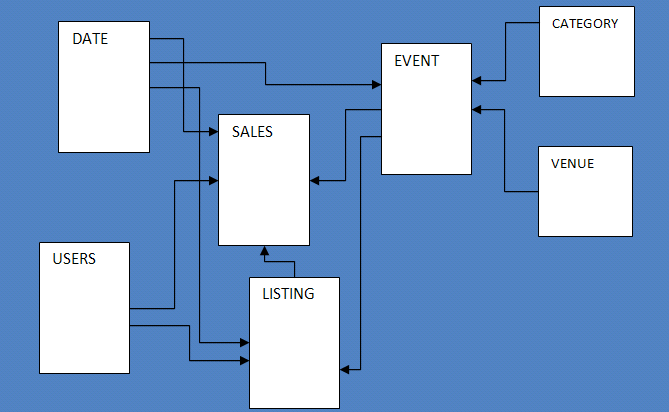
Objs-

1. Connect to Amazon Redshift
2. Select a distribution key for a column
3. Select a sort key

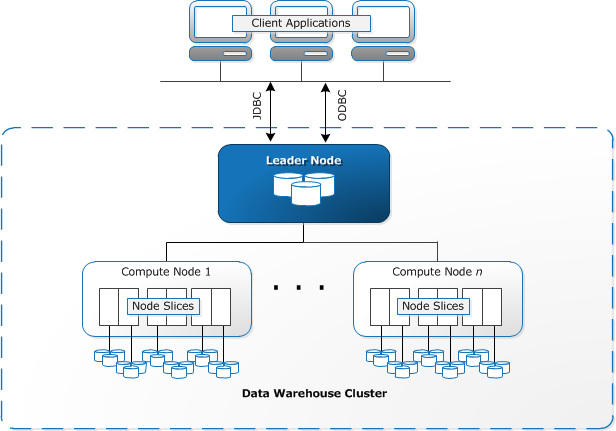
Distribution Key:

* Purpose: Distributes data across different nodes in the cluster.
* Benefit: Improves query performance by ensuring data is evenly spread out, avoiding hotspots.

DB to be used



Node arch

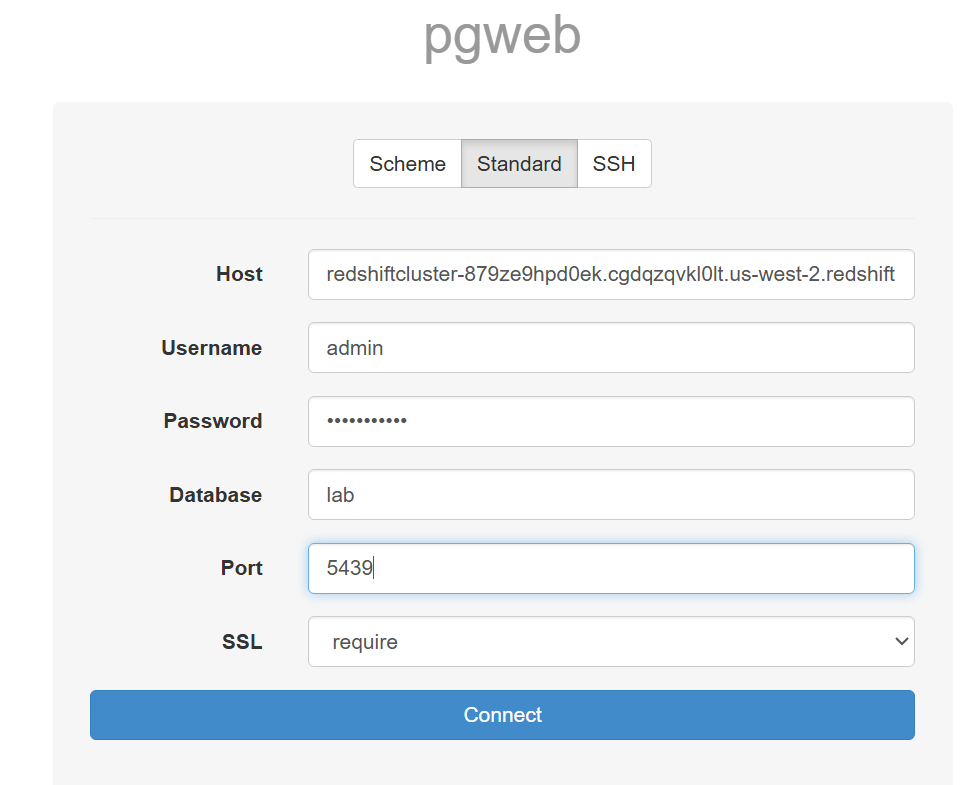


client applications send requests to the leader node, which then directs the compute nodes to process the data. The compute nodes, through their slices, work on the data and return the results to the leader node, which then sends it back to the client applications.

**Task 1: Connecting to Amazon Redshift, thru postgreSQL**

1.1 Open pgweb using URL given

1.2 use these values to connect



**Task 2: Selecting a distribution key for a column**

2.1 In this cluster - 2 nodes, 2 slices each, 4 slices in total

Key Distribution:

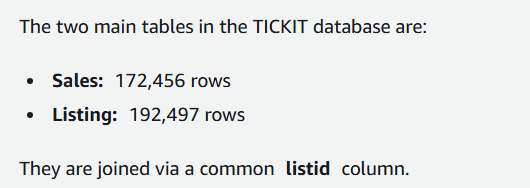
* Description: Rows are distributed across nodes based on the values in a specified column.
* When to Use: When you frequently join tables on this column.
* Example: Distributing a sales table by customer\_id if you often join with a customers table on customer\_id.

Even Distribution:

* Description: Rows are distributed evenly across all nodes.
* When to Use: When there is no clear column for key distribution, or you need to evenly distribute the load.
* Example: A large log table where you do not frequently join on a specific column.

All Distribution:

* Description: A full copy of the table is stored on each node.
* When to Use: For small tables that are often joined with larger tables, to avoid data movement between nodes.
* Example: A small reference table with currency exchange rates used in joins with a large transactions table.



2.2 join w/out any dist key

*SET enable\_result\_cache\_for\_session TO OFF;*

*EXPLAIN*

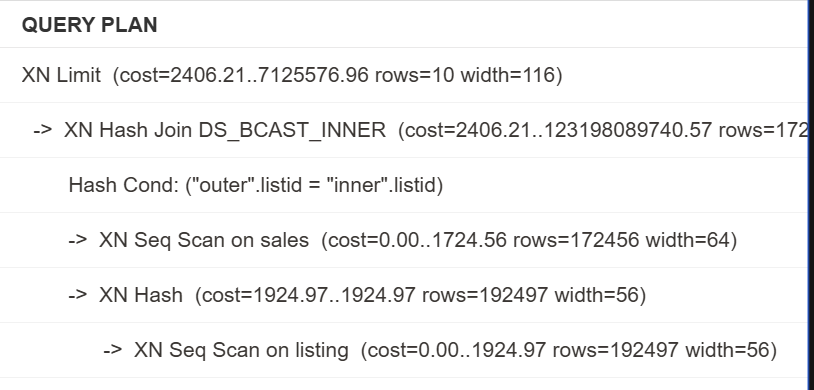
*SELECT*

*\**

*FROM sales*

*JOIN listing USING (listid)*

*LIMIT 10*



Note that the query plan includes DS\_BCAST\_INNER (Distribution: Broadcast Inner Table), which indicates that a copy of the entire inner table (the listing table) is broadcast to all of the compute nodes.

The cost in this query plan is very large (100098448011.03), indicating that a lot of data is moving around the Amazon Redshift cluster.

How to optimize -

* Common Problem:
  + When you join two tables, data might need to move between nodes, which can slow down the query.
* Solution: Use a Distribution Key (DISTKEY)
  + A distribution key helps by placing rows with the same value on the same node.
* How It Works:
  + If you often join tables on a specific column (like listid), use that column as the DISTKEY.
  + This way, rows with the same listid are stored on the same slice, reducing data movement during joins.

2.3 do this by using

*CREATE TABLE sales\_distkey*

*DISTKEY (listid)*

*AS*

*SELECT \* FROM sales;*

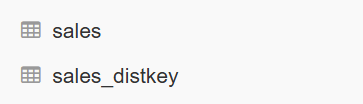
*CREATE TABLE listing\_distkey*

*DISTKEY (listid)*

*AS*

*SELECT \* FROM listing;*

o/p - new table formed



2.4 run this (same as 2.2)

*EXPLAIN*

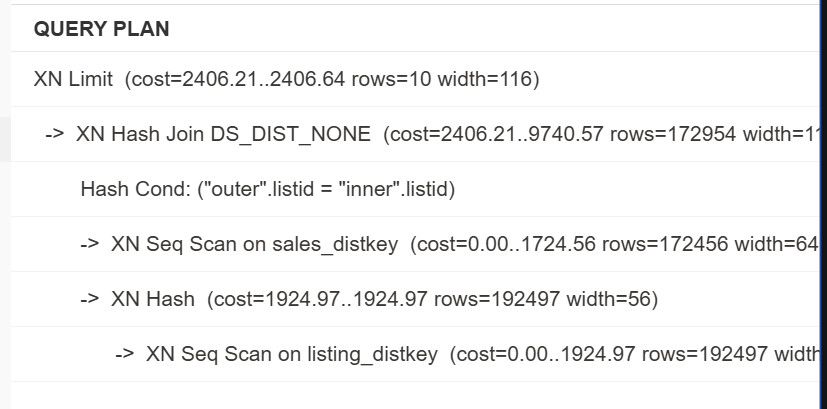
*SELECT*

*\**

*FROM sales\_distkey*

*JOIN listing\_distkey USING (listid)*

*LIMIT 10*



The query plan now says DS\_DIST\_NONE (Distribution: None), which means that no redistribution is required. This is because corresponding slices are co-located on the compute nodes because they had the same DISTKEY.

The cost has also decreased from 100,098,448,011 to 2406, making the query much more efficient.

**Task 4: Applying a sort key**

4.1 query for sorting

*CREATE TABLE big\_sales\_sorted(*

*sale\_id INT,*

*sale\_date DATE,*

*amount DECIMAL(10, 2)*

*)*

*SORTKEY (saletime);*

o/p-

New table



4.1 we use

*SELECT SUM(qtysold \* pricepaid) AS revenue*

*FROM big\_sales*

*WHERE saletime between '2008-08-01 00:00:00'::TIMESTAMP AND '2008-08-01 23:59:59'::TIMESTAMP*

Query on two tables, one is sorted, one isn’t (they store the same data)

We notice time taken in sorted one is less

4.2

| Unsorted | Sorted |
| --- | --- |