accenture

DW WITH AWS REDSHIFT

MODULE : REDSHIFT ARCHITECTURE

Learning and Knowledge Management

MODULE OBJECTIVES

At the end of this module, you should be able to:

- Explain Overview of Amazon RedShift
- Explain Amazon Redshift Architecture
- Introduce Columnar Databases
- Design Tables in Amazon RedShift
- Load Data in Amazon RedShift
- Perform Operations in Amazon RedShift

Amazon RedShift Architecture and Columnar Databases

OVERVIEW OF AMAZON REDSHIFT

What is RedShift?

- RedShift is Amazon’s fast and fully managed peta-byte scale data warehouse service in the cloud.
- Announced in the year 2012 and it is the fastest growing service from AWS.
- Designed for OLAP and BI. DO NOT use it for OLTP.
- ANSI SQL compatible.
- Column-Oriented.
- We can provision multi-terabyte clusters in minutes at a lower cost
- Refer below link to have a look at different types of nodes in RedShift Cluster.

https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-clusters.html

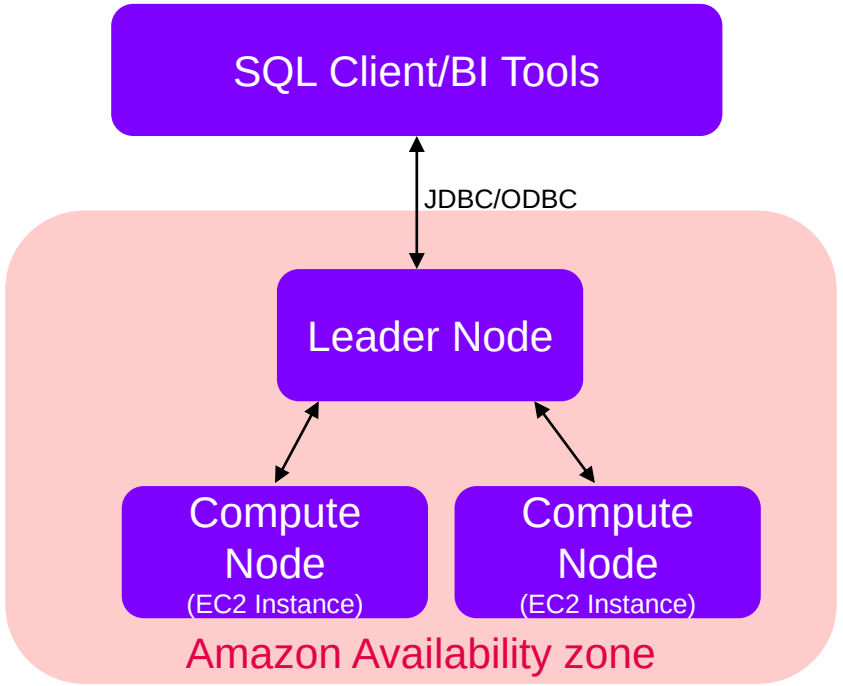
REDSHIFT ARCHITECTURE

Leader Node

- Each leader node has an endpoint.
- It stores metadata and coordinates parallel query execution.
- It manages distribution of data and query processing tasks to the compute nodes.

Compute Node

- Multiple nodes executes query in parallel and the results are returned back to leader node and in turn to SQL clients.
- Has dedicated CPU, memory and local storage.
- Can scale out/in, up/down.
- Backups to S3 are done in parallel

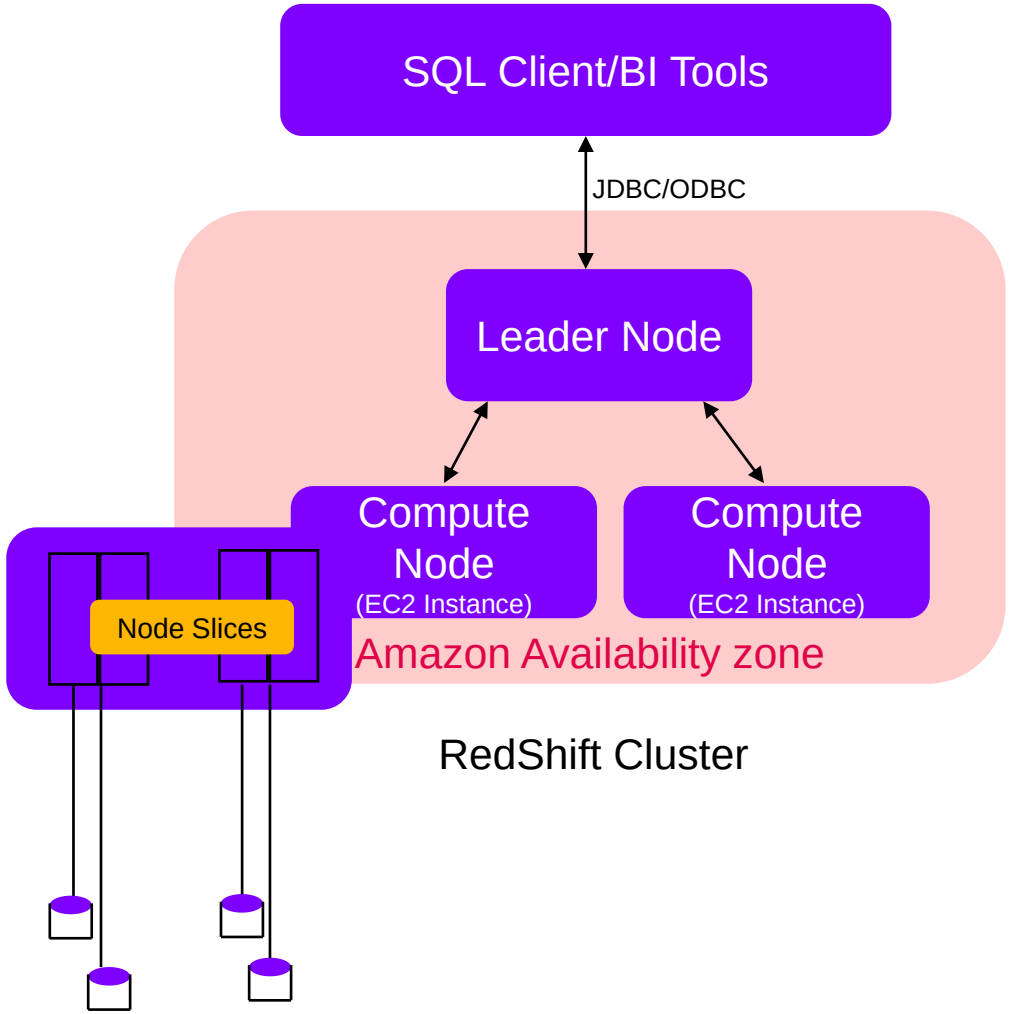


RedShift Cluster

REDSHIFT ARCHITECTURE

Slices

- Each compute node consist of slices.
- Slices are portion of memory and disk.
- Data is loaded to slices in parallel
- Processes a portion of query assigned to compute node
- Number of slices per node depends on size of node.
- Designing the tables to use slices helps to achieve optimum performance.



REDSHIFT ARCHITECTURE

Node Types

- https://aws.amazon.com/redshift/pricing/

MPP

- Massively parallel processing database
- Data stored on multiple nodes
- Nodes have dedicated cpu, memory and local storage
- “shared nothing” architectures. It means that each node is self-sufficient in terms of cpu, memory and storage.

REDSHIFT IN AWS ECOSYSTEM

Which AWS services Integrate with Redshift?

COLUMNAR DATABASES

- Columnar Databases are the database management system that stores data tables as columns rather than as rows.
- Efficient read/write operations to and from disk storage in order to speed up the time to return the query results.
  - It does this by reducing the amount of data needed to be written and read to and from the disk.
  - Columnar databases have compression algorithms which allows to reduce the amount of data to be stored on disks
  - Avoids scanning and discarding unwanted rows which results in increased query performance.
- Columnar Vs Row based databases

Logical Structure

	State	Product	Sales
Row 1	Maharashtra	Mobile Phone	100000
Row 2	Rajasthan	iPad	200000
Row 3	AP	Washing Machine	300000
Row 4	MP	AC	400000

Logical structure is same. Difference lies in Physical structure

COLUMNAR DATABASES

- Physical Structure – For Row store, data is stored on disk row by row.

Offset			
1000	Maharashtra	Mobile Phone	100000
2000	Rajasthan	iPad	200000
3000	AP	Washing Machine	300000
4000	MP	AC	400000

- For column store, data is stored column by column

Offset				
1000	Maharashtra	Rajasthan	AP	MP
2000	Mobile Phone	iPad	Washing Machine	AC
3000	100000	200000	300000	400000

COLUMNAR DATABASES

Benefits

- Queries on few columns
- Data Aggregation
- Compression
- Lower total cost of ownership

Use cases where Columnar Databases will not be helpful:

- Needle in hay stack query
- Deal with small amount of data
- BLOB
- OLTP

Design Tables in RedShift

DESIGN TABLES IN REDSHIFT

Topics Covered

- RedShift Architecture and it’s relationship to Table Design
- Distribution Styles
- Sort Keys
- Compression
- Constraints
- Column Sizing
- Data Types

TABLE DESIGN IN REDSHIFT

A Different Approach

Designing will be different than typical row-based RDBMS

- Linking Tables / Referential Integrity
- Indexes

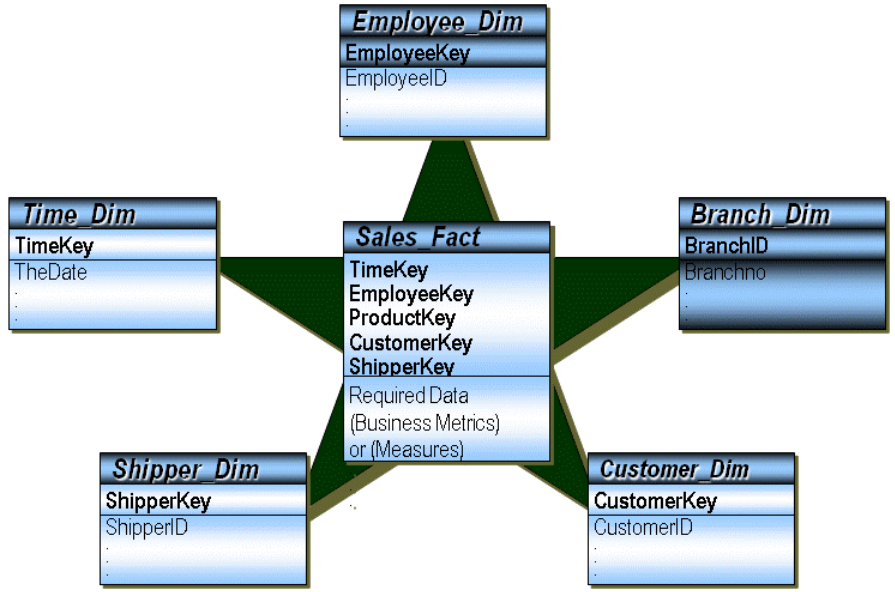
Data Model

- We will use star schema to illustrate certain points.
- Star Schema is a style of data mart schema and is the approach

most widely used to develop data warehouses and dimensional data marts.

- The star schema consists of one or more fact tables

referencing any number of dimension tables.



https://en.wikipedia.org/wiki/Star_schema

TABLE DESIGN IN REDSHIFT

Distribution Styles

- When data is loaded into a table, RedShift distributes table rows to compute nodes and slices according to the distribution style chosen at the time of creating the table
- 4 Distribution Styles
  - Even
  - Key
  - ALL
  - AUTO

TABLE DESIGN IN REDSHIFT

Even Distribution

- Leader node distributes the rows across the slices in a **round-robin fashion** regardless of the values in any one particular column.
- Even distribution is appropriate when a table doesn't participate in joins OR when there is not a clear choice between KEY and ALL distribution.
- It is the **default distribution** style.

crime_id
1000
1001
1002
1003
1004
1005
1006
1007

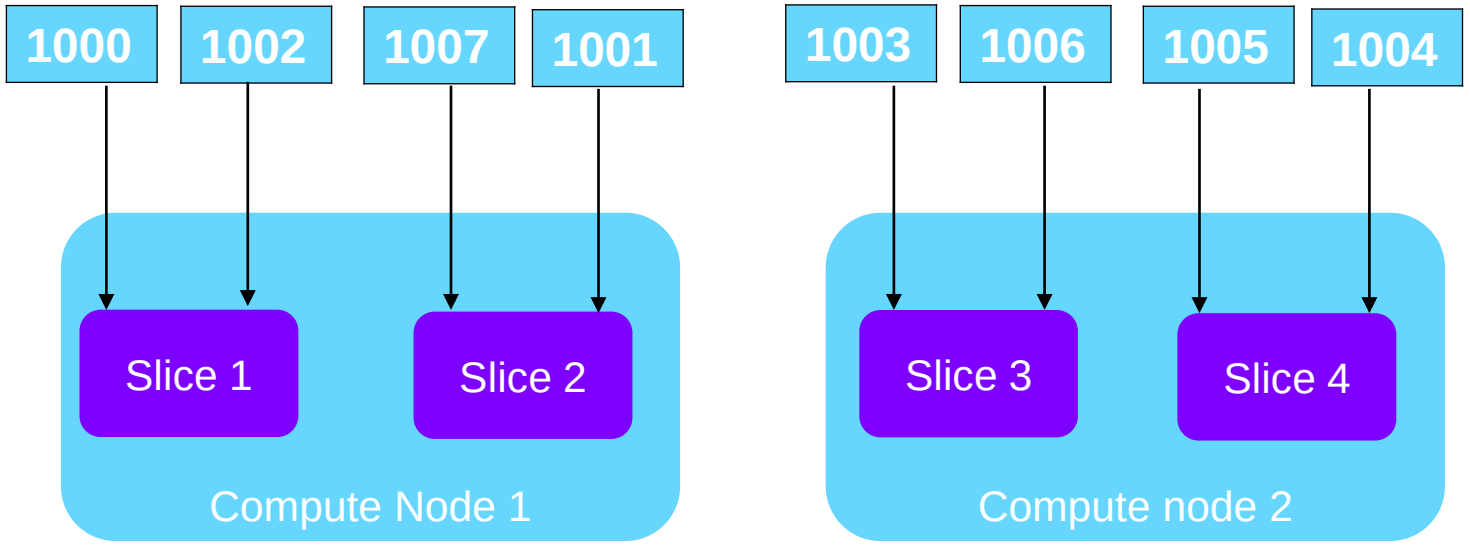
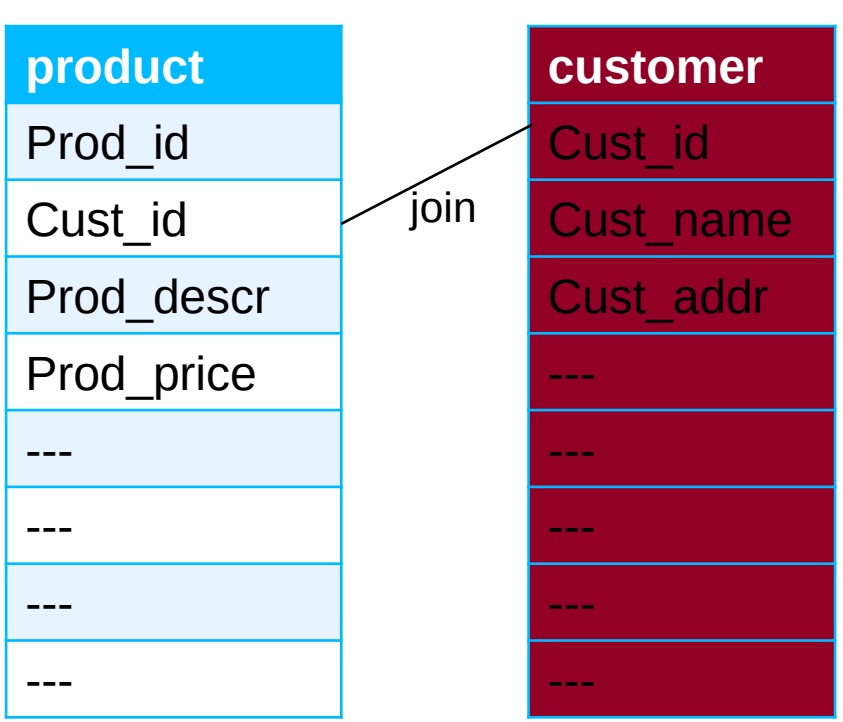


TABLE DESIGN IN REDSHIFT

Key Distribution

- Rows are distributed according to the values in one column.
- Leader node collocates matching values on the same node slice.



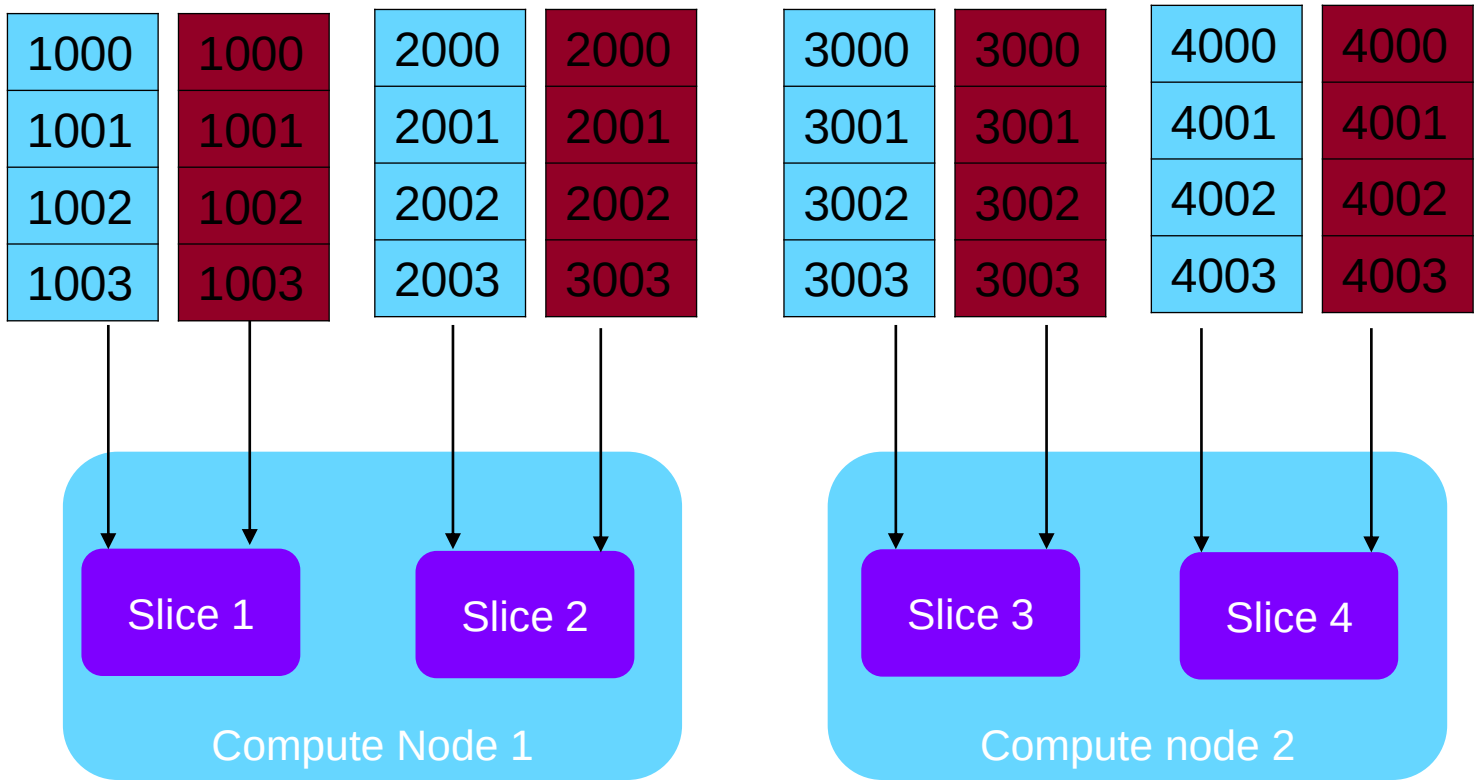


TABLE DESIGN IN REDSHIFT

Key Distribution

- If key distribution style is not used then we would risk storing matching values on different nodes.
- If you now want to look at customer' data along with their products data then matching rows will be on different nodes and this query would result in cross node traffic and will slow down the query performance.

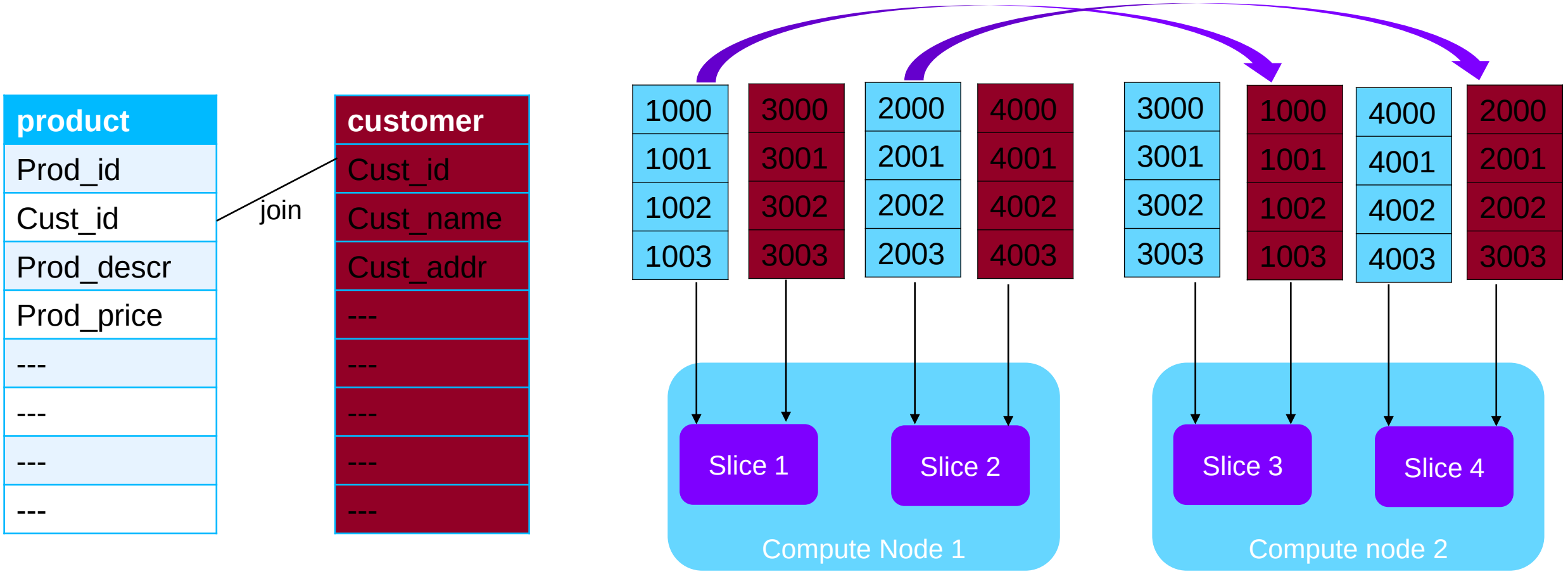


TABLE DESIGN IN REDSHIFT

ALL Distribution

- Copy of entire table is distributed to every node. It ensures that every row is collocated for every join that the table participates in.
- ALL multiples storage required by the number of nodes in the cluster and therefore, it takes much longer to load, update or insert data into multiple tables.
- ALL is appropriated for relatively slow moving tables. Small dimension tables DO NOT benefit significantly because the cost of re-distribution is low.

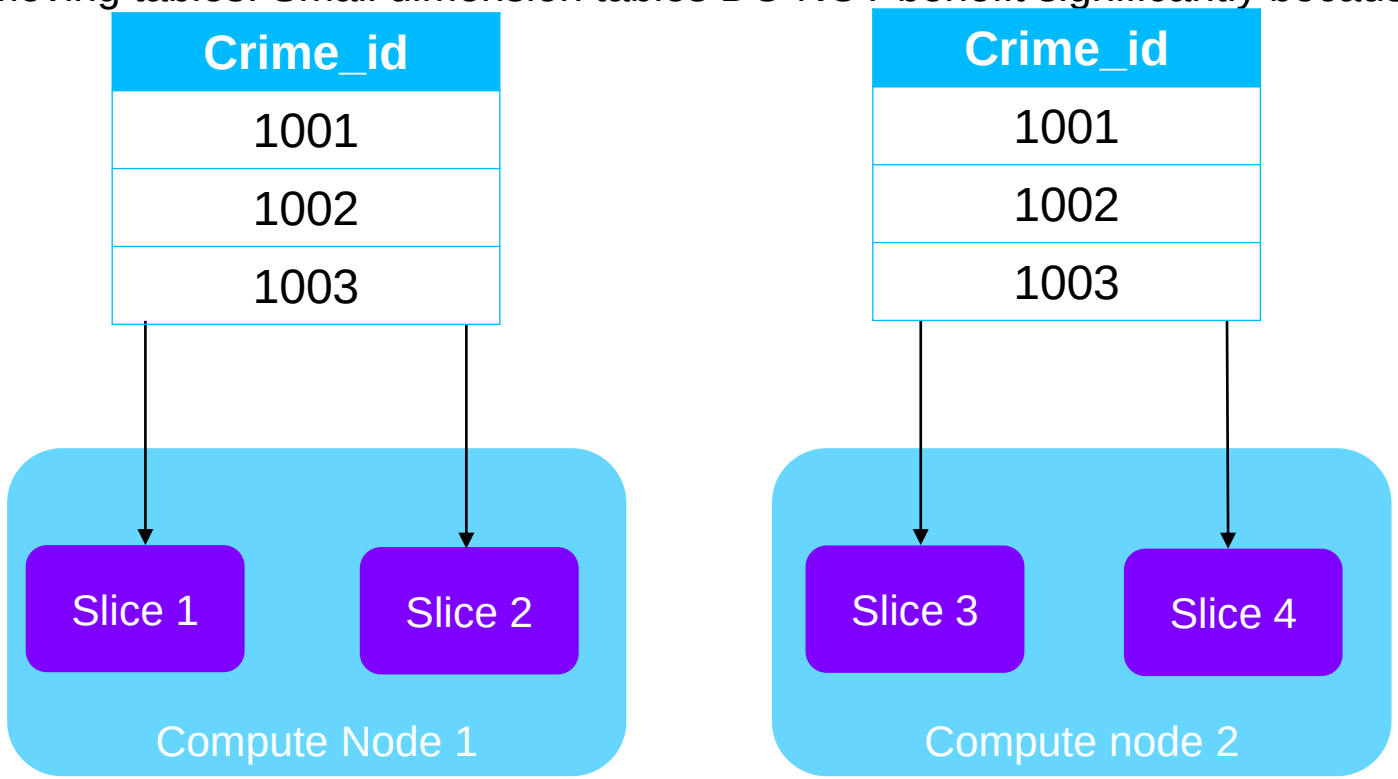


TABLE DESIGN IN REDSHIFT

AUTO Distribution

- RedShift assigns an optimal redistribution style based on the size of table.
  - For e.g.: RedShift assigns ALL distribution to a small table and changes to EVEN distribution when table grows larger.
- When distribution style is changed from ALL to EVEN, storage utilization may change slightly and this change occurs in background within seconds.
- RedShift never changes from EVEN to ALL.

TABLE DESIGN IN REDSHIFT

Sort Keys

- When we create table, we can specify one more columns as sort keys.
- Amazon RedShift stores data on disk in a sorted order.
- Sort keys are processed into an index and therefore if chose incorrectly, it may hamper query performance.
- Typical DB's uses block size of less than 32KB, however RedShift uses 1MB block size.
- RedShift has zone maps (min and max values)

SELECT date, id, case_number, location_desc FROM chicago_crime WHERE date = '02/05/2017';

date
1/1/2017
1/31/2017

date
2/1/2017
2/28/2017

date
3/1/2017
3/31/2017

chicago_crime
id
case_number
date
location_desc

TABLE DESIGN IN REDSHIFT

Sort Key

Sort Order is important

- Let's consider that data is loaded in following order:

copy chicago_crime from 's3://mybucket/data/march.gz' gzip delimiter '|';

copy chicago_crime from 's3://mybucket/data/jan.gz' gzip delimiter '|';

copy chicago_crime from 's3://mybucket/data/feb.gz' gzip delimiter '|';

SELECT date, id, case_number, location_desc FROM chicago_crime WHERE date = '02/05/2017';

Even though sort key is defined but because sort order is not maintained while loading table, above query will result into reading all the blocks rather than the one that we are looking for. Query performance is reduced.

date
3/1/2017
3/31/2017

date
1/1/2017
1/28/2017

date
2/1/2017
2/31/2017

chicago_crime
id
case_number
date
location_desc

TABLE DESIGN IN REDSHIFT

Sort Key

Types of Sort Keys:

- **Compound**
  - Default
  - Compound sort keys speed up joins, GROUP BY, ORDER BY operations and window functions that use PARTITION BY and ORDER BY
  - When you load large data with compound keys then you are left with unsorted region. This means that there are large parts of table which is unsorted and hence affects performance. As a result, VACUUM operations have to be run
  - Table sorted by columns listed in sort key order.
  - Poor performance if the query does not include primary sort column.

TABLE DESIGN IN REDSHIFT

Sort Key

Types of Sort Keys:

- **Interleaved**
  - Gives equal weight to each column in the sort key. Therefore, query predicates can use any subset of the columns that make up the sort key in any order.
  - Don’t use interleaved sort key on columns with monotonically increasing attributes such as identity columns, dates or timestamps.
  - It is useful when multiple queries are run by BI Analysts but have different filters on those queries. Query performance will be much better.
  - Table maintenance operations like data load/VACUUM are slower.
  - Interleaved keys are most useful on very large tables only – 100 million+ rows.
  - Not good for data being loaded in sort order because Interleaved keys don’t preserve this order.

TABLE DESIGN IN REDSHIFT

Data Types

Supported Data Types:

https://docs.aws.amazon.com/redshift/latest/dg/c_Supported_data_types.html

- **Use Data Types correctly**
  - YYYY/MM/DD – DATE
  - Comments – VARCHAR
  - FLAG (Y/N) – CHAR
  - Price – DECIMAL
  - How CHAR and VARCHAR datatypes are defined in RedShift?
    - https://docs.amazonaws.cn/en_us/redshift/latest/dg/r_Character_types.html
- Column sizing is important. Wide columns impacts performance.

TABLE DESIGN IN REDSHIFT

Compression

What is Compression?

- Reduces the amount of data that is stored on disk.
- Storing less data reduces cost
- Reduces Disk I/O
- Each column can have compression encoding applied to them.

  https://docs.aws.amazon.com/redshift/latest/dg/c_Compression_encodings.html

- If no compression is applied in CREATE TABLE or ALTER TABLE statement then RedShift automatically assigns compression encoding as follows:
  - Boolean, REAL or DOUBLE PRECISION data types are assigned RAW compression.
  - SMALL INT, BIG INT, DECIMAL, DATE, TIMESTAMP or TIMESTAMPZ datatypes are assigned AZ64 compression.
  - CHAR or VARCHAR datatypes are assigned LZO compression.
  - Recommendation : AWS recommends Automatic Compression.

TABLE DESIGN IN REDSHIFT

Constraints

- Column level and Table level constraints in RedShift
  - PRIMARY KEY
  - UNIQUE
  - NOT NULL
  - REFERENCES

- YOU can create constraints in RedShift, but they are not enforced
  - PRIMARY KEY – NOT ENFORCED
  - UNIQUE – NOT ENFORCED
  - NOT NULL - ENFORCED
  - REFERENCES and FOREIGN KEY – NOT ENFORCED

TABLE DESIGN IN REDSHIFT

Workload Management

- Manage long running and short running queries
- Rules to route queries to queues
- Configure memory allocations to queues
- Improve performance and experience

LIMITS

- Concurrent user connections : 500
- Total concurrency level for all user-defined queues : 50
- # user defined queues : 8
- # super queue : 1
- Default concurrency per queue : 5

RedShift Spectrum

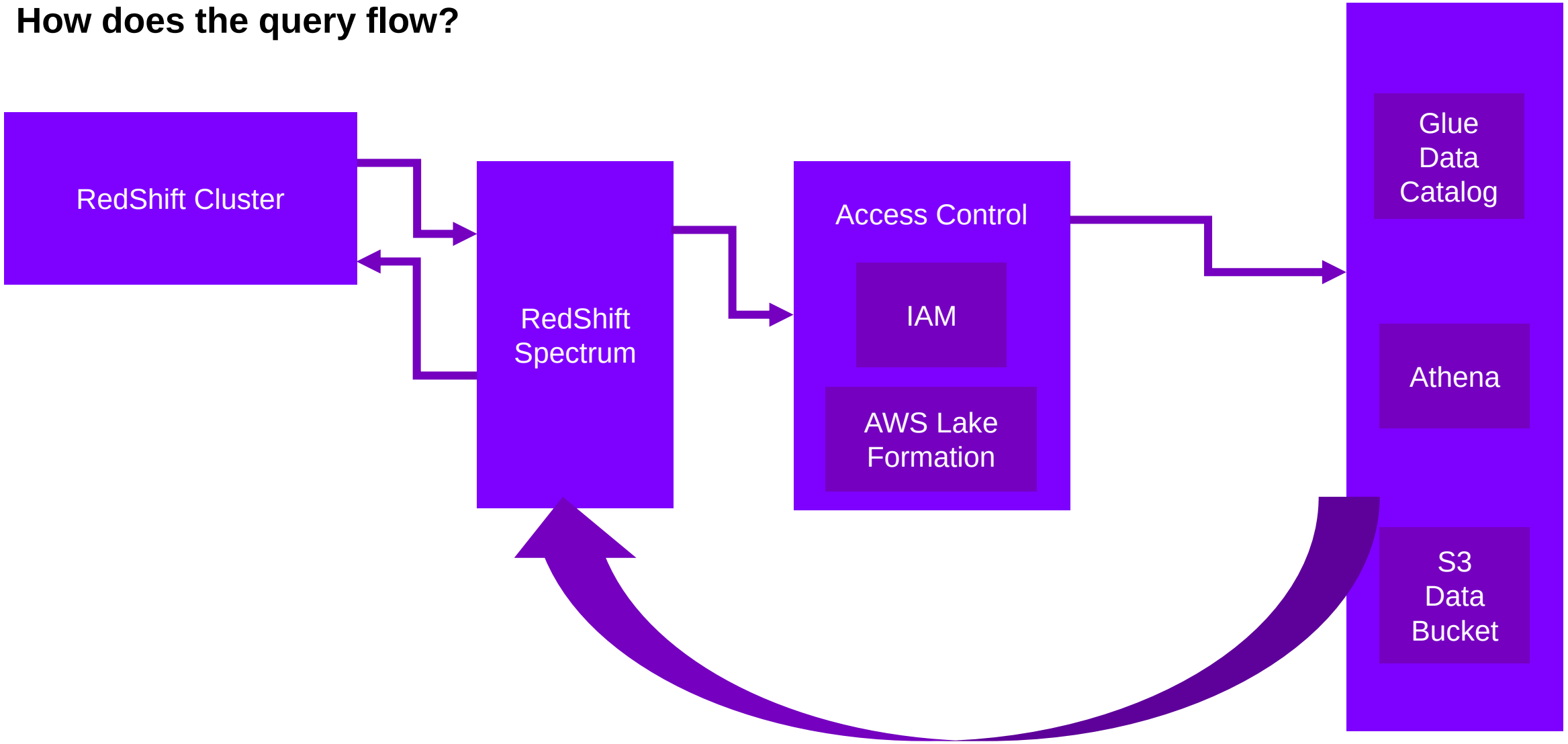
REDSHIFT SPECTRUM

What is RedShift Spectrum?

- RedShift spectrum is used to create foreign tables from data stored in S3.
- It uses EXTERNAL keyword for creating schema and tables.
- It supports INSERT and SELECT operations.
- It **does not** support UPDATE and DELETE operations.

REDSHIFT SPECTRUM

How does the query flow?



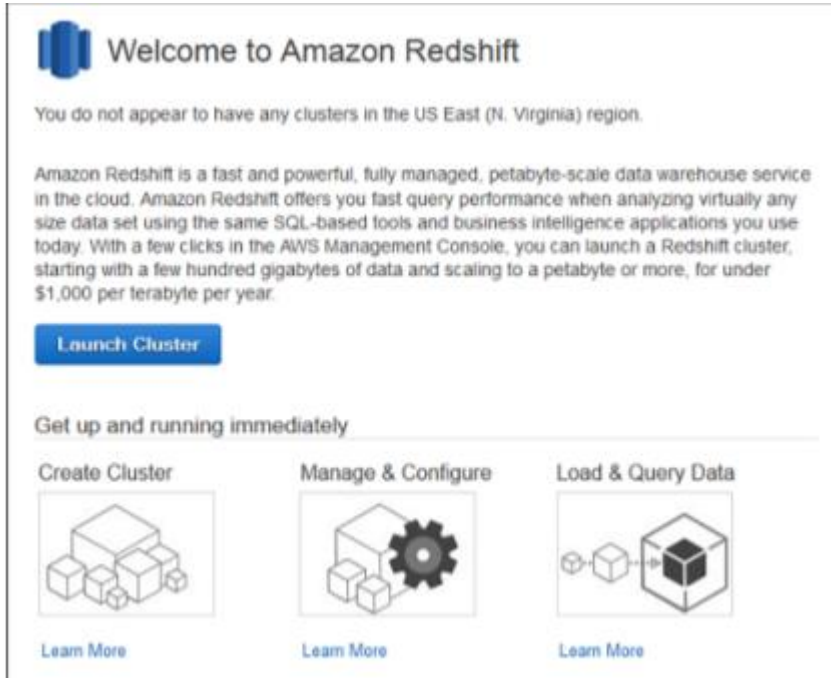
DEMO ON REDSHIFT SPECTRUM

RedShift Operations

REDSHIFT CLUSTER

Launching a Redshift Cluster

AWS Management Console



AWS CLI

https://docs.aws.amazon.com/cli/latest/reference/redshift/create-cluster.html

AWS SDK

```python
import boto3
from botocore.exceptions import ClientError


def create_redshift_cluster(ClusterId):
    """Create an Amazon Redshift cluster

    The function returns without waiting for the cluster to be fully created.

    :param ClusterId: string; Name to assign to the cluster
    :return: dictionary containing cluster information, otherwise None.
    """

    redshift_client = boto3.client('redshift')
    try:
```

REDSHIFT CLUSTER

Considerations

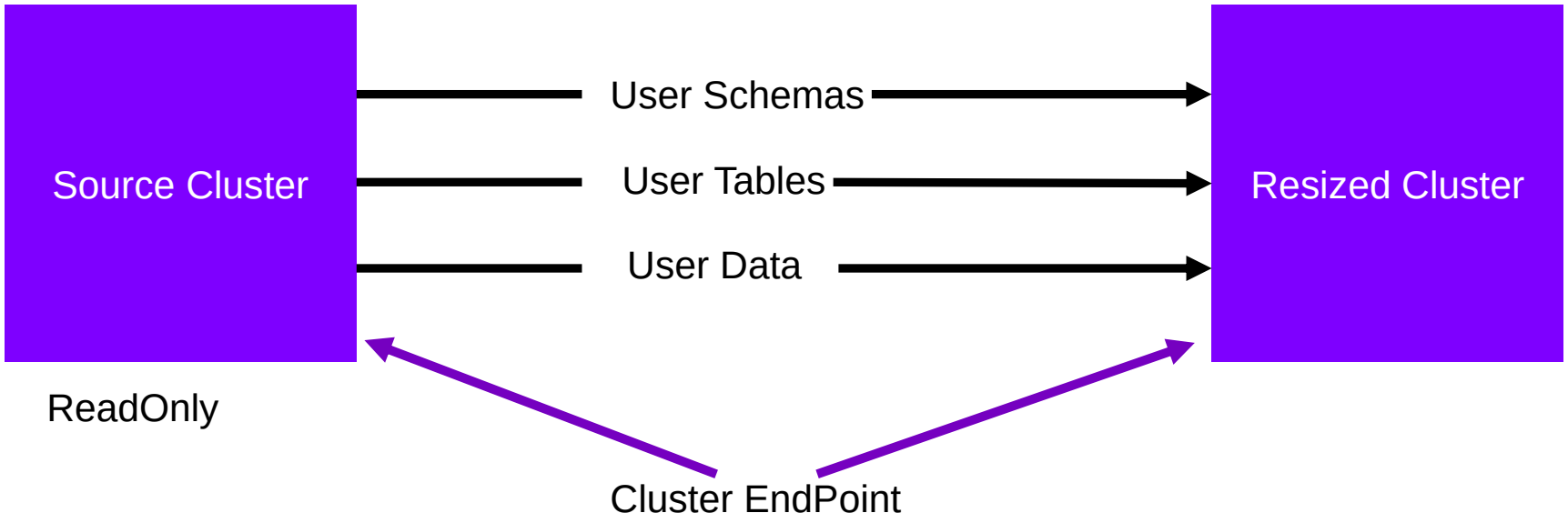
- Be aware of workloads
- To select appropriate Nodes, Identify how much storage is required and how much compute intensive is your worlload
- Also, Identify whether we are going to load any pre-existing data from another service like DynamoDB or S3

DEMO ON CREATING REDSHIFT CLUSTER USING AWS MANAGEMENT CONSOLE

REDSHIFT CLUSTER

Resizing Redshift Cluster

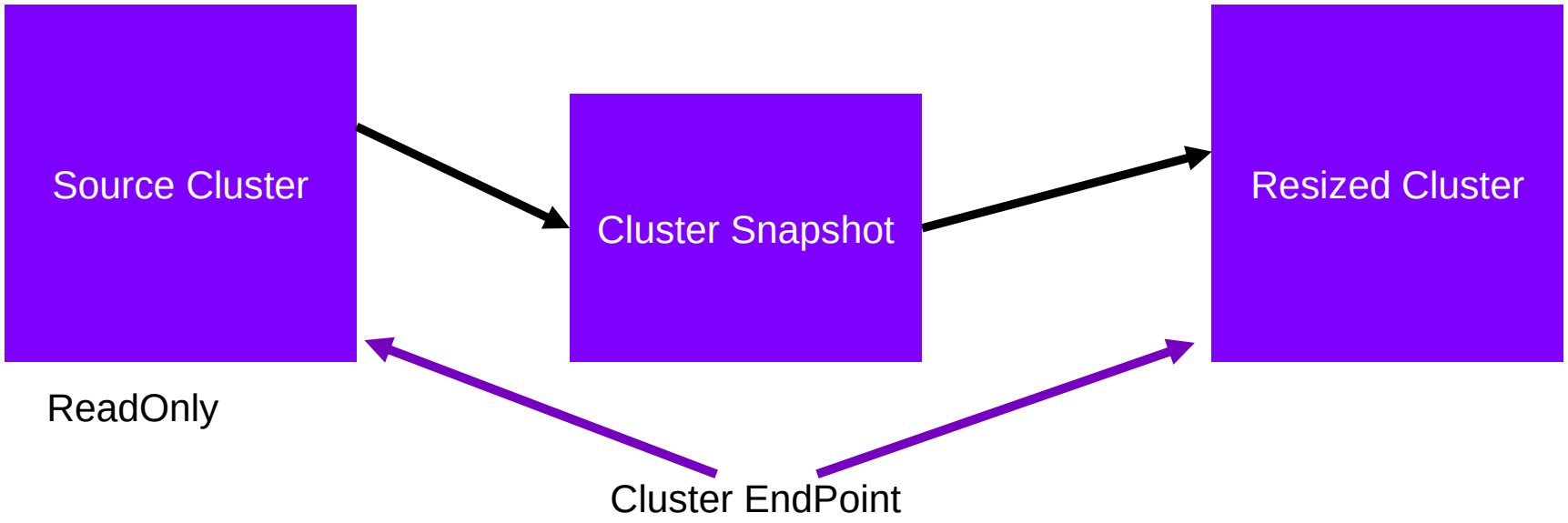
- Classic Resize
  - Source Cluster goes into ReadOnly mode while resizing the cluster.



REDSHIFT CLUSTER

Resizing Redshift Cluster

- Elastic Resize
  - Source Cluster goes into ReadOnly mode while resizing the cluster.



DEMO ON RESIZING REDSHIFT CLUSTER USING AWS MANAGEMENT CONSOLE

REDSHIFT CLUSTER

Utilizing VACUUM and Deep Copy

- Why VACUUM ?

UPDATE 1 = Dorothy

DELETE 3

INSERT Eva

Cust_ID	Cust_Name
3 X	Catherine
1 X	Anna
2	Bob
1	Dorothy
4	Eva

REDSHIFT CLUSTER

Utilizing VACUUM and Deep Copy

- VACUUM Process will remove unwanted rows
- Sort the rows
- Reindex the table

Cust_ID	Cust_Name
2	Bob
1	Dorothy
4	Eva

Cust_ID	Cust_Name
1	Dorothy
2	Bob
4	Eva

Interleaved tables require explicit VACUUM REINDEX

REDSHIFT CLUSTER

VACUUM Options

- VACUUM FULL
  - Reclaims disk space
  - Sorts Rows
  - Reindexes Tables
- VACUUM SORT ONLY
  - Sorts Rows
  - Default threshold 95% sorted
- VACUUM DELETE ONLY
  - Reclaims Disk Space
  - Default threshold 5% marked for deletion
- VACUUM REINDEX *table_name*
  - Performs VACUUM FULL option on Interleaved tables

REDSHIFT CLUSTER

Automatic VACUUM Delete

- It is triggered by high percentage of rows marked for deletion.
- With Activity monitoring, it can identify when to trigger the Delete operation.
- Automatically reclaims Disk Space.

Automatic VACUUM Table Sort

- It is triggered by high percentage of unsorted rows
- Utilizes scan operations to identify unsorted tables.

Automatic Analyze

- Automatically updates table statistics
- Waits for low activity periods to run analyze jobs
- Utilizes table Statistic age for triggering

REDSHIFT CLUSTER

What is a Deep Copy?

- It is triggered by high percentage of rows marked for deletion.
- With Activity monitoring, it can identify when to trigger the Delete operation.
- Automatically reclaims Disk Space.

Customer_ Table

Cust_ID	Cust_Name
1	Dorothy
2	Bob
4	Eva

Create a new Table

Copy Data

Drop Source

Rename Target Table

Customer_Table

Cust_ID	Cust_Name
3 X	Catherine
1 X	Anna
2	Bob
1	Dorothy
4	Eva

REDSHIFT CLUSTER

Deep Copy Methods

- CREATE TABLE copy_customer_table(...);
- INSERT INTO copy_customer_table(SELECT * FROM Customer_Table);
- DROP TABLE customer;
- ALTER TABLE copy_customer_table RENAME TO Customer_Table;

REDSHIFT CLUSTER

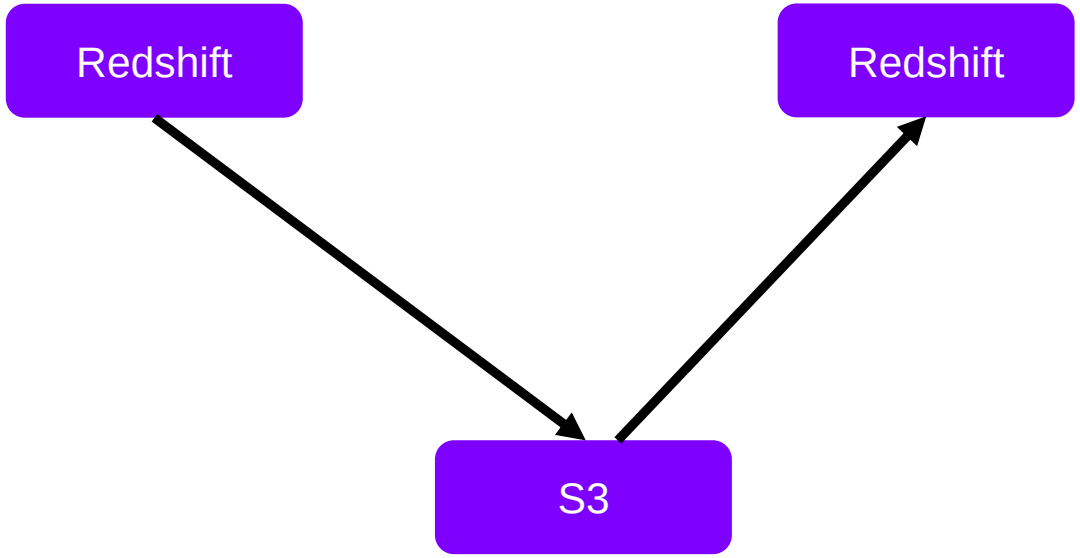
Backup and Restore

- Snapshot
  - Point in time backup of whole cluster
  - We can manually trigger.
  - Scheduled Automated Backups
  - Retention period can be configurable

REDSHIFT CLUSTER

Restore from Snapshot

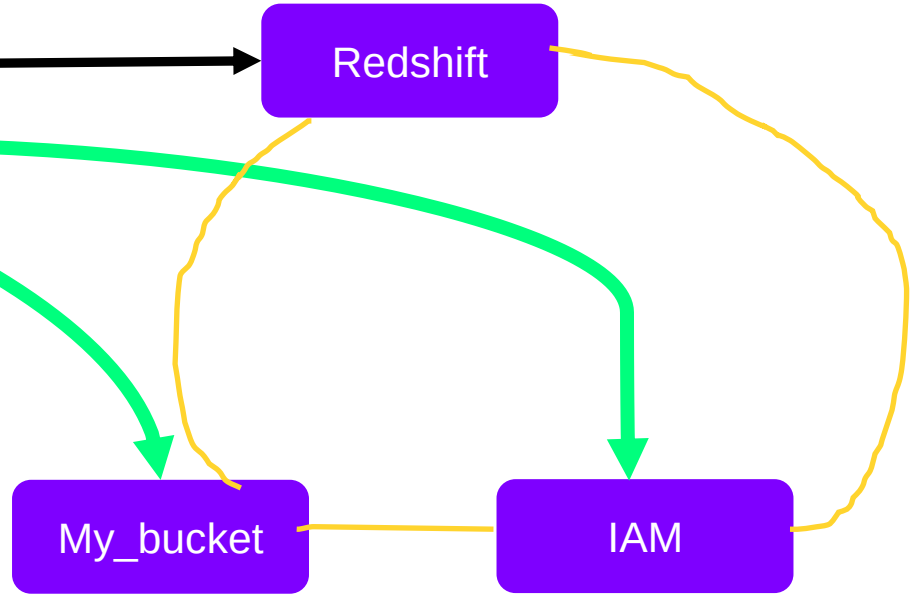
- Creates a new Cluster
- Snapshot data is lazy loaded as query request it



REDSHIFT CLUSTER

Loading Data from S3

```
copy customer_table
from 's3://my_bucket/mydata'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```



Copy command identifies target table i.e. customer_table

from directive identifies the source files

Iam_role identifies role with S3 read permissions

RedShift retrieves and loads data into a table

REDSHIFT CLUSTER

Unload Data to S3

```
unload ('select * from customer_table')
to 's3://my_bucket/customer_unload'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
FORMAT PARQUET;
```

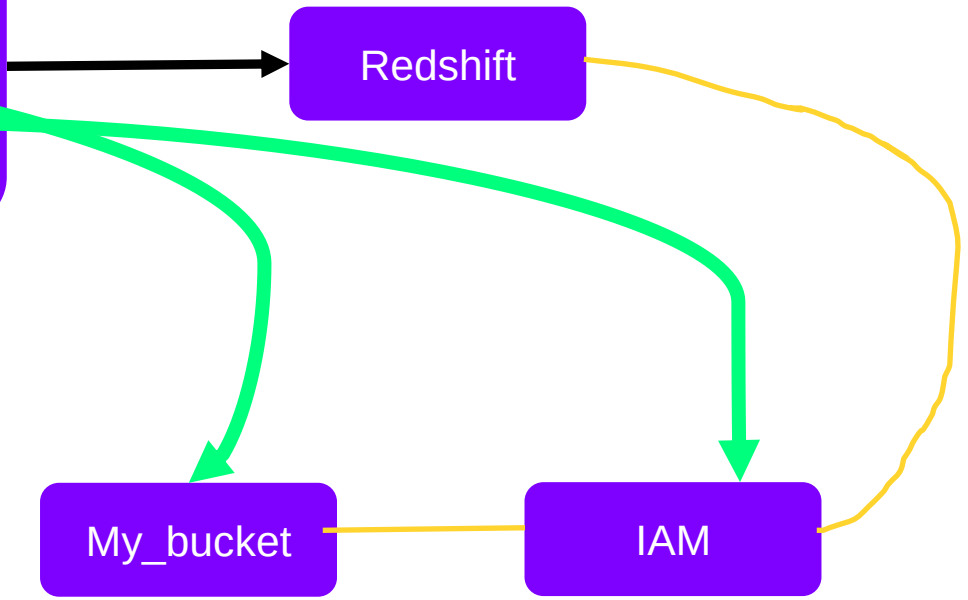
Copy command identifies target table i.e. customer_table

to directive identifies the source files

Iam_role identifies role with S3 write permissions

FORMAT directive specifies the target file format

RedShift puts the generated file in S3 bucket



DEMO ON BACKUP AND RESTORE

REDSHIFT CLUSTER

Monitoring using RedShift Console

Cluster
- CPU Utilization
- Maintenance Mode

Storage
- Percentage Disk Space Used
- Auto VACUUM Space freed
- Read Throughput
- Read Latency
- Write Throguhput
- Write Latency

Storage
- Query Duration
- Query Throughput
- Query duration per WLM queue
- Concurrency Scaling Activity
- Concurrency Scaling Usage
- Average Query Queue time By Priority

Usage Limits
- Usage Limit for Concurrency Scaling
- Usage Limit for RedShift Spectrum

Database
- Database Connections
- Total Table Count
- Health Status

REDSHIFT CLUSTER

Monitoring using CloudWatch

Cluster
- Commit Queue Length
- Concurrency Scaling Seconds
- Database Connections
- Health Status
- Maintenance Mode

Node
- CPU Utilization
- Read IOPS
- Write IOPS

DEMO ON MONITORING USING REDSHIFT CONSOLE

REDSHIFT CLUSTER

Checkpoint

- When Should we use RedShift over other Datawarehouse or Datalake?
- Does RedShift have AutoScaling features?
- Which data formats does Redshift Spectrum supports?

MODULE SUMMARY

Now, you should be able to:

- Explain Overview of Amazon RedShift
- Explain Amazon Redshift Architecture
- Introduce Columnar Databases
- Design Tables in Amazon RedShift
- Load Data in Amazon RedShift
- Perform Operations in Amazon RedShift

THANK YOU