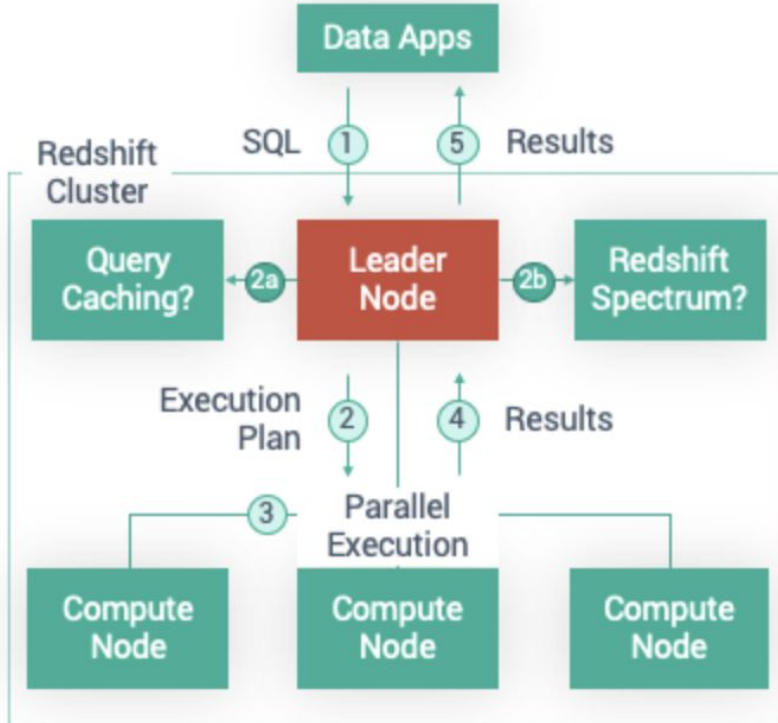


Amazon Redshift is a fully managed, cloud-based data warehouse service offered by AWS. It allows you to store and analyze large amounts of structured and semi-structured data at scale, using standard SQL and integration with business intelligence (BI) and analytics tools. The query handling efficiency is achieved through the combination of:

- **Massively Parallel Processing (MPP):** Amazon Redshift leverages a **shared-nothing architecture** where multiple processors work in parallel to execute complex queries. Each processor performs independent, coordinated computations on its own slice of data, ensuring high performance and scalability.
- **Columnar Database Design:** Redshift uses a **column-oriented storage format**, which is optimized for analytical workloads. This design makes it faster and more efficient for aggregating and analyzing large datasets. As a fully managed service, it provides a cost-effective, easy-to-use, and quick-to-deploy solution for organizations.
- **Data Compression:** Thanks to its columnar storage architecture, Redshift applies **adaptive compression encoding** based on the data type of each column. This reduces storage requirements significantly and speeds up query execution by minimizing I/O operations.
- **Advanced Query Optimization:** The Redshift query optimizer analyzes table metadata, including statistics and data distribution, to create **efficient query execution plans**. This ensures that queries run with minimal latency, even on large datasets.
- **Compiled Query Execution:** Redshift query engine converts queries into machine-compiled code, which is distributed across the cluster nodes. By eliminating the overhead of interpretation during execution, the compiled code delivers faster query performance.

Amazon Redshift Architecture: Query Caching & Redshift Spectrum

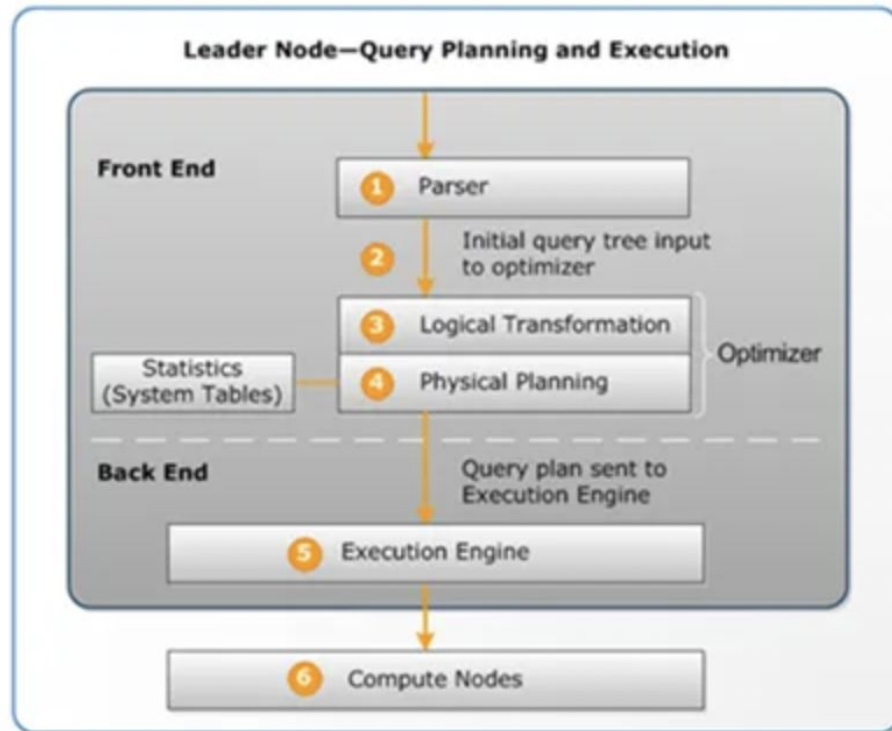


- ① The cluster receives a query coming from a data app and parses the SQL in the leader node.
- ② The leader node creates an execution plan that breaks a query down into a discrete sequence of steps.
 - ②a Determines if cached result is available
 - ②b Determines if query goes to Redshift Spectrum
- ③ The leader node distributes the work of executing the steps in parallel across the compute nodes.
- ④ The compute nodes send the results back to the leader node to merge data into a single result.
- ⑤ The leader node addresses any final sorting or aggregation and returns the results to the data app.

Leader node

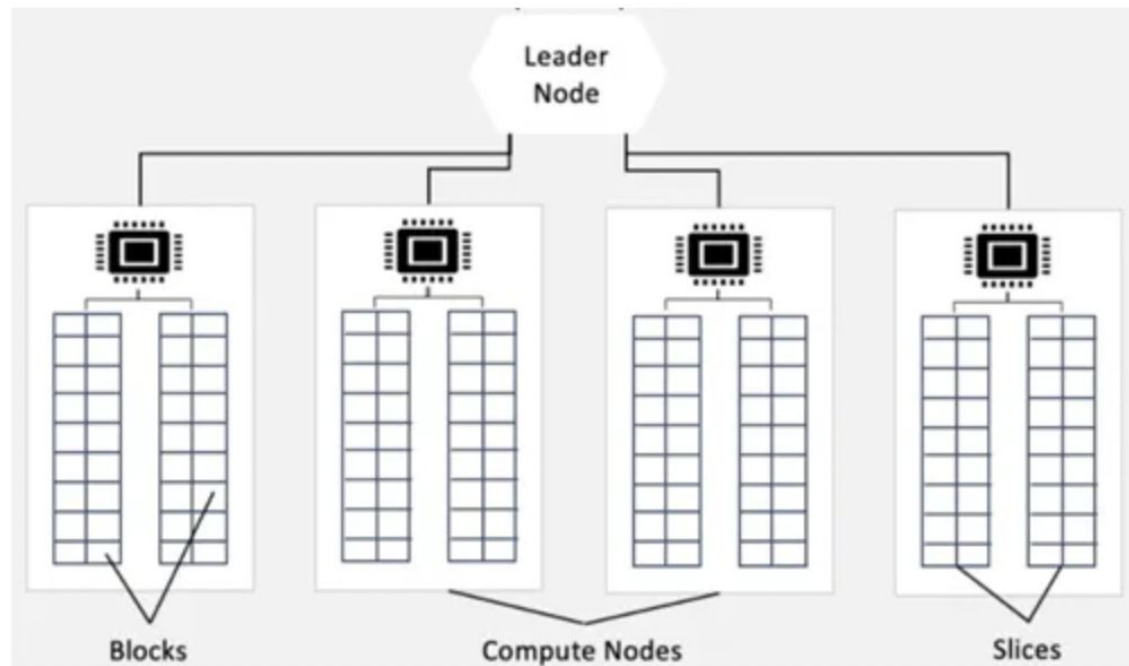
The leader node manages all external and internal communications with data applications and all communication with compute nodes. It is responsible for:

- Parsing and query rewriting,
- Planning of query processing,
- Query compiling to C++ and further distribution to compute nodes,
- Task scheduler and WLM,
- Executing queries (limited functionality).



Compute node

A **compute node** in Amazon Redshift is a worker node responsible for storing data and performing query processing. It works in parallel with other compute nodes to execute queries efficiently and return results to the leader node.

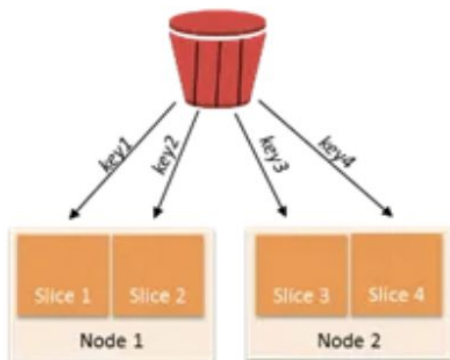


Slices - A **slice** in Amazon Redshift is a logical subdivision of a compute node, responsible for processing a portion of the node's data in parallel. Each slice independently executes queries on its assigned data, contributing to the overall performance and scalability of the cluster.

Each computational node is composed of two or more slices, which allow parallel access and processing across slices on each node. The objective of this concept is to distribute the workload of queries evenly across all nodes in order to leverage the parallel processing and to increase efficiency. The three distribution styles supported by Redshift are EVEN, KEY, and ALL (also AUTO, but it is not considered to be a separate style).

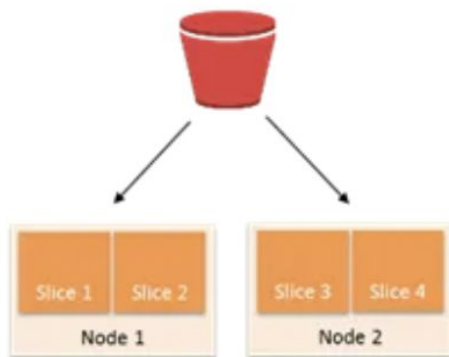
Distribution Key

key value to same location



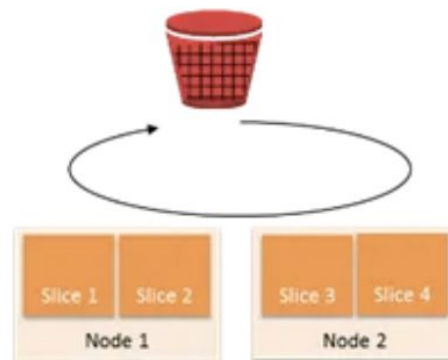
All

All data on every node



Even

Round robin distribution



Data Distribution in Slices

- **KEY distribution:** Rows are distributed according to the values in one column. The leader node places matching values on the same node slice. It is usually done with high cardinality columns [CJ1] to improve performance between two tables while joining them or aggregating values.
- **ALL distribution:** A copy of the entire table is distributed to every node. ALL distribution multiplies the storage required by the number of nodes in the cluster, and so it takes much longer to load, update, or insert data into multiple tables. ALL distribution is appropriate only for relatively slow-moving tables.
- **EVEN distribution:** The leader node distributes the rows across the slices in a round-robin fashion, regardless of the values in any particular column. EVEN distribution is appropriate when a table doesn't participate in joins. It is also appropriate when there is not a clear choice between KEY distribution and ALL distribution.

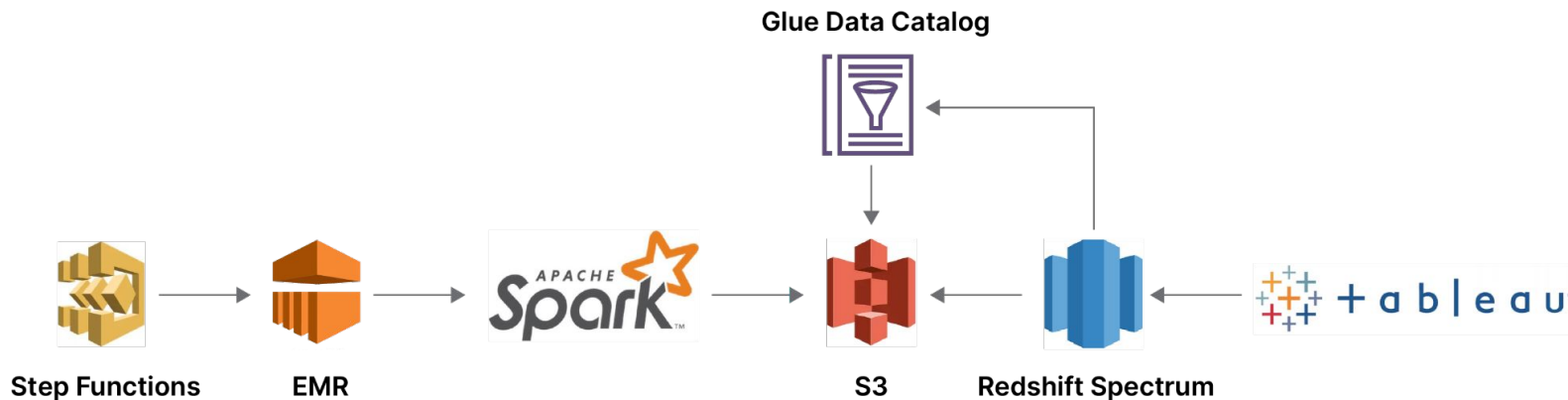
What is Redshift Spectrum?

Amazon Redshift Spectrum is a feature of Amazon Redshift, a fully managed data warehouse service in AWS. Redshift Spectrum allows users to run complex SQL queries directly against vast amounts of data in Amazon S3, **without needing to load or ETL** (Extract, Transform, Load) the data into Redshift itself. Here are some of its key characteristics and features:

- **Extensive Data Querying:** Redshift Spectrum lets you query data that resides in Amazon S3, as if they were regular Redshift tables. This makes it seamless to analyze data in its raw, stored format.
- **No ETL Required:** Since you can query data directly in Amazon S3, there's no need for an ETL process to load data into the Redshift warehouse, which can be a time-saving advantage for ad-hoc or one-off analyses.
- **Scalability:** Redshift Spectrum scales out query processing by dynamically allocating thousands of parallel query processors as required. This ensures quick results even on large datasets.
- **Cost-Effective:** With Redshift Spectrum, you pay only for the queries you run against the data stored in S3. This makes it a flexible solution, especially if you don't need to analyze your entire dataset in Redshift itself.
- **Integrated with AWS Glue:** AWS Glue's Data Catalog is used as the native metadata repository for Redshift Spectrum, making it easy to discover and manage the metadata of datasets.
- **Support for Various Data Formats:** Redshift Spectrum supports popular data formats like Parquet, ORC, JSON, Avro, and more.

What is Redshift Spectrum?

- **Security:** Redshift Spectrum offers multiple layers of security, including options to encrypt data at rest and in transit. It also integrates with AWS Identity and Access Management (IAM) for access controls.
- **Unified Data Analysis:** Users can run SQL queries that span their data warehouse and data lake, allowing for complex analyses that blend structured data in Redshift with semi-structured or unstructured data in Amazon S3.
- **Performance Optimization:** The system uses sophisticated query optimization and columnar storage on S3, reducing the amount of data that needs to be read from disk, which speeds up queries.



- **Create Table (Internal Table)**

```
CREATE TABLE sales (  
    id INT ENCODE lzo,  
    date DATE ENCODE bytedict,  
    product VARCHAR(255) ENCODE lzo,  
    quantity INT ENCODE delta,  
    revenue DECIMAL(10,2) ENCODE delta  
)  
DISTSTYLE KEY  
DISTKEY (date)  
SORTKEY (date, product);
```

- **Load data from S3 table**

```
COPY sales  
FROM 's3://your-bucket/path/to/data/sales-data.csv'  
IAM_ROLE 'arn:aws:iam::your-account-id:role/YourRedshiftRole'  
DELIMITER ','  
IGNOREHEADER 1  
REGION 'us-west-2';
```

- **Unload Query Results into S3 (Multi Parts)**

```
UNLOAD ('SELECT * FROM sales WHERE date = "2023-08-01"')  
TO 's3://your-bucket/path/to/export/sales-data-'  
IAM_ROLE 'arn:aws:iam::your-account-id:role/YourRedshiftRole'  
DELIMITER ','  
ADDQUOTES  
ALLOWOVERWRITE  
REGION 'us-west-2';
```

- **Unload Query Results into S3 (Single File)**

```
UNLOAD ('SELECT * FROM sales WHERE date = "2023-08-01"')  
TO 's3://your-bucket/path/to/export/sales-data.csv'  
IAM_ROLE 'arn:aws:iam::your-account-id:role/YourRedshiftRole'  
DELIMITER ','  
ADDQUOTES  
ALLOWOVERWRITE  
REGION 'us-west-2'  
PARALLEL OFF;
```

- **Unload Query Results into S3 with Manifest File**

```
UNLOAD ('SELECT * FROM sales')  
TO 's3://your-bucket/path/to/export/sales-data-'  
IAM_ROLE 'arn:aws:iam::your-account-id:role/YourRedshiftRole'  
DELIMITER ','  
ADDQUOTES  
ALLOWOVERWRITE  
MANIFEST;
```

- **Load into S3 with Manifest File**

```
COPY sales  
FROM 's3://your-bucket/path/to/import/sales-data.manifest'  
IAM_ROLE 'arn:aws:iam::your-account-id:role/YourRedshiftRole'  
DELIMITER ','  
MANIFEST;
```

The usefulness of a manifest file when loading data into Redshift includes:

- **Explicitness**: By using a manifest file, you're being explicit about which files to load. This can be useful in scenarios where there may be other unrelated files in the same S3 location.
- **Error Handling**: If a COPY operation fails for some files but succeeds for others, you can adjust the manifest file to only retry the files that failed, rather than retrying the entire dataset.
- **Parallelism**: Redshift can load multiple files in parallel. If your data is split across many files (which is often the case when unloading data from Redshift or using distributed data processing frameworks), using a manifest file ensures that all files are loaded concurrently, utilizing Redshift's MPP (massively parallel processing) capabilities.
- **Flexibility**: Manifest files give you flexibility in data loading. For example, you can have files from different S3 locations or even from different S3 buckets in a single manifest.
- **Ensuring Completeness**: Especially in environments where new data files might be continuously added to an S3 location, using a manifest ensures that you know exactly which files were loaded into Redshift.

- **Create External Schema**

```
CREATE EXTERNAL SCHEMA your_external_schema
FROM DATA CATALOG
DATABASE 'your_external_database'
IAM_ROLE 'arn:aws:iam::your-account-id:role/YourRedshiftRole'
CREATE EXTERNAL DATABASE IF NOT EXISTS;
```

- **Create Table - Redshift Spectrum (External)**

```
CREATE EXTERNAL TABLE spectrum_schema.sales (
  id INT,
  date DATE,
  product VARCHAR(255),
  quantity INT,
  revenue DECIMAL(10,2)
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 's3://your-bucket/path/to/data/'
TABLE PROPERTIES ('skip.header.line.count'='1');
```

Materialized view in Redshift

A materialized view in Amazon Redshift is a database object that contains the results of a query. Unlike a regular view which is just a saved SQL query that runs against the base tables every time it's queried, a materialized view stores the actual query results in the Redshift cluster.

Here are some reasons why you would want to use materialized views in Redshift:

- **Performance:** Since materialized views store the query results, accessing a materialized view can be much faster than running the underlying query, especially if the query is complex and involves multiple joins, aggregations, and filtering.
- **Minimize Redundant Computation:** If you find yourself running the same heavy query frequently, it's wasteful to compute the results from scratch every single time. Materialized views allow you to compute once and reuse the results.
- **Automated Refresh:** You can set materialized views to be refreshed automatically at regular intervals or manually as needed, ensuring that the data remains relatively up-to-date without constant manual intervention.
- **Simplified Querying:** Materialized views can simplify complex business logic into a single table-like structure. This can make it easier for analysts and other end-users to fetch data without having to understand the intricacies of the base tables and joins.
- **Optimization:** Redshift can further optimize how the data in a materialized view is stored, making reads even faster. For instance, using sort keys and data distribution strategies.
- **Cost-effective:** For heavy queries that would otherwise be run multiple times, using a materialized view can save computational resources, translating to cost savings.

- **Create Materialized Views**

```
CREATE MATERIALIZED VIEW sales_summary AS  
SELECT product, SUM(quantity) AS total_quantity, SUM(revenue) AS total_revenue  
FROM sales  
GROUP BY product;
```

- **Refresh Materialized Views**

```
REFRESH MATERIALIZED VIEW sales_summary;
```