Data Warehousing

Amazon Redshift

References/Acknowledgements

http://docs.aws.amazon.com/redshift

Agenda

- **Use Case**
- **Data Warehouse (Intro)**
  - OLAP vs OLTP
- **Amazon Redshift**
  - AWS Service for Data Warehousing
  - Redshift Cluster Architecture
  - Key Characteristics
- **Design Considerations**
- **Launching and Accessing Redshift Cluster**

Use Case/Scenario

- **Assume Dulles Airport has become overly-crowed lately**
  - Runways are running at their full capacities
- **Govt/FAA seeks GMU's help to alleviate overcrowding of Dulles airport**
- **Option includes**
  - Expand existing airport(s)
  - Identify a new location to build a new airport
    - in Washington DC Metro area
- **What will be our approach to help guide the government to make the right decision?**
  - Which option should we recommend and how do we decide that option?
  - What is needed to answer this question?

Use Case/Scenario

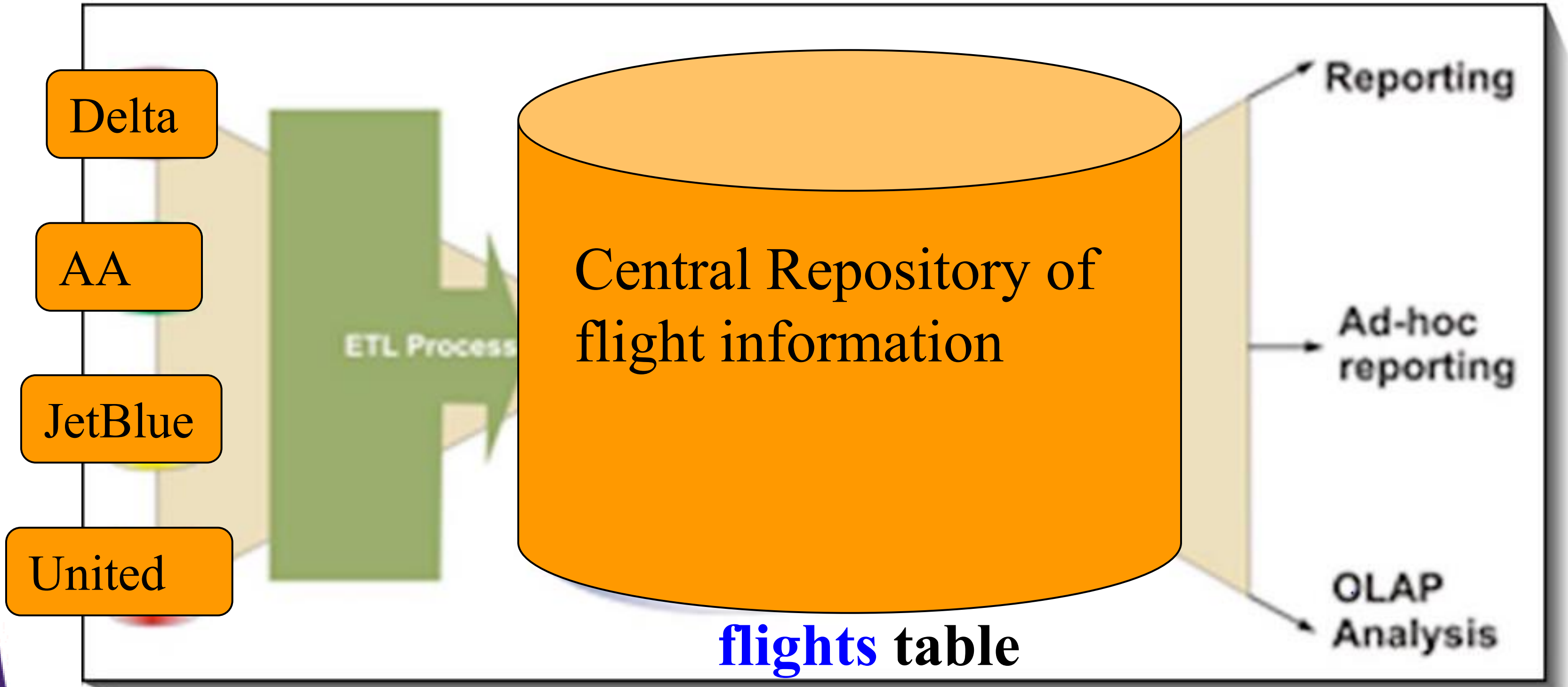
- **A crucial information:** **What is the geographical area where majority of travelers live?**

```
select zipcode, count(*) from flights group by zipcode;
```

Use Case/Scenario

- **A crucial information: What is the geographical area where majority of travelers live?**

select zipcode, count(*) from flights group by zipcode;



What is a Data Warehouse?

- **A large store of data accumulated from a wide range of sources within an enterprise**
  - Used for reporting and data analysis to help guide management make right decisions
  - Central repositories of integrated data from one or more disparate sources

Source: http://www.ibmbigdatahub.com/blog/ensuring-data-warehouse-quality-business-mandate

What is a Data Warehouse?

- A data warehouse, in general, is based on a **relational database**
  - designed for query and analysis rather than for transaction processing.
- It usually **contains historical data**
  - derived from transaction data
- It **separates analysis workload**
  - from transaction workload
- Typically includes
  - an ETL solution,
  - an OLAP engine, and
  - client analysis tools

What is Amazon Redshift

Amazon Redshift

- **Amazon's relational data warehouse service in AWS**
- **Fully Managed**
  - Most administrative tasks are fully automated
    - Provisioning, configuration, backups, patching, replication
    - Continuous monitoring and recovery from failures
- **Petabyte Scale**
  - Optimized for datasets ranging from a few hundred gigabytes to a petabyte or more
  - 1PB = 1000 terabytes = 1000000 gigabytes
    - Think the size of 10 billion photos on FACEBOOK

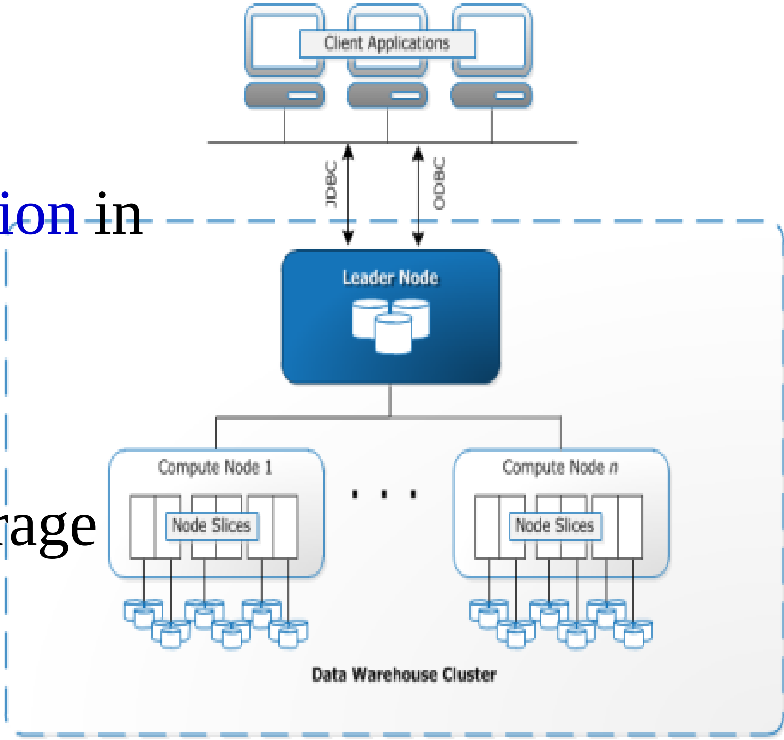
What is Amazon Redshift

- **Delivers fast query and I/O performance for virtually any size dataset using**
  - Columnar storage technology
  - Massively parallel processing (MPP) architecture
    - Parallelizes and distributes queries across multiple nodes
- **Designed for OLAP and BI applications**
  - ANSI SQL compliant
- **A powerful and cost-effective data warehouse solution**
  - “Pay as you go” pricing model
  - $1000 per TB per year

Amazon Redshift

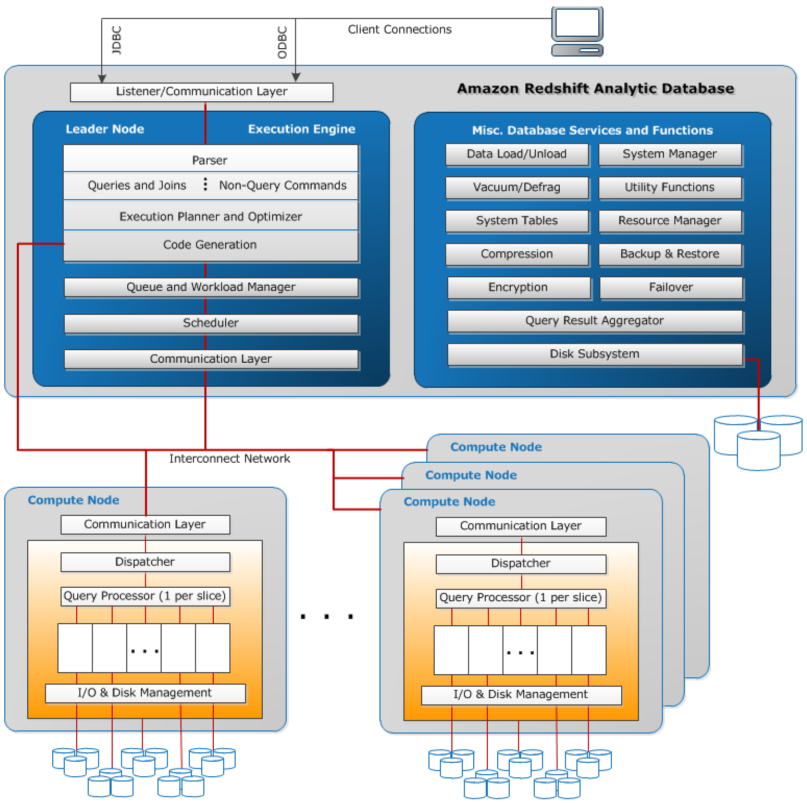
Redshift Architecture

- **Redshift data warehouse is a cluster of nodes (computing resources)**
- **Leader Node (One)**
  - Has SQL endpoint
    - For client connectivity
  - Stores metadata about the cluster
  - Coordinates parallel query execution in compute nodes
  - The brain behind Redshift cluster
- **Compute Nodes**
  - Stores data in local, columnar storage
  - Execute queries in parallel
  - Load, backup, restore



Redshift Architecture

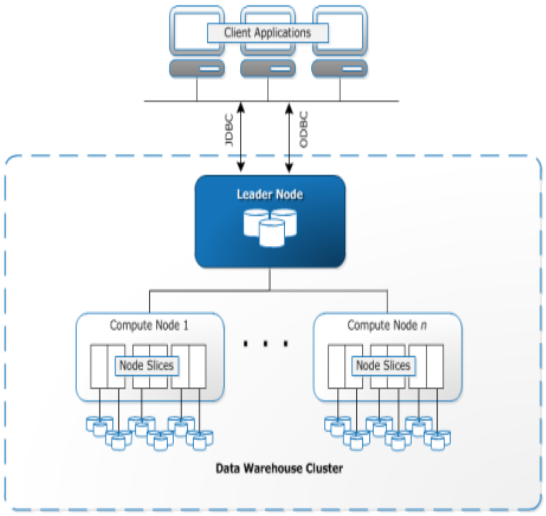
- **Massively parallel processing architecture**
  - Redshift distributes workload to each node in the cluster and processes work in parallel



High throughput b/w components
- Direct Attached Storage
- 10 GigE mesh network

Hardware Platforms used by Redshift

- **Dense Storage (DS2) Nodes**
  - Based on Hard Disk Drives (HDD)
  - Can scale from 2TB to 2PB
  - Ideal for large data warehouses
- **Dense Compute (DC1) Nodes**
  - Use Solid State Drive (SSD)
  - Can scale from 160GB to 326TB
  - Ideal for high performance databases



Family	Node Type	Storage type	Storage per node	Maximum Storage
Dense Compute	dc1.large	SSD	0.16TB	32 nodes = 5.12TB
Dense Compute	dc1.8xlarge	SSD	2.56TB	128 nodes = 326TB
Dense Storage	ds2.xlarge	Magnetic	2TB	32 nodes = 64TB
Dense Storage	ds2.8xlarge	Magnetic	16TB	128 nodes = 2PB

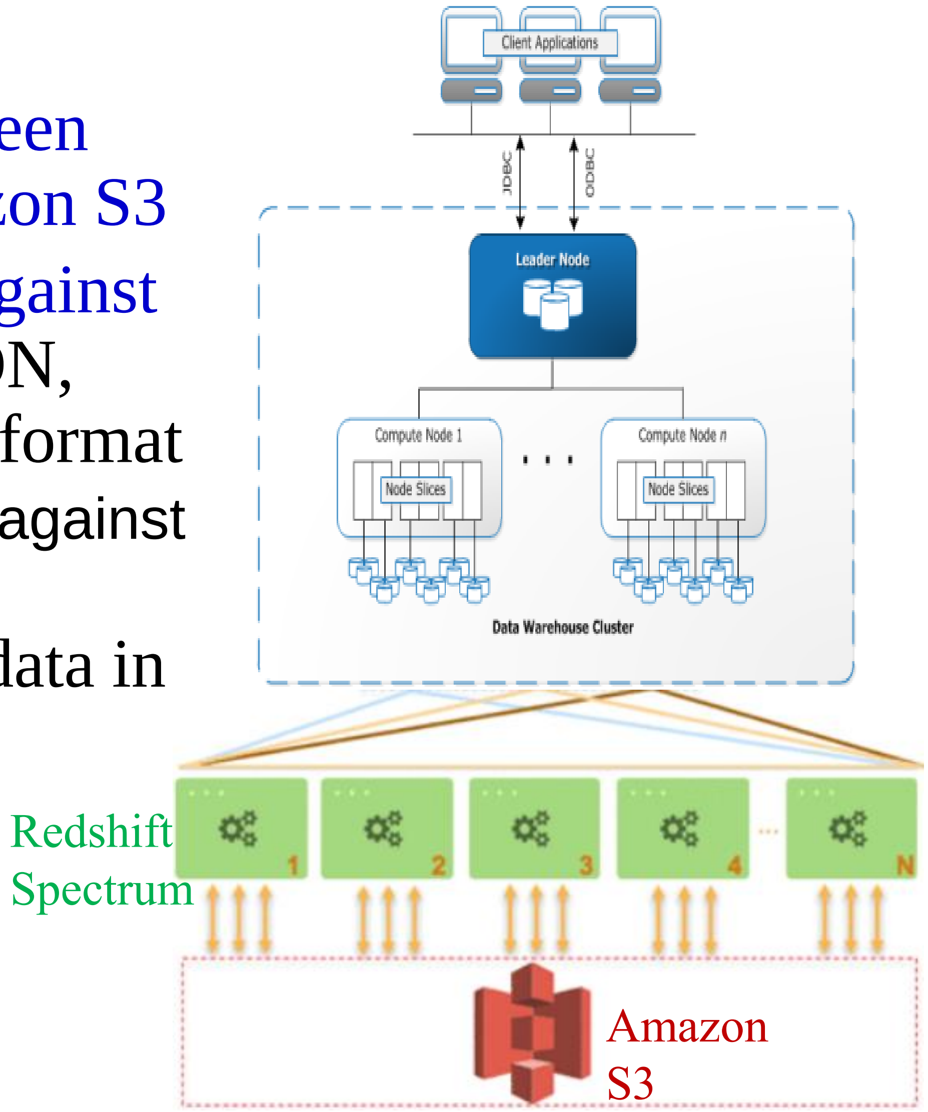
- **Optimized for data processing**

Redshift Spectrum

- **An extension to Amazon Redshift**
- **Allows you to query and retrieve structured and semi-structured data from files in Amazon S3**
  - **without having to load the data into Amazon Redshift tables.**
  - Much of the processing occurs in the Redshift Spectrum layer, and most of the data remains in Amazon S3.
  - Multiple clusters can concurrently query the same dataset in Amazon S3 without the need to make copies of the data for each cluster.
- **Redshift Spectrum queries employ massive parallelism to execute very fast against large datasets.**

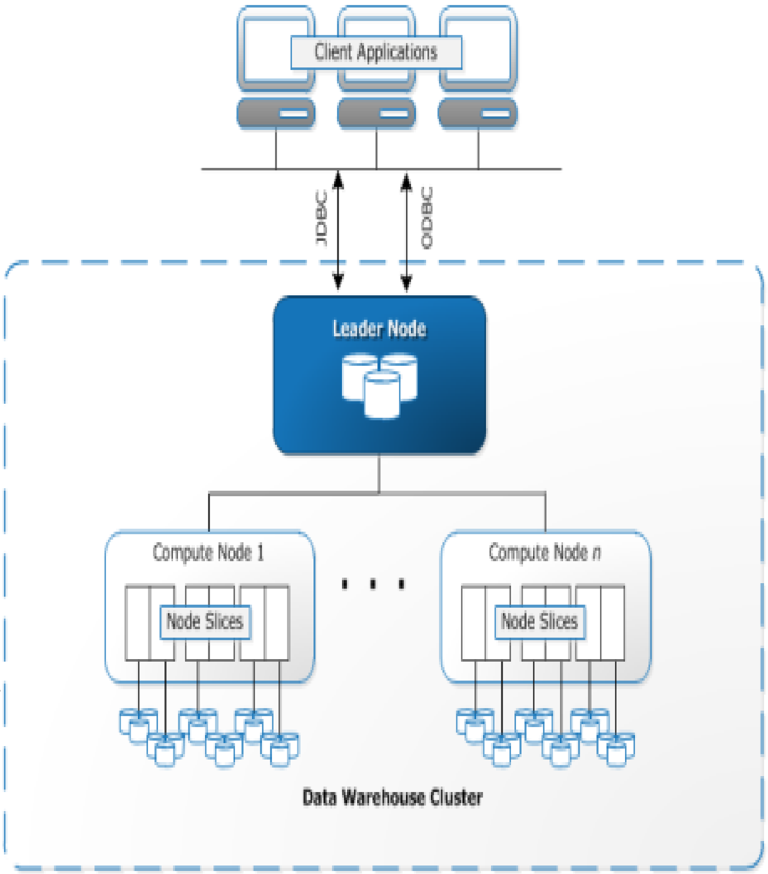
Redshift Spectrum

- **Redshift Spectrum**
  - Spectrum layer lays between compute nodes and Amazon S3
  - Executes query directly against Amazon S3 stored in JSON, ORC, CSV, Parquet, etc. format
    - Executes Redshift SQL against those files in S3
  - Can join data in S3 with data in Redshift cluster



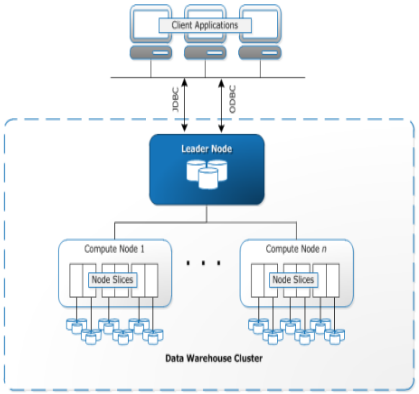
Provisioning Redshift Cluster

- **To provision a cluster:**
  - Log into the Redshift console
  - Specify the type and number of nodes that will make up the cluster
    - Node type determines the storage size, memory, CPU, and price of each node
  - Select a few configurations and launch the cluster
  - See an example later



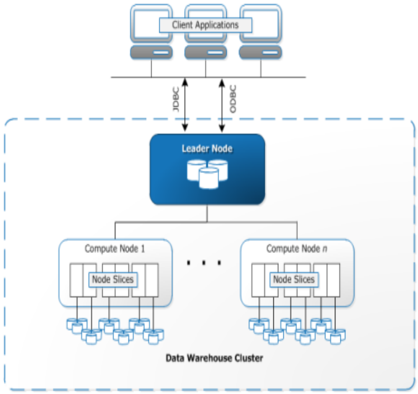
Scalability in Redshift cluster

- **Achieved by resizing the cluster**
  - Scale out or in by adding or removing nodes
  - Scale up or down by specifying a different node type
  - Or, you can do both
- **Resizing replaces the old cluster at the end of the resize operation**
  - The source cluster remains in read-only mode until the resize operation is complete
  - Resizing involves minimal downtime



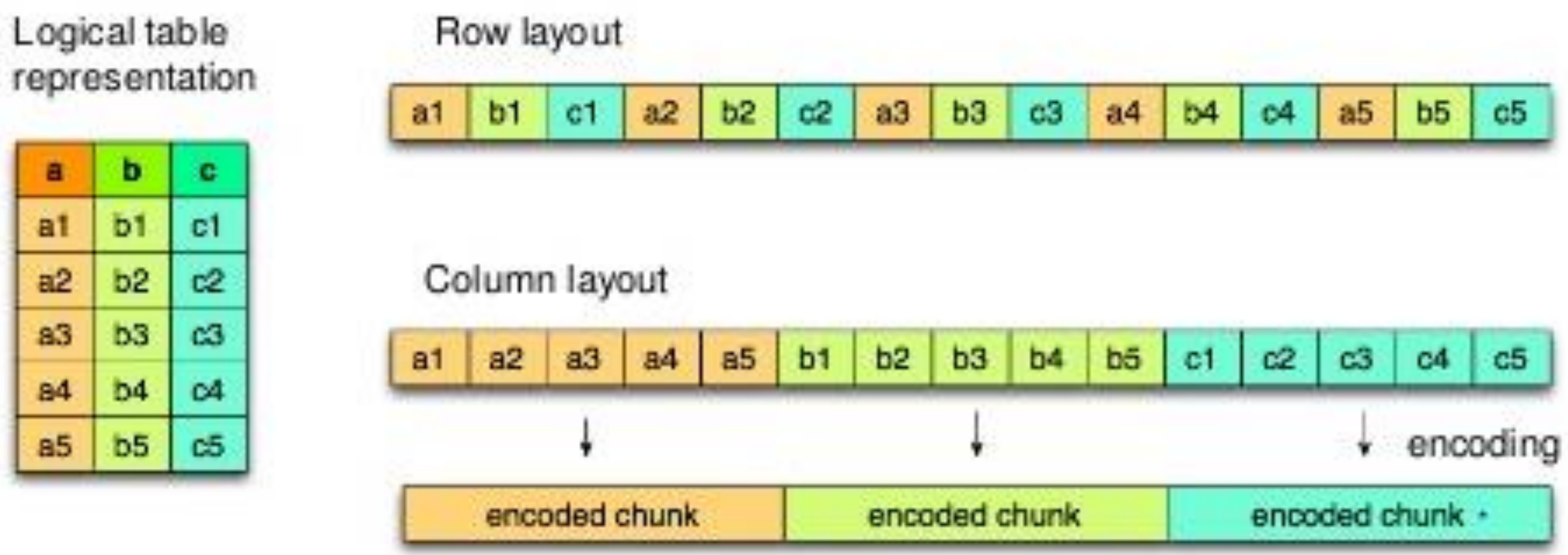
Characteristics that makes Redshift unique

- **Variety of innovations to reduce disk I/O**
  - Columnar Storage
  - Data Compression
  - Zone Maps
  - Data Sorting
    - Sort Keys
  - Node Slices
    - Support for massive parallelism
  - Data Distribution
    - Dist Key
- **Data protection and Redshift security**



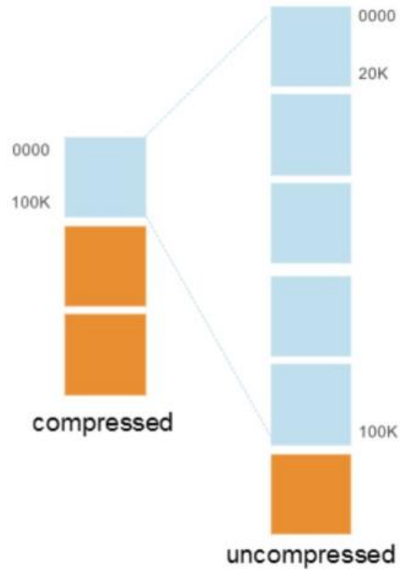
Columnar Storage

- **Redshift stores data on disk by columns**
  - rather than storing data values together for a whole row
- **For analytics queries, which typically reads only subset of columns, only scans blocks relevant to columns**
  - Results in massive reduction of disk I/O
    - Less data to load from disk



Columnar Compression

- **Compression is a column-level operation that reduces the size of the data when stored**
  - Reduces storage requirements by 50 to 75%
    - Storing less data lowers the cost of ownership
  - Reduces disk I/O during query execution
    - Improves query performance
  - In Redshift, compression and encoding are used interchangeably



Columnar Compression

- **Enables effective compression for each data type per column due to like data**
  - Compression encoding is specified in DDL statement
    - As are sort key(s) and distribution style

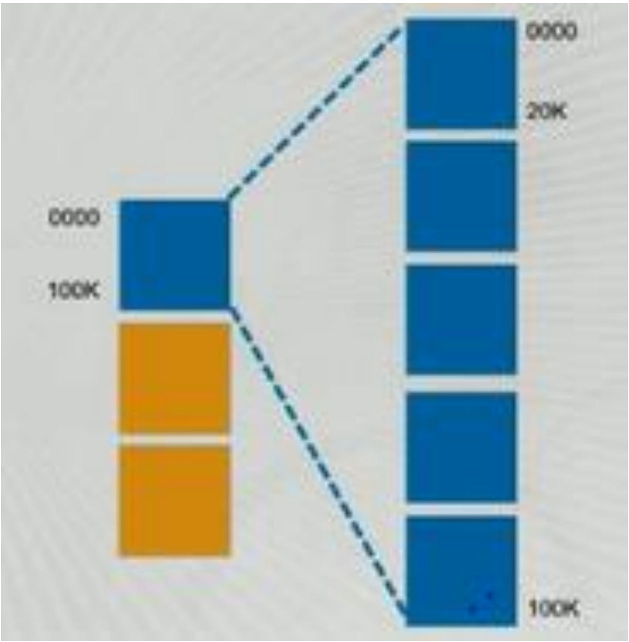
Compression encodings

- Raw (RAW)
- Byte-dictionary (BYTEDICT)
- Delta (DELTA / DELTA32K)
- Mostly (MOSTLY8 / MOSTLY
- Runlength (RUNLENGTH)
- Text (TEXT255 / TEXT32K)
- LZO

```
CREATE TABLE public.orders
(
    o_orderkey BIGINT NOT NULL ENCODE mostly32 DISTKEY,
    o_custkey BIGINT NOT NULL ENCODE mostly32,
    o_orderstatus CHAR(1) NOT NULL ENCODE lzo,
    o_totalprice NUMERIC (12, 2) NOT NULL ENCODE mostly32,
    o_orderdate DATE NOT NULL ENCODE delta,
    o_orderpriority CHAR(15) NOT NULL ENCODE bytedict,
    o_clerk CHAR(15) NOT NULL ENCODE lzo,
    o_shippriority INTEGER NOT NULL ENCODE runlength,
    o_comment VARCHAR(79) NOT NULL ENCODE text255
)
SORTKEY
(
    o_orderdate
);
```

Blocks

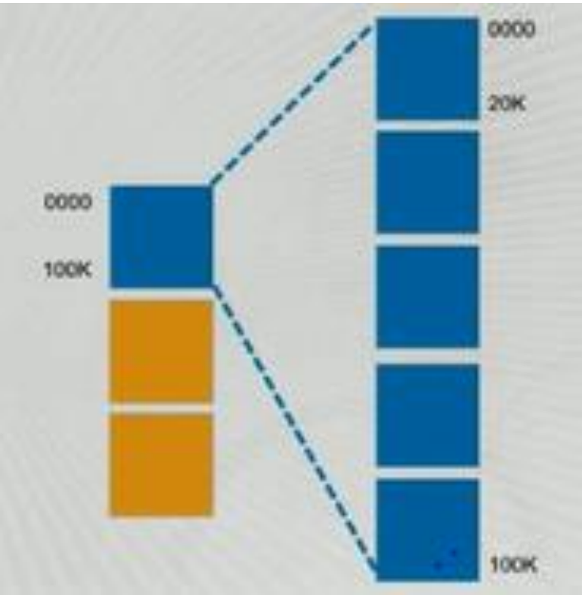
- **Blocks are where the column data is stored in**
- **Typically, in 1 MB immutable blocks**
- **Blocks are individually encoded with 1 of 11 encodings**
- **A full block can contain millions of values**



Data Blocks (disk)

Zone Maps

- In-memory block metadata
- Store min and max values for each block
  - All blocks have zone maps
- Allows to prune data (blocks) out at query time
  - For queries with predicates on “where” clause, checks min-max values in memory before reading the disk
  - Exclude the non-relevant blocks
  - Minimize unnecessary I/O
- An Indexing strategy



Zone Map (memory)

Data Block (disk)

Data Sorting

- **Redshift stores data in tables in sorted order based on sort keys**
  - Sorting makes zone maps more effective to skip over the data blocks to minimize I/Os
  - Sorting makes pruning much more effective

Sort Keys

- **Picking a sort key is typically based on query patterns**
  - Also, on business requirements, and/or cardinality of data
  - Sort key can span more than one column
- **Single Column Sort Key**
  - Consists of a single column in the table
- **Compound Sort Keys**
  - Made up of nested collection of columns
  - Columns specified in their order of priority
    - Primary sort column is given more weight
  - Useful with JOINS, ORDER BY, GROUP BY operations
- **Interleaved Sort Keys**
  - Made up of two or more columns
  - Gives equal weight to each column in the sort key
    - Order in which columns are listed does not matter
  - Useful when don't know which column will be used in the query

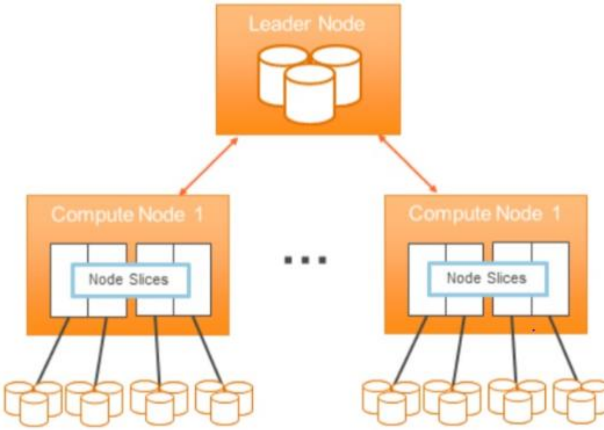
```
CREATE TABLE
CUST_SALES_DT_SINGLE
SORTKEY (C_CUSTKEY)
AS SELECT * FROM
CUST_SALES_DATE;
```

```
CREATE TABLE
CUST_SALES_DT_COMPOUND
COMPOUND SORTKEY
(C_CUSTKEY, C_REGION,
C_MKTSEGMENT, D_DATE)
AS SELECT * FROM
CUST_SALES_DATE;
```

```
CREATE TABLE
CUST_SALES_DT_INTERLEAVED
INTERLEAVED SORTKEY
(C_CUSTKEY, C_REGION,
C_MKTSEGMENT, D_DATE)
AS SELECT * FROM
CUST_SALES_DATE;
```

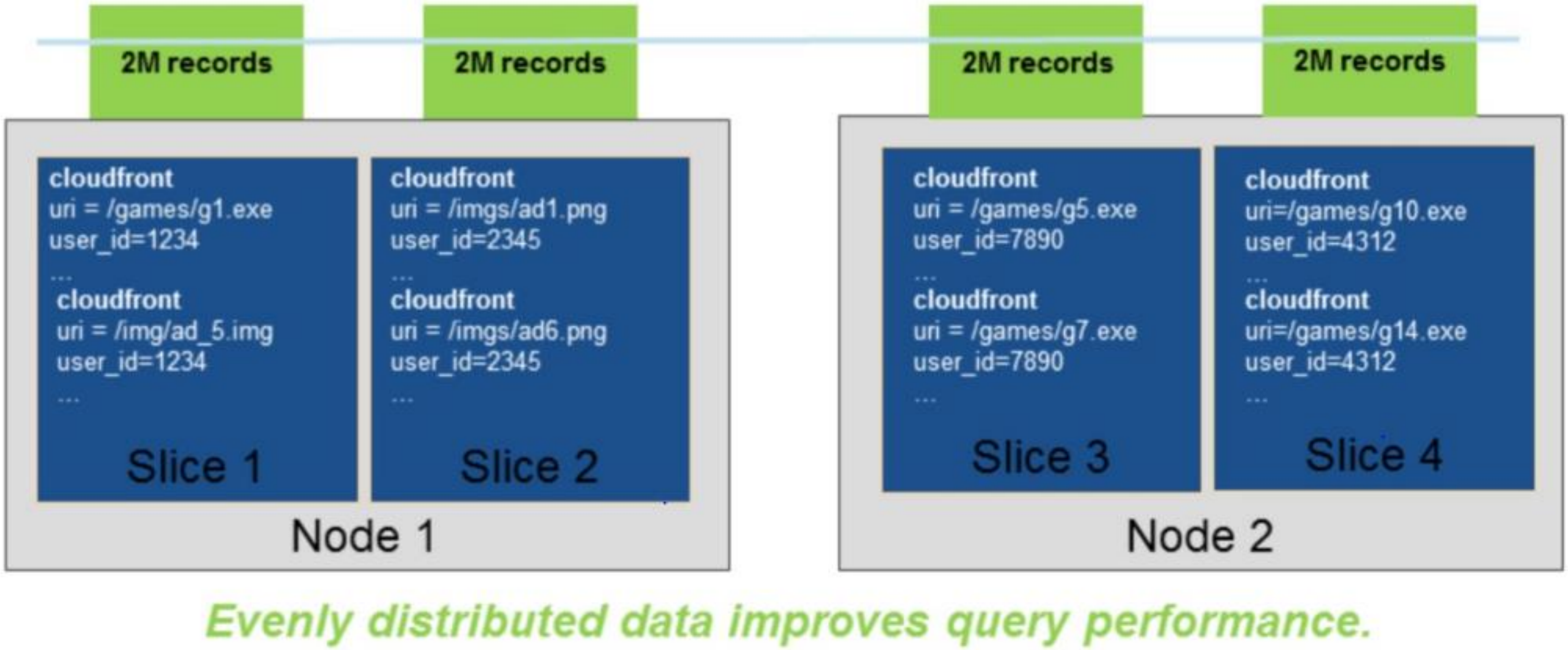
Node Slices

- **A slice can be thought of as a "virtual compute node"**
  - This is how parallelism is achieved in each of the Redshift cluster nodes
- **Compute Node is partitioned into Slices**
  - Each slice is allocated a portion of the node's CPU, memory, and disk space
  - When data is written to Redshift, the data is actually stored in slices
  - A slice processes its own data
- **Support parallel query processing in compute nodes**
  - Each slice processes a piece of workload in parallel
  - Number of slices per node depends on the node size
  - A compute node has either 2, 16, 32 slices



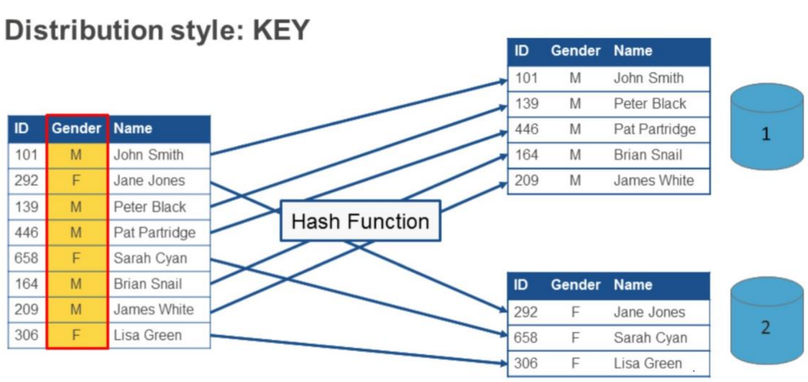
Distribution Styles

- **Used to distribute data (table rows) across compute nodes and slices**
  - The goal is to distribute data evenly across the nodes
  - Minimize data movement: co-located joins and aggregates



Three Distribution Styles: Key, Even, and All

- **KEY (hash function)**
  - Here a column is selected to be the distribution key
  - Value of the column in the row is hashed, and the hash corresponds to one of the slices in the cluster
    - The same value goes to the same location (slice)
  - Use this when the Key/column has high cardinality values and creates an even distribution of the data across all nodes
    - Useful in query performance in table joins and group by
    - Place matching rows on the same node slice

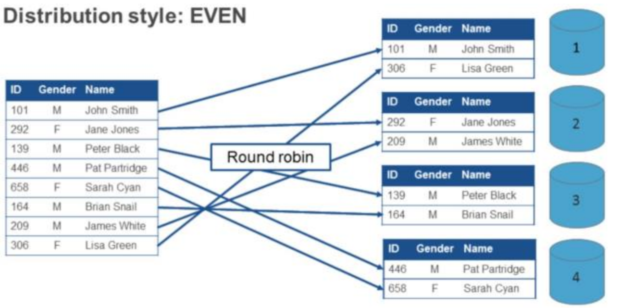



```
CREATE TABLE loft_deep_dive (
    aid   INT        --audience_id
   ,loc  CHAR(3)    --location
   ,dt   DATE       --date
) DISTSTYLE KEY DISTKEY (aid);
```

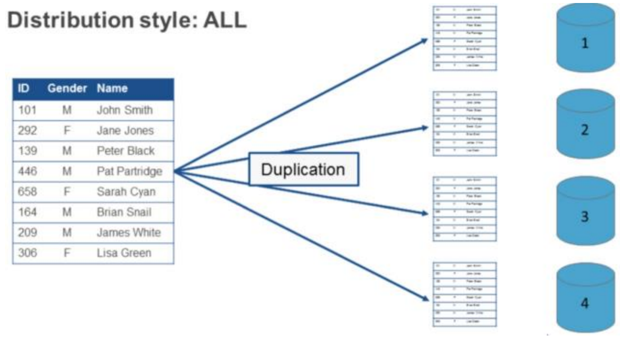
Three Distribution Styles

- **EVEN (round robin – default choice)**
  - Rows distributed across the slices on compute nodes in a round-robin fashion

```
CREATE TABLE loft_deep_dive (
    aid  INT        --audience_id
    ,loc CHAR(3)    --location
    ,dt  DATE       --date
) DISTSTYLE EVEN;
```



- **ALL (duplication)**
  - A copy of the entire table is distributed to every compute node
  - <u>Useful for reasonably sized</u> data (2-3 millions rows)



Design Consideration: Column Properties

- **DISTKEY, SORTKEY, Compression Style (in that order) can influence performance significantly**
  - Can have an order of magnitude difference in performance

- **Distribution Keys**
  - A poor DISTKEY can introduce data skew and an unbalanced workload
  - A query completes only as fast as the slowest slice completes
- **Sort Keys**
  - Picking SORT Key(s) depends on business requirements and query patterns
    - Based on range or equality predicate (WHERE clause)
    - If you access recent data frequently, sort based on timestamp
- **Compression**
  - COPY automatically analyzes and compresses data when loading into empty table
  - ANALYZE COMPRESSION checks existing table and proposes optimal compression algorithm for each column
  - Changing column encoding requires a table rebuild

Analyzing Table Design

- **Properly specifying dist keys, sort keys, and compression encodings can significantly improve storage, I/O, and query performance**
- **AWS provides a SQL script to identify tables where sort keys, dist keys, and column encodings are missing or performing poorly**
- **table_inspector.sql**

Analyzing Table Design

- **table_inspector.sql**

```
SELECT SCHEMA schemaname,
       "table" tablename,
       table_id tableid,
       size size_in_mb,
       CASE
         WHEN diststyle NOT IN ('EVEN','ALL') THEN 1
         ELSE 0
       END has_dist_key,
       CASE
         WHEN sortkey1 IS NOT NULL THEN 1
         ELSE 0
       END has_sort_key,
       CASE
         WHEN encoded = 'Y' THEN 1
         ELSE 0
       END has_col_encoding,
       CAST(max_blocks_per_slice - min_blocks_per_slice AS FLOAT) /
GREATEST(NVL (min_blocks_per_slice,0)::int,1) ratio_skew_across_slices,
       CAST(100*dist_slice AS FLOAT) /(SELECT COUNT(DISTINCT slice) FROM
stv_slices) pct_slices_populated
FROM svv_table_info ti
  JOIN (SELECT tbl,
               MIN(c) min_blocks_per_slice,
               MAX(c) max_blocks_per_slice,
               COUNT(DISTINCT slice) dist_slice
        FROM (SELECT b.tbl,
                     b.slice,
                     COUNT(*) AS c
              FROM STV_BLOCKLIST b
              GROUP BY b.tbl,
                       b.slice)
        WHERE tbl IN (SELECT table_id FROM svv_table_info)
        GROUP BY tbl) iq ON iq.tbl = ti.table_id;
```

Analyzing Table Design

- **table_inspector.sql**

```sql
SELECT SCHEMA schemaname,
       "table" tablename,
       table_id tableid,
       size size_in_mb,
       CASE
         WHEN diststyle NOT IN ('EVEN','ALL') THEN 1
         ELSE 0
       END has_dist_key,
       CASE
         WHEN sortkey1 IS NOT NULL THEN 1
         ELSE 0
       END has_sort_key,
       CASE
         WHEN encoded = 'Y' THEN 1
         ELSE 0
       END has_col_encoding,
       CAST(max_blocks_per_slice - min_blocks_per_slice AS FLOAT) /
GREATEST(NVL (min_blocks_per_slice,0)::int,1) ratio_skew_across_slices,
       CAST(100*dist_slice AS FLOAT) /(SELECT COUNT(DISTINCT slice) FROM
stv_slices) pct_slices_populated
FROM svv_table_info ti
  JOIN (SELECT tbl,
               MIN(c) min_blocks_per_slice,
               MAX(c) max_blocks_per_slice,
               COUNT(DISTINCT slice) dist_slice
        FROM (SELECT b.tbl,
                     b.slice,
                     COUNT(*) AS c
              FROM STV_BLOCKLIST b
              GROUP BY b.tbl,
                       b.slice)
        WHERE tbl IN (SELECT table_id FROM svv_table_info)
        GROUP BY tbl) iq ON iq.tbl = ti.table_id;
```

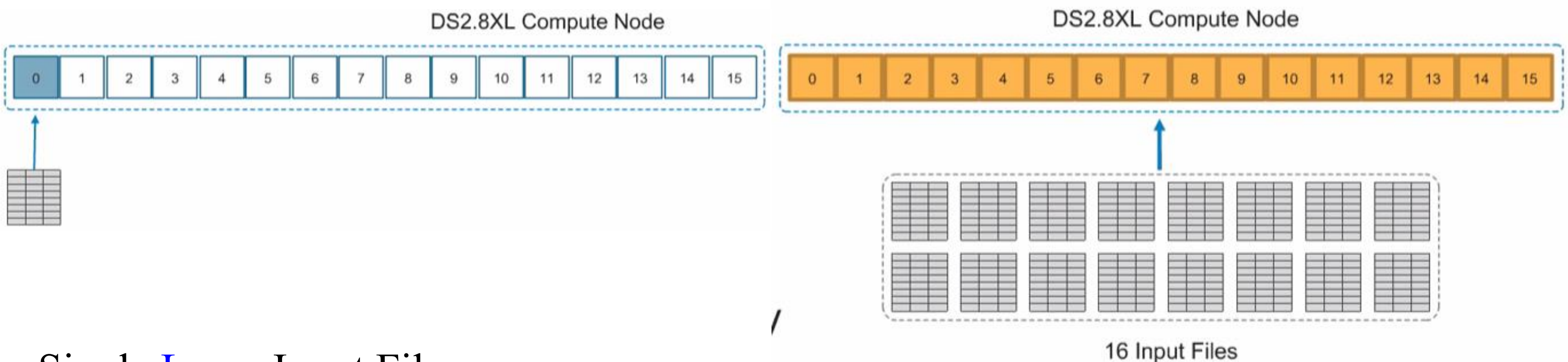
schemaname	tablename	tableid	size_in_mb	has_dist_key	has_sort_key	has_col_encoding	ratio_skew_across_slices	pct_slices_populated
public	category	100553	28	1	1	0	0	100
public	date	100555	44	1	1	0	0	100
public	event	100558	36	1	1	1	0	100
public	listing	100560	44	1	1	1	0	100
public	nation	100563	175	0	0	0	0	39.06
public	region	100566	30	0	0	0	0	7.81
public	sales	100562	52	1	1	0	0	100
public	skew1	100547	18978	0	0	0	.15	50
public	skew2	100548	353	1	0	0	0	1.56
public	venue	100551	32	1	1	0	0	100
public	users	100549	82	1	1	1	0	100
public	venue	100551	32	1	1	0	0	100

Loading data into Redshift

- **COPY command** can **load multiple input files simultaneously** from S3, DynamoDB, or EMR

Loading data into Redshift

- **Node Slices can be loaded simultaneously**
  - To maximize throughput, have at least as many input files as number of slices
  - Single input file means only one slice is ingesting the data
    - Avoid single insert, update, delete operations (or use for small inserts/updates only)



Single Large Input File: Effective throughput=6.25 MB/s

16 small input files (splitting the Large Input File): Effective throughput=100 MB/s

Loading Data into Redshift

- **An example of COPY command:**

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

- **Can use a Manifest file to control exactly what is loaded**
  - An example of Manifest file called cust.manifest

```
{
  "entries": [
    {"url":"s3://mybucket-alpha/2013-10-04-custdata", "mandatory":true},
    {"url":"s3://mybucket-alpha/2013-10-05-custdata", "mandatory":true},
    {"url":"s3://mybucket-beta/2013-10-04-custdata", "mandatory":true},
    {"url":"s3://mybucket-beta/2013-10-05-custdata", "mandatory":true}
  ]
}
```

  - An example of COPY command using the manifest file

```
copy customer
from 's3://mybucket/cust.manifest'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole'
manifest;
```

Data Protection in Redshift

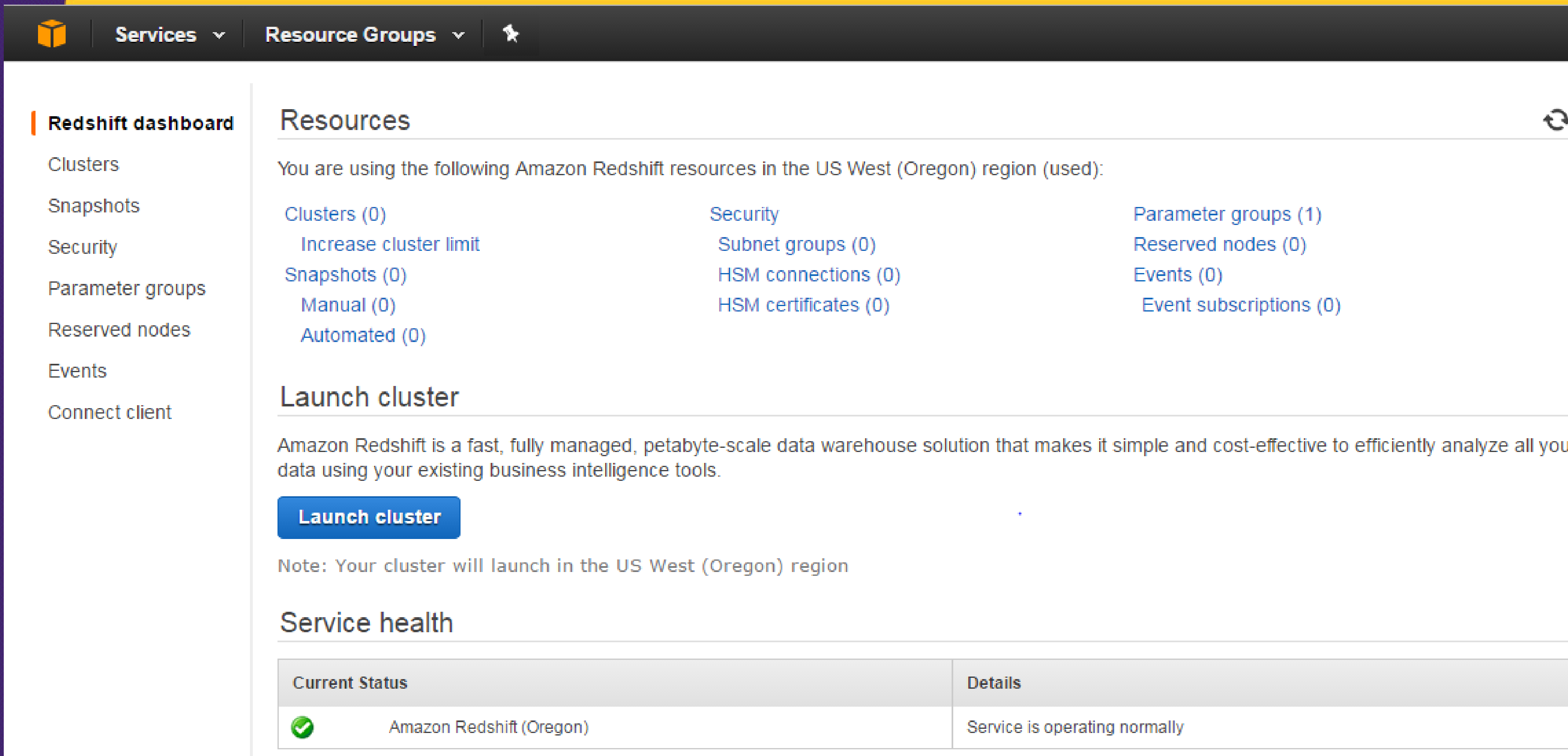
- **Replicates all the data in data warehouse when data is loaded**
  - Replicates in compute nodes
- **Continuously backs up all the data in Amazon S3**
  - Backups are incremental and restores are quick
    - with frequently accessed data streamed first
- **Can asynchronously replicate your snapshots to Amazon S3 in another region**
  - Useful for disaster recovery

Redshift Security

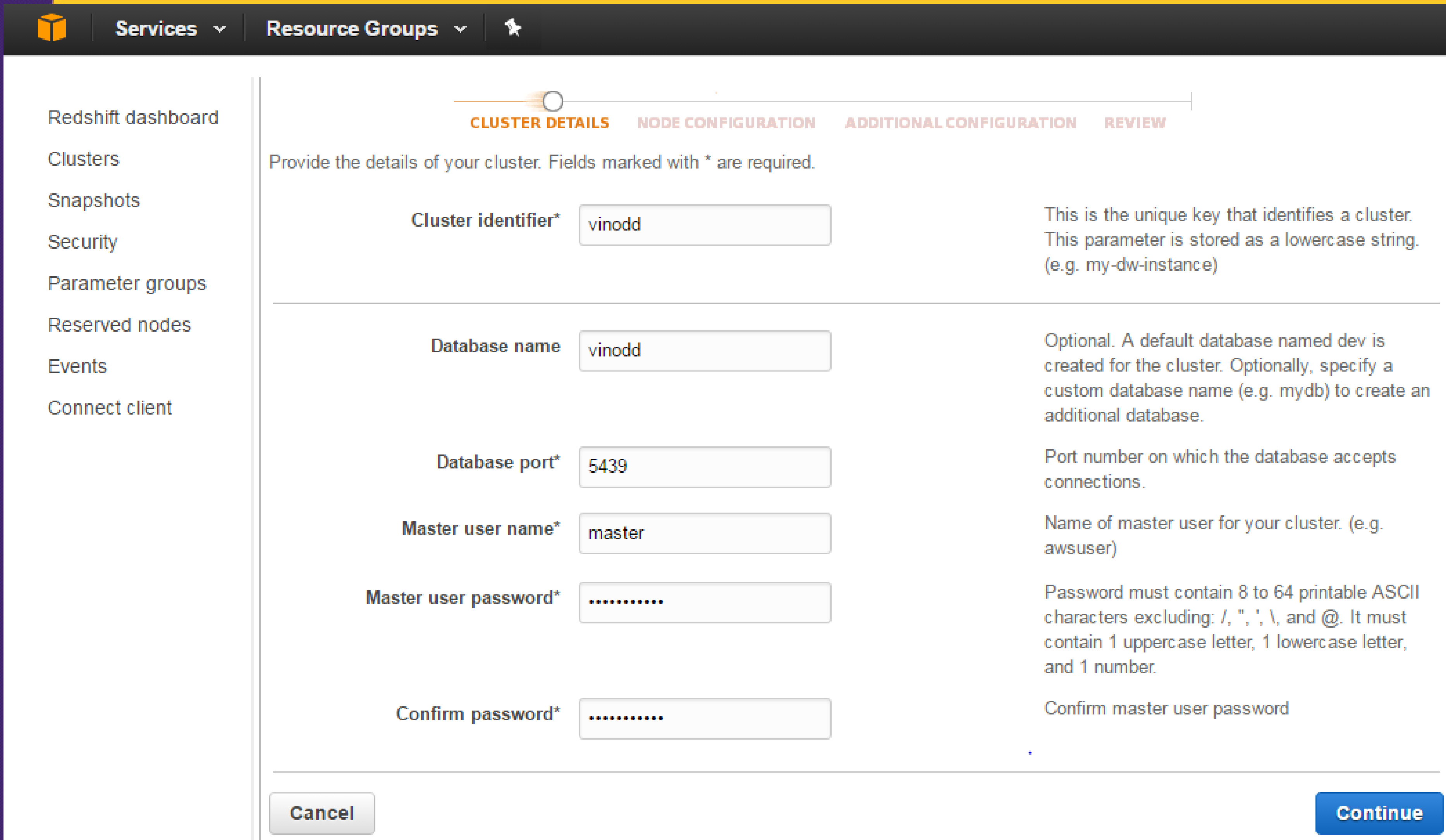
- **Encryption at rest and in transit are built-in**
  - Includes data and backups
  - By default, takes care of key management
- **Option to manage your keys using your own hardware security modules**
  - AWS CloudHSM or
  - AWS KMS
- **SSL and Amazon VPC usage support built in**

Launching Redshift Cluster

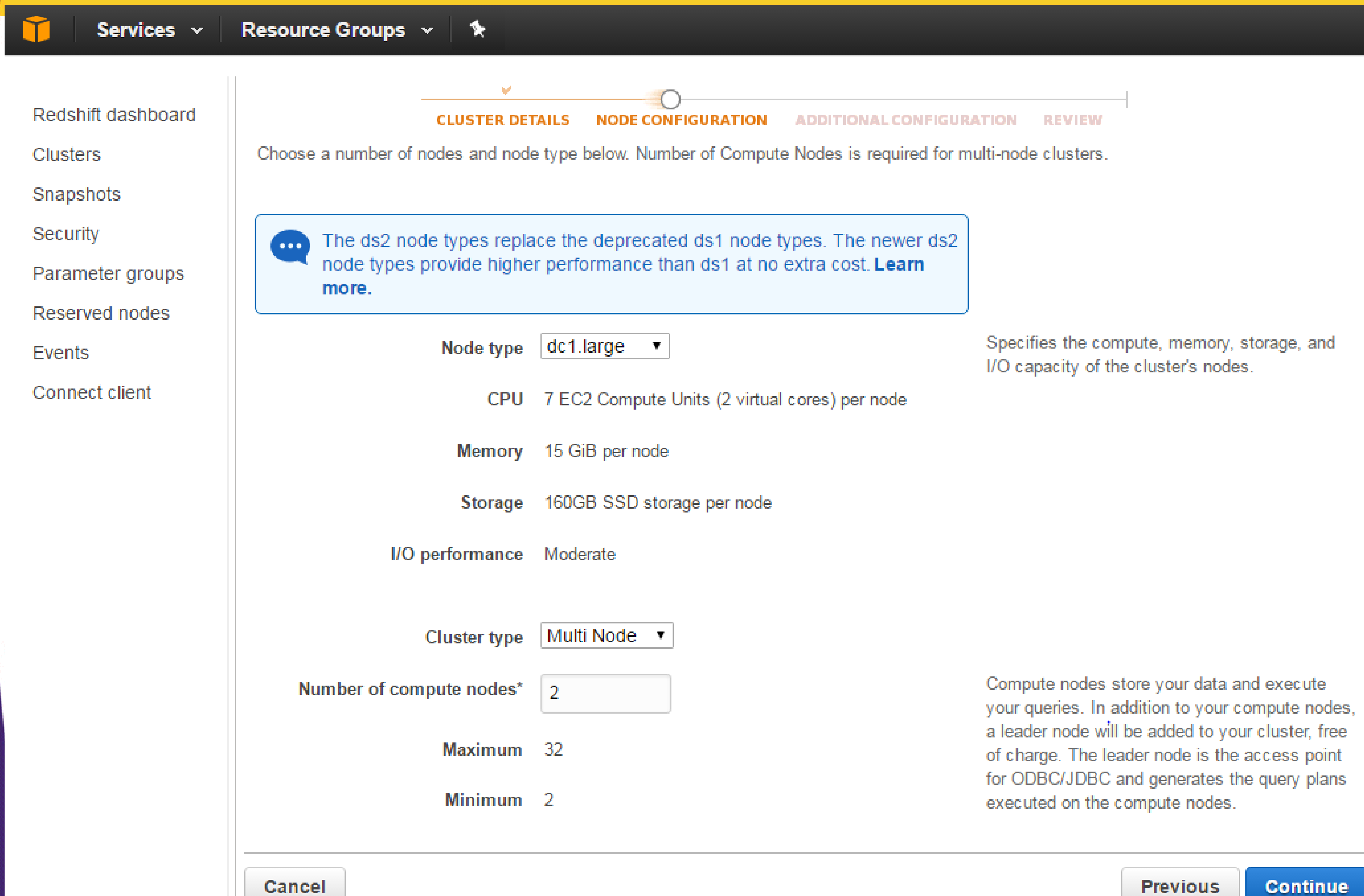
Launching Redshift Cluster



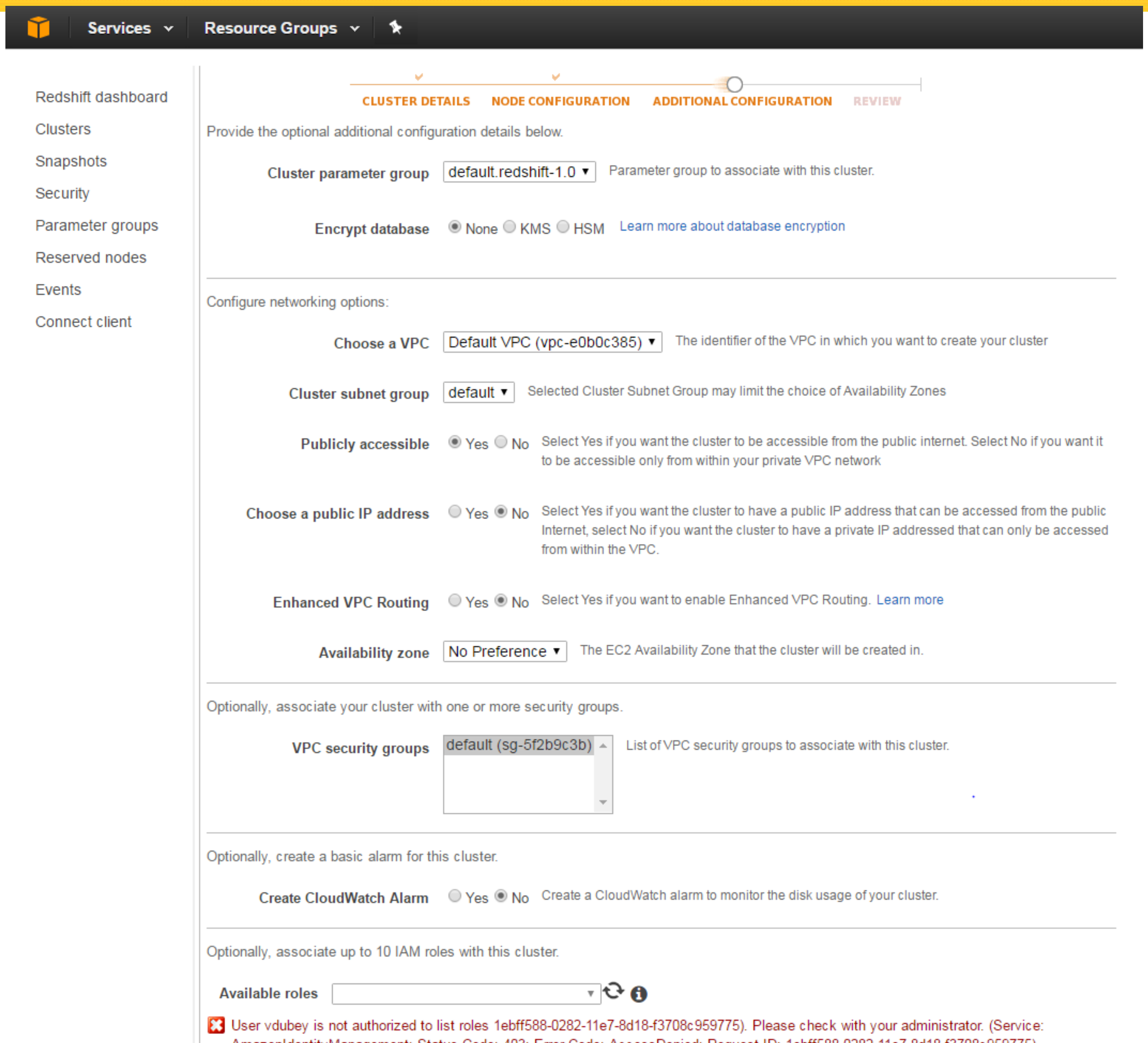
Launching Redshift Cluster



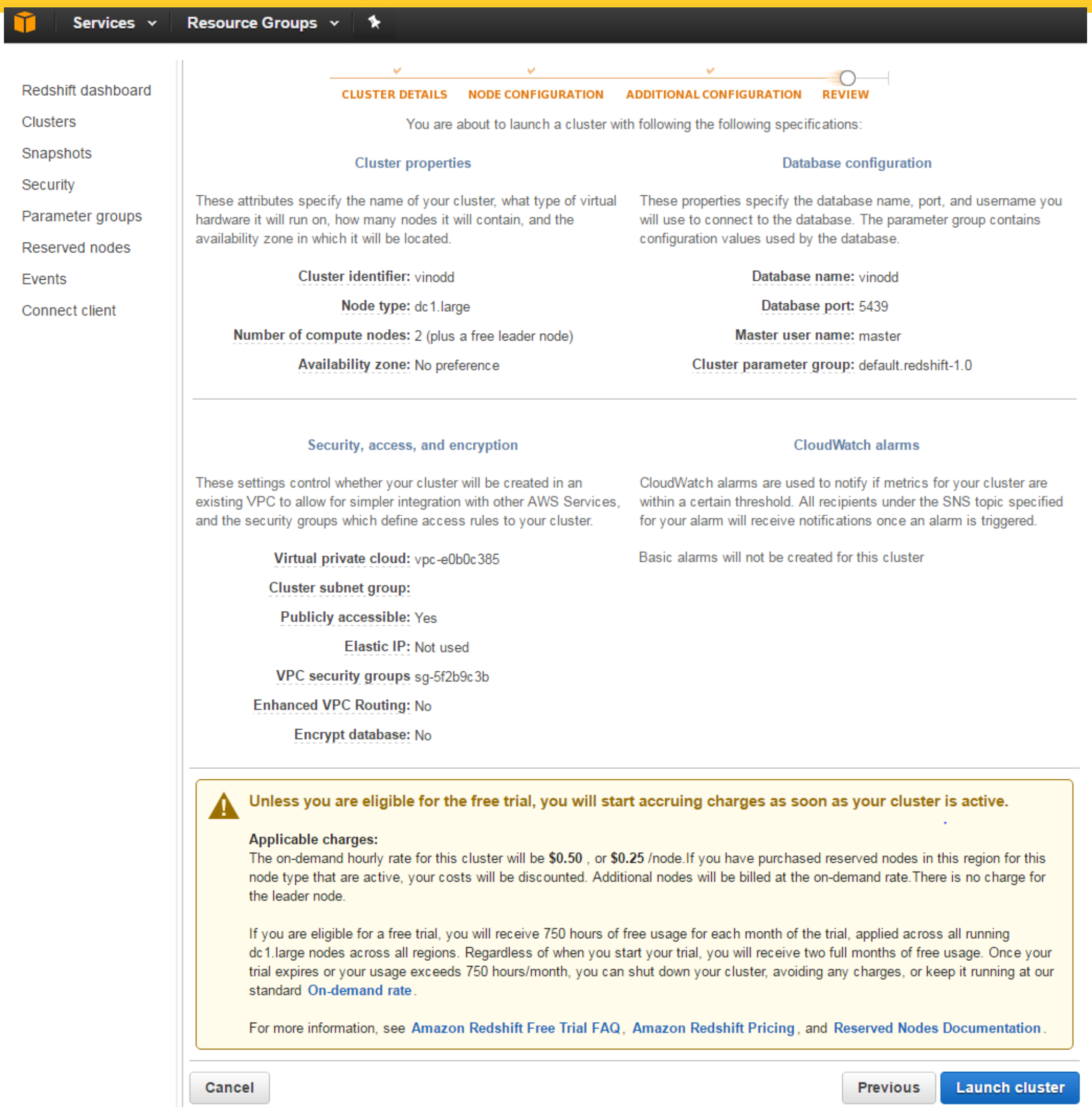
Launching Redshift Cluster



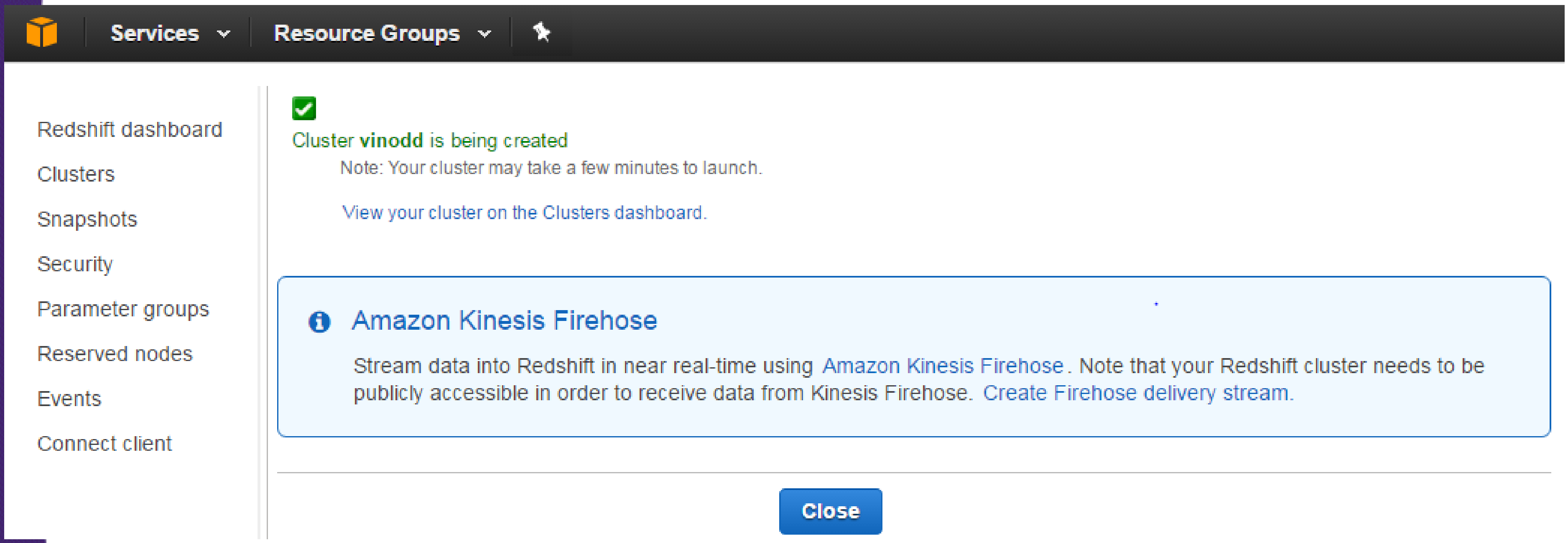
Launching Redshift Cluster



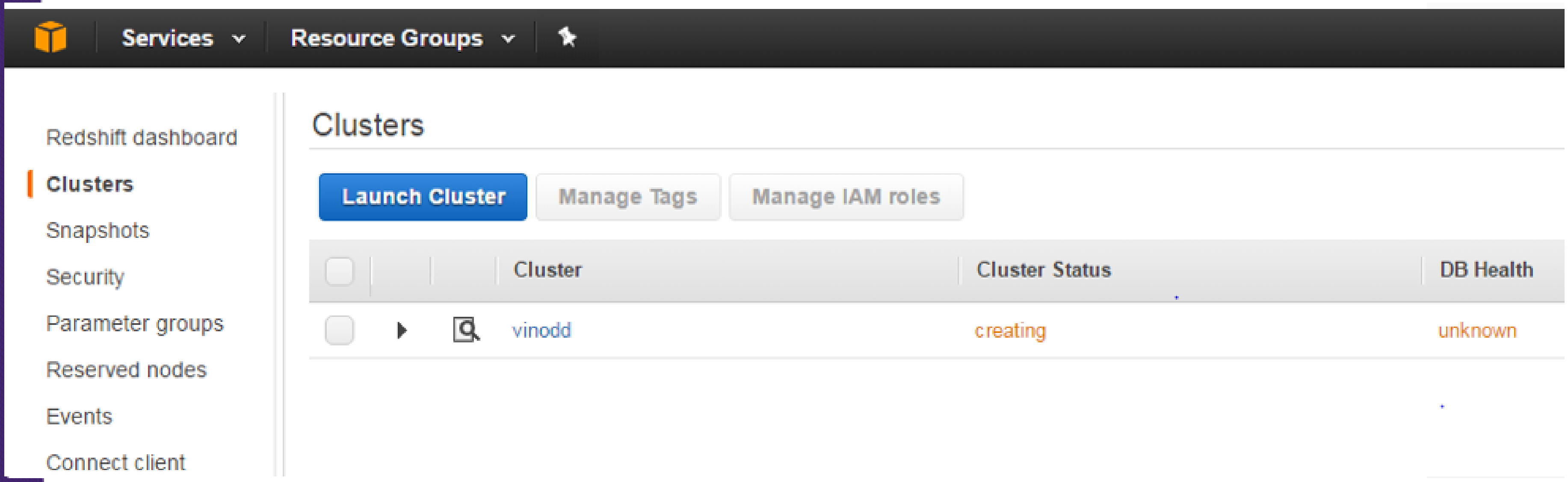
Launching Redshift Cluster

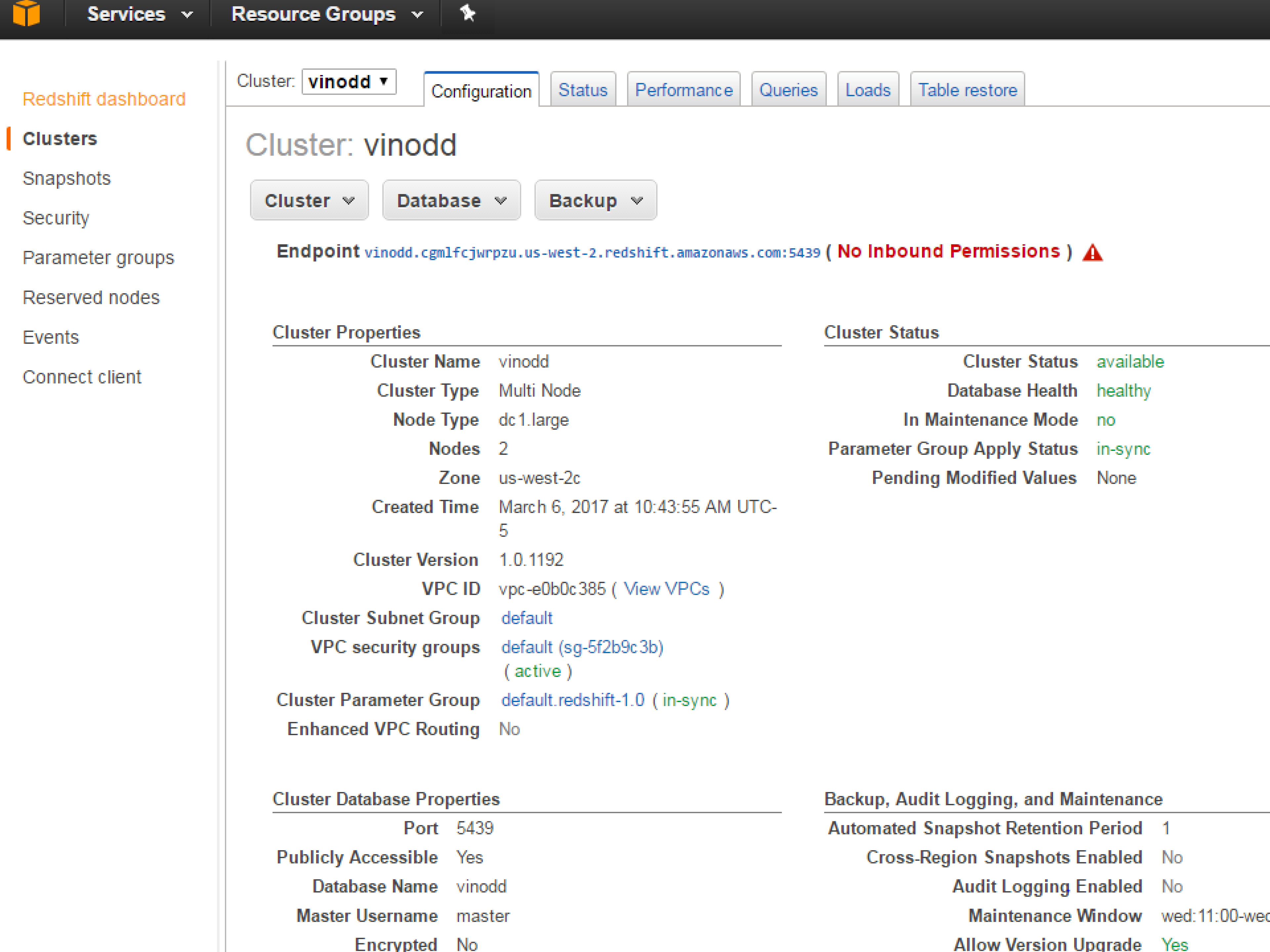


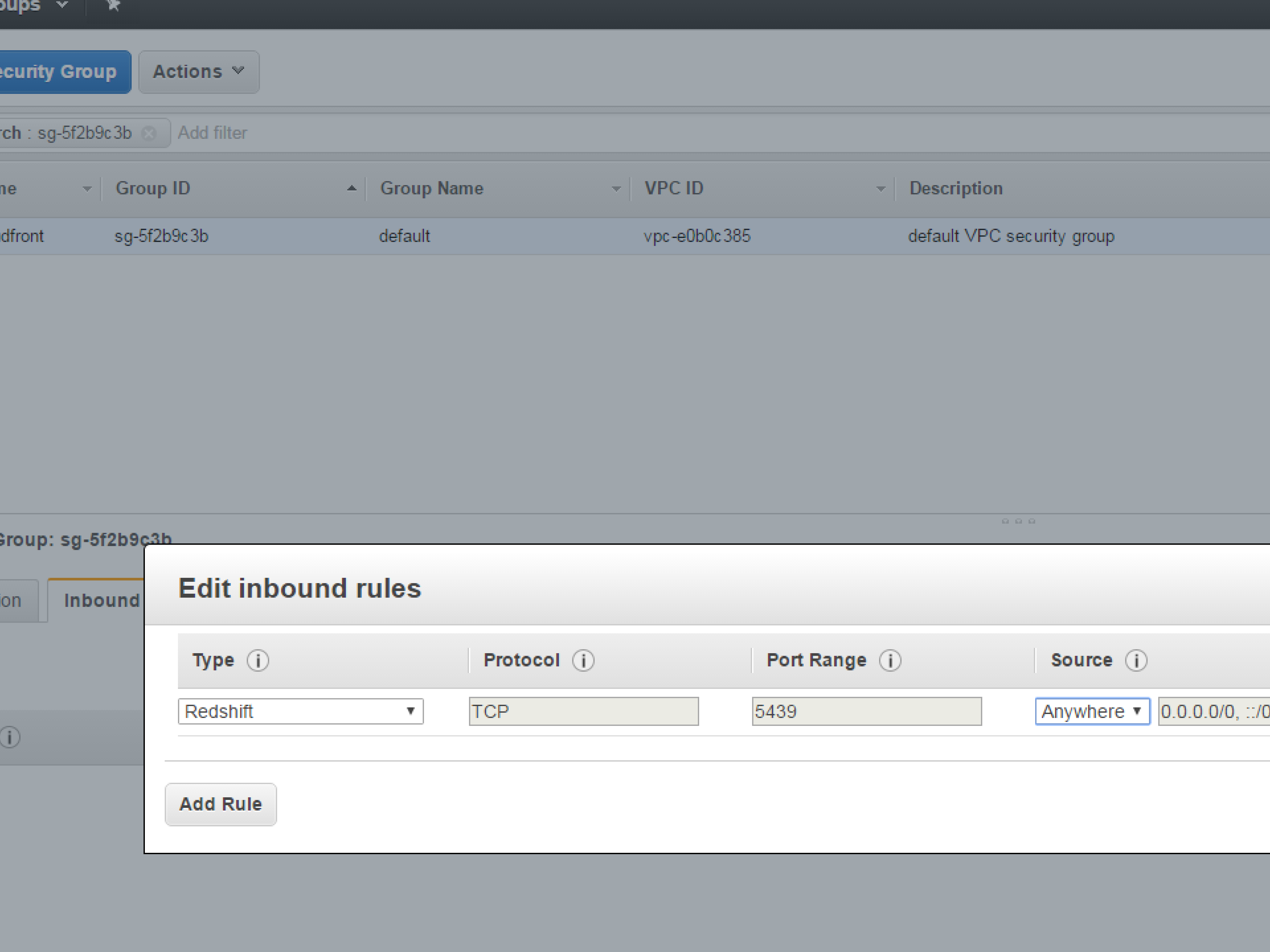
Launching Redshift Cluster



Launching a Redshift Cluster




Services
Resource Groups
Redshift dashboard
Clusters
Snapshots
Security
Parameter groups
Reserved nodes
Events
Connect client
Cluster: vinodd
Configuration
Status
Performance
Queries
Loads
Table restore
Cluster: vinodd
Cluster
Database
Backup
Endpoint vinodd.cgmlfcjwrpzu.us-west-2.redshift.amazonaws.com:5439 (No Inbound Permissions)
Cluster Properties
Cluster Name vinodd
Cluster Type Multi Node
Node Type dc1.large
Nodes 2
Zone us-west-2c
Created Time March 6, 2017 at 10:43:55 AM UTC-5
Cluster Version 1.0.1192
VPC ID vpc-e0b0c385 (View VPCs)
Cluster Subnet Group default
VPC security groups default (sg-5f2b9c3b) (active)
Cluster Parameter Group default.redshift-1.0 (in-sync)
Enhanced VPC Routing No
Cluster Status
Cluster Status available
Database Health healthy
In Maintenance Mode no
Parameter Group Apply Status in-sync
Pending Modified Values None
Cluster Database Properties
Port 5439
Publicly Accessible Yes
Database Name vinodd
Master Username master
Encrypted No
Backup, Audit Logging, and Maintenance
Automated Snapshot Retention Period 1
Cross-Region Snapshots Enabled No
Audit Logging Enabled No
Maintenance Window wed:11:00-wed
Allow Version Upgrade Yes


oups
ecurity Group
Actions
rch : sg-5f2b9c3b
Add filter
me
Group ID
Group Name
VPC ID
Description
dfront
sg-5f2b9c3b
default
vpc-e0b0c385
default VPC security group
Group: sg-5f2b9c3b
ion
Inbound
Edit inbound rules
Type
Protocol
Port Range
Source
Redshift
TCP
5439
Anywhere
0.0.0.0/0, ::/0
Add Rule

rce Groups

sters

aunch Cluster | Manage Tags | Manage IAM roles

Cluster	Cluster Status	DB Health
vinodd	available	healthy

Endpoint vinodd.cgmlfcjwrpzu.us-west-2.redshift.amazonaws.com:5439 (authorized)

Cluster Properties

Cluster Name	vinodd
Node Type	dc1.large
Nodes	2
Zone	us-west-2c
Cluster Parameter Group	default.redshift-1.0 (in-sync)
Enhanced VPC Routing	No

Cluster Status

Cluster
Database
In Maintenance
Parameter Group Apply
Pending Modified V

Cluster Database Properties

Port	5439
Database Name	vinodd
Master Username	master
Encrypted	No

Backup, Audit Logging, a

Automated Snapshot Re
Cross-Region Snaps
Audit Log
Mainter
Allow Ve

Tags

You have not created any tags. Please add tags using the **Manage Tags** button above.

Launching a Redshift Cluster

Capacity Details

Current Node Type	dc1.large
CPU	7 EC2 Compute Units (2 virtual cores) per node
Memory	15 GiB per node
Storage	160GB SSD storage per node
I/O Performance	Moderate
Platform	64-bit

SSH ingestion settings

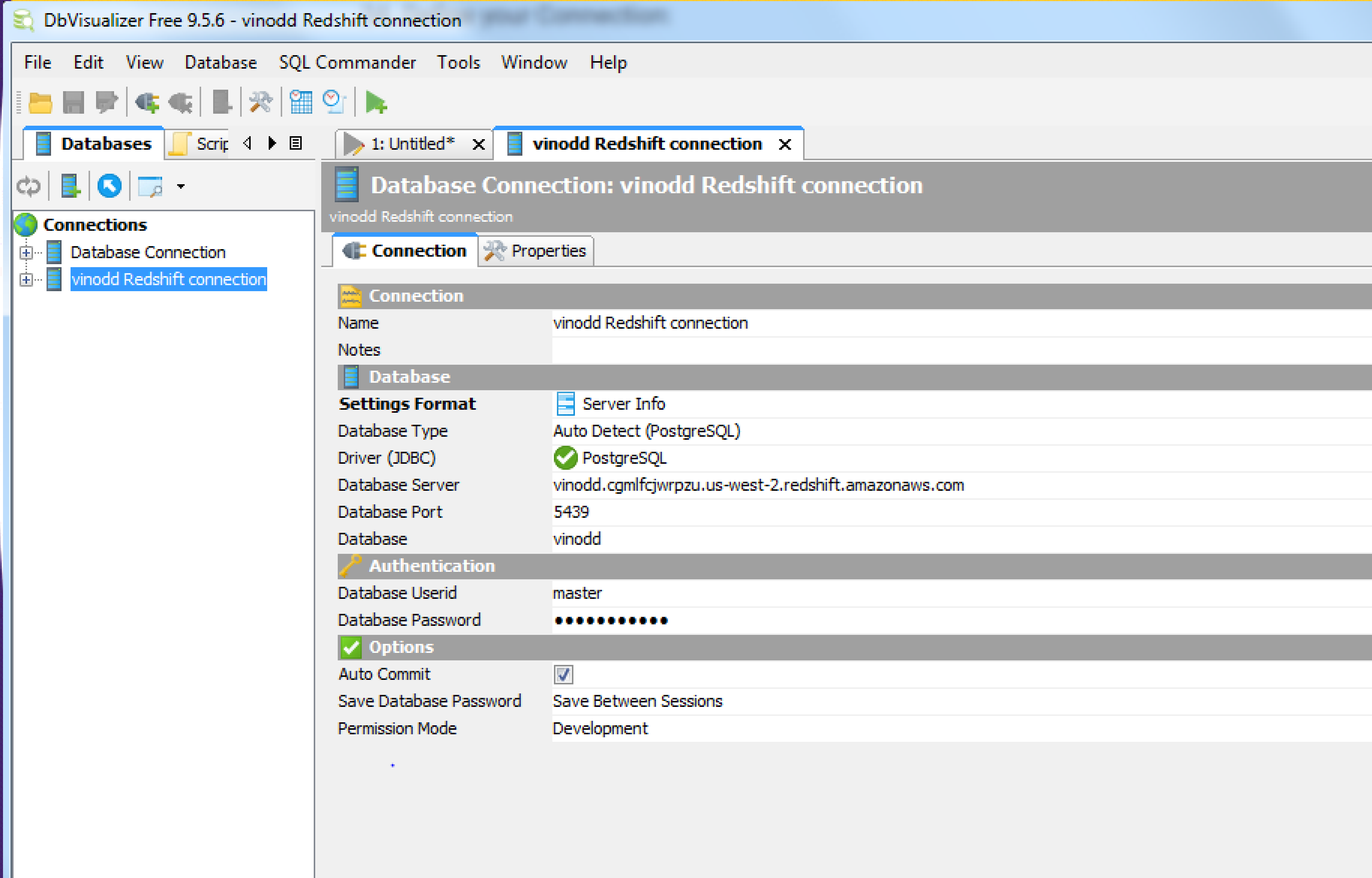
Cluster public key:

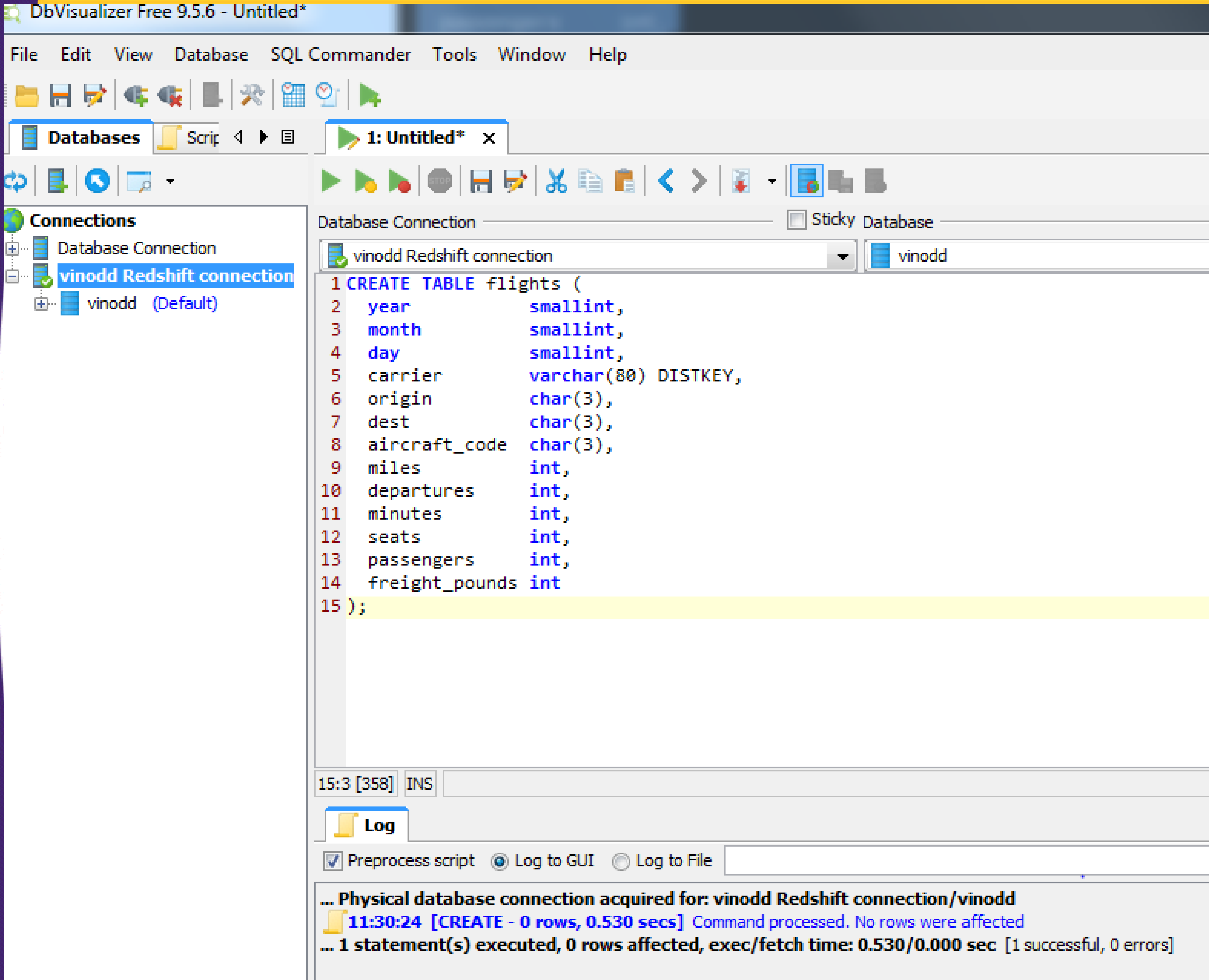

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCdHxCsZGCwHySvJ/h32d
jXKqU41XYtlZ0ChTPvWYxe4TueOVmuydqiZcd9pbXr4Ag6p2Gw
8yrFjc+ig1gYK6CV6FWZxxxz/otNYZ286dTI7sOntK7VS8VHh6
Kozt6w1Qa4YQ0Z5F30D4e8bh6mluGOyLegFIPGl+P0l+IFAExf
2AIUi5EfdhACEIXPpOyo3+azBZnpgYZliDuOyx10q3LEdDat2G
SEw+rWdj4yee2arLBqK/5+pZZb1UdO3bvaYiD9ucnQP4Z5aumg
HAJ2+zkp5lX79QjVKew7InLkbRxeO8eM1jfI+fn2kjibr0klHg
3F+Y4olBcjrjgSFQY3I/xz Amazon-Redshift
```


Node IP addresses:

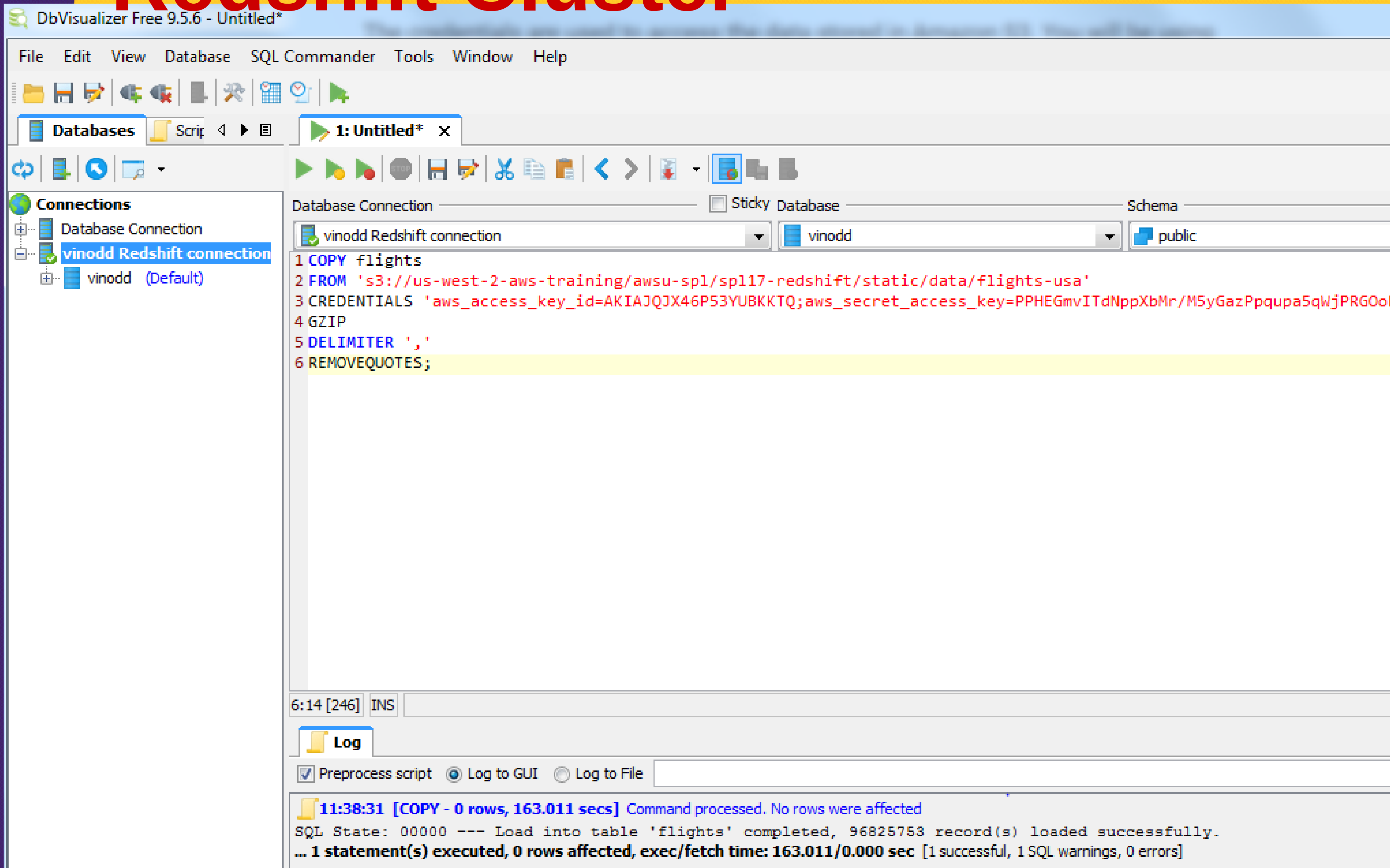
Node	Public IP	Private IP
Compute-0	35.164.78.240	172.31.7.102
Leader	52.24.125.80	172.31.1.136
Compute-1	35.163.154.58	172.31.4.110

Connecting to Redshift Cluster Using DbVisualiser

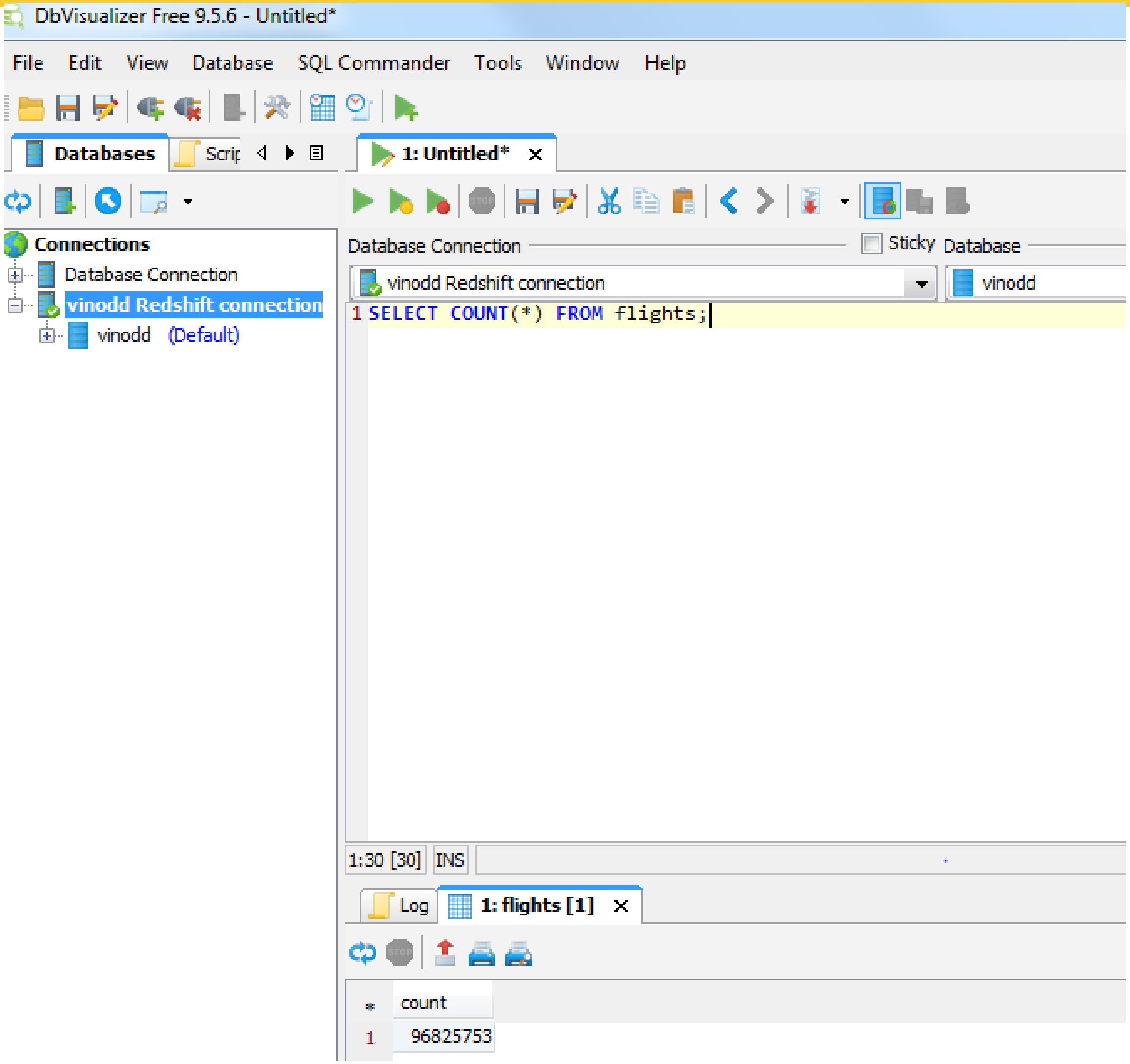


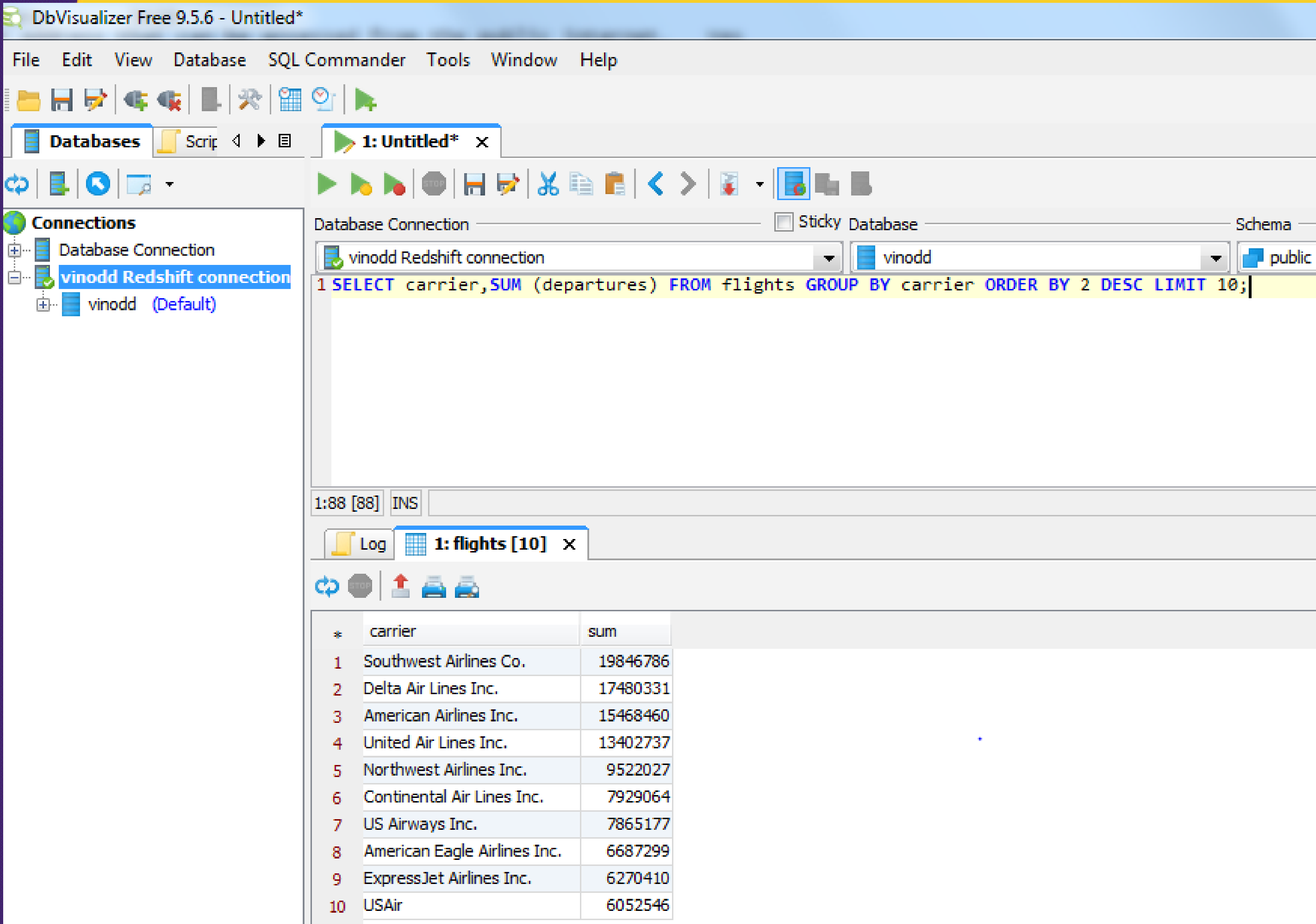

DbVisualizer Free 9.5.6 - Untitled*
File Edit View Database SQL Commander Tools Window Help
Databases
Scrip
1: Untitled*
Connections
Database Connection
vinodd Redshift connection
vinodd (Default)
Database Connection
Sticky
Database
vinodd Redshift connection
vinodd
1 CREATE TABLE flights (
2 year smallint,
3 month smallint,
4 day smallint,
5 carrier varchar(80) DISTKEY,
6 origin char(3),
7 dest char(3),
8 aircraft_code char(3),
9 miles int,
10 departures int,
11 minutes int,
12 seats int,
13 passengers int,
14 freight_pounds int
15);
15:3 [358]
INS
Log
Preprocess script
Log to GUI
Log to File
... Physical database connection acquired for: vinodd Redshift connection/vinodd
11:30:24 [CREATE - 0 rows, 0.530 secs] Command processed. No rows were affected
... 1 statement(s) executed, 0 rows affected, exec/fetch time: 0.530/0.000 sec [1 successful, 0 errors]

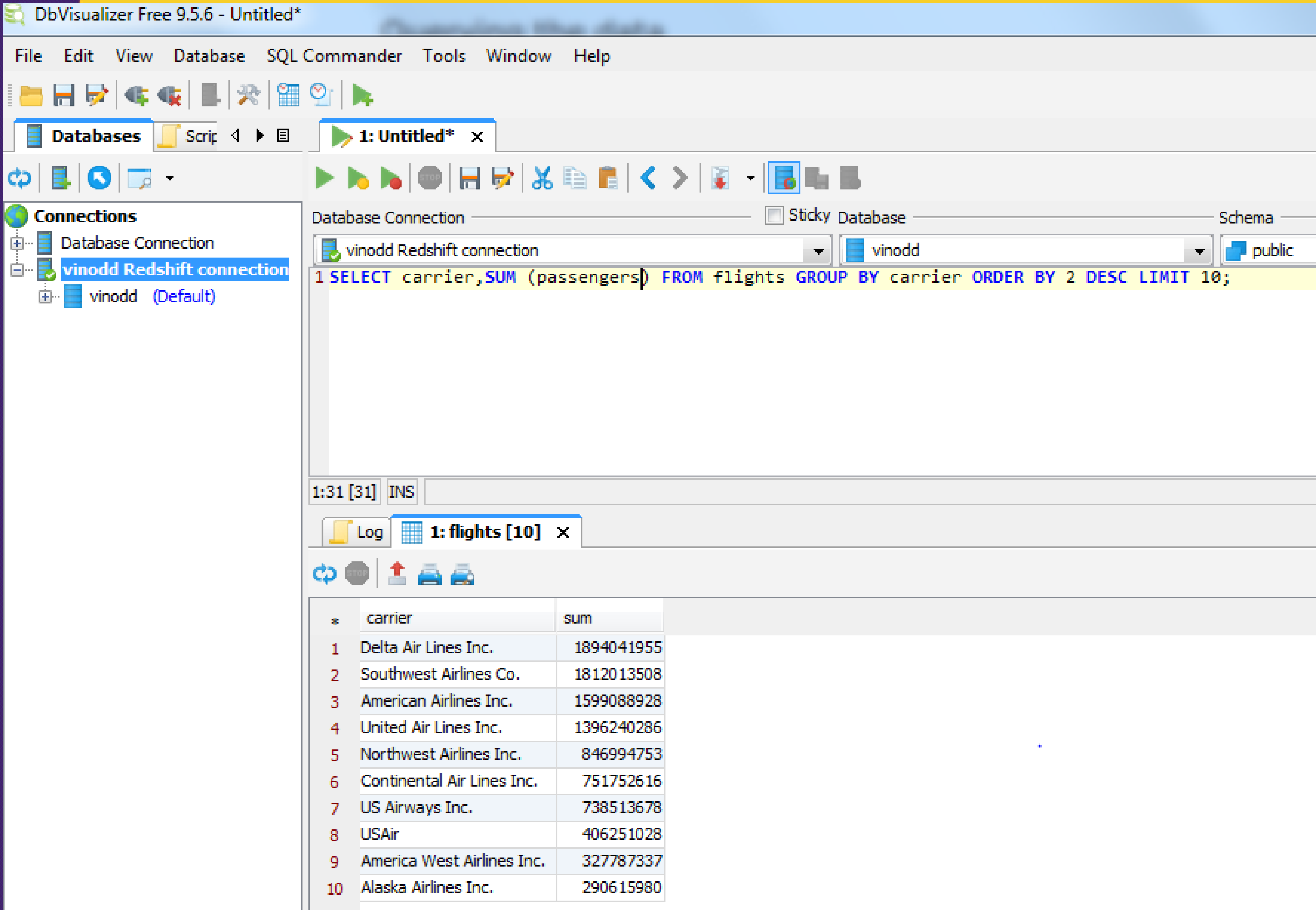
Copying data from S3 to Redshift Cluster

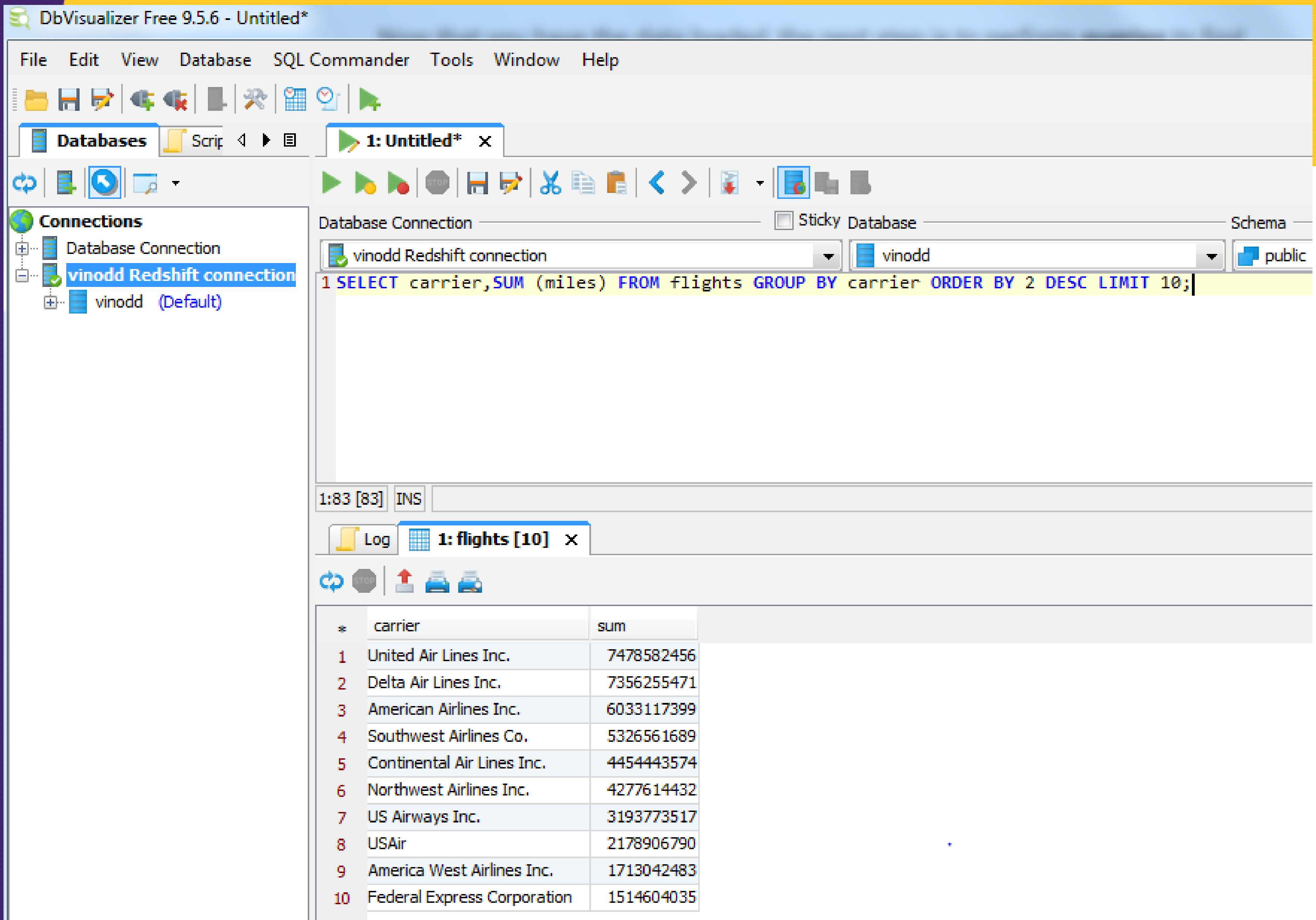


Querying Redshift Cluster

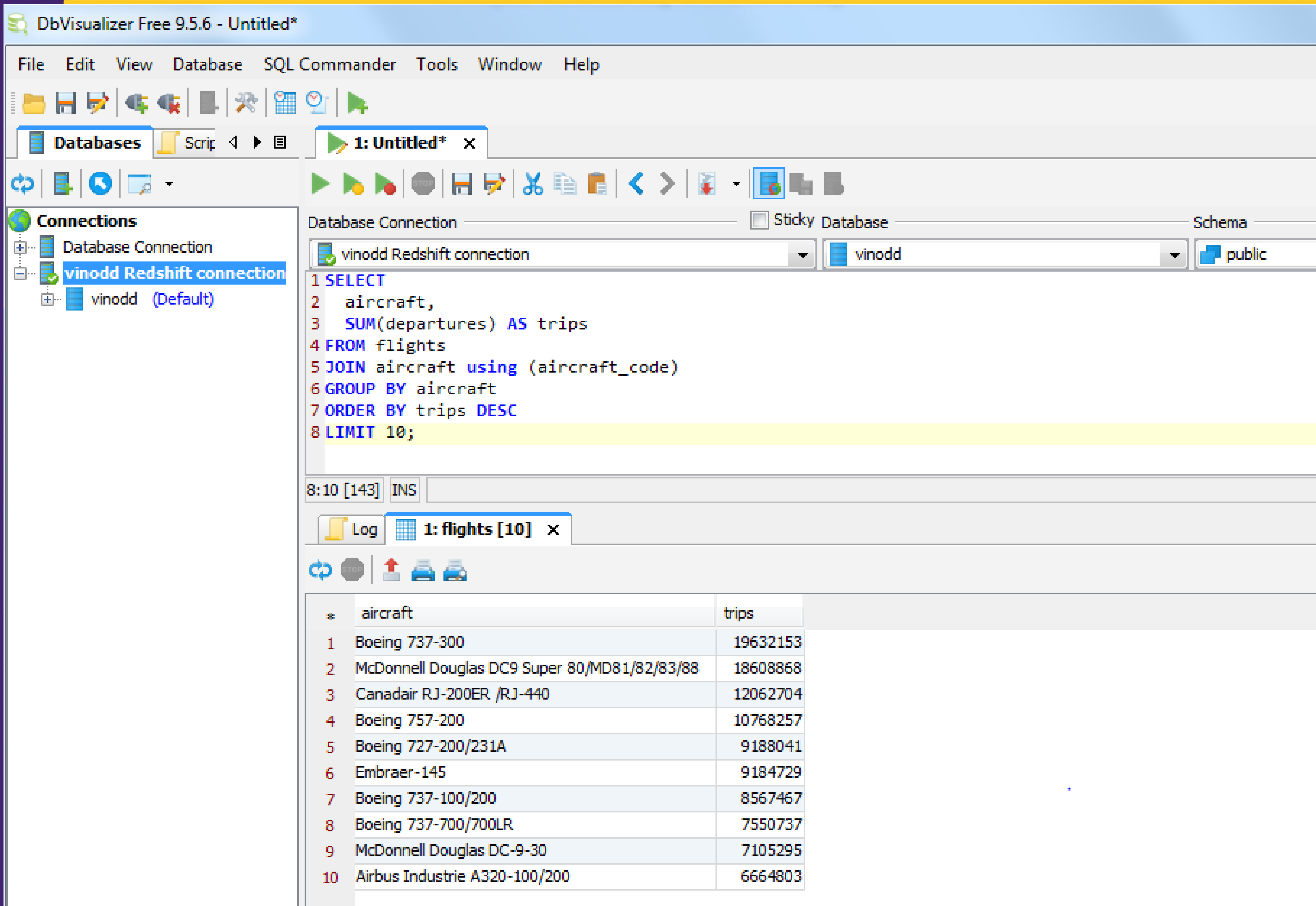


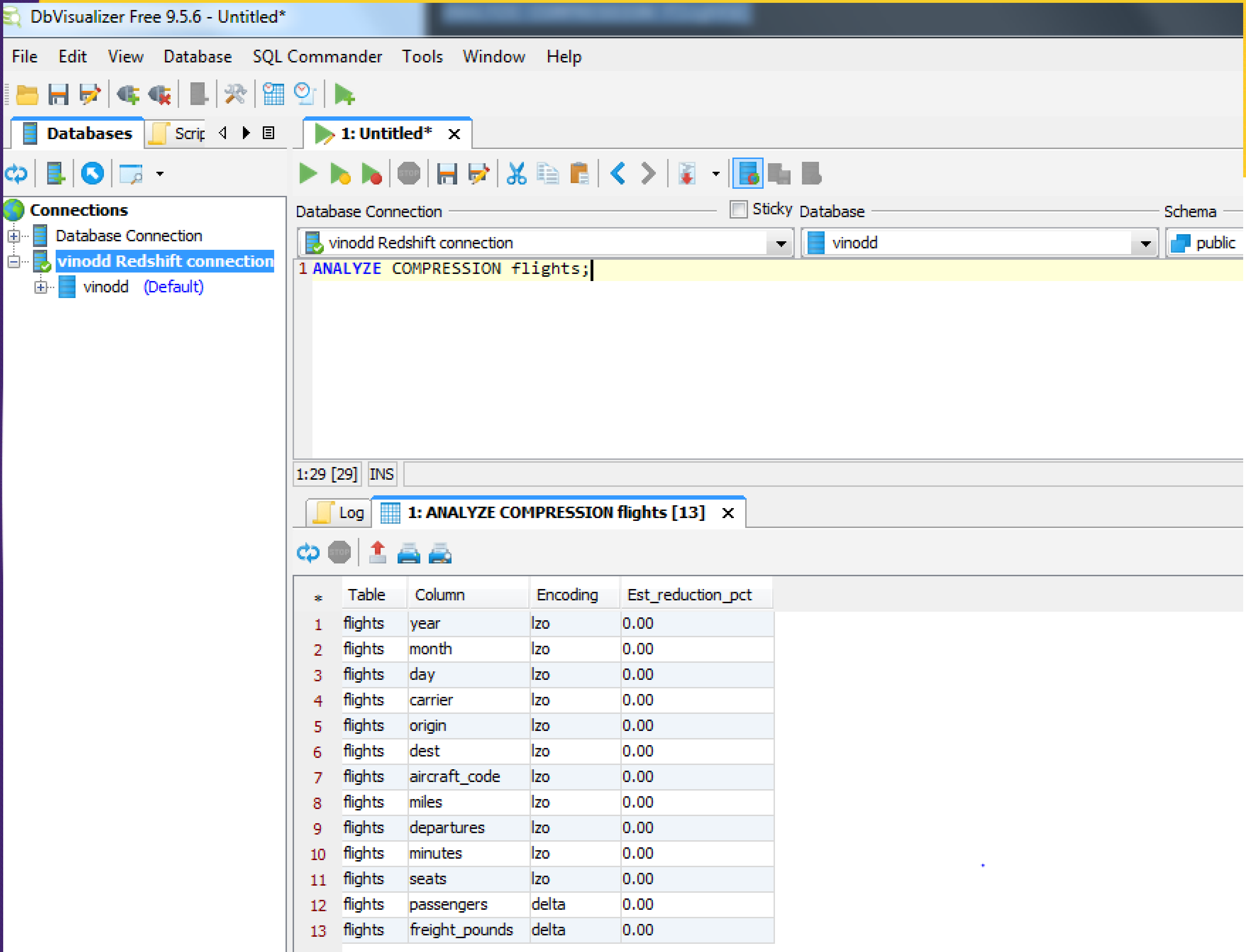

DbVisualizer Free 9.5.6 - Untitled*
File Edit View Database SQL Commander Tools Window Help
Databases
Scrip
1: Untitled*
Connections
Database Connection
vinodd Redshift connection
vinodd (Default)
Database Connection
Sticky
Database
Schema
vinodd Redshift connection
vinodd
public
1 SELECT carrier,SUM (departures) FROM flights GROUP BY carrier ORDER BY 2 DESC LIMIT 10;
1:88 [88]
INS
Log
1: flights [10]
carrier
sum
1 Southwest Airlines Co. 19846786
2 Delta Air Lines Inc. 17480331
3 American Airlines Inc. 15468460
4 United Air Lines Inc. 13402737
5 Northwest Airlines Inc. 9522027
6 Continental Air Lines Inc. 7929064
7 US Airways Inc. 7865177
8 American Eagle Airlines Inc. 6687299
9 ExpressJet Airlines Inc. 6270410
10 USAir 6052546

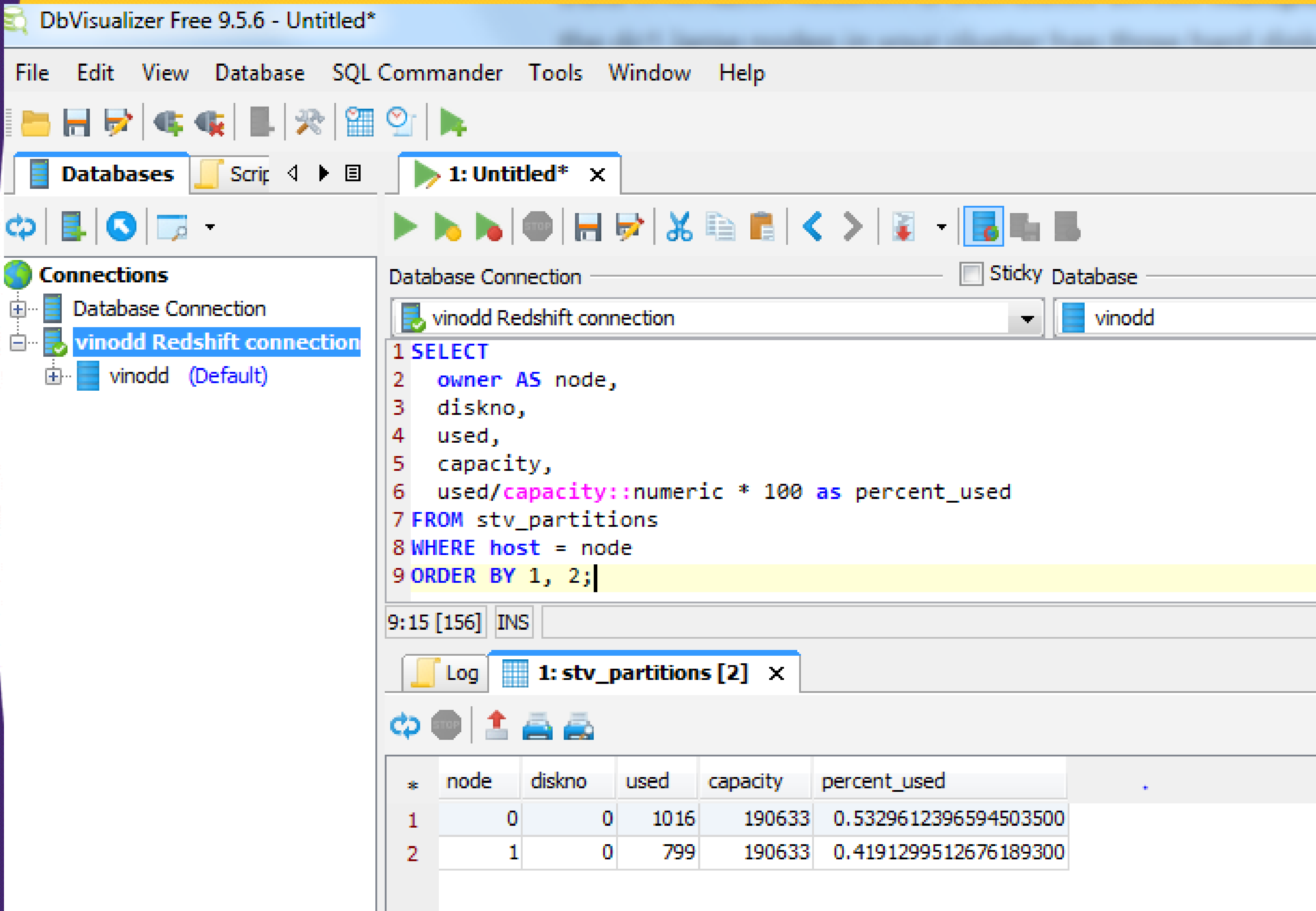

DbVisualizer Free 9.5.6 - Untitled*
File Edit View Database SQL Commander Tools Window Help
Databases
Scrip
1: Untitled*
Connections
Database Connection
vinodd Redshift connection
vinodd (Default)
Database Connection
Sticky
Database
Schema
vinodd Redshift connection
vinodd
public
1 SELECT carrier,SUM (passengers) FROM flights GROUP BY carrier ORDER BY 2 DESC LIMIT 10;
1:31 [31]
INS
Log
1: flights [10]
carrier
sum
1 Delta Air Lines Inc. 1894041955
2 Southwest Airlines Co. 1812013508
3 American Airlines Inc. 1599088928
4 United Air Lines Inc. 1396240286
5 Northwest Airlines Inc. 846994753
6 Continental Air Lines Inc. 751752616
7 US Airways Inc. 738513678
8 USAir 406251028
9 America West Airlines Inc. 327787337
10 Alaska Airlines Inc. 290615980



DbVisualizer Free 9.5.6 - Untitled*
File Edit View Database SQL Commander Tools Window Help
Databases
Scrip
1: Untitled*
Connections
Database Connection
vinodd Redshift connection
vinodd (Default)
Database Connection
Sticky
Database
Schema
vinodd Redshift connection
vinodd
public
1 SELECT carrier,SUM (miles) FROM flights GROUP BY carrier ORDER BY 2 DESC LIMIT 10;
1:83 [83]
INS
Log
1: flights [10]
carrier
sum
1 United Air Lines Inc. 7478582456
2 Delta Air Lines Inc. 7356255471
3 American Airlines Inc. 6033117399
4 Southwest Airlines Co. 5326561689
5 Continental Air Lines Inc. 4454443574
6 Northwest Airlines Inc. 4277614432
7 US Airways Inc. 3193773517
8 USAir 2178906790
9 America West Airlines Inc. 1713042483
10 Federal Express Corporation 1514604035


DbVisualizer Free 9.5.6 - Untitled*
File Edit View Database SQL Commander Tools Window Help
Databases
Connections
Database Connection
vinodd Redshift connection
vinodd (Default)
1: Untitled*
Database Connection
Sticky
Database
Schema
vinodd Redshift connection
vinodd
public
SELECT
aircraft,
SUM(departures) AS trips
FROM flights
JOIN aircraft using (aircraft_code)
GROUP BY aircraft
ORDER BY trips DESC
LIMIT 10;
8:10 [143]
INS
Log
1: flights [10]
aircraft
trips
1 Boeing 737-300 19632153
2 McDonnell Douglas DC9 Super 80/MD81/82/83/88 18608868
3 Canadair RJ-200ER /RJ-440 12062704
4 Boeing 757-200 10768257
5 Boeing 727-200/231A 9188041
6 Embraer-145 9184729
7 Boeing 737-100/200 8567467
8 Boeing 737-700/700LR 7550737
9 McDonnell Douglas DC-9-30 7105295
10 Airbus Industrie A320-100/200 6664803


DbVisualizer Free 9.5.6 - Untitled*
File Edit View Database SQL Commander Tools Window Help
Databases
Scrip
1: Untitled*
Connections
Database Connection
vinodd Redshift connection
vinodd (Default)
Database Connection
Sticky
Database
Schema
vinodd Redshift connection
vinodd
public
1 ANALYZE COMPRESSION flights;
1:29 [29]
INS
Log
1: ANALYZE COMPRESSION flights [13]
Table Column Encoding Est_reduction_pct
1 flights year lzo 0.00
2 flights month lzo 0.00
3 flights day lzo 0.00
4 flights carrier lzo 0.00
5 flights origin lzo 0.00
6 flights dest lzo 0.00
7 flights aircraft_code lzo 0.00
8 flights miles lzo 0.00
9 flights departures lzo 0.00
10 flights minutes lzo 0.00
11 flights seats lzo 0.00
12 flights passengers delta 0.00
13 flights freight_pounds delta 0.00


DbVisualizer Free 9.5.6 - Untitled*
File Edit View Database SQL Commander Tools Window Help
Databases
Scrip
1: Untitled*
Connections
Database Connection
vinodd Redshift connection
vinodd (Default)
Database Connection
Sticky
Database
vinodd Redshift connection
vinodd
SELECT
owner AS node,
diskno,
used,
capacity,
used/capacity::numeric * 100 as percent_used
FROM stv_partitions
WHERE host = node
ORDER BY 1, 2;
9:15 [156]
INS
Log
1: stv_partitions [2]
node
diskno
used
capacity
percent_used
1 0 0 1016 190633 0.53296123965945035OO
2 1 0 799 190633 0.41912995126761893OO

SQL

```
COPY flights     FROM
    's3://us-west-2-aws-training/awsu-spl/spl17-redshift/static/data/flights-usa'     CREDENTIALS ''     GZIP
DELIMITER ','     REMOVEQUOTES
```

Query Execution Details

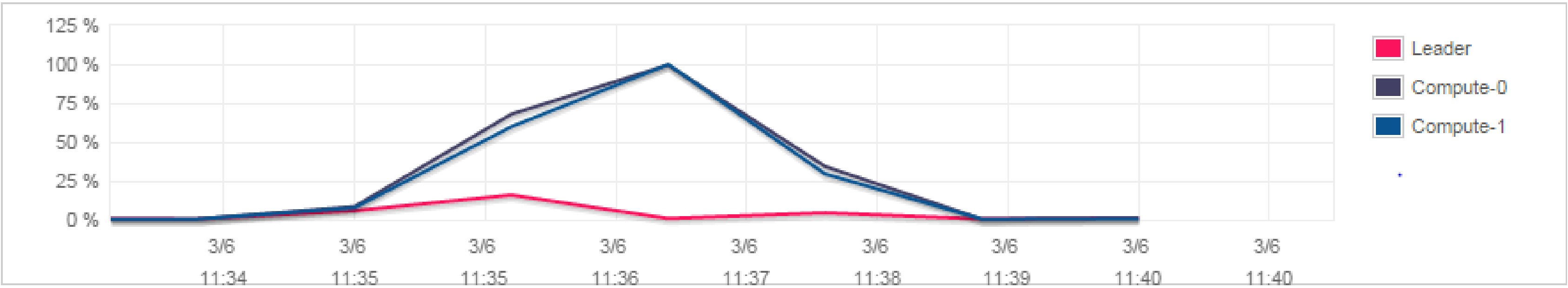
Loaded Files

Filename

Cluster Performance During Query Execution

Cluster performance is shown from 3 minutes prior to query execution through 3 minutes after query completion.

CPU utilization



Resource Groups

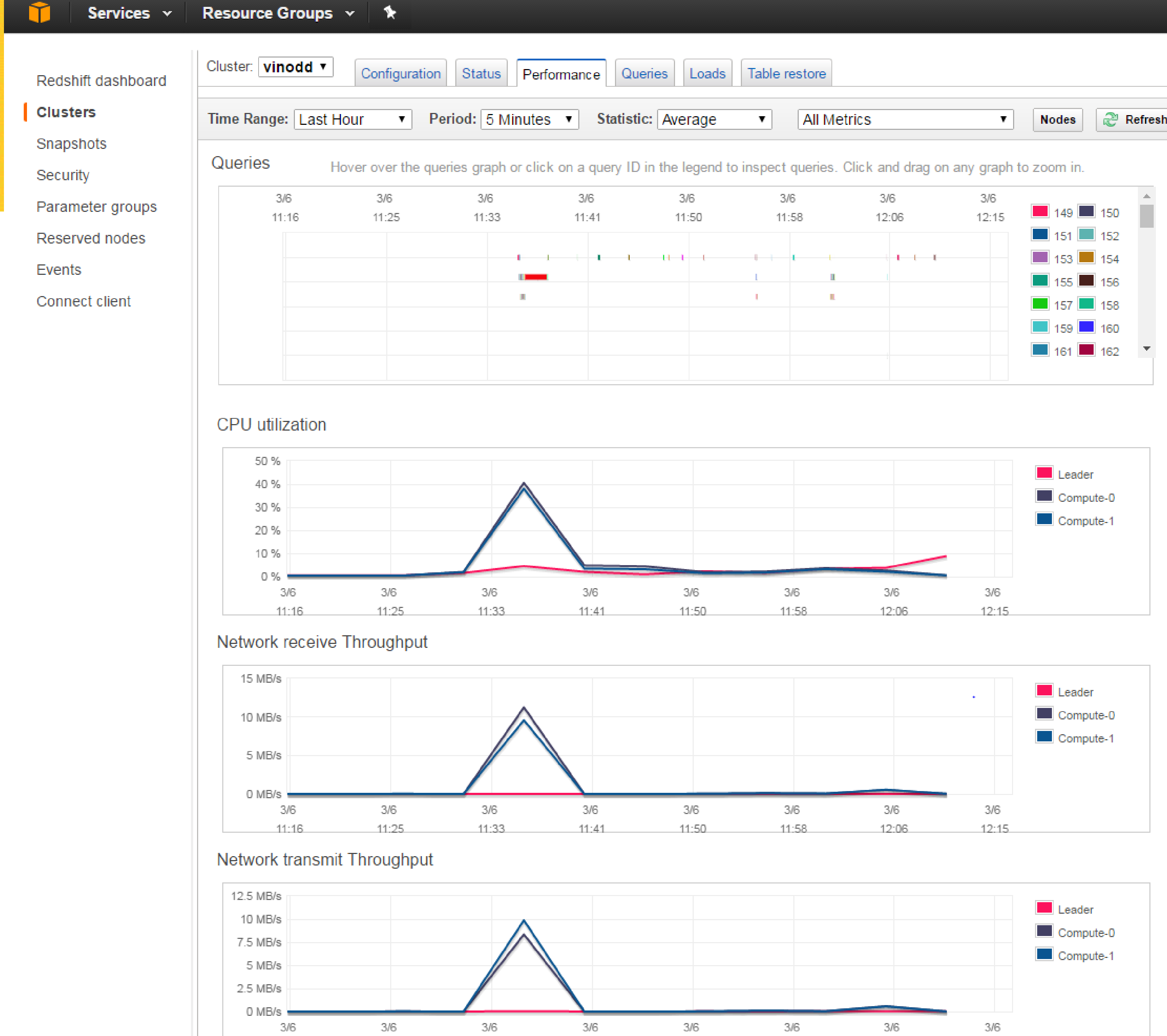
Cluster: vinodd

Configuration | Status | Performance | Queries | Loads | Table restore

Terminate Query

Filter: Last 24 Hours | Search...

Query	Run time	Start time	Status	User	SQL
321	8.64s	March 6, 2017 at 12:10:17 PM UTC-5	completed	master	SELECT name, count(*) FROM stv_blocklist JOIN (SE
318	3.96s	March 6, 2017 at 12:08:42 PM UTC-5	completed	master	SELECT owner AS node, diskno, used, capacity, use
311	9.17s	March 6, 2017 at 12:07:16 PM UTC-5	completed	master	CREATE TABLE vegas_flights DISTKEY (origin) SORTK
308	1.02s	March 6, 2017 at 12:06:26 PM UTC-5	completed	master	padb_fetch_sample: select * from airports
307	4ms	March 6, 2017 at 12:06:26 PM UTC-5	completed	master	padb_fetch_sample: select count(*) from airports
306	1.58s	March 6, 2017 at 12:06:25 PM UTC-5	completed	master	COPY airports FROM 's3://us-west-2-aws-training/a
305	40ms	March 6, 2017 at 12:06:25 PM UTC-5	completed	master	analyze compression phase 2
304	11ms	March 6, 2017 at 12:06:25 PM UTC-5	completed	master	analyze compression phase 1
303	37ms	March 6, 2017 at 12:06:25 PM UTC-5	completed	master	analyze compression phase 2
302	6ms	March 6, 2017 at 12:06:25 PM UTC-5	completed	master	analyze compression phase 1
301	1.08s	March 6, 2017 at 12:06:24 PM UTC-5	completed	master	COPY ANALYZE airports
290	39ms	March 6, 2017 at 12:02:06 PM UTC-5	completed	master	analyze compression phase 2
289	2.07s	March 6, 2017 at 12:02:04 PM UTC-5	completed	master	analyze compression phase 1


Services
Resource Groups
Redshift dashboard
Clusters
Snapshots
Security
Parameter groups
Reserved nodes
Events
Connect client
Cluster: vinodd
Configuration
Status
Performance
Queries
Loads
Table restore
Time Range: Last Hour
Period: 5 Minutes
Statistic: Average
All Metrics
Nodes
Refresh
Queries
Hover over the queries graph or click on a query ID in the legend to inspect queries. Click and drag on any graph to zoom in.
149 150 151 152 153 154 155 156 157 158 159 160 161 162
CPU utilization
50 % 40 % 30 % 20 % 10 % 0 %
3/6 11:16 3/6 11:25 3/6 11:33 3/6 11:41 3/6 11:50 3/6 11:58 3/6 12:06 3/6 12:15
Leader
Compute-0
Compute-1
Network receive Throughput
15 MB/s 10 MB/s 5 MB/s 0 MB/s
Network transmit Throughput
12.5 MB/s 10 MB/s 7.5 MB/s 5 MB/s 2.5 MB/s 0 MB/s

Services | Resource Groups

- Redshift dashboard
- **Clusters**
- Snapshots
- Security
- Parameter groups
- Reserved nodes
- Events
- Connect client

Cluster: vinodd

Configuration | Status | Performance | Queries | Loads | Table restore

Cluster Status

Cluster Status	available
Database Health	healthy
In Maintenance Mode	no
Parameter Group Apply Status	in-sync
Pending Modified Values	None

Recent Events

Time	Event	Source ID	Source type
Mar 6 10:43 AM	Amazon Redshift cluster 'vinodd' has been created at 2017-03-06 15:43 UTC and is ready for use.	vinodd	cluster
Mar 6 10:43 AM	Cluster restart is complete.	vinodd	cluster
Mar 6 10:43 AM	Cluster is being restarted.	vinodd	cluster

CloudWatch Alarms

No alarms configured

No alarms created. You can create an alarm using the Create Alarm button above