paper looker

Looker for Amazon Redshift

Technical Overview

In the post-cloud era, the traditional data warehousing (DW) model of moving "all data from everywhere" into an on-premise megaserver for advanced analytics breaks down. Many data-savvy, born-of-the-web companies are looking for a more streamlined solution that can exploit the scale and economics of the cloud. This is what Looker and Amazon Redshift provide.

Looker is proud to be a Technology Partner in the AWS Partner Network.

Looker and Redshift

Amazon Redshift is a fully managed, high-performance MPP data warehouse solution in the cloud that can scale up to a petabyte or more, while costing an order-of-magnitude less than legacy data warehousing solutions.

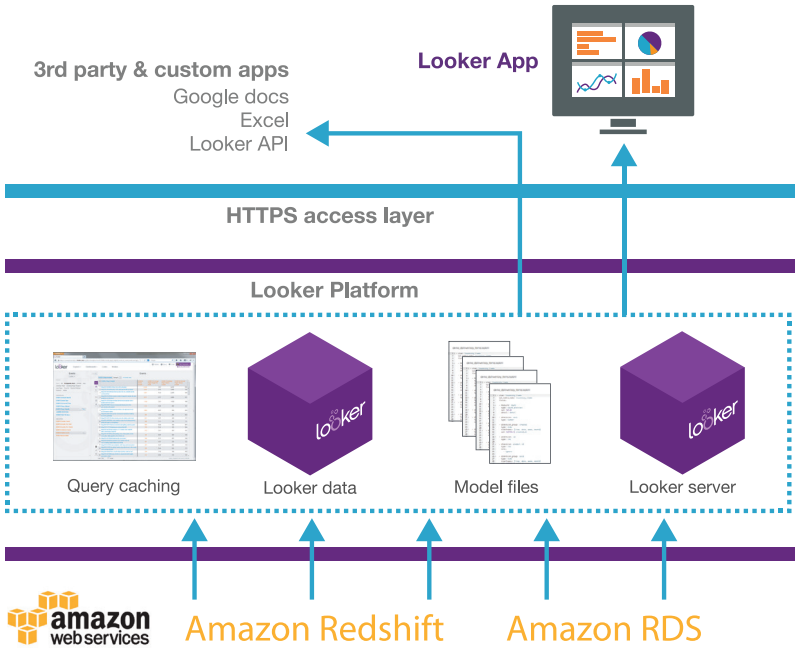
Looker is a business intelligence and data exploration platform that allows users of all skill levels to explore and visualize data stored in AWS Redshift and other SQL databases. Looker unlocks the most powerful, advanced functions of the underlying SQL data source (such as Redshift) without creating unmanageable complexity for data analysts and line-of-business users.

Looker leverages LookML, a data modeling language that provides modularity and reusability to SQL. LookML unleashes the potential in many Redshift optimizations, such as: Redshift dialect-specific SQL constructs; time-zone conversion and filters; the ability to create tables on the fly with sort keys and distribution keys on user-designated columns.

The Looker/AWS Ecosystem

Looker was built with MPP databases like Redshift in mind. By querying an MPP data warehouse directly for just the data needed to answer a question, Looker is the most efficient BI path in terms of hardware, storage, and computing power.

How Looker works with Amazon Redshift



The Looker platform includes a web application as well as a set of APIs.

At a high level, the factors that affect the performance, scale, and cost of the Looker for Redshift solution are:

- Moving data into Redshift
- In-database tuning—optimizing Redshift (cluster types, schema design, and compression) for Looker
- Looker model tuning—optimizing Looker (data model, caching, derived tables) for Redshift

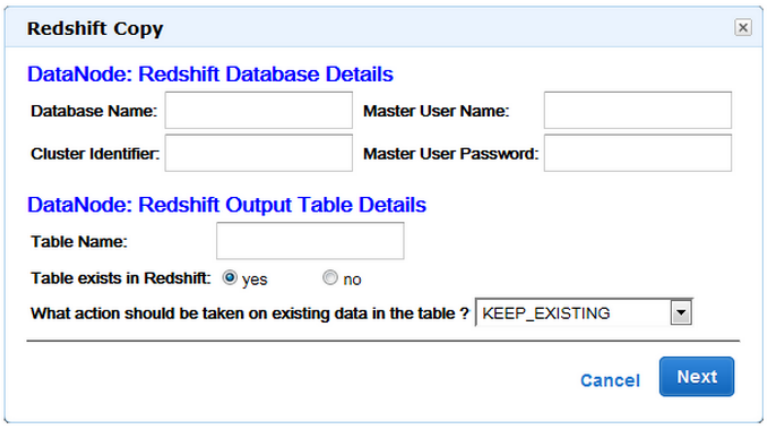
Moving Data into Redshift

Data is most commonly loaded into AWS Redshift via the COPY command, which exploits the Amazon Redshift MPP architecture to read and write files in parallel. You can follow any of these four methods to move data into Redshift:

- Using the Redshift COPY command to copy data from Amazon S3
- Using Redshift COPY to copy data from DynamoDB
- Copying data from Hadoop Amazon Elastic MapReduce (EMR)
- Using Redshift COPY with SSH to copy data from remote hosts

COPY from S3

The workflow using COPY usually involves a script to automate database dumps from an OLTP database (e.g., MySQL / PostgreSQL) to files in an AWS S3 bucket.* Amazon also offers AWS Pipeline to help streamline this process.



COPY from DynamoDB

For Looker customers using a non-relational schema, such as DynamoDB, the same Redshift COPY command is used to load a Redshift table with data from a single Amazon DynamoDB table. For more information about the DynamoDB > Redshift pipeline, see the AWS website.

Note: If your data is in a non-relational format and makes heavy use of JSON blocks, please contact a Looker analyst for tips on Redshift's JSON_EXTRACT_PATH_TEXT function.

Moving data from Amazon Elastic MapReduce (EMR)

For customers who are running a Hadoop/MapReduce environment in Amazon EMR, there are several methods of extracting data to Redshift. The Redshift COPY command can be used, or data can be extracted using EMR's Hadoop tools, such as Pig, Hive, or Cloudera Impala.

Note: Each of these methods has specific requirements for the Amazon EMR environment. Please contact a Looker analyst for tips on the best method to extract data from Hadoop/EMR environments.

COPY from remote hosts using SSH

Using the COPY command with SSH enables you to move data from one or more remote hosts. For the syntax used to do this, see the Redshift documentation on the AWS website.

* When using the COPY command from PostgreSQL, it is important to handle unsupported data types (covered in the AWS docs). There are also a number of third-party vendors that sell solutions for this problem, including SnapLogic and FlyData, but they can be expensive.

Optimizing Redshift for Looker Performance: In-Database Tuning

Choosing the right cluster size and type

AWS offers two cluster types:

• Dense Storage (DW1) node clusters use HDDs and are cheaper when you need to store and query very large amounts of data in Redshift.

• Dense Compute (DW2) node clusters use SSDs and more RAM, which costs more—especially when you have many terabytes of data—but can allow for much faster querying and a better interactive experience for your business users. If you have under 1TB, it usually makes sense to use SSDs; if over 1 TB, it depends on your use case.

AWS Redshift Node Configurations					
	vCPU	ECU	Memory	Storage	I/O
Dense Storage (DW1)					
dw1.xlarge	2	4.4	15GB	2TB HDD	0.30 GB/s
dw1.8xlarge	16	35	120GB	16TB HDD	2.40 GB/s
Dense Compute (DW2)					
dw2.large	2	7	15	0.16TB SSD	0.20 GB/s
dw2.8xlarge	32	104	244	2.56TB SSD	3.70 GB/s

Schema design and architecture

Redshift (and other MPP databases) distribute data across nodes and slices that eschew the concept of a table index, favoring instead sort keys and distribution keys that define where data is stored and how it is distributed.

Sort keys

Every table in Redshift can have one or more sort keys. Redshift stores data in 1MB blocks, storing the min and max values for each sort key present in that block. The main benefit of sort keys is that Redshift can skip over blocks of data when a sort key is present and the query is limited by that column. Sort keys are most important on large fact tables, often on a timestamp column. They can also be important for join performance when used on dimension/object tables, as Redshift does a sequential scan if the columns are not sort keys.

For example, if you have a 50 billion–row event table with three years of data, you may often need to run queries on just “today” or “last 7 days” in Looker. If you have a sort key on `created_at` and include a filter on that field in Looker, Redshift will be able to skip over 99% of rows (i.e., blocks of data) when executing the query.

```
SELECT COUNT(DISTINCT user_id)
FROM events
WHERE created_at < dateadd(day,-30,current_date)
```

If you load rows incrementally into a table in your ETL process, you will also need to run VACUUM on the table every so often to keep things sorted. Until you run VACUUM, the incrementally added rows will live separately on the node.

Distribution styles and DISTKEY

When a SQL query is sent from Looker to Redshift, it first goes to a leader node that plans out how to execute the query. Data needed for the query is then moved to compute nodes where the query runs. The time to move data to compute nodes to execute a query can have a major impact on performance, so it helps to distribute your data in a way that minimizes how much of it has to be moved.

There are three distribution options:

- **Even —** Even is the default distribution method, and will distribute rows across the slices in a "round-robin" as they are created, so they are not explicitly sorted.
- **Key —** Rows are distributed according to a specified column.
- **ALL —** All rows are distributed on every node, multiplying the amount of storage and write time required, but minimizing the amount of network traffic needed to move data to other nodes for each query. This is only recommended for tables that are not updated frequently or extensively.

A key-distributed table may have one distribution key (DISTKEY). It is recommended to distribute on a dimension table's primary key and a fact table's corresponding foreign key.

For example, in an e-commerce schema, an orders table should likely be distributed by `customer_id,` and a `customers` table should be distributed by the customer's `id` (the primary key). This design minimizes network traffic when joining the `orders` and `customers` tables, because all of each customer's data in both tables is likely on the same node already.

There may be other considerations—such as how often you join `orders` and `order_items` (in which case you probably join on `order_id`)—that you might also want to take into account.

If there are multiple foreign keys in a table (e.g., Star Schema), there can be only one DISTKEY per table, so you may have to make a tradeoff. This may vary depending on your use case, but our recommendation is usually to put the DISTKEY on the foreign key column that corresponds to the largest joined table—which will minimize network traffic on large joins. (So if you often join a 1,000-row table and a 100 million–row table to the fact table, you should distribute on the column that joins to the 100 million–row table.)

Joins and schema design

A rule of thumb we use at Looker is that each join can cost a 10-40% speed reduction on a query—sometimes much more if not optimized and distributed efficiently or when joining large tables. Adding this factor to the columnar design of databases like Redshift, it is often better to have long and wide tables than many separate tables.

Compression

Data stored in Redshift can be compressed by column, reducing the amount of disk space it takes up and also the amount of network I/O needed when executing queries.

If you load data into your Redshift cluster via the COPY command, automatic compression is applied. You can also set COMPUPDATE to OFF if you don't want to apply automatic compression, and run ANALYZE COMPRESSION to see compression recommendations for each column in a table.* More info on AWS docs.

Optimizing Looker for Redshift Performance: Looker Model Tuning

This section covers Looker best practices when building a LookML model on a Redshift data source.

Looker joins

Looker does LEFT JOINs by default (when using Looker's `foreign_key:` syntax), but you can specify any join type you prefer. In most cases it helps to have a DISTKEY and a SORT KEY on join columns for optimal performance.

Conditional and always filters

When setting up a LookML model for your business users, you often want to guide them toward writing efficient queries that are limited by a column with a sort key on it. (See example in previous section about adding a sort key on `created_at` on a large events table.)

For example, to bound a query to 30 days, unless filtering by specific users' events:

```
14 - explore: events
15   conditionally_filter:
16     events.created_date: 30 days
17     unless: [users.name, users.id]
```

The same concept applies when using `always_filter` instead of `conditionally_filter`, except that the filter can never be removed.

Caching

Caching is helpful when many of your users are running the same exact queries many times a day, such as for use in a dashboard or common Look. All query results are cached for five minutes on the Looker server by default, but this can be extended using the `persist_for` parameter:

```
- explore: events
  persist_for: 24 hours   # cache all results from events explore
```

* Looker customers report that Redshift recommends LZO (a lossless data compression algorithm focused on decompression speed) most often, especially for strings.

Scheduled reports

The Looker scheduling feature allows you to run reports at a designated schedule so you don't have to wait for the results live. For example, a very complex report that takes two minutes to run can be scheduled to run in the middle of the night and emailed to you as an HTML, CSV, TXT, or JSON format when it is ready.

Derived tables

Looker allows two types of derived tables to be created as Looker views: non-persistent (ephemeral) and persistent derived tables. For an overview of derived tables in Looker, please see the full documentation.

Non-persistent derived tables

If a derived table is constructed without a `persist_for` or `sql_trigger_value` parameter, its SQL will be treated as a Common Table Expression (CTE).* This will run a query using the Redshift WITH( ) clause:

```
WITH some_table AS (SELECT * FROM some_table)
SELECT * FROM some_table;
```

A WITH() clause is functionally the same as using a subquery in Redshift. If a WITH( ) clause or subquery is used often, Redshift may reuse the results from internal memory on subsequent runs if the query optimizer determines it is optimal to do so.

Persistent derived tables

Sometimes a table takes a long time to compute and is best stored as a persistent table in the database. This can be achieved using trigger values and `persist_for` parameters on derived tables. Persistent derived tables can be an advanced technique. See the full documentation for examples, and reach out to your Looker analyst if you need assistance.

```
- view: users_sales_facts
  derived_table:
    sql: |
      SELECT
        o.user_id AS user_id
        , COUNT(*) AS lifetime_items
        , ROUND(AVG(oi.sale_price), 2) AS average_item_price
        , ROUND(SUM(oi.sale_price), 2) AS lifetime_revenue
      FROM order_items AS oi
      LEFT JOIN orders AS o
      ON oi.order_id = o.id
      GROUP BY user_id
    sortkeys: [user_id]
    distkey: user_id
    sql_trigger_value: SELECT MAX(id) FROM orders # rebuild when a new order is added
```

* Note that Redshift does not allow CTEs inside of CTEs.

Monitoring Query Performance

Looker Usage — "LookInside" Model

Looker maintains query metadata and statistics from all Looks and Explore queries run in the application. To see and explore how your Looker models are being used, go to the Admin > Usage page in Looker.

Query Runtime Performance

	QUERY Runtime Tiers		T01 [0,5)		T02 [5,10)		T03 [10,30)	
	QUERY Model	QUERY View	HISTORY Query Run Count	USER Count	HISTORY Query Run Count	USER Count	HISTORY Query Run Count	USER Count
87	faa	aircraft	98	14	10	6	1	
88	faa	airports	428	22	1	1	1	
89	faa	flights	1,796	40	202	25	36	1
90	finance	fmac_performance	1	1	1	1	∅	
91	finance	fnm_acquisitions	1	1	∅	∅	∅	
92	fod_demo	inventory_items	6	1	∅	∅	∅	
93	foobar	order_items	3	1	∅	∅	∅	
94	frans_data	order_items	5	1	∅	∅	∅	

Commonly Used Fields

	FIELD USAGE Base View	FIELD USAGE Field	FIELD USAGE Model	FIELD USAGE Times Used
1	order_items	order_items.count	thelook	24,492
2	order_items	order_items.count	thelook_redshift	20,879
3	order_items	order_items.total_sale_price	thelook	14,920
4	orders	orders.count	thelook	14,848
5	order_items	order_items.total_sale_price	thelook_redshift	13,487
6	order_items	category.name	thelook	11,308
7	order_items	orders.created_date	thelook_redshift	11,149

pg_catalog — "LookBelow" Model

Each Redshift instance maintains history and statistics about such details as data loads, queries, and columns in the pg_catalog schema.

It is possible to build a Looker model on top of the pg_catalog schema if your Looker Redshift user has sufficient permissions to do so. This can help with DBA dashboards, automated alerts about failed batch jobs, etc. Contact Looker support for help setting up a pg_catalog model.

Additional Reading

AWS Redshift Documentation

AWS maintains superb, detailed, and searchable documentation for Redshift. You can probably answer 95% of your questions from reading their docs.

Looker Support

Contact Looker Support for specific questions about your setup and Looker. We have advised more than 100 leading companies on Redshift optimization.

Snowplow Analytics: Open Source Event Analytics Platform

Looker customers on Redshift often use the Snowplow platform as a more flexible and free alternative to commercial trackers, such as Google Analytics or Mixpanel.

For a fresh look at your own data, sign up for a free Looker trial.

Just tell us how to connect to your analytics database, and you'll experience the full Looker functionality free of charge.

Try Looker for free.
Or schedule a demo
at: looker.com/free-trial

About Looker

Looker is an inventive software company that's pioneering the next generation of business intelligence (BI). We believe that businesses can only thrive when data is consistently defined and easily accessible across the entire organization.

Our web-based platform powers the work of data analysts while fueling (and fulfilling) the business user's curiosity. Looker is creating true discovery-driven businesses and unlocking the value of their data, one customer at a time.

Looker is based in Santa Cruz, CA | looker.com