Writing optimal queries

INTRODUCTION TO REDSHIFT

Jason Myers
Principal Engineer

datacamp

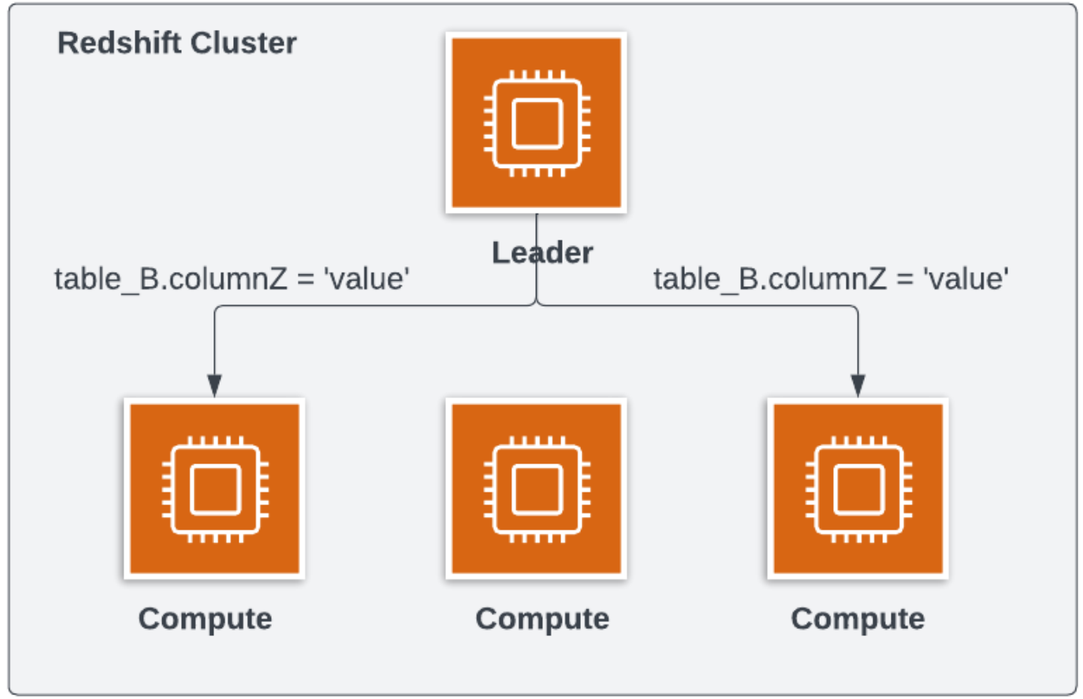
Limit columns

- Avoid `SELECT *`
- Don't select columns you don't need in the result
  - Remember that Redshift is columnar and pulls data by column

Use DISTKEY and SORTKEYs

Use in the following clauses whenever possible

- `JOIN`
- `WHERE`
- `GROUP BY`



Use `SORTKEYs` in order in `ORDER BY`

- Highly optimized
  `sortkey_1, sortkey_2, sortkey_3`
- Not optimized `sort_key_1, sort_key_3`

Building good predicates

- Use `DISTKEY` and `SORTKEY`
- Close to the table join
- Avoid using functions in them

```
SELECT receipts.cookie_id,
       sum(receipts.total)
FROM receipts
JOIN cookies ON receipts.cookie_id = cookies.cookie_id
  -- Keep cookies predicates in the join to push down to nodes holding the records for cookies
 AND cookies.available_on < '2023-11-14'
 AND cookies.end_of_sale IS null
-- Predicates that are not part of the join or on the joined table stay in the WHERE clause
WHERE receipts.order_time > '2023-11-13'
GROUP BY 1 ORDER BY 1;
```

Be consistent with column ordering

When using:

- `GROUP BY`
- `ORDER BY`

Bad

```
GROUP BY col_one, col_two, col_three
ORDER BY col_two, col_three, col_one
```

Good

```
GROUP BY col_two, col_three, col_one
ORDER BY col_two, col_three, col_one
```

Use subqueries wisely

- Use proper join strategies instead of just using a subquery
- Use `EXISTS` in your predicates when just checking for the truthfulness of a subquery result

```
SELECT column_name
FROM table_name
WHERE EXISTS
  (SELECT column_name
   FROM table_name
   WHERE active is True);
```

- If reusing subqueries, use CTEs to take advantage of caching

Let's practice!

INTRODUCTION TO REDSHIFT

Understanding query performance

INTRODUCTION TO REDSHIFT

Jason Myers
Principal Architect

datacamp

Query optimization process

1. Check the `STL_ALERT_EVENT_LOG` table
2. Run `EXPLAIN` on your query
3. Check the `SVL_QUERY_SUMMARY` and `SVL_QUERY_REPORT` tables

STL_ALERT_EVENT_LOG table

Contains any Redshift alerts triggered by a query

```
SELECT *
  FROM stl_alert_event_log
 WHERE query = 1337;
```

```
 query |              event                |          solution          |     event_time
=======+===================================+============================+====================
  1337 | Missing query planner statist | Run the ANALYZE command | 2023-11-13 18:20:58
```

EXPLAINing the query execution plan

- Shows all the steps of a query and their relative costs and rows scanned
- Works on all Data Manipulation Language (DML) statements:
  - `SELECT`
  - `SELECT INTO`
  - `CREATE TABLE AS`
  - `INSERT`
  - `UPDATE`
  - `DELETE`

Explain example

```
-- Running EXPLAIN on our top ten divisions by
-- revenue query
EXPLAIN WITH top_ten_divisions_by_rev AS
(
  SELECT division_id,
         SUM(revenue) AS revenue_total
    FROM sales_data
   GROUP BY division_id
   ORDER BY revenue_total DESC
   LIMIT 10
),
```

```
division_names AS
(
  SELECT id AS division_id,
         name AS division_name
    FROM division_names
)
SELECT division_name,
       revenue_total
  FROM top_ten_divisions_by_rev
  JOIN division_names USING (DIVISION_ID)
 WHERE revenue_total > 100000;
```

Explain results

```
|QUERY PLAN                                                                                                  |
+------------------------------------------------------------------------------------------------------------+
|Hash Join  (cost=47.11..58.89 rows=3 width=524)                                                             |
|  Hash Cond: (division_names.id = top_ten_divisions_by_rev.division_id)                                     |
|  ->  Seq Scan on division_names  (cost=0.00..11.40 rows=140 width=520)                                     |
|  ->  Hash  (cost=47.07..47.07 rows=3 width=12)                                                             |
|        ->  Subquery Scan on top_ten_divisions_by_rev  (cost=46.92..47.07 rows=3 width=12)                  |
|              Filter: (top_ten_divisions_by_rev.revenue_total > 100000)                                     |
|              ->  Limit  (cost=46.92..46.95 rows=10 width=12)                                               |
|                    ->  Sort  (cost=46.92..47.42 rows=200 width=12)                                         |
|                          Sort Key: (sum(sales_data.revenue)) DESC                                          |
|                          ->  HashAggregate  (cost=40.60..42.60 rows=200 width=12)                          |
|                                Group Key: sales_data.division_id                                           |
|                                ->  Seq Scan on sales_data  (cost=0.00..30.40 rows=2040 width=8)|
```

[1] https://www.postgresql.org/docs/current/using-explain.html

Let's practice!

INTRODUCTION TO REDSHIFT

Redshift security

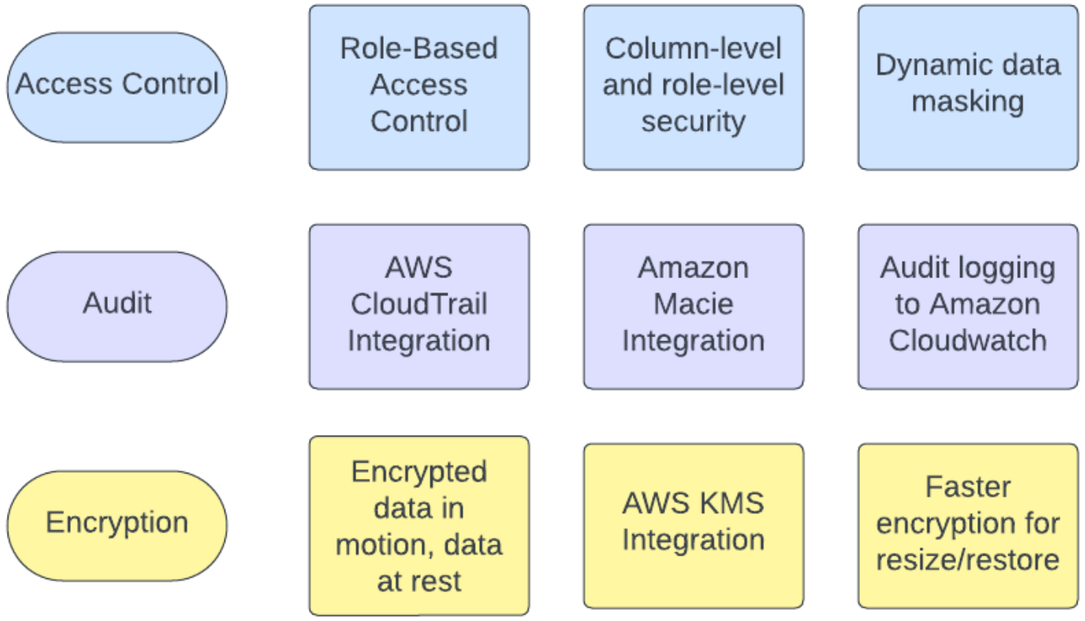
INTRODUCTION TO REDSHIFT

Jason Myers
Principal Engineer

datacamp

Redshift security

- Column level access control
- Row level security via policies
- Data masking via policies



Column level permissions

- Hides a column completely
- Can verify via `SVV_COLUMN_PRIVILEGES`

```
SELECT *
  FROM SVV_COLUMN_PRIVILEGES
 WHERE relation_name = 'products';
```

```
relation_name | column_name  | privilege_type | identity_name | identity_type
==============|==============|================|===============|==============
products      | product_name | SELECT         | amelia        | user
products      | product_name | SELECT         | analytics     | role
```

Row-level security

- Policies that prefilter data

```
CREATE RLS POLICY policy_books
WITH (category VARCHAR(255))
USING (category = 'Dark Academia');
```

```
SELECT product_line, category, product_name
FROM products;
```

```
product_line | category      | product_name
=============|===============|====================
Books        | Dark Academia | A Deadly Education
```

Row-level security

- SVV_RLS_POLICY to view policies

```
SELECT polname AS policy_name,
       polatts AS column_details,
       polqual AS condition
  FROM SVV_RLS_POLICY;
```

```
policy_name  | column_details                                    | condition
 ============|===================================================|==========================
policy_books | [{"colname":"category","type":"VARCHAR(255)"}] | category = 'Dark Academia'
```

Row-level security admin view

- SVV_RLS_APPLIED_POLICY can be used by Superusers to see affected queries

```
SELECT username,
       command,
       relschema,
       relname,
       polname,
FROM SVV_RLS_APPLIED_POLICY;
```

```
username | command | relschema | relname  | polname
=========|=========|===========|==========|=============
aashvi   |    s    | public    | products | policy_books
```

Dynamic Masking overview

- Policy that obscures values returned by a query
- Only a super user or someone granted can see them
- Uses
  - National ID numbers (e.g. Social Security Number)
  - Credit cards

```
SELECT name, social_security_number
  FROM customers;
```

```
name     | social_security_number
======== | ======================
John Doe | XXX-XX-1234
Jane Doe | XXX-XX-5678
```

Let's practice!

INTRODUCTION TO REDSHIFT

Congratulations

INTRODUCTION TO REDSHIFT

Jason Myers
Principal Engineer

Architecture and Data

Redshift Architecture

- Cluster node types
- How queries execute
- Spectrum Internal/external schemas

Data Types

- Standard types
- Semistructured data with the super type
- Conversions

Queries, Performance and Security

Redshift SQL and Functions

- Common Table Expressions
- Time traveling date functions
- Window functions for analysis
- Lead and Lag

Performance & Security

- Writing performant queries
- Investigating query performance issues
- Column level
- Row level
- Data masking

Tables and Views

Table Name	Description
SVV_ALL_SCHEMAS	View schema details
SVV_ALL_TABLES	View table details
SVV_ALL_COLUMNS	View table columns details
STV_PARTITIONS	View details about data across the cluster
SVV_TABLE_INFO	View all details about tables like DISTKEY, SORTKEY, and skew
STL_ALERT_EVENT_LOG	Queries that trigger alerts and possible resolutions
SVL_QUERY_SUMMARY	View query details like size, rows, and time
SVV_COLUMN_PRIVILEGES	View details about column-level security
SVV_RLS_POLICY	View row-level security policy details
SVV_RLS_APPLIED_POLICY	View show queries affected by row-level security policies

Next Steps

- Datacamp Redshift Projects
- AWS Redshift Immersion Labs
- Transactions and Error Handling Datacamp Course

Thank you and Congratulations!

INTRODUCTION TO REDSHIFT