Personal Notes: Redshift & Analytics Deep Dive

Table of Contents

Amazon Redshift Fundamentals

What is Amazon Redshift?

- **Definition**: Fully managed, petabyte-scale cloud data warehouse service
- **Purpose**: Fast query performance using SQL-based analytics on structured and semi-structured data
- **Category**: OLAP (Online Analytical Processing) system, not OLTP
- **Launch**: 2012, based on PostgreSQL 8.0.2

Key Characteristics

- **Columnar Storage**: Data stored in columns rather than rows
- **Massively Parallel Processing (MPP)**: Distributes queries across multiple nodes
- **Compression**: Automatic compression reduces storage and I/O
- **Machine Learning Integration**: Built-in ML capabilities with Redshift ML

Why Columnar Storage Matters

Traditional Row Storage:

```
Row 1: [ID=1, Name=John, Age=25, Salary=50000]
Row 2: [ID=2, Name=Jane, Age=30, Salary=60000]
Row 3: [ID=3, Name=Bob, Age=35, Salary=70000]
```

Columnar Storage:

```
ID Column:     [1, 2, 3, ...]
Name Column:   [John, Jane, Bob, ...]
Age Column:    [25, 30, 35, ...]
Salary Column: [50000, 60000, 70000, ...]
```

Benefits:

- Only reads columns needed for query (reduced I/O)
- Better compression (similar data types together)
- Vectorized processing (operates on entire columns)
- Cache efficiency (related data stored together)

Redshift Architecture Deep Dive

Cluster Architecture

Leader Node:

- Receives client connections
- Parses and develops query execution plans
- Coordinates parallel query execution
- Aggregates results from compute nodes
- One leader node per cluster

Compute Nodes:

- Execute queries in parallel
- Store data locally
- Communicate with leader node and each other
- Can have 1-128 compute nodes

Node Slices:

- Each compute node divided into slices
- Each slice processes portion of workload in parallel
- Number of slices depends on node type
- dc2.large = 2 slices, dc2.8xlarge = 32 slices

Data Distribution

Distribution Styles:

1. **AUTO**: Redshift chooses optimal distribution
2. **EVEN**: Rows distributed evenly across slices
3. **KEY**: Rows distributed based on values in one column
4. **ALL**: Copy of entire table on every slice

Why Distribution Matters:

- Minimizes data movement during joins
- Balances workload across nodes
- Affects query performance significantly

Example:

```sql
-- Key distribution for frequently joined tables
CREATE TABLE sales (
  sale_id INT,
  customer_id INT DISTKEY, -- Distribute by customer_id
  amount DECIMAL(10,2)
);

CREATE TABLE customers (
  customer_id INT DISTKEY, -- Same distribution key
  name VARCHAR(100)
);
-- Joins on customer_id will be local to each slice
```

Sort Keys

Purpose: Physical ordering of data on disk **Types:**

1. **Compound Sort Key**: Multi-column, prioritized left to right

2. **Interleaved Sort Key**: Equal weight to all columns

Example:

```sql
-- Compound sort key - good for range queries on date
CREATE TABLE sales (
  sale_date DATE,
  region VARCHAR(20),
  amount DECIMAL(10,2)
) COMPOUND SORTKEY (sale_date, region);

-- Queries like "WHERE sale_date BETWEEN '2024-01-01' AND '2024-01-31'"
-- will scan fewer blocks
```

Redshift Spectrum Explained

What is Redshift Spectrum?

- Feature that extends Redshift queries to data in S3
- No need to load data into Redshift first
- Uses external compute resources (separate from cluster)
- Supports various file formats: Parquet, ORC, CSV, JSON, Avro

Architecture Components

External Schema:

- Logical grouping of external tables
- References AWS Glue Data Catalog database
- Defines IAM roles for S3 access

External Tables:

- Metadata definitions pointing to S3 data
- Define structure (columns, data types)
- Specify file location and format

AWS Glue Data Catalog:

- Centralized metadata repository
- Stores table definitions, schemas, partitions
- Shared across multiple AWS analytics services

How Spectrum Works

1. Query submitted to Redshift leader node
2. Leader node identifies external tables in query
3. Generates execution plan mixing local and external data
4. Spectrum compute layer reads from S3
5. Results returned to Redshift for final processing
6. Leader node combines all results

Spectrum vs Traditional ETL

Traditional Approach:

```
S3 → ETL Process → Load to Redshift → Query
- Time: Hours/Days for large datasets
- Cost: Storage in both S3 and Redshift
- Complexity: ETL pipeline maintenance
```

Spectrum Approach:

```
S3 → Direct Query via Spectrum
- Time: Immediate querying
- Cost: Pay only for queries, not storage
- Complexity: Simple table definitions
```

Data Warehousing Concepts

Hot vs Cold Data

Hot Data:

- **Definition**: Frequently accessed, business-critical data
- **Characteristics**: Recent transactions, active customers, current inventory
- **Storage**: Expensive but fast (SSD, memory)

- **Example**: Last 90 days of sales data

Cold Data:

- **Definition**: Infrequently accessed, historical data
- **Characteristics**: Archived logs, old transactions, compliance data
- **Storage**: Cheap but slower (S3, tape)
- **Example**: Sales data from 5+ years ago

Warm Data:

- **Definition**: Occasionally accessed data
- **Storage**: Mid-tier options
- **Example**: Last year's monthly reports

OLAP vs OLTP

OLTP (Online Transaction Processing):

- Handles day-to-day operations
- Many small, fast transactions
- Row-based storage
- Examples: Order processing, user registration
- Database: RDS, Aurora

OLAP (Online Analytical Processing):

- Handles analysis and reporting
- Fewer, complex queries
- Columnar storage
- Examples: Sales analysis, trend reporting
- Database: Redshift, analytics engines

Data Warehouse Design Patterns

Star Schema:

```
Dimension Tables
    |
Fact Table (center)
    |
Dimension Tables
```

Snowflake Schema:

- Normalized dimension tables
- More complex but reduces redundancy

In Our Demo:

- `sales_data` = Fact table (transactions)
- `web_clicks` = Behavioral dimension data
- Region, customer info = Dimension attributes

Demo Technical Details

Why This Demo Architecture Works

Hot Data (sales_data in Redshift):

- Transactional data requiring fast aggregations
- Frequently queried for business reporting
- Benefits from columnar compression and parallel processing
- Small dataset shows immediate performance

Cold Data (web_clicks in S3):

- Behavioral data, less frequently analyzed
- Large volume potential (clickstream grows rapidly)
- Cost-effective to store in S3
- Occasional analysis needs don't justify warehouse storage

Hybrid Query Benefits:

- Customer journey analysis combining sales + behavior
- Single SQL interface for both data sources

- No complex ETL to move clickstream into warehouse
- Demonstrates real-world analytics pattern

Query Execution Deep Dive

Query 1: Hot Data Analysis

```sql
SELECT region, SUM(amount) as total_sales
FROM sales_data
GROUP BY region;
```

Execution Path:

1. Leader node receives query
2. Pushes down aggregation to compute nodes
3. Each slice processes its data portion
4. Local aggregation on each slice
5. Results sent back to leader for final aggregation
6. Fast execution due to columnar storage and parallel processing

Query 2: Cold Data Analysis

```sql
SELECT page, COUNT(*) as visits
FROM spectrum_demo.web_clicks
GROUP BY page;
```

Execution Path:

1. Leader node identifies external table
2. Generates Spectrum execution plan
3. Spectrum compute reads from S3
4. S3 Select optimizations (server-side filtering)
5. Results returned to Redshift
6. Slower than hot data but no storage cost

Query 3: Hybrid Analysis

```sql
SELECT s.region, w.page, COUNT(*)
FROM sales_data s
JOIN spectrum_demo.web_clicks w ON s.customer_id = w.user_id
GROUP BY s.region, w.page;
```

Execution Path:

1. Leader analyzes join requirements
2. Determines optimal join strategy
3. Loads hot data into memory
4. Streams cold data from S3
5. Performs join operation
6. Aggregates results

Performance Optimization

Redshift Performance Factors

1. Distribution Keys

- Choose frequently joined columns
- Ensure even distribution of data
- Avoid small dimension tables as distribution keys

2. Sort Keys

- Use columns in WHERE clauses
- Date columns are excellent sort keys
- Consider query patterns

3. Compression

- Redshift automatically chooses compression
- ANALYZE COMPRESSION command for recommendations
- Can achieve 3-4x compression ratios

4. Vacuuming

- Reclaims space after deletes/updates
- VACUUM FULL for heavy operations
- Auto VACUUM available

5. Table Design

```sql
-- Optimized table design
CREATE TABLE sales_optimized (
  sale_date DATE SORTKEY,        -- Range queries
  customer_id INT DISTKEY,       -- Join performance
  product_id INT,
  amount DECIMAL(10,2)
) DISTSTYLE KEY;
```

Spectrum Performance Optimization

1. File Format

- Parquet: Best compression and query performance
- ORC: Good alternative to Parquet
- CSV: Human readable but slower
- Avoid small files (< 100MB)

2. Partitioning

```sql
```

```sql
-- Partitioned external table
CREATE EXTERNAL TABLE spectrum_demo.web_clicks_partitioned (
  user_id INT,
  page VARCHAR(50),
  timestamp TIMESTAMP
)
PARTITIONED BY (year INT, month INT)
STORED AS PARQUET
LOCATION 's3://bucket/web-clicks/';

-- Query with partition pruning
SELECT * FROM spectrum_demo.web_clicks_partitioned
WHERE year = 2024 AND month = 1;
-- Only scans January 2024 partition
```

3. Projection Pushdown

- Spectrum only reads needed columns
- Especially effective with columnar formats
- Reduces network transfer

4. Predicate Pushdown

- WHERE clauses processed at S3 level
- Reduces data transfer to Redshift
- Works with S3 Select

Cost Management

Redshift Pricing Model

On-Demand Pricing:

- Pay per hour for compute nodes
- dc2.large: ~$0.25/hour
- dc2.8xlarge: ~$4.80/hour
- No upfront costs

Reserved Instances:

- 1 or 3-year commitments
- Up to 75% savings
- Good for predictable workloads

Redshift Serverless:

- Pay for actual usage (RPUs - Redshift Processing Units)
- No infrastructure management
- Good for variable workloads

Spectrum Pricing

- $5 per TB of data scanned
- No charges for failed queries
- Compression reduces costs significantly
- Partitioning reduces scanned data

Cost Optimization Strategies

1. Pause/Resume Clusters

```bash
# AWS CLI commands
aws redshift pause-cluster --cluster-identifier demo-cluster
aws redshift resume-cluster --cluster-identifier demo-cluster
```

2. Right-sizing

- Monitor CPU, memory, storage utilization
- Use CloudWatch metrics
- Consider node types based on workload

3. Reserved Capacity

- Analyze usage patterns
- Commit to reserved instances for steady workloads

4. Data Lifecycle

- Move old data to S3

- Use Spectrum for historical analysis
- Archive unused data

Demo Cost Breakdown

```
Demo Infrastructure Costs (1 hour):
- dc2.large Redshift cluster: $0.25
- S3 storage (minimal): $0.01
- Spectrum queries: $0.00 (small dataset)
- Total: ~$0.26

Monthly Production Estimate:
- 3-node dc2.large cluster: ~$540
- 10TB S3 data: ~$230
- 100 Spectrum queries (1TB each): ~$500
- Total: ~$1,270/month
```

Common Questions & Answers

Q: When should I use Redshift vs RDS?

A:

- **Use Redshift for**: Analytics, reporting, data warehousing, OLAP workloads
- **Use RDS for**: Transactional systems, OLTP workloads, operational databases
- **Key difference**: Redshift optimized for read-heavy analytical queries, RDS for transactional operations

Q: How does Redshift compare to other data warehouses?

A:

- **vs Snowflake**: Similar capabilities, Snowflake has compute/storage separation
- **vs BigQuery**: Google's serverless approach, different pricing model
- **vs Synapse**: Microsoft's offering, integrated with Azure ecosystem
- **Redshift advantages**: AWS ecosystem integration, mature service, cost-effective

Q: What's the difference between Redshift and Athena?

A:

- **Redshift**: Provisioned compute, faster for frequent queries, requires cluster management
- **Athena**: Serverless, pay-per-query, slower but no infrastructure management
- **Use Redshift**: Regular reporting, dashboards, frequent analysis
- **Use Athena**: Ad-hoc queries, infrequent analysis, exploration

Q: Can I query real-time data in Redshift?

A:

- Redshift is not designed for real-time streaming
- Near real-time possible with frequent batch loads
- Consider Kinesis Data Streams → Kinesis Data Firehose → Redshift for streaming
- Use Redshift Streaming Ingestion for real-time capabilities (newer feature)

Q: How do I handle schema changes?

A:

- Redshift supports ALTER TABLE for some changes
- Major schema changes may require table recreation
- Use external tables in Spectrum for schema evolution
- Plan schema design carefully upfront

Q: What about data security?

A:

- Encryption at rest and in transit
- VPC isolation
- IAM integration
- Database-level users and permissions
- Column-level security
- AWS PrivateLink support

Real-World Use Cases

Retail Analytics

```sql
```

```sql
-- Customer segmentation analysis
WITH customer_metrics AS (
  SELECT
    customer_id,
    COUNT(*) as order_count,
    SUM(amount) as total_spent,
    AVG(amount) as avg_order_value,
    MAX(order_date) as last_order_date
  FROM sales_data
  GROUP BY customer_id
)
SELECT
  CASE
    WHEN total_spent > 5000 THEN 'VIP'
    WHEN total_spent > 2000 THEN 'Premium'
    WHEN total_spent > 500 THEN 'Regular'
    ELSE 'New'
  END as customer_segment,
  COUNT(*) as customer_count,
  AVG(total_spent) as avg_lifetime_value
FROM customer_metrics
GROUP BY 1;
```

IoT Data Analysis

- Store sensor data in S3 (cold storage)
- Keep recent alerts in Redshift (hot storage)
- Use Spectrum to analyze historical patterns
- Join real-time alerts with historical trends

Financial Services

- Transaction data in Redshift for fraud detection
- Historical trades in S3 via Spectrum
- Regulatory reporting combining both sources
- Risk analysis across time periods

Media & Entertainment

- User engagement metrics in Redshift
- Content consumption logs in S3

- Recommendation engine training data
- A/B testing analysis

Troubleshooting & Best Practices

Common Performance Issues

1. Query Running Slowly

- Check query execution plan
- Look for missing join conditions
- Verify distribution and sort keys
- Consider VACUUM and ANALYZE

Diagnostic Queries:

```sql
-- Check table statistics
SELECT
  schemaname,
  tablename,
  n_tup_ins as inserts,
  n_tup_upd as updates,
  n_tup_del as deletes
FROM pg_stat_user_tables;

-- Check query performance
SELECT
  query,
  total_time,
  rows,
  starttime
FROM stl_query_metrics
WHERE userid > 1
ORDER BY starttime DESC;
```

2. Spectrum Queries Slow

- Check file format (prefer Parquet)
- Verify partitioning strategy

- Look for small files
- Consider compression

3. High Costs

- Monitor cluster utilization
- Check for unnecessary full table scans
- Review Spectrum data scanned
- Consider pause/resume for dev environments

Best Practices

1. Data Loading

- Use COPY command, not INSERT
- Load from multiple files in parallel
- Compress files before loading
- Use manifest files for consistency

2. Schema Design

- Choose appropriate data types (smallest possible)
- Use NOT NULL when possible
- Consider denormalization for performance
- Plan distribution and sort keys

3. Maintenance

- Schedule VACUUM operations
- Run ANALYZE after significant data changes
- Monitor cluster health with CloudWatch
- Set up alerting for failures

4. Security

- Use IAM roles instead of access keys
- Enable encryption at rest
- Implement least privilege access
- Regular security audits

Monitoring Checklist

Daily:

- Query performance alerts
- Failed query notifications
- Cluster health status

Weekly:

- Storage utilization trends
- Query pattern analysis
- Cost optimization opportunities

Monthly:

- Capacity planning review
- Security audit
- Performance optimization assessment

Advanced Concepts for Deep Understanding

Query Compilation and Execution

Redshift Query Processing:

1. **Parse**: SQL syntax validation
2. **Rewrite**: Query optimization rules
3. **Plan**: Generate execution plan
4. **Compile**: Convert to machine code
5. **Execute**: Run across compute nodes

Query Plan Analysis:

```sql
```

```sql
-- Get query execution plan
EXPLAIN SELECT region, SUM(amount) FROM sales_data GROUP BY region;

-- Analyze actual query performance
SELECT * FROM stl_explain WHERE query = [query_id];
```

Workload Management (WLM)

Purpose: Control resource allocation among queries

Queue Configuration:

```sql
-- Create custom WLM configuration
{
  "query_concurrency": 5,
  "max_execution_time": 3600,
  "memory_percent_to_use": 80,
  "query_group": "analytics"
}
```

Queue Types:

- **Superuser queue**: Administrative queries
- **User queues**: Regular workloads
- **Short query acceleration**: Fast queries bypass queues

Advanced Spectrum Features

1. Server-Side Filtering

```sql
-- Predicate pushdown example
SELECT * FROM spectrum_demo.web_clicks
WHERE timestamp >= '2024-01-01'  -- Filtered at S3 level
  AND session_duration > 100;    -- Further filtered in Redshift
```

2. Partition Projection

```sql
```

```sql
-- Dynamic partitioning
CREATE EXTERNAL TABLE spectrum_demo.logs_partitioned (
  event_time TIMESTAMP,
  user_id INT,
  action VARCHAR(50)
)
PARTITIONED BY (
  year INT,
  month INT,
  day INT
)
STORED AS PARQUET
LOCATION 's3://bucket/logs/year=${year}/month=${month}/day=${day}/';
```

Presentation Tips

Demo Flow Narrative

Opening (30 seconds): "Today we'll see how modern organizations handle both hot operational data and cold historical data using Amazon Redshift's hybrid architecture."

Hot Data Demo (2 minutes): "First, let's analyze our recent sales data stored in Redshift. Notice how quickly we can aggregate millions of transactions across multiple dimensions."

Cold Data Transition (1 minute): "Now, what if we want to understand customer behavior from our web logs? Traditionally, we'd need to load terabytes of clickstream data into our warehouse. Instead, we'll query it directly from S3."

Spectrum Demo (2 minutes): "Using Redshift Spectrum, we can analyze historical behavioral data without the cost and complexity of loading it into our warehouse."

Hybrid Power (3 minutes): "The real magic happens when we combine both datasets. We can now understand the complete customer journey from web interaction to purchase."

Closing (1 minute): "This architecture gives us the performance we need for operational reporting while maintaining cost-effective access to all our historical data."

Handling Questions

Performance Questions:

- Have query execution times ready

- Show CloudWatch metrics if available
- Compare with traditional approaches

Cost Questions:

- Prepare realistic cost scenarios
- Show cost calculators
- Discuss optimization strategies

Technical Questions:

- Be ready to explain architecture diagrams
- Discuss alternative approaches
- Know competitor comparisons

Visual Aids

Architecture Diagram:

```
[Applications] → [Redshift Cluster] ←→ [S3 Data Lake]
                  ↓              ↓
             [Hot Data Queries]   [Spectrum Queries]
                  ↓              ↓
             [Fast Results]     [Cost-Effective Analysis]
```

Performance Comparison Chart:

- Query response times: Hot vs Cold data
- Cost comparison: All-in-warehouse vs Hybrid
- Storage growth: Linear vs Optimized

This comprehensive set of notes should give you deep understanding to confidently demonstrate and explain Amazon Redshift, Redshift Spectrum, and modern data warehousing concepts to any technical audience.