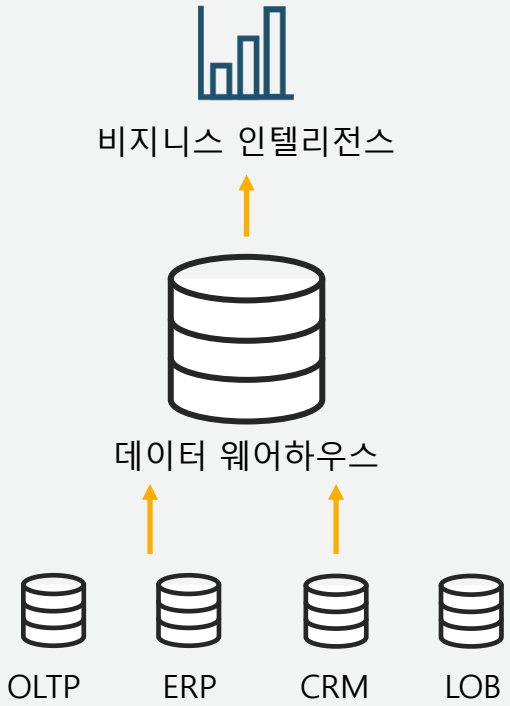
Data Warehouse

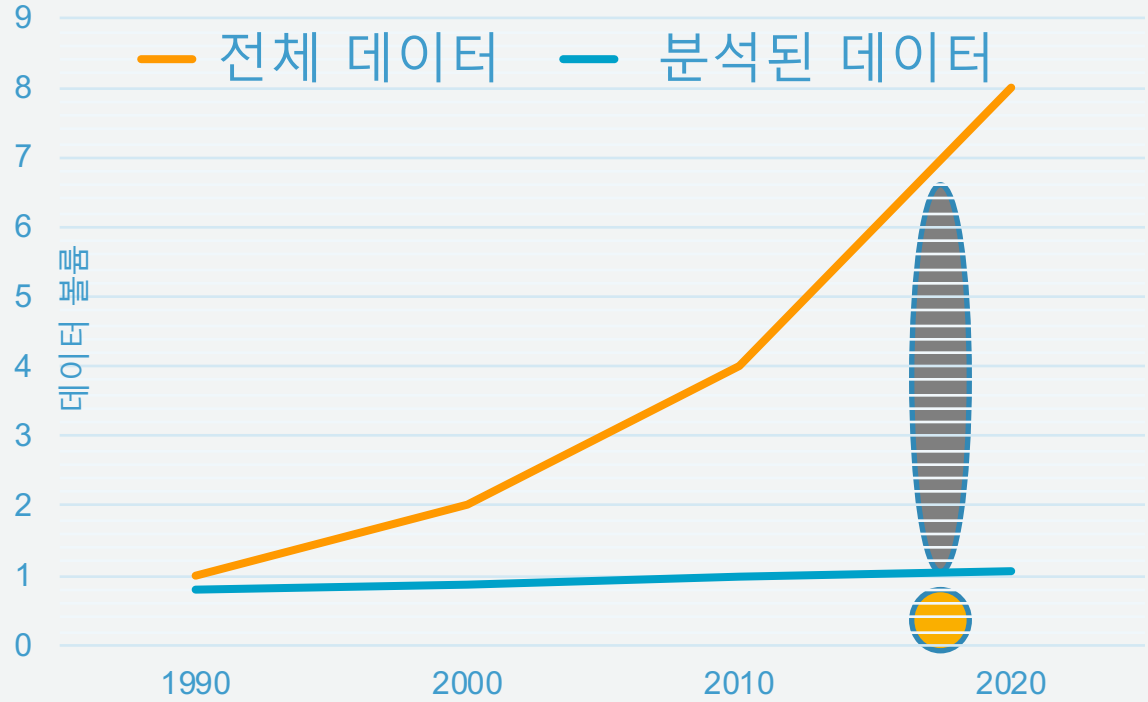
Yoojeong Choi, Database SA, AWS

전통적인 데이터 Analytic 접근 방법



- 정형 데이터 형식
- 수TB ~ 수PB 바이트 크기
- 데이터 로딩 전 스키마 지정
- 지정된 인원이 한정된 자원을 기반으로 활용
- 높은 초기 투자 비용 및 유지보수 비용

초기 아키텍처로 인한 Dark Data 발생



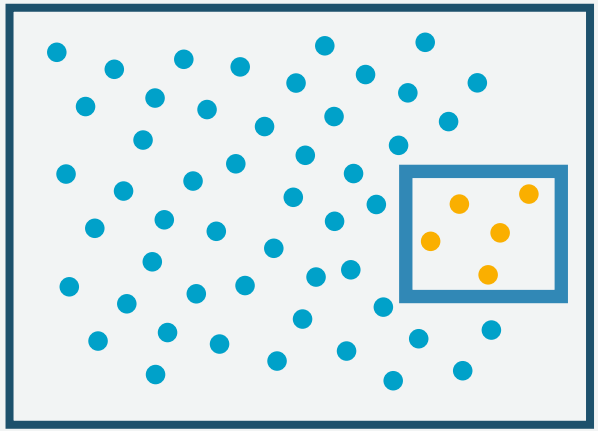
많은 데이터들이 비지니스에 활용되지 못하고 있습니다.

Sources:
Gartner: User Survey Analysis: Key Trends Shaping the Future of Data Center Infrastructure Through 2011
IDC: Worldwide Business Analytics Software 2012–2016 Forecast and 2011 Vendor Shares

페러다임의 전환

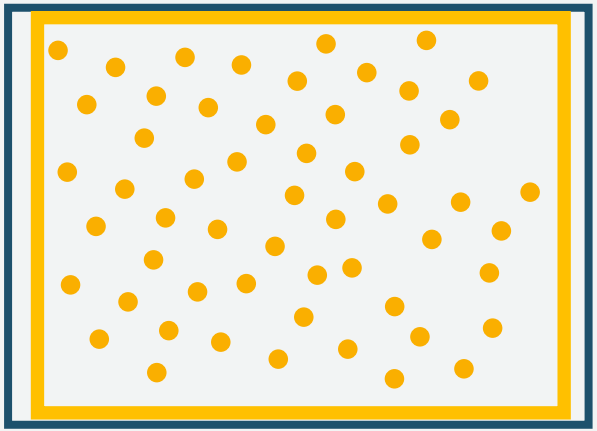
기업의 모든 데이터를 기반으로한 비니지스 인사이트 도출이 필요

전통적인 데이터 웨어하우스



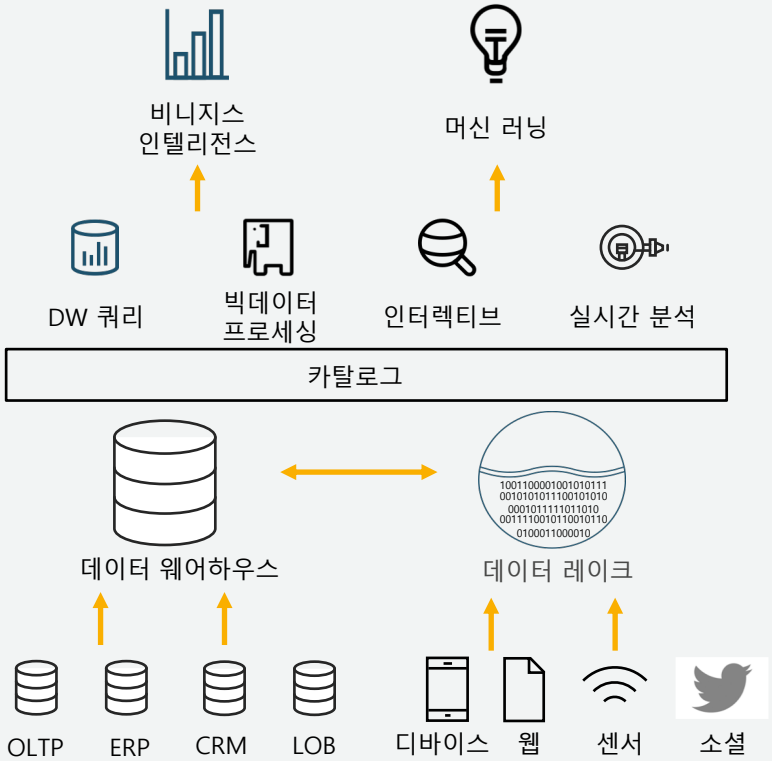
데이터 웨어하우스에
저장된
데이터만 분석

모던 데이터 웨어하우스



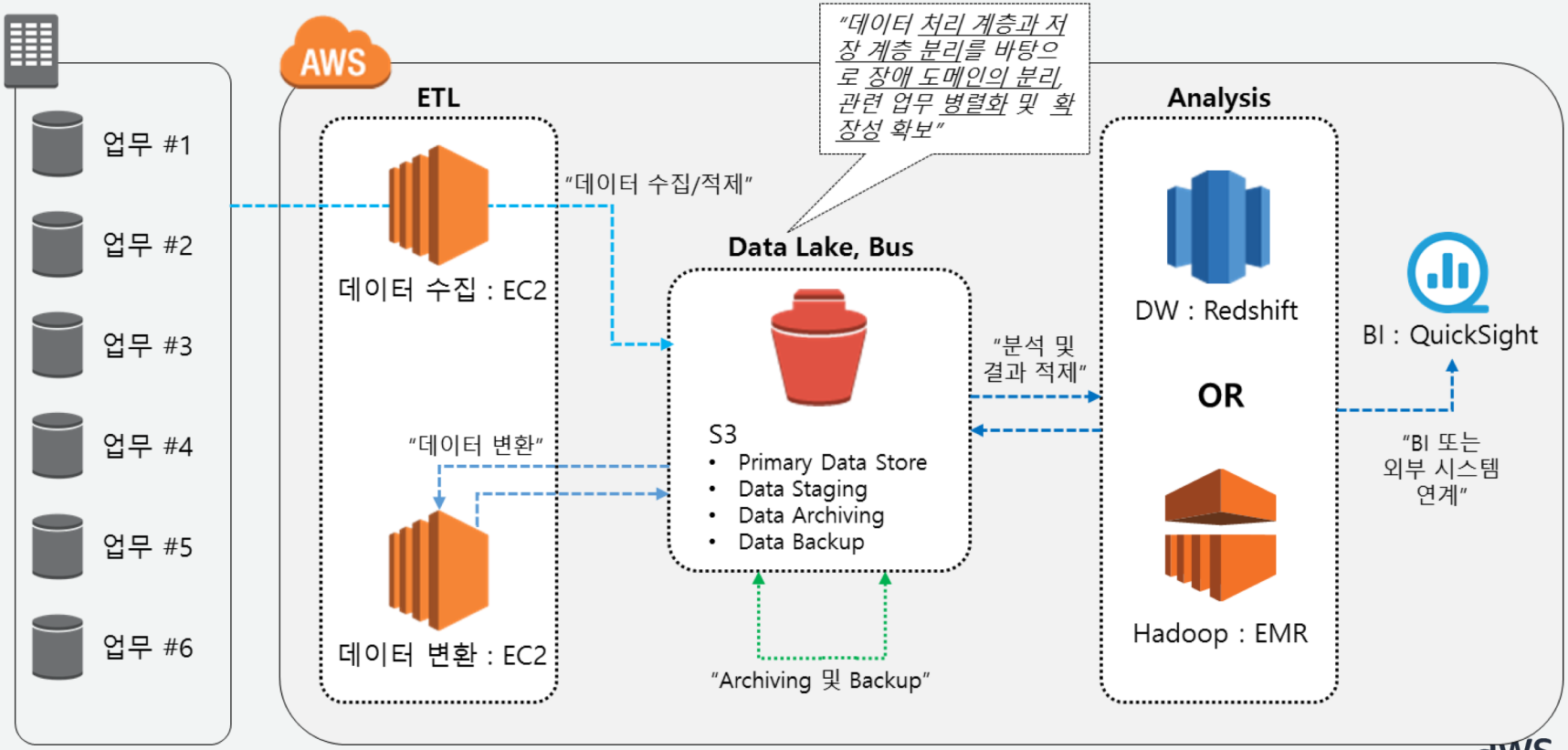
데이터레이크를 기반으로
모든 데이터에 대한 분석

데이터 레이크를 기반한 데이터 분석 역량의 강화



- 정형, 비정형 데이터 분석
- 수TB ~ 수EB 바이트 크기
- 분석 시 스키마 지정
- 다양한 분석 도구
- 비용 최적화

만약 빅데이터 분석을 처음 시작하신다면...



Amazon Redshift

1/10 비용으로 빠르고, 손쉬운 확장을 지원하는 완전 관리형 DW

빠른 성능

Pinterest "Hive 대비 50~100배 빠른 성능"

amazon "210억 행과 100억행 조인에 2시간"

비용 효율적

"시간당 $0.25로 시작"

"년간 미압축 데이터 기준 1TB에 $250~$333"

보안

"ㄷ 에

"ㅈ 지

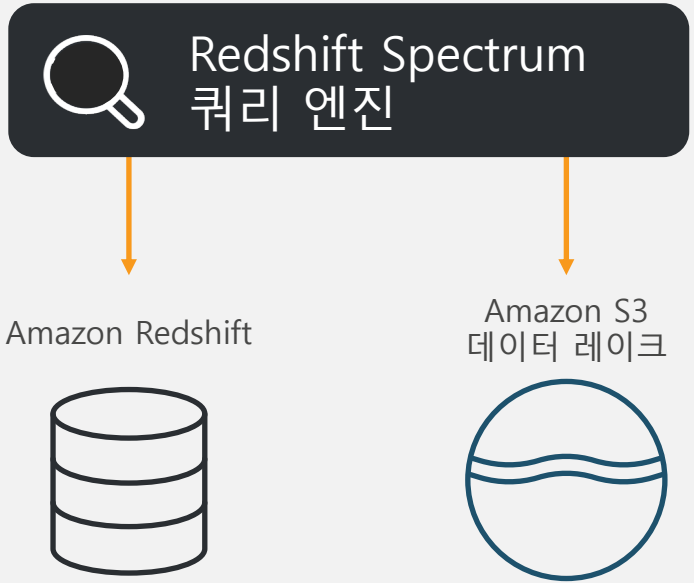
데이터 레이크와의 통합

"S3 (데이터레이크) 대한 쿼리"

"AWS 다양한 서비스와의 유연한 통합"

aws

데이터레이크와의 통합



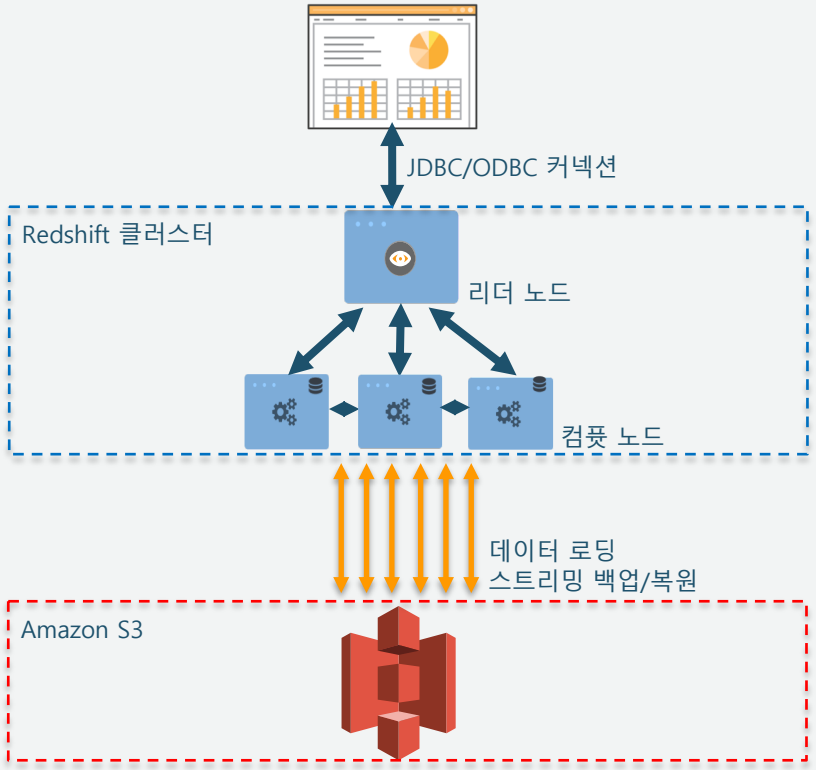
- Amazon Redshift에서 S3데이터에 대한 쿼리
- Compute과 Storage 에 대한 별도의 용량 관리
- 빠른 쿼리 성능
- 무제한 병렬화
- CSV, ORC, Gro, Avro, Parquet 지원
- On demand, 쿼리당 스켄 데이터에 기준한 비용 (TB 당 $5)

아키텍처

Amazon Redshift 클러스터 아키텍처

MPP 기반의 Shared Nothing 아키텍처
Amazon S3로의 지속적엔 데이터 백업
리더 노드
- SQL 엔드포인트
- 메타데이터 저장소
- 병렬 SQL 쿼리 코디네이터

컴퓨팅 노드
- 로컬, 컬럼 기반의 스토리지
- 병렬 쿼리 수행 주체
- 데이터 로드, 백업 및 복원 주체
- 2, 16, 32개 슬라이스



노드와 Slice

노드

~~DC1~~ - Do Not Use

DC2
- Based on EC2 i3 Instance Family
- SSD based storage
- Intel Broadwell Processors
- DC2.large
- DC2.8xlarge

DS2
- Based on EC2 d2 Instance Family
- Magnetic based storage
- Intel Haswell Processors
- DS2.xlarge
- DS2.8xlarge

슬라이스

Redshift is a distributed system:
- A compute node contains slices (one per virtual core)
- A slice contains data

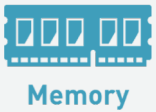
Slices are chosen based on two types of distribution:
- Round Robin (automated)
- Based on a distribution key (hash of a defined column)

Queries run on all slices in parallel: optimal query throughput can be achieved when data is evenly spread across slices

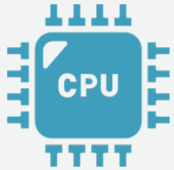
새로운 고밀도 컴퓨팅 노드 – DC2

동일 가격에 DC1 대비 3배 높아진 I/O 성능과 30% 개선된 컴퓨팅 성능

NVMe SSD



DDR4 Memory



Intel E5-2686 v4 (*Broadwell*)

Liberty Mutual INSURANCE

*"Amazon Redshift DC1에서 DC2로 이전한 후 월간 레포트 작성이 **9배**가 빨라 졌습니다."*

- Bradley Todd,
Technical Architect, Liberty Mutual

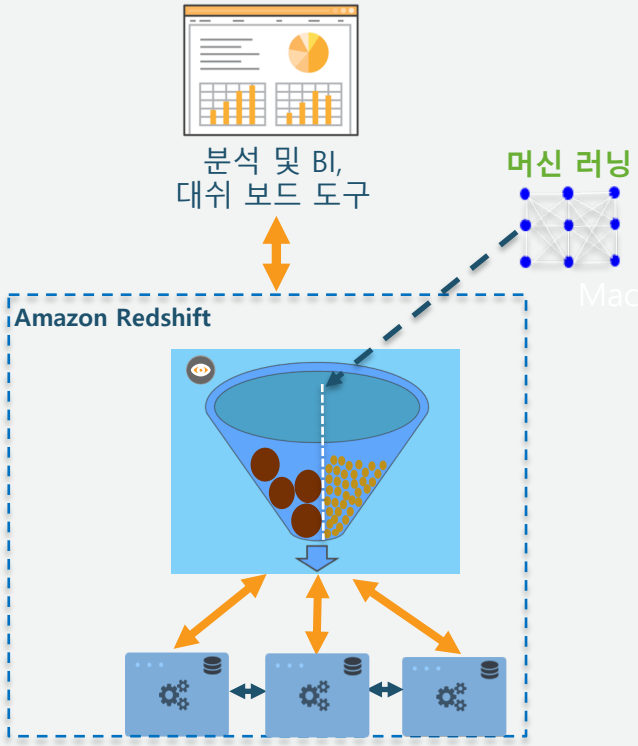
결과 캐싱 - 일초 미만의 쿼리 응답 속도

쿼리 실행 시간을 단축을 위한 쿼리 결과 메모리 케싱

- 기본 활성화 (관련 파라메터 enable_result_cache_for_session)
- 캐시 조건
  - 쿼리에 사용되는 객체에 대한 액세스 권한 보유
  - Redshift 쿼리 관련 파라메터 및 조회하는 테이블내 데이터의 미변경
  - 쿼리 실행 시마다 실행해야 하는 함수 (예: GETDATE) 미사용
- 리더 노드 사이즈 및 쿼리 결과 크기를 기준으로 캐싱
- 세션에 관계 없이 캐쉬 조회
- SVL_QLOG 뷰의 "source_query"항목을 통해 확인

단기 쿼리 가속화 Short Query Acceleration (SQA)

장기 쿼리로 인한 체감 성능 저하 완화 및 동시성 확보



- 머신 러닝 알고리즘을 기반으로 단기 쿼리 판단
- 장기 쿼리뒤에서 단기 쿼리가 대기하지 않도록 SQA 전용 큐에서 단기 쿼리 실행
- SQA 조건
  - GUI에서는 WLM의 총 슬롯 수가 15개 이하여야 함 (CLI, API를 통해 SQA를 활성화 한 경우 해당 없음)
  - CTAS 및 읽기 전용 쿼리에 사용
  - 기본값은 5초 (권장), 최대 20초까지 설정
- 'STL_WLM_Query'의 'service_class' 컬럼값이 14인 쿼리가 SQA를 통해 Acceleration된 쿼리 문이며, 'final_state'를 통해 'Completed' 또는 'Evicted' 여부를 확인할 수 있음

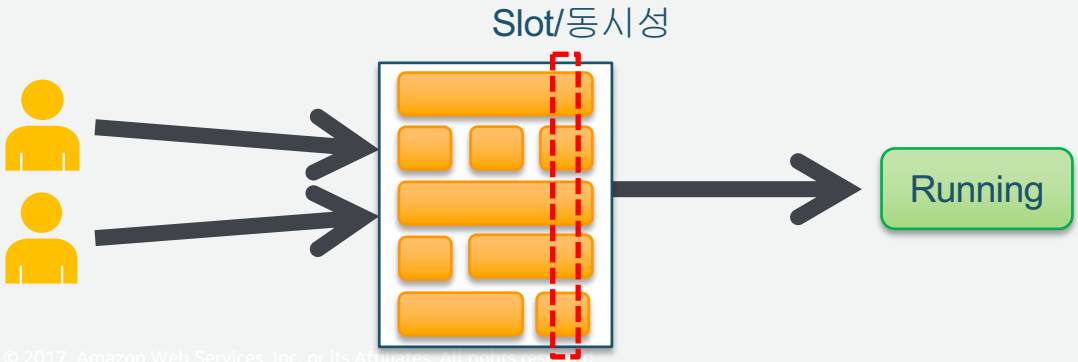
Workload Management (WLM)

Concurrent Query Execution

Long-running queries may cause short-running queries to wait in a queue
As a result users may experience unpredictable performance

By default, a Redshift cluster is configured with one queue with 5 execution slots by default (50 max, 동시 연결 수량은 500)

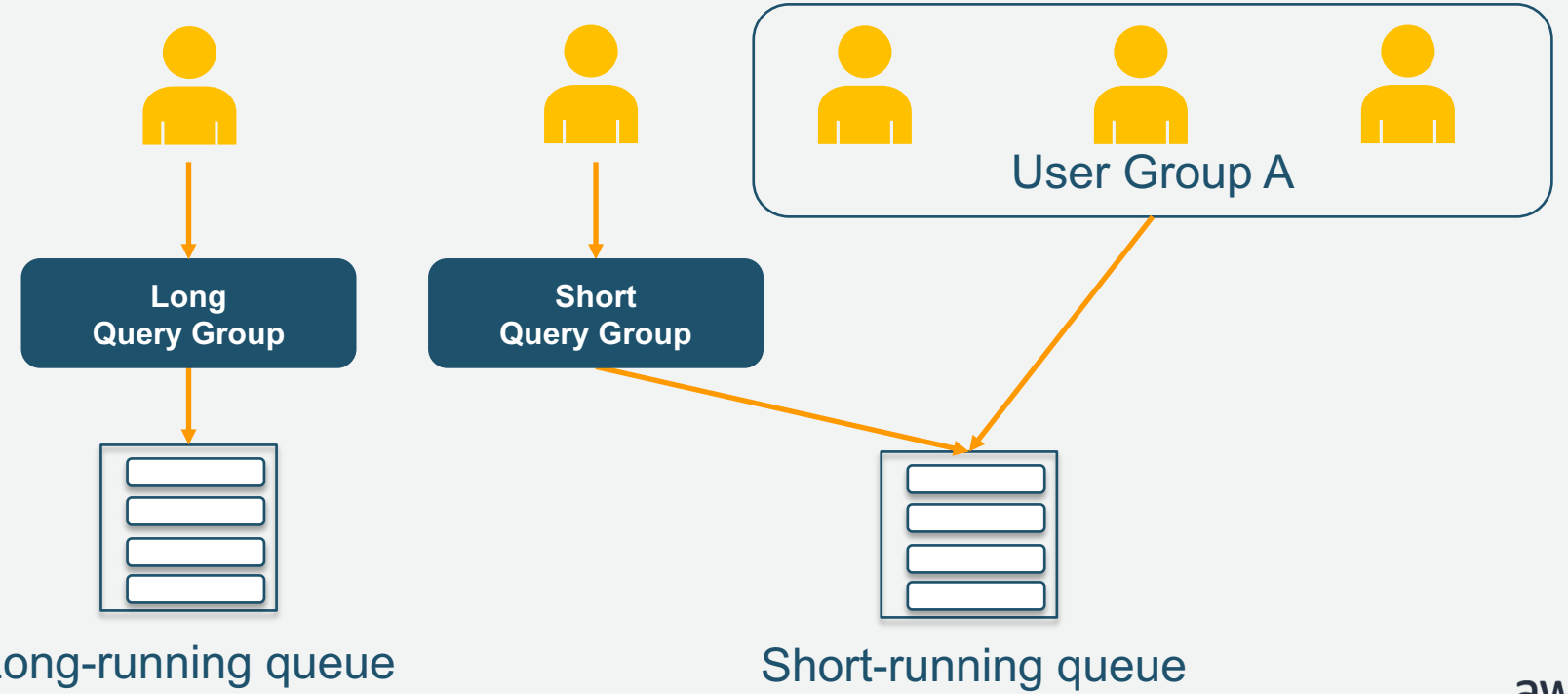
전체 가용 메모리는 설정된 쿼리 Slot/동시성 를 기준으로 전체 메모리를 배분



"원치 않는 페이징으로 인한 성능 저하"

Workload Management (WLM)

실행 시간 및 비지니스 목적성에 따라 쿼리를 적합한 대기열로 라우팅



Workload Management (WLM)

- 결과 캐싱 및 단기 쿼리 가속화는 WLM을 거치지 않음으로 성능 뿐 아니라 병렬성을 높히는데 큰 영향을 줌
- 병렬성을 높히면 그 만큼 단위 쿼리의 리소스가 줄어듬
- **메모리 부족 여부 확인**
  'SVL_QUERY_SUMMARY' 또는 'v_my_last_query_summary'의 'is_diskbased' 컬럼 확인

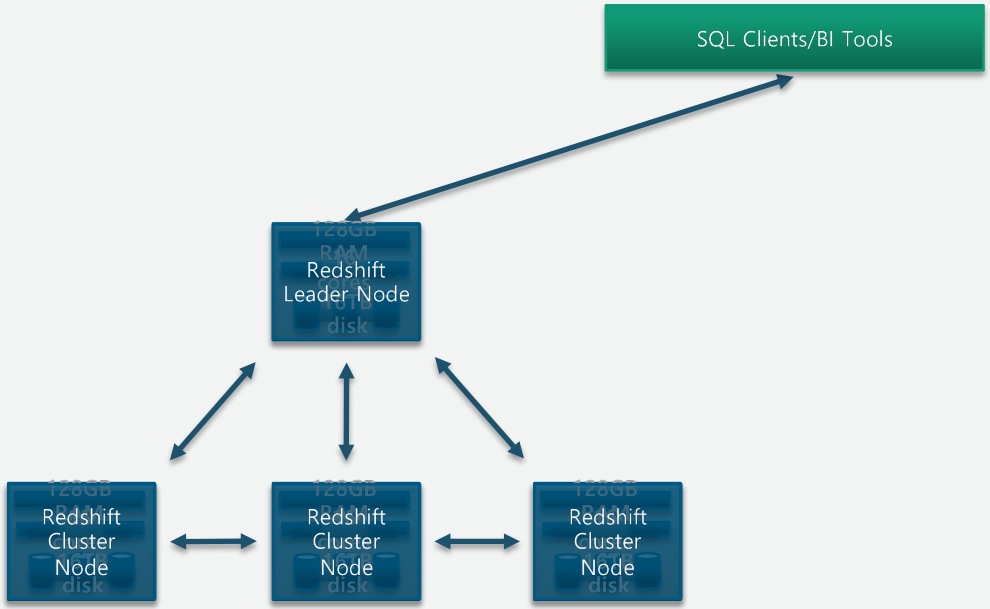
노드 선택의 기준

- IO Latency 요구 사항 및 필요 스토리지 용량
- 쿼리의 수행 패턴 (Slice Level, Node Level)
  - Slice Level vs Node Level
  - Broadcast
- 결과 캐싱 사용 여부
- 리더 노드 전용 함수 및 최종 Sorting, Aggregation (집계) 의 사용
- 최신 세대의 노드

리더 노드 전용 함수

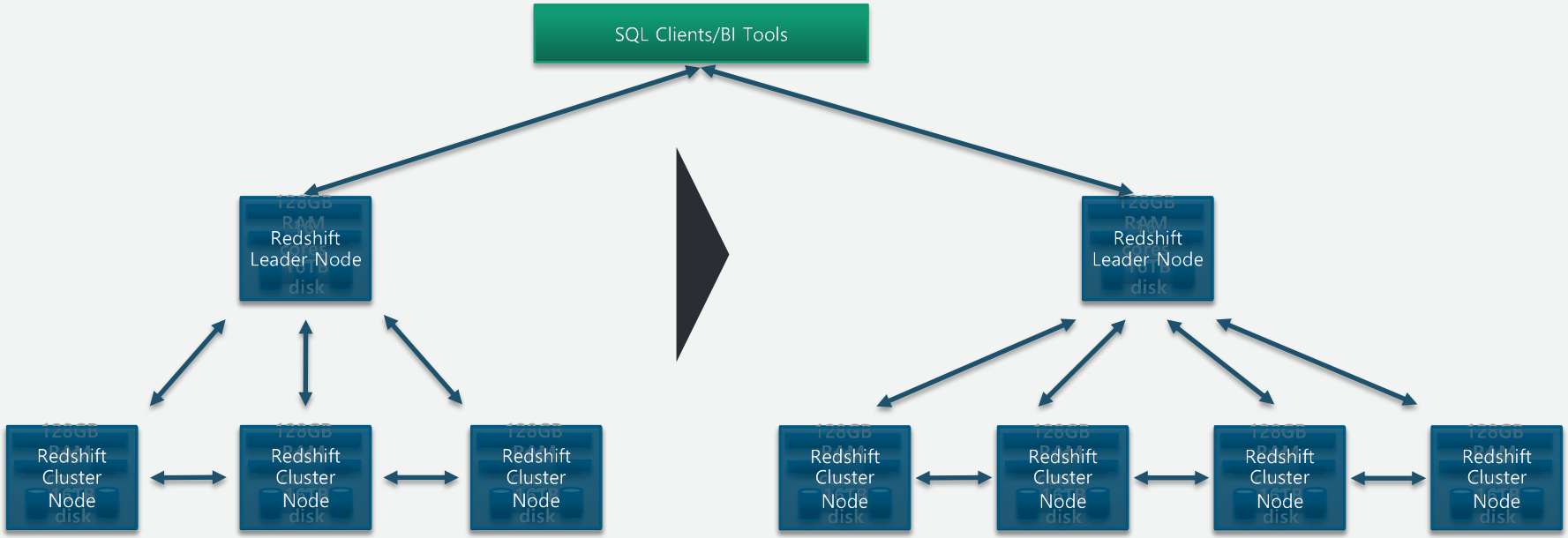
날짜 함수	문자열 함수
• AGE	• ASCII
• CURRENT_TIME	• GET_BIT
• CURRENT_TIMESTAMP	• GET_BYTE
• LOCALTIME	• SET_BIT
• ISFINITE	• SET_BYTE
• NOW	• TO_ASCII

클러스터 및 노드 사이즈 변경



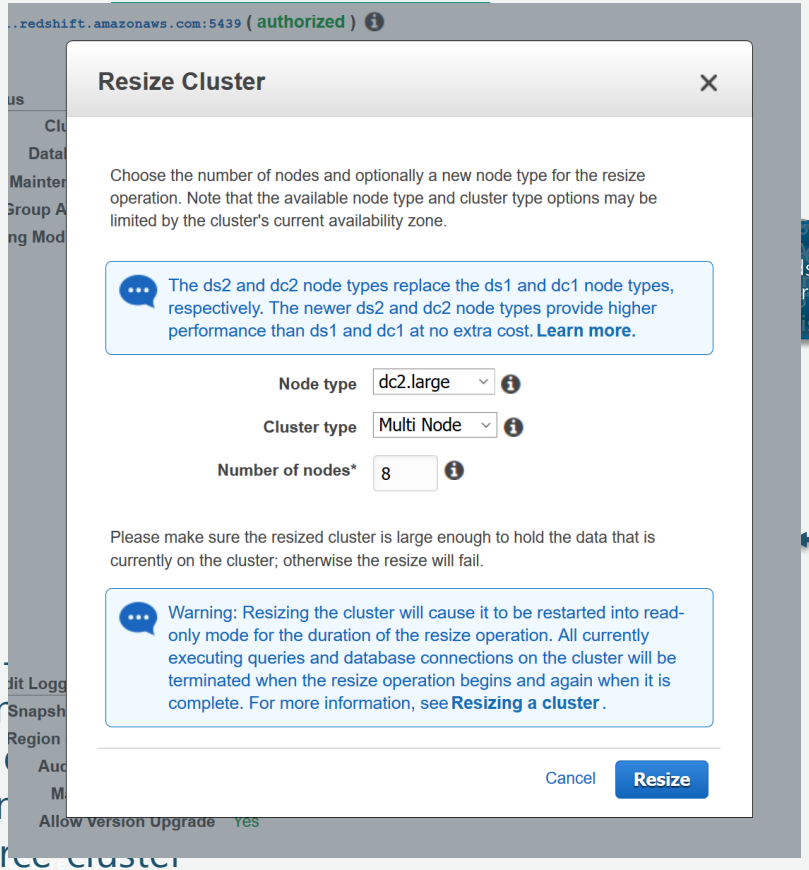
1. Cluster is put into read-only mode
2. New cluster is provisioned according to resizing needs
3. Node-to-node parallel data copy
4. Automatic SQL endpoint switchover via DNS
5. Decommission the source cluster

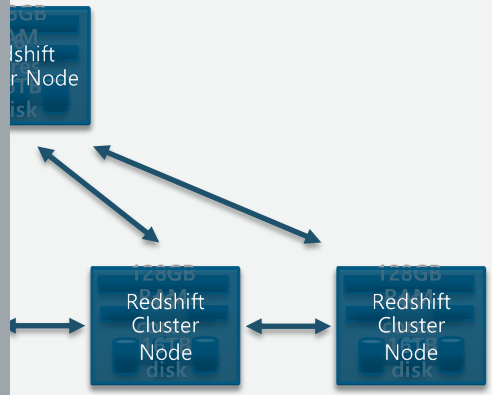
사이즈 변경



1. Cluster is put into read-only mode
2. New cluster is provisioned according to resizing needs
3. Node-to-node parallel data copy
4. Automatic SQL endpoint switchover via DNS
5. Decommission the source cluster

사이즈 변경





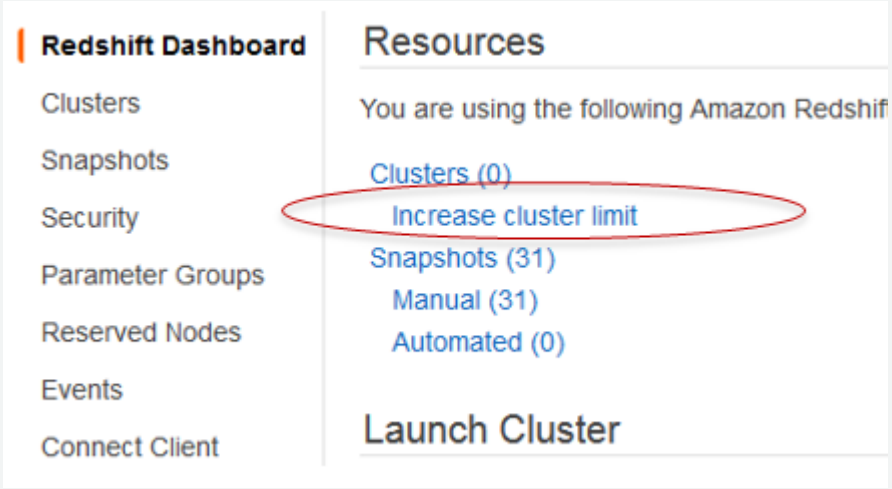
1. Cluster is put into read-
2. New cluster is provisior
3. Node-to-node parallel
4. Automatic SQL endpoir
5. Decommission the sour

제한

Amazon Redshift Limits

Resource	Default Limit
Nodes per cluster	101
Nodes	200
Reserved Nodes	200
Snapshots	20
Parameter Groups	20
Security Groups	20
Subnet Groups	20
Subnets per Subnet Group	20
Event Subscriptions	20

Limit Increase 요청

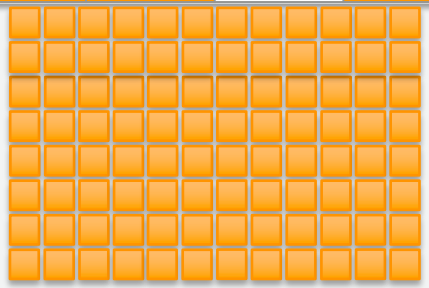


데이터 및 스키마 모델링

Row 기반 저장소와 Column 기반 저장소

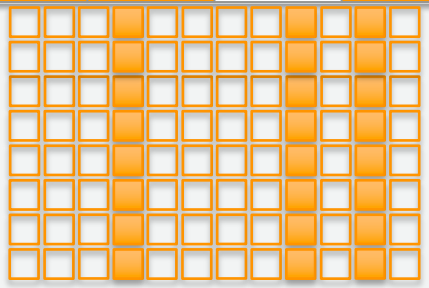
Select ID, Age, Amount from mytable

ID	Age	State	Amount
123	20	CA	500
345	25	WA	250
678	40	FL	125
957	37	WA	375



Row 기반 저장소

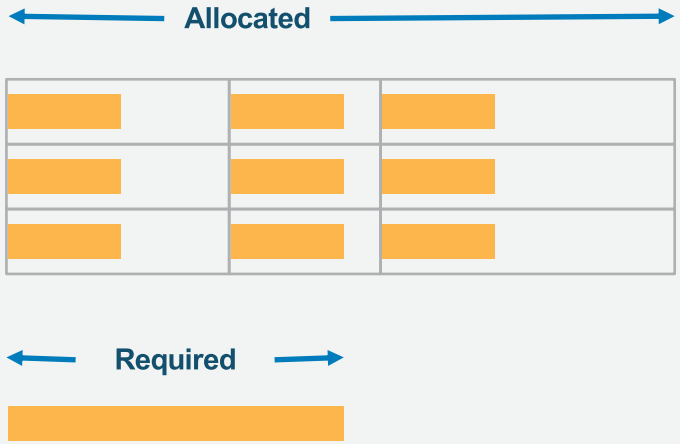
ID	Age	State	Amount
123	20	CA	500
345	25	WA	250
678	40	FL	125
957	37	WA	375



Column 기반 저장소

Column 기반의 스토리지

Column간의 간격 최소화를 통한 성능 향상



- During queries and ingestion, the system allocates buffers based on declared column width
- **Wider than needed columns mean memory is wasted**
- Fewer rows fit into memory; increased likelihood of queries spilling to disk
- SVV_TABLE_INFO 테이블의 'max_varchar'을 통해 가장 큰 열을 확인

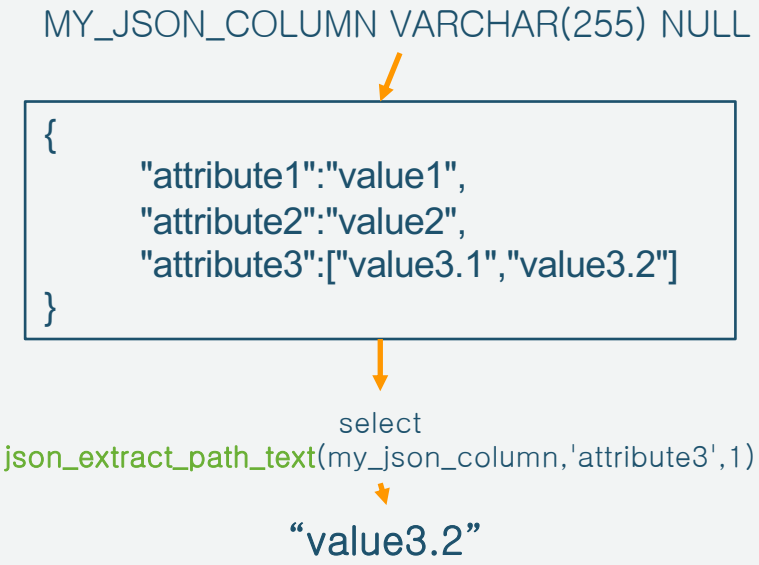
지원 자료형

Data Type	Aliases	Description
SMALLINT	INT2	Signed two-byte integer
INTEGER	INT, INT4	Signed four-byte integer
BIGINT	INT8	Signed eight-byte integer
DECIMAL	NUMERIC	Exact numeric of selectable precision
REAL	FLOAT4	Single precision floating-point number
DOUBLE PRECISION	FLOAT8, FLOAT	Double precision floating-point number
BOOLEAN	BOOL	Logical Boolean (true/false)
CHAR	CHARACTER, NCHAR, BPCHAR	Fixed-length character string
VARCHAR	CHARACTER VARYING, NVARCHAR, TEXT	Variable-length character string with a user-defined limit
DATE		Calendar date (year, month, day)
TIMESTAMP	TIMESTAMP WITHOUT TIME ZONE	Date and time (without time zone)
TIMESTAMPTZ	TIMESTAMP WITH TIME ZONE	Date and Time (with time zone)

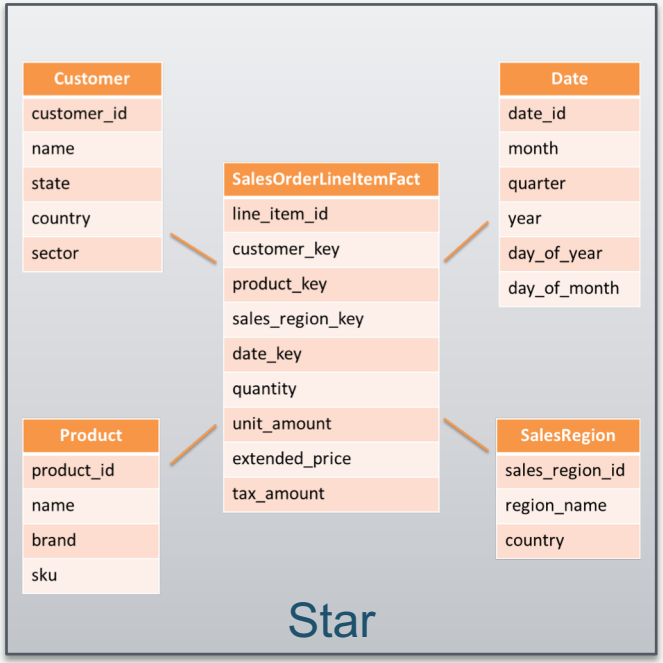
JSON 지원

- Redshift tables can include arbitrary string data stored as JSON
- JSON Object or Array

- DB Functions for working with JSON
  - IS_VALID_JSON
  - IS_VALID_JSON_ARRAY
  - JSON_ARRAY_LENGTH
  - JSON_EXTRACT_ARRAY_ELEMENT_TEXT
  - JSON_EXTRACT_PATH_TEXT



스키마 디자인



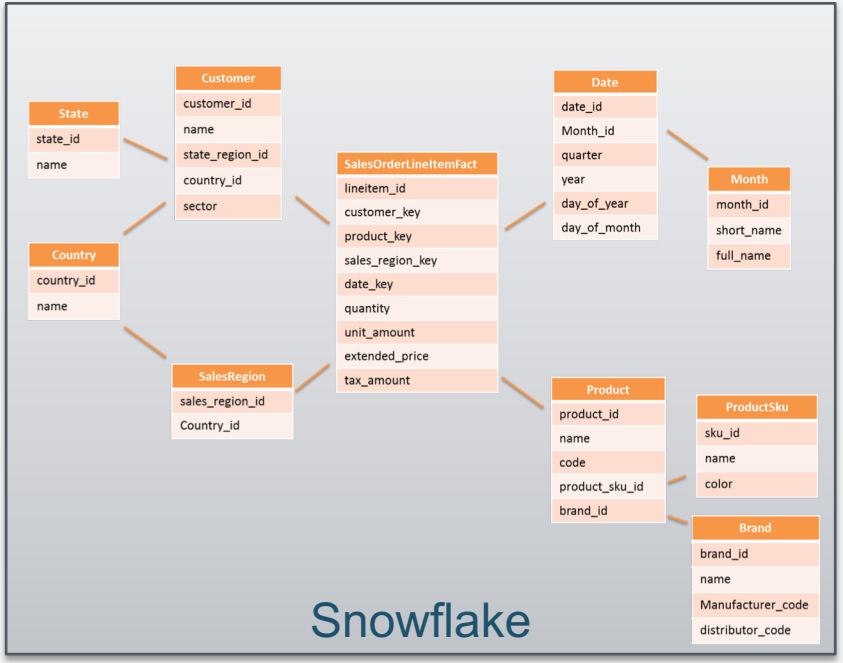
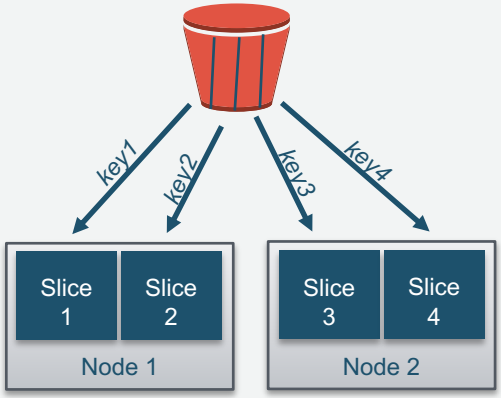


Table Distribution Styles

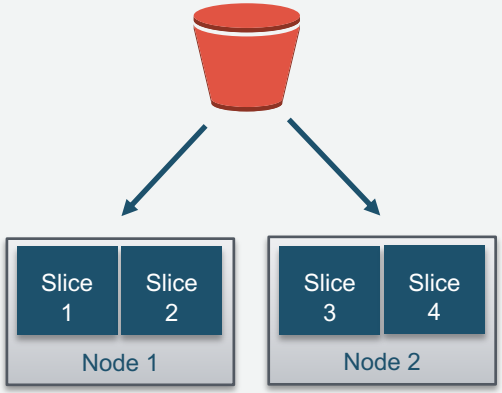
Distribution Key

Same key to same location



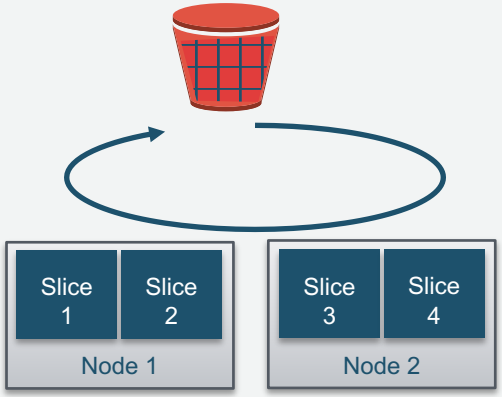
All

All data on every node



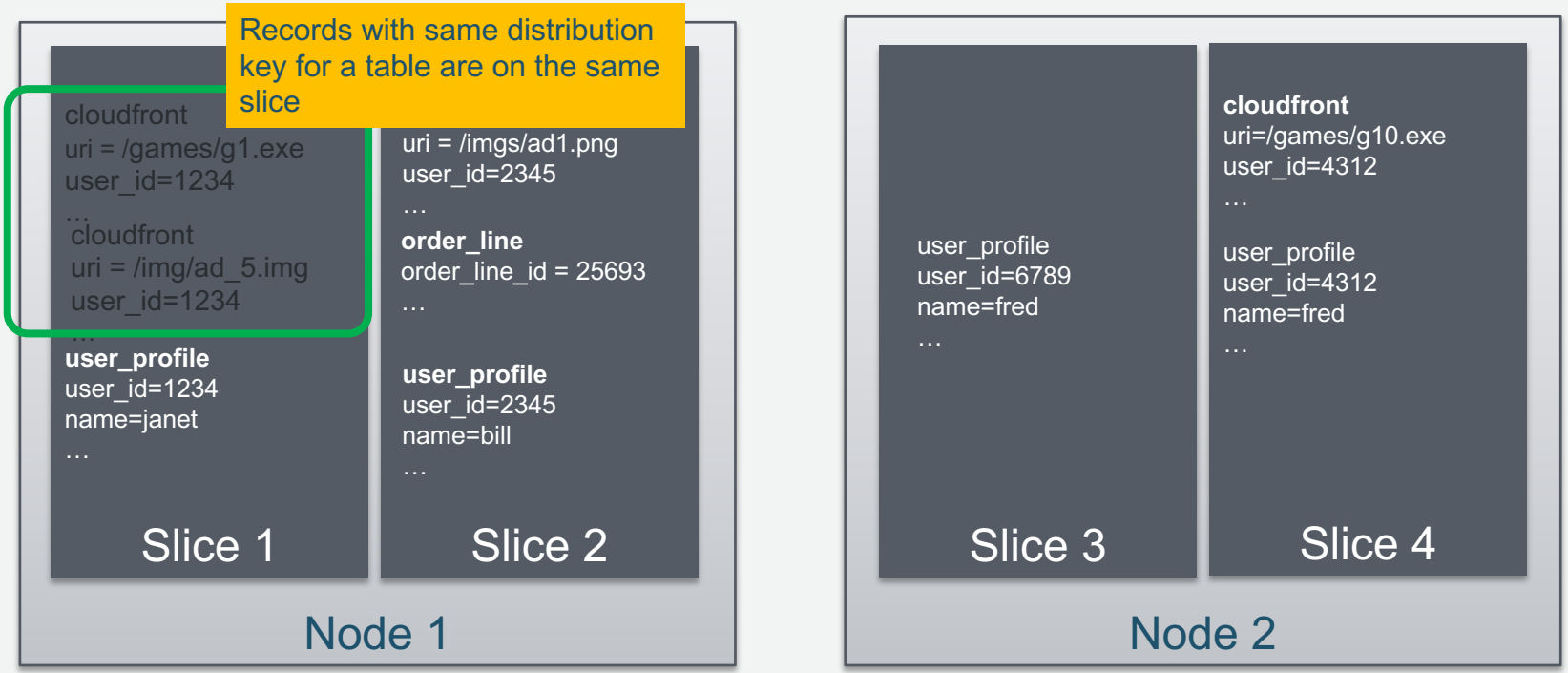
Even

Round robin distribution



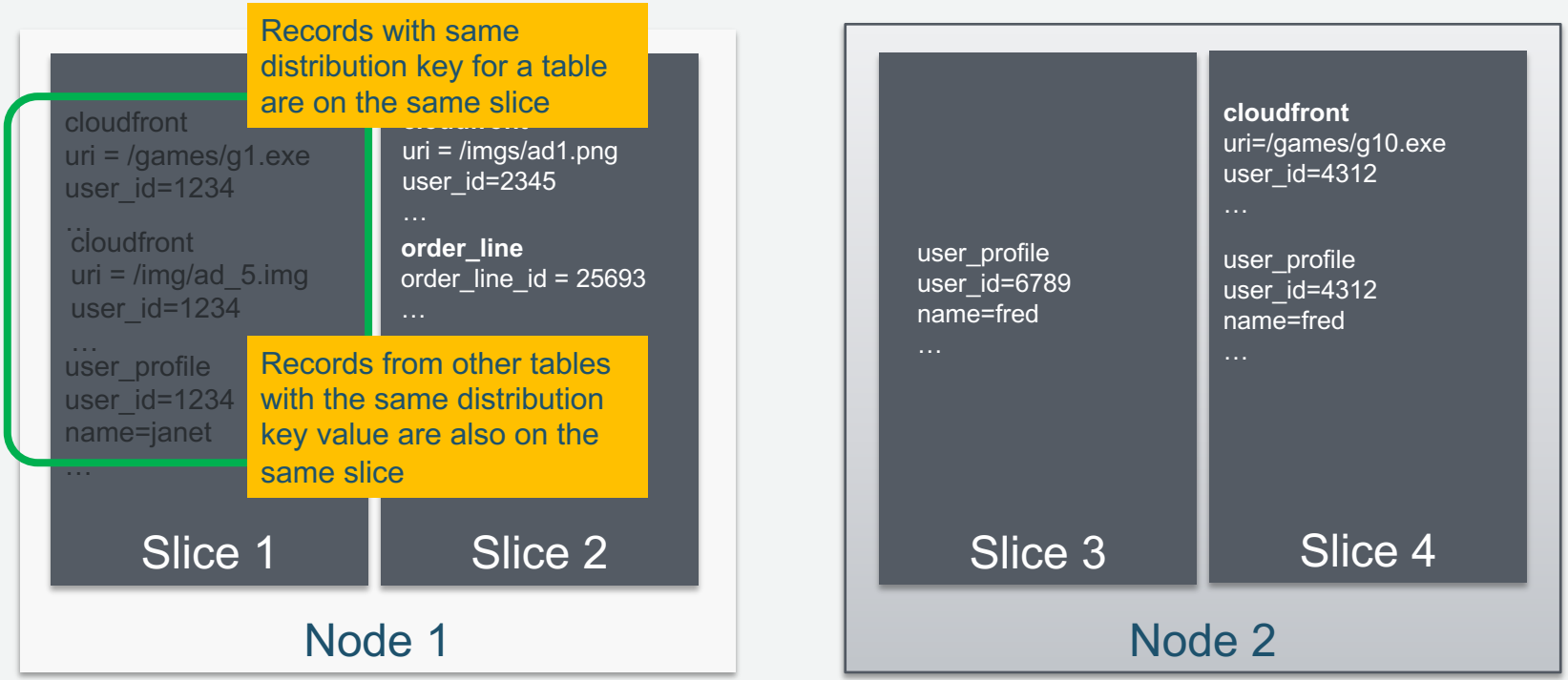
Data Distribution with Distribution Keys

Data Distribution and Distribution Keys



Distribution Keys determine which data resides on which slices

Data Distribution and Distribution Keys



Distribution Keys help with data locality for join evaluation

Data Distribution - Comparison

적절한 분산 처리

```
XN Merge  (cost=1000634197251.69..1000634197251.69 rows=1 width=33)
  Merge Key: lineitem.l_shipmode
  ->  XN Network  (cost=1000634197251.69..1000634197251.69 rows=1 width=33)
        Send to leader
        ->  XN Sort  (cost=1000634197251.69..1000634197251.69 rows=1 width=33)
              Sort Key: lineitem.l_shipmode
              ->  XN HashAggregate  (cost=634197251.66..634197251.67 rows=1 width=33)
                    ->  XN Hash Join DS_DIST_NONE  (cost=153674846.86..633973363.89 r
```

DS_DIST_NONE, DS_DIST_ALL_NONE

Broadcast 발생

```
XN Merge  (cost=9055884343615.08..9055884343615.09 rows=1 width=33)
  Merge Key: lineitemnodist.l_shipmode
  ->  XN Network  (cost=9055884343615.08..9055884343615.09 rows=1 width=33)
        Send to leader
        ->  XN Sort  (cost=9055884343615.08..9055884343615.09 rows=1 width=33)
              Sort Key: lineitemnodist.l_shipmode
              ->  XN HashAggregate  (cost=8055884343615.06..8055884343615.08 rows=1 width=33)
                    ->  XN Hash Join DS_BCAST_INNER  (cost=153663142.28..8055884154841.01 rows=25
```

DS_BCAST_INNER, DS_DIST_BOTH

Network Broadcast

During join steps, it may be necessary for a slice to work with data that is not stored locally

Network transmission is by far the most expensive operation a query will do

The DS_DIST* information in an explain plan tells you how data is being accessed between tables and slices

DS_DIST*	INFO
DS_DIST_NONE	Preferred, no data transfer between nodes.
DS_DIST_ALL_NONE	One table has DISTSTYLE ALL, no data transfer between nodes.
DS_DIST_ALL_INNER	Inner table is being sent to to Single Node. Very Slow.
DS_DIST_INNER	Inner table is being redistributed in an outer join.
DS_DIST_OUTER	Outer table is being redistributed in an outer join.
DS_BCAST_INNER	Inner table is being broadcast to all nodes.
DS_BCAST_BOTH	Both tables are being broadcast to all nodes.

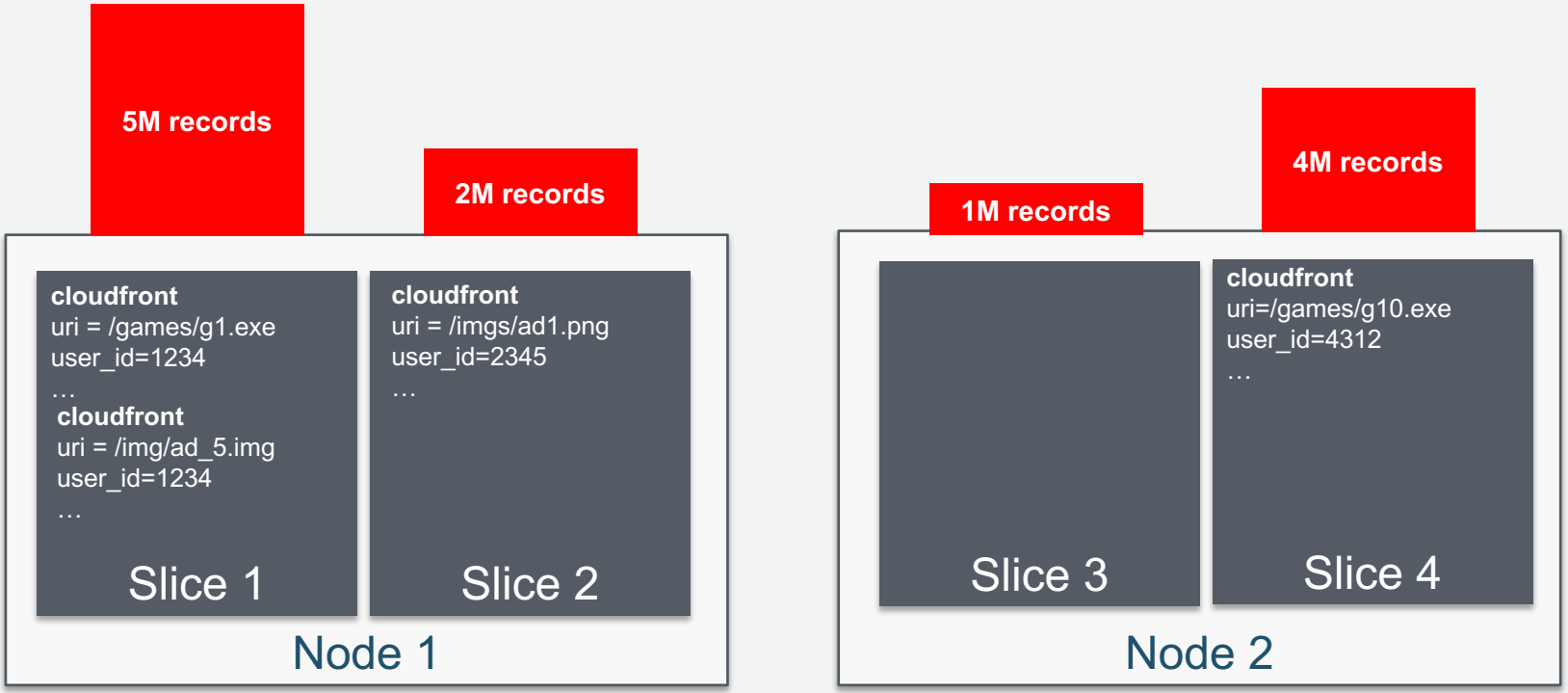
Distribution Style의 선택

- **KEY**
  - 스타 스키마의 Fact Table과 같이 큰 테이블
  - 조인 또는 집계의 대상이 되는 큰 테이블
  - 변경이 잦은 테이블
  - KEY 값을 Join에서 사용하지 않더라도 성능 향상 효과가 있음

- **ALL**
  - 변경이 거의 없는 테이블
  - 사이즈가 작은 테이블 (몇억껀 수준 미만)
  - 조인에 자주 사용되지만 특별한 Distribution Key 선정이 어려운 테이블
  - 상품의 카탈로그와 같이 작고 조인에 자주 활용되는 테이블에 대해 고려 가능

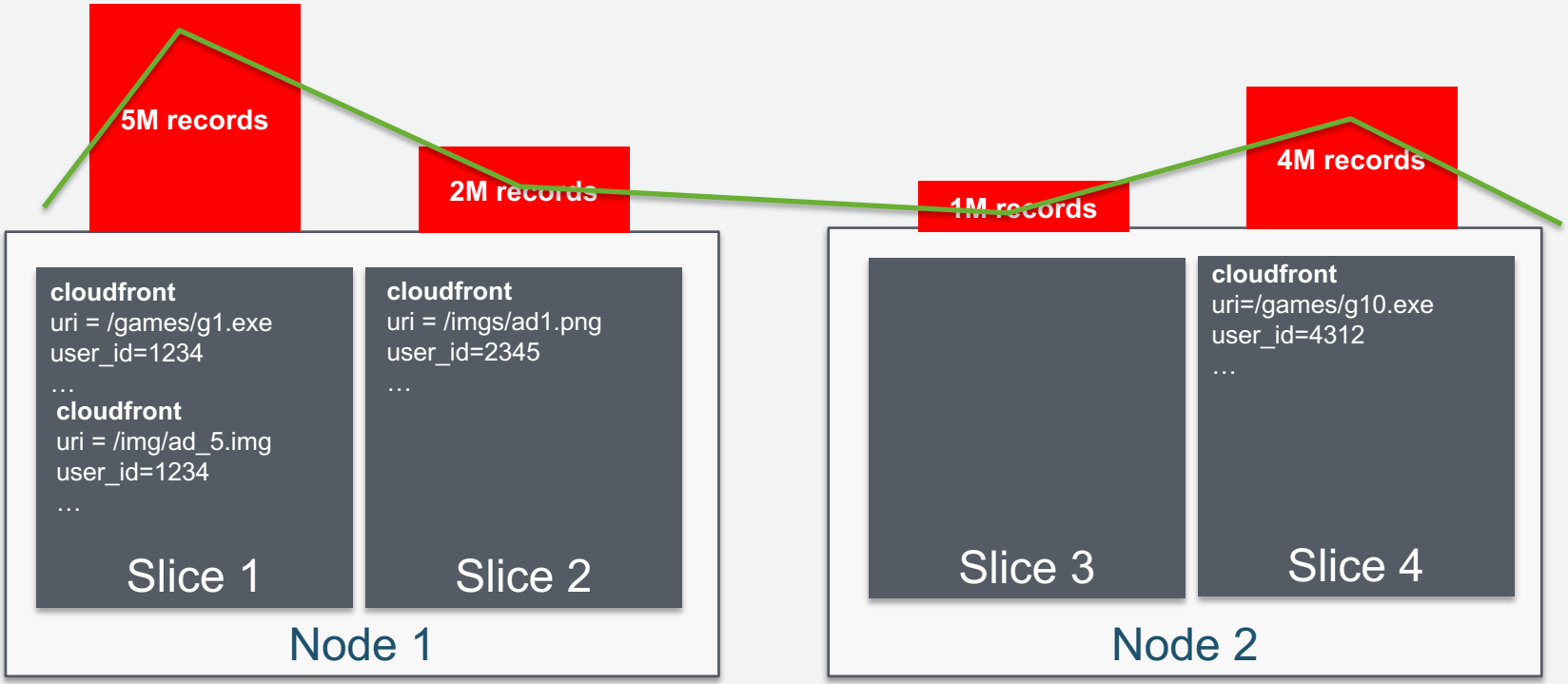
- **EVEN:** 조인 및 집계가 없는 경우

Data Distribution and Distribution Keys



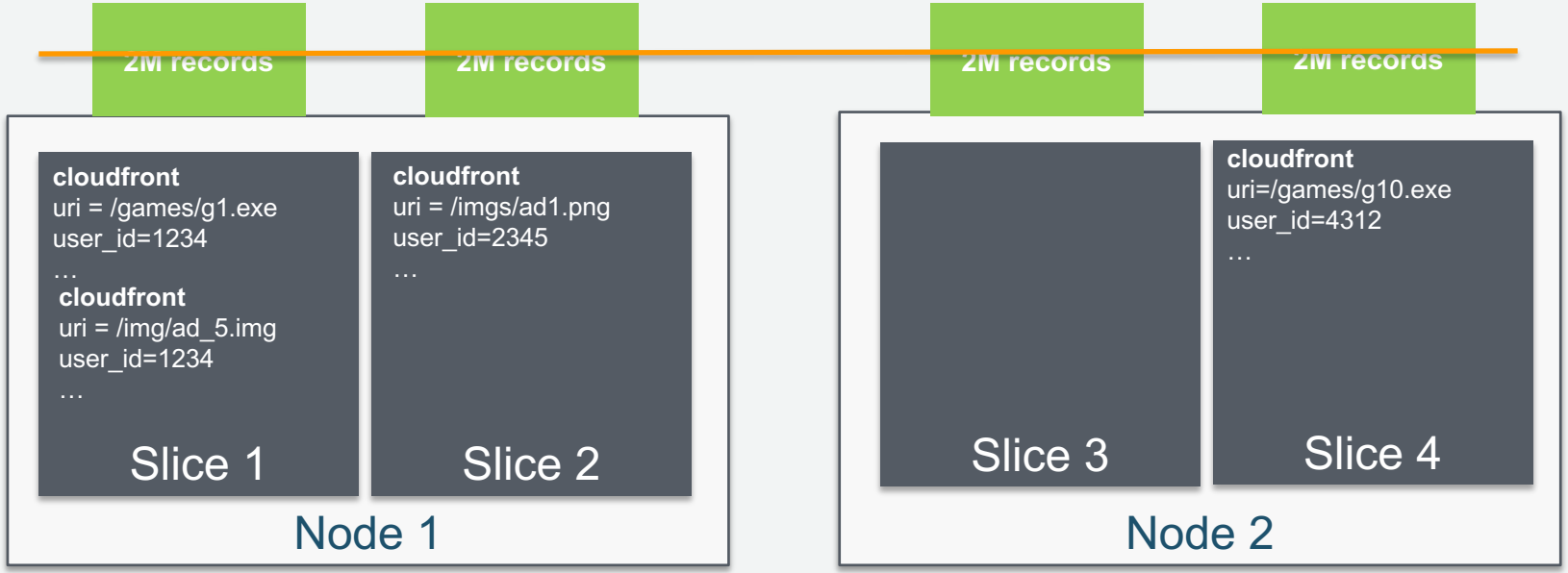
Poor key choices lead to uneven distribution of records…

Data Distribution and Distribution Keys



Unevenly distributed data cause processing imbalances!

Data Distribution and Distribution Keys



Evenly distributed data improves query performance

Choosing a Good Distribution Key

목적

- Distribute data evenly across nodes
- **Minimize data movement: Co-located Joins & Aggregates**

Best Practice

- Use the **joined columns** for largest commonly joined tables as key (example: fact table and large dimension table)
- Consider using **Group By** column as a key (GROUP BY clause)
- Never use a distribution key that causes severe data skew
- Choose a key with **high cardinality**; large number of discrete values

Avoid

- **Keys used as equality filter** as your distribution key (Concentrates processing on one node)

Dist Key 어떻게 선택할까?

1. 가장 자주 활용되는 쿼리를 확인합니다.
   예 "awslabs/amazon-redshift-utils/AdminScripts/top-queries.sql" 활용 등

2. 조인 및 Group By에 활용되며 고른 데이터 분포 및 높은 Cardinality 보유한 컬럼을 선택 합니다.

3. 그게 아니라면....

"Fact Table은 Even으로 Dimension Table은 ALL로"

"참고 : MPP 형태의 스키마는 테이블 내에
모든 데이터를 보유하기 때문에 Distribution style 은 Even으로"

"3정규화 형태의 Inmon 모델은 노드 수량은 줄이고
노드 크기는 키우거나 Star Schema로 전환 "

참고,

1. 가장 큰 Dimension Table의 Primary키와 Fact Table의 Foreign키를 Dist Key로
2. 나머지 조인이 되는 조건의 Dimension Table은 Distribution ALL을 검토 한다.

성능

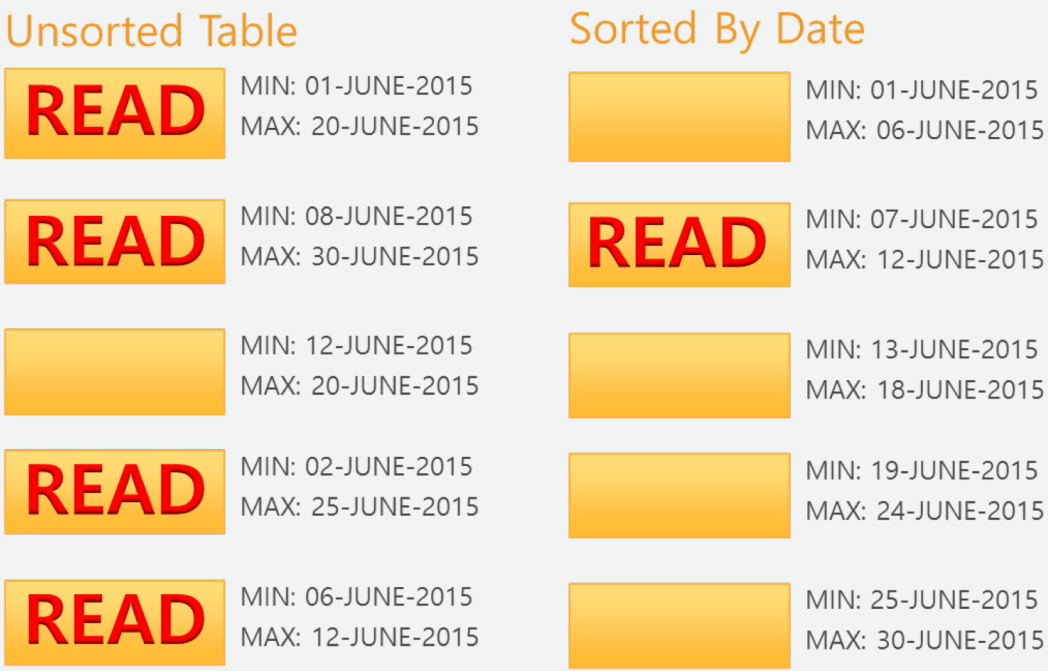
성능 향상

1. 보다 높은 하드웨어 스펙
2. 병렬화의 달성
3. 네트워크를 통한 트레픽의 최소화

"스토리지 I/O의 최소화"

Sort Keys – Zone Maps

SELECT COUNT(*) FROM LOGS WHERE DATE = '09-JUNE-2015'



Zone Map?

각 Block에 대한 최대값 및 최소값에 대한 정보 저장 (RDB의 Index와 유사)

Sort Keys – Single Column

Table is sorted by 1 column

[SORTKEY (date)]

Date	Region	Country
2-JUN-2015	Oceania	New Zealand
2-JUN-2015	Asia	Singapore
2-JUN-2015	Africa	Zaire
2-JUN-2015	Asia	Hong Kong
3-JUN-2015	Europe	Germany
3-JUN-2015	Asia	Korea

- Queries that use 1st column (i.e. *date)* as primary filter
- Can speed up joins and group bys
- Quickest to VACUUM

Sort Keys – Compound

Table is sorted by 1st column , then 2nd column etc.

Date	Region	Country
2-JUN-2015	Oceania	New Zealand
2-JUN-2015	Asia	Singapore
2-JUN-2015	Africa	Zaire
2-JUN-2015	Asia	Hong Kong
3-JUN-2015	Europe	Germany
3-JUN-2015	Asia	Korea

- Queries that use 1st column as primary filter, then other cols
- Can speed up joins and group bys
- Slower to VACUUM

Sort Keys – Interleaved

Equal weight is given to each column.

[SORTKEY INTERLEAVED (date, region, country)]

Date	Region	Country
2-JUN-2015	Oceania	New Zealand
2-JUN-2015	Asia	Singapore
2-JUN-2015	Africa	Zaire
2-JUN-2015	Asia	Hong Kong
3-JUN-2015	Europe	Germany
3-JUN-2015	Asia	Korea

- Queries that use different columns in filter
- Queries get faster the more columns used in the filter (up to 8)
- Slowest to VACUUM

Example – Compound Sort vs Interleaved Sort

- Product (**type, color, size**)

- Compound sort on **type, color, size**
  - Where **type** = ‘shirt’ and **color**=‘red’ and **size** = ‘xl’: *full advantage of sort*
  - Where **type** = ‘shirt’ and **color**=‘red’: *some efficiency*
  - Where **type** = ‘shirt’ and **size** = ‘xl’: ***limited efficiency depending on cardinality of type column***

- Interleaved sort on **type, color, size**
  - Using any of **type**, **color** and **size** in a where clause will improve performance

aws

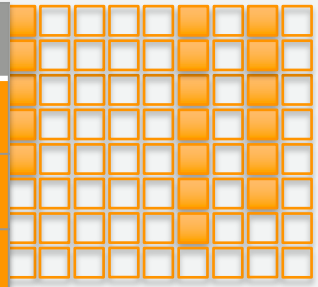
압축

```
analyze compression listing;

  Table   |     Column      | Encoding
---------+-----------------+----------
 listing | listid          | delta
 listing | sellerid        | delta32k
 listing | eventid         | delta32k
 listing | dateid          | bytedict
 listing | numtickets      | bytedict
 listing | priceperticket  | delta32k
 listing | totalprice      | mostly32
 listing | listtime        | raw
```

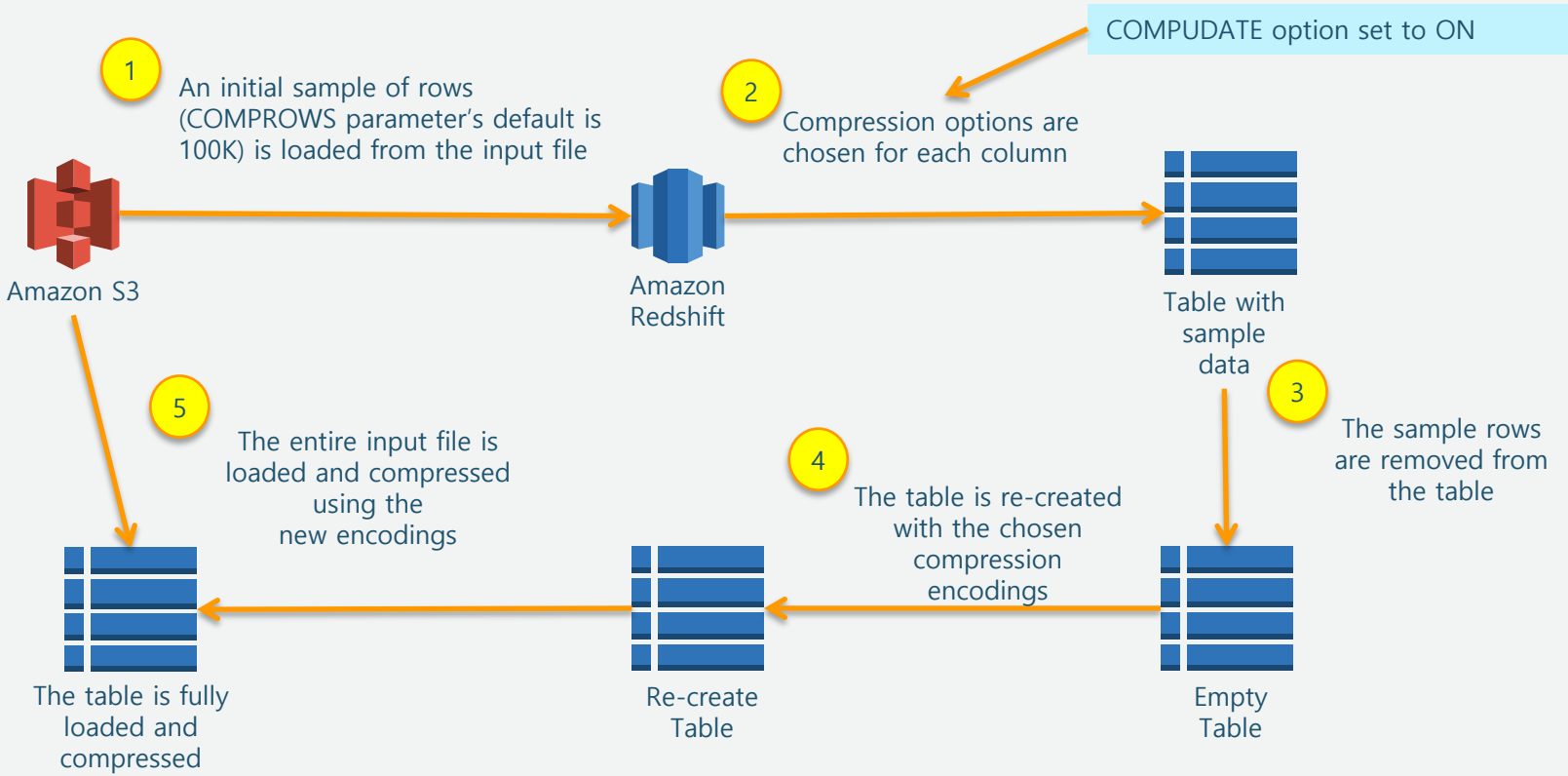
- 압축을 기반으로 디스크 IO 더욱 줄일 수 있습니다.

ID	Age	State	Amount
123	20	CA	500
345	25	WA	250
678	40	FL	125
957	37	WA	375



aws

Automatic Data Compression



쿼리 분석

EXPLAIN command followed by query string:

```
explain select avg(datediff(day, listtime, saletime)) as avgwait from sales,
listing where sales.listid = listing.listid;
                        QUERY PLAN
XN Aggregate (cost=6350.30..6350.31 rows=1 width=16)
   -> XN Hash Join DS_DIST_NONE (cost=47.08..6340.89 rows=3766 width=16)
         Hash Cond: ("outer".listid = "inner".listid)
      -> XN Seq Scan on listing (cost=0.00..1924.97 rows=192497 width=12)
      -> XN Hash (cost=37.66..37.66 rows=3766 width=12)
         -> XN Seq Scan on sales (cost=0.00..37.66 rows=3766 width=12)
```

The EXPLAIN plan has the following information about query execution steps:

- Operations to be performed
- Tables and columns involved
- Data being processed

운영 및 모니터링

VACUUM

ID	LOCATION_CODE	ADDRESS
1,2,3,4	RFK,JFK,LBJ,GWB	900 Columbus,800 Washington, 700 Foxborough,600 Kansas

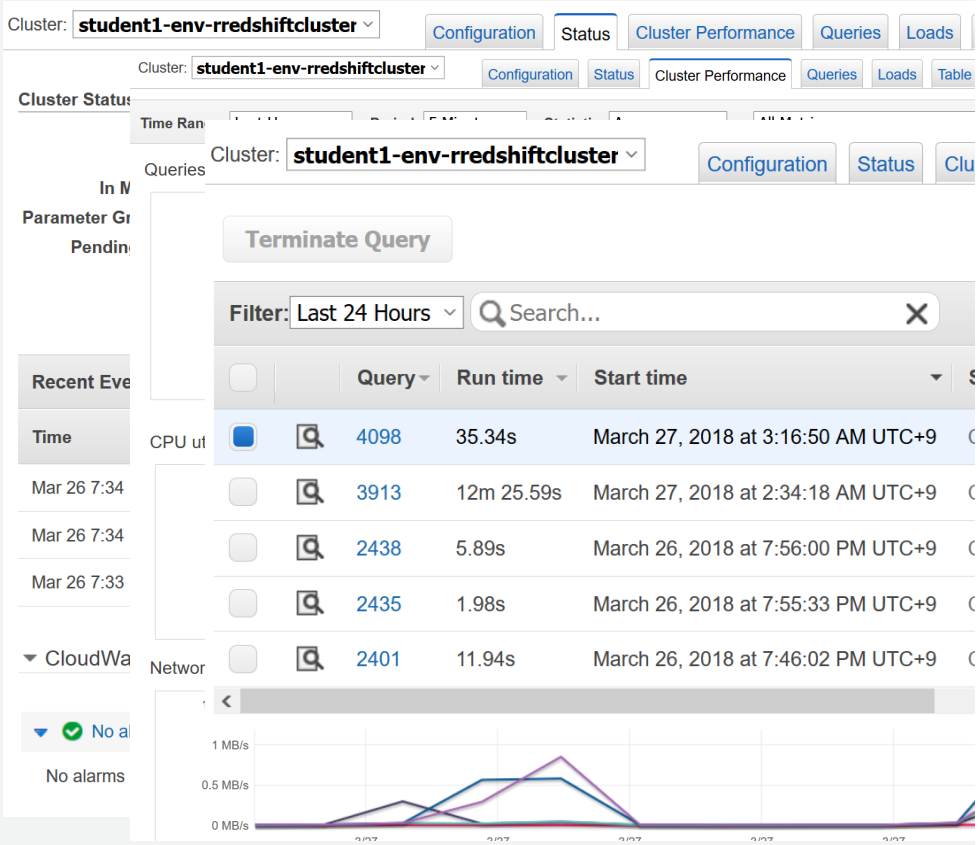
```
delete from customer where id = 3;
```

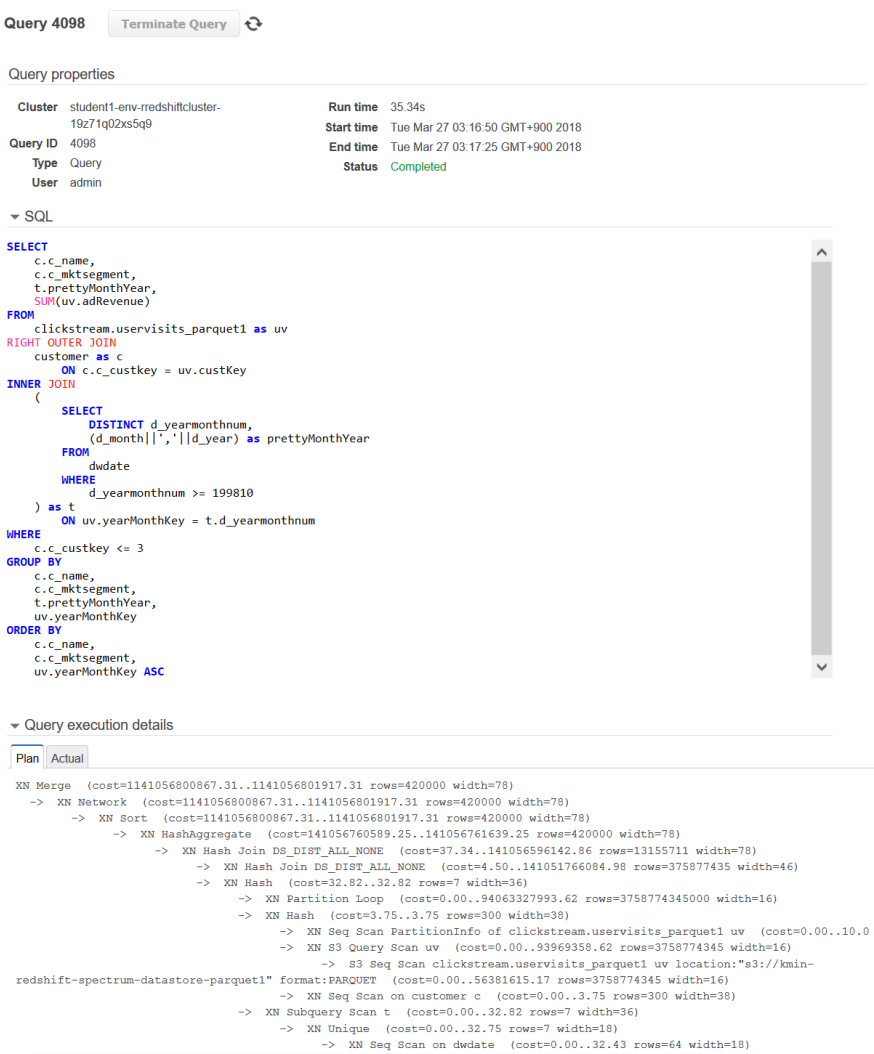
1,2 x	RFK,JFK,LBJ,GWB	900 Columbus,800 Washington xxxxxxxxxxxxxxx 00 Kansas

```
VACUUM Customer;
```

1,2,4	RFK,JFK,GWB	900 Columbus,800 Washington,600 Kansas

모니터링



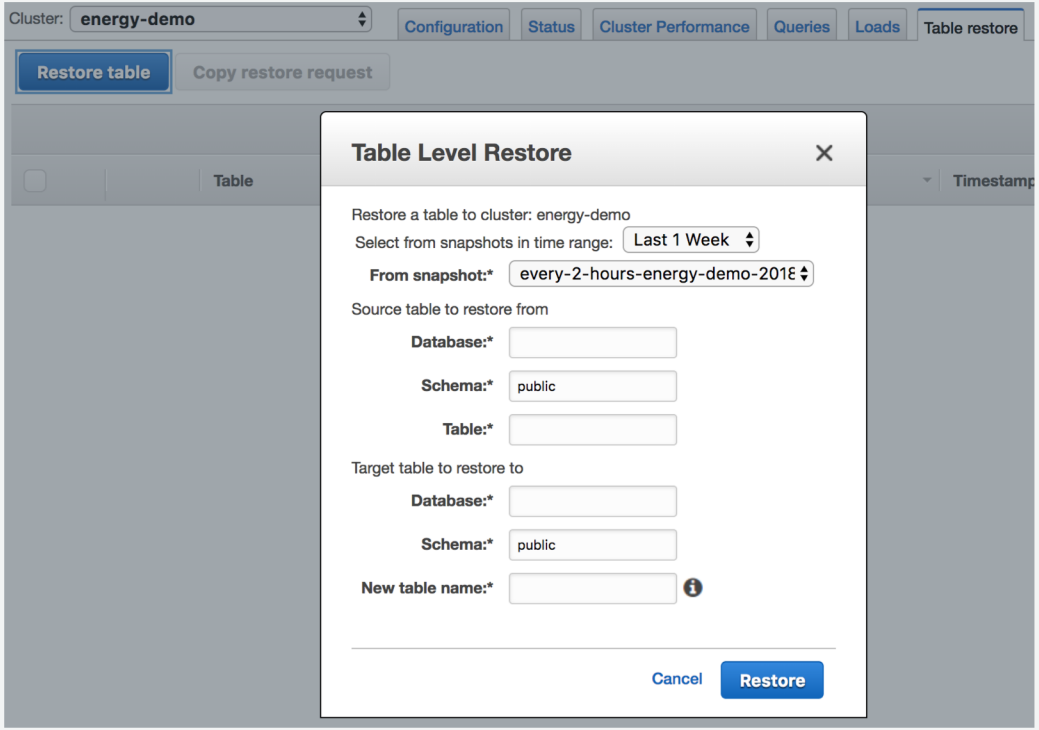


단일 테이블 복원

Restore doesn't have to be full cluster

Select the Snapshot, Source Database, Schema & Table

Restore to the same cluster that created the snapshot, but a new Database, Schema & Table Name

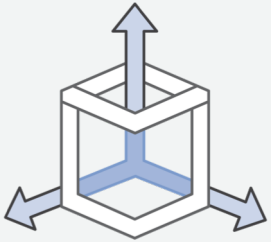


S3 데이터레이크와 통합
Amazon Redshift Spectrum

External Tables with Redshift Spectrum

Run SQL queries directly against data in S3 using thousands of nodes

High concurrency: Multiple clusters access same data



No ETL: Query data in-place using open file formats

Full Amazon Redshift SQL support

Fast @ exabyte scale

Elastic & highly available

On-demand, pay-per-query

1 Query
SELECT COUNT(*)
FROM S3.EXT_TABLE
GROUP BY…

JDBC/ODBC

Amazon Redshift

2 Query is optimized and compiled at the leader node. Determine what gets run locally and what goes to Amazon Redshift Spectrum

3 Query plan is sent to all compute nodes

4 Compute nodes obtain partition info from Data Catalog; dynamically prune partitions

5 Each compute node issues multiple requests to the Amazon Redshift Spectrum layer

6 Amazon Redshift Spectrum nodes scan your S3 data

7 Amazon Redshift Spectrum projects, filters, joins and aggregates

8 Final aggregations and joins with local Amazon Redshift tables done in-cluster

9 Result is sent back to client

1 2 3 4 … N

Amazon S3
Exabyte-scale object storage

Data Catalog
Apache Hive Metastore

Redshift Spectrum에서의 성능 향상 기법

어떻게 하면 IO를 줄일것인가?

Columnar 포멧

파티셔닝

```
SELECT
        c.c_name,
        c.c_mktsegment,
        t.prettyMonthYear,
        SUM(uv.adRevenue)
FROM
        clickstream.uservisits_csv10 as uv
RIGHT OUTER JOIN
        customer as c
                ON c.c_custkey = uv.custKey
INNER JOIN
        (
                SELECT
                        DISTINCT d_yearmonthnum,
                        (d_month||','||d_year) as prettyMonthYear
                FROM
                        dwdate
                WHERE
                        d_yearmonthnum >= 199810
        ) as t
                ON uv.yearMonthKey = t.d_yearmonthnum
WHERE
        c.c_custkey <= 3
GROUP BY
        c.c_name,
        c.c_mktsegment,
        t.prettyMonthYear,
        uv.yearMonthKey
ORDER BY
        c.c_name,
        c.c_mktsegment,
        uv.yearMonthKey ASC
```

clickstream.uservisits_csv10 as uv

```
XN Merge (cost=1141056800867.31..1141056801917.31 rows=420000 width=78)
  -> XN Network (cost=1141056800867.31..1141056801917.31 rows=420000 width=78)
    -> XN Sort (cost=1141056800867.31..1141056801917.31 rows=420000 width=78)
      -> XN HashAggregate (cost=141056760589.25..141056761639.25 rows=420000 width=78)
        -> XN Hash Join DS_DIST_ALL_NONE (cost=37.34..141056596142.86 rows=13155711 width=78)
          -> XN Hash Join DS_DIST_ALL_NONE (cost=4.50..141051766094.98 rows=3758774345 width=46)
            -> XN Hash (cost=3.75..3.75 rows=300 width=38)
              -> XN Seq Scan on ... width=38)
            -> XN Hash (cost=32...
              -> XN Subquery Sca...
                -> XN Unique (cost=0.00..32.75 rows=7 width=18)
                  -> XN Seq Scan on dwdate (cost=0.00..32.43 rows=64 width=18)
  -> XN Seq Scan PartitionInfo of clickstream.uservisits_csv10 uv (cost=0.00..10.00 rows=1000 width=0)
  -> XN S3 Query Scan uv (cost=0.00..93969358.62 rows=3758774345 width=16)
```

rows=3758774345

Query 76037 Terminate Query

Run time 4m 47.06s

Query properties

Cluster	student1-env-rredshiftcluster-19z71q02xs5q9	**Run time**	4m 47.06s
Query ID	76037	**Start time**	Mon Apr 09 12:59:54 GMT+900 2018
Type	Query	**End time**	Mon Apr 09 13:04:41 GMT+900 2018
User	admin	**Status**	Completed

- 결과 : 약 4분 47초
- CSV 포멧
- 파티셔닝 없음

```
SELECT
        c.c_name,
        c.c_mktsegment,
        t.prettyMonthYear,
        SUM(uv.adRevenue)
FROM
        clickstream.uservisits_parquet1 as uv
RIGHT OUTER JOIN
        customer as c
                ON c.c_custkey = uv.custKey
INNER JOIN
        (
                SELECT
                        DISTINCT d_yearmonthnum,
                        (d_month||','||d_year) as prettyMonthYear
                FROM
                        dwdate
                WHERE
                        d_yearmonthnum >= 199810
        ) as t
                ON uv.yearMonthKey = t.d_yearmonthnum
WHERE
        c.c_custkey <= 3
GROUP BY
        c.c_name,
        c.c_mktsegment,
        t.prettyMonthYear,
        uv.yearMonthKey
ORDER BY
        c.c_name,
        c.c_mktsegment,
        uv.yearMonthKey ASC
```

clickstream.uservisits_parquet1 as uv

```
XN Merge (cost=1141056800867.31..1141056801917.31 rows=420000 width=78)
  -> XN Network (cost=1141056800867.31..1141056801917.31 rows=420000 width=78)
                (cost=1141056800867.31..1141056801917.31 rows=420000 width=78)
                shAggregate (cost=141056760589.25..141056761639.25 rows=420000 width=78)
                ash Join DS_DIST_ALL_NONE (cost=37.34..141056596142.86 rows=13155711 width=78)
        -> XN Hash Join DS_DIST_ALL_NONE (cost=4.50..141051766084.08 rows=3758774345 width=46)
          -> XN Hash (cost=3.75..3.75 rows=300 width=38)
            -> XN Seq Scan on customer c (cost=0.00..3.75 rows=300 width=38)
          -> XN Hash (co
            -> XN Subque                                                =36)
              -> XN Unique (cost=0.00..32.75 rows=73 width=18)
                -> XN Seq Scan on dwdate (cost=0.00..32.43 rows=64 width=18)
  -> XN Seq Scan PartitionInfo of clickstream.uservisits_parquet1 uv (cost=0.00..10.00 rows=1000
width=0)
  -> XN S3 Query Scan uv (cost=0.00..93969358.62 rows=3758774345 width=16)
```

rows=3758774345

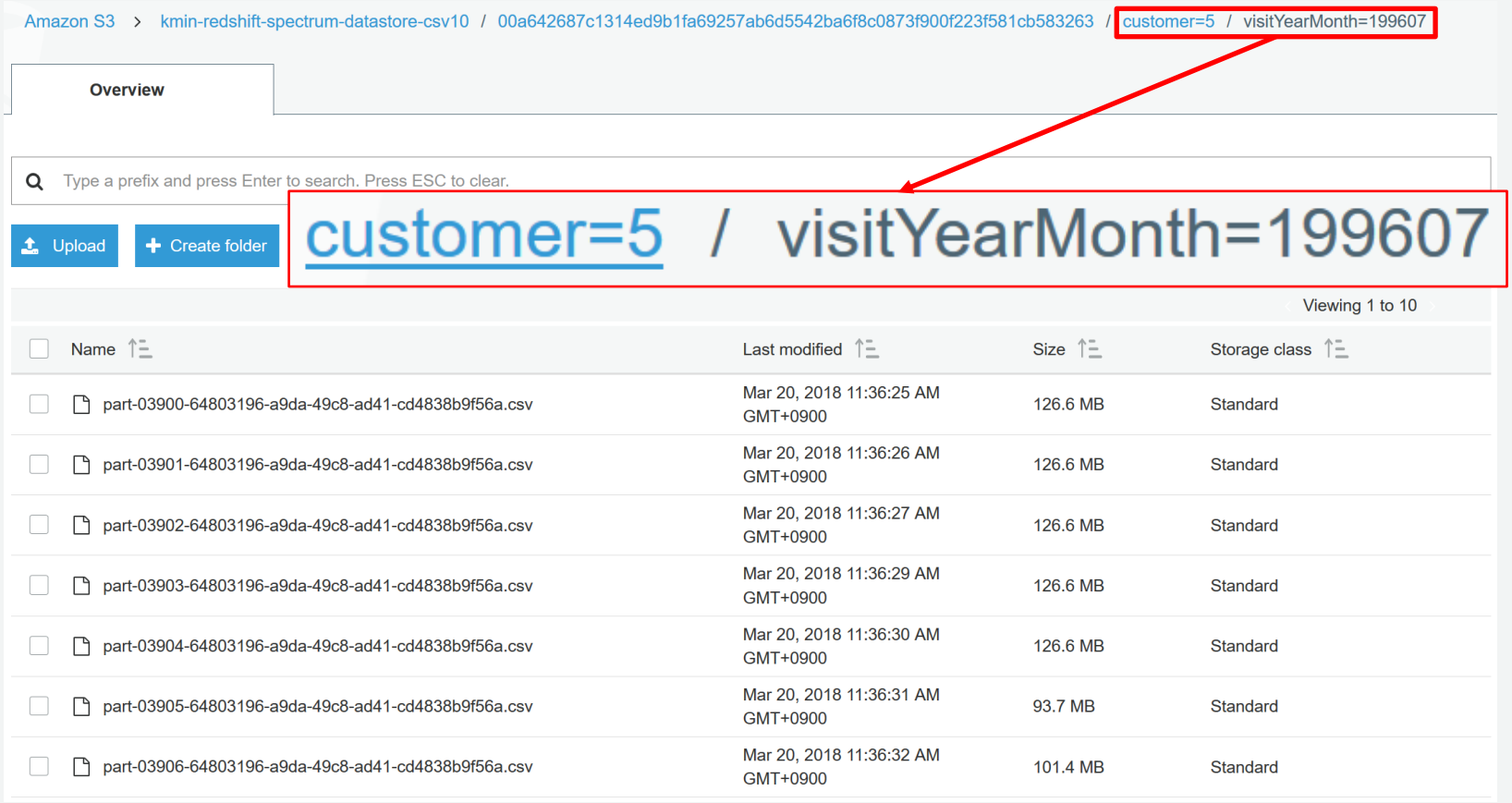
Query 76160 Terminate Query

Run time 25.81s

Query properties

Cluster	student1-env-rredshiftcluster-19z71q02xs5q9	**Run time**	25.81s
Query ID	76160	**Start time**	Mon Apr 09 13:33:08 GMT+900 2018
Type	Query	**End time**	Mon Apr 09 13:33:33 GMT+900 2018
User	admin	**Status**	Completed

- 결과 : 약 26초
- Parquet 포멧 (Columnar 포멧)
- 파티셔닝 없음



```
SELECT
        c.c_name,
        c.c_mktsegment,
        t.prettyMonthYear,
        uv.totalRevenue
FROM
        ((SELECT
                customer,
                visitYearM
                SUM(adR
        FROM
                clickstream.uservisits_parquet1
        WHERE
                customer <= 3
                and visitYearMonth >= 199810
        GROUP BY
                customer,
                visitYearMonth) as uv
RIGHT OUTER JOIN
        customer as c
                ON c.c_custkey = uv.customer
INNER JOIN
        (
                SELECT
                        DISTINCT d_yearmonthnum,
                        (d_month||','||d_year) as prettyMonthYear
                FROM
                        dwdate
                WHERE
                        d_yearmonthnum >= 199810
        ) as t
                ON uv.visitYearMonth = t.d_yearmonthnum
        )
ORDER BY
        c.c_name,
        c.c_mktsegment,
        uv.visitYearMonth ASC
```

customer <= 3
and visitYearMonth >= 199810

```
XN Merge (cost=1000094008880.16..1000094008880.18 rows=7 width=78)
  -> XN Network (cost=1000094008880.16..1000094008880.18 rows=7 width=78)
    -> XN Sort (cost=1000094008880.16..1000094008880.18 rows=7 width=78)
      -> XN Hash Join DS_DIST_ALL_NONE (cost=94008878.97..94008880.06 rows=7 width=78)
        -> XN Hash Join DS_DIST_ALL_NONE (cost=93971378.97..93971379.61 rows=7 width=48)
          -> XN Subquery Scan uv (cost=93971346.13..93971346.42 rows=23 width=16)
            -> XN HashAggregate (cost=93971346.13..93971346.19 r
          -> XN Hash (cost=32.82..32.82 rows=7 width=36)
            -> XN Subquery Scan t (cost=0.00..32.82 rows=7 width=3
              -> XN Unique (cost=0.00..32.75 rows=7 width=18)
                -> XN Seq Scan on dwdate (cost=0.00..32.43 rows=64 width=18)
        -> XN Hash (cost=30000.00..30000.00 rows=3000000 width=38)
          -> XN Seq Scan on customer c (cost=0.00..30000.00 rows=3000000 width=38)
  -> XN Seq Scan PartitionInfo of clickstream.uservisits_parquet1 (cost=0.00..17.50 rows=112 width=8)
  -> XN S3 Query Scan uservisits_parquet1 (cost=46984679.32..46984689.32 rows=1000 width=8)
```

rows=3000000

Query 76241 Terminate Query

Run time 2.8s

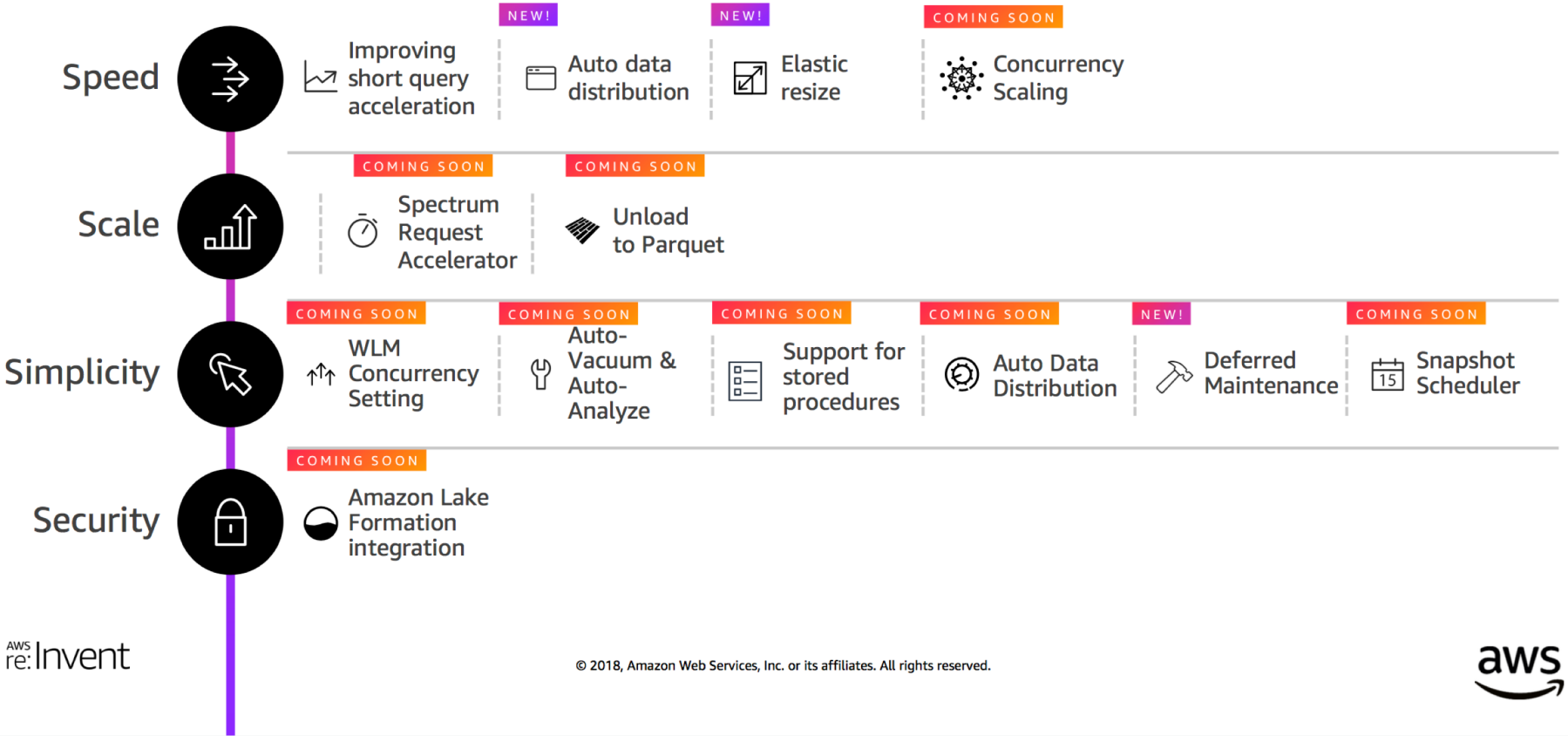
Query properties

Cluster	student1-env-rredshiftcluster-19z71q02xs5q9
Query ID	76241
Type	Query
User	admin
Run time	2.8s
Start time	Mon Apr 09 13:55:29 GMT+900 2018
End time	Mon Apr 09 13:55:32 GMT+900 2018
Status	Completed

- 결과 : 약 3초
- Parquet 포멧 (Columnar 포멧)
- 파티셔닝 : customer, visitYearMonth

Amazon Redshift

New features



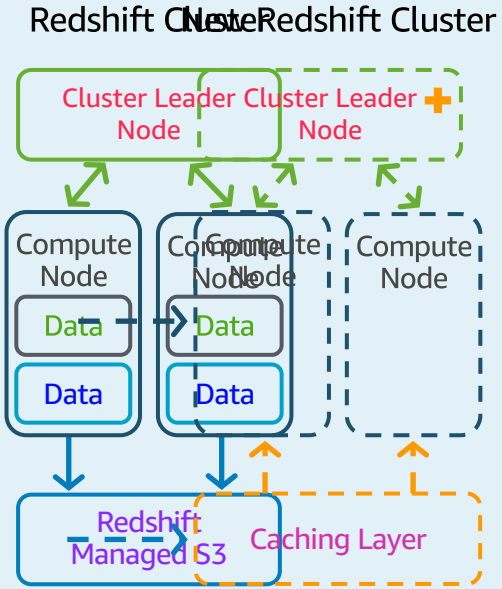
AWS re:Invent

aws

Amazon Redshift Concurrency Scaling (Preview)

NEW!

Consistently fast performance at virtually unlimited concurrency

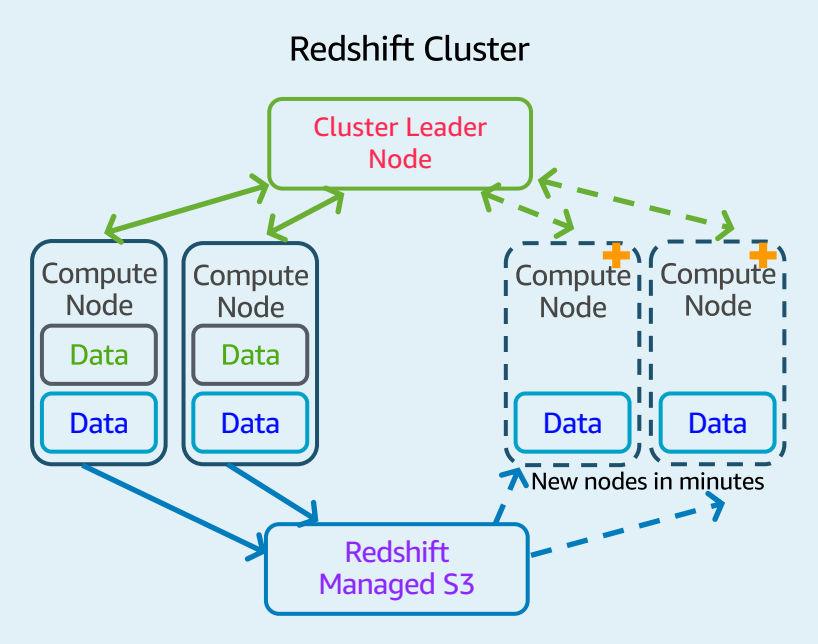


- Automatically spins up additional clusters on-demand
- Handles virtually unlimited number of concurrent users
- Accrued minutes make it free for most customers

aws

Amazon Redshift Elastic Resize (GA) NEW!

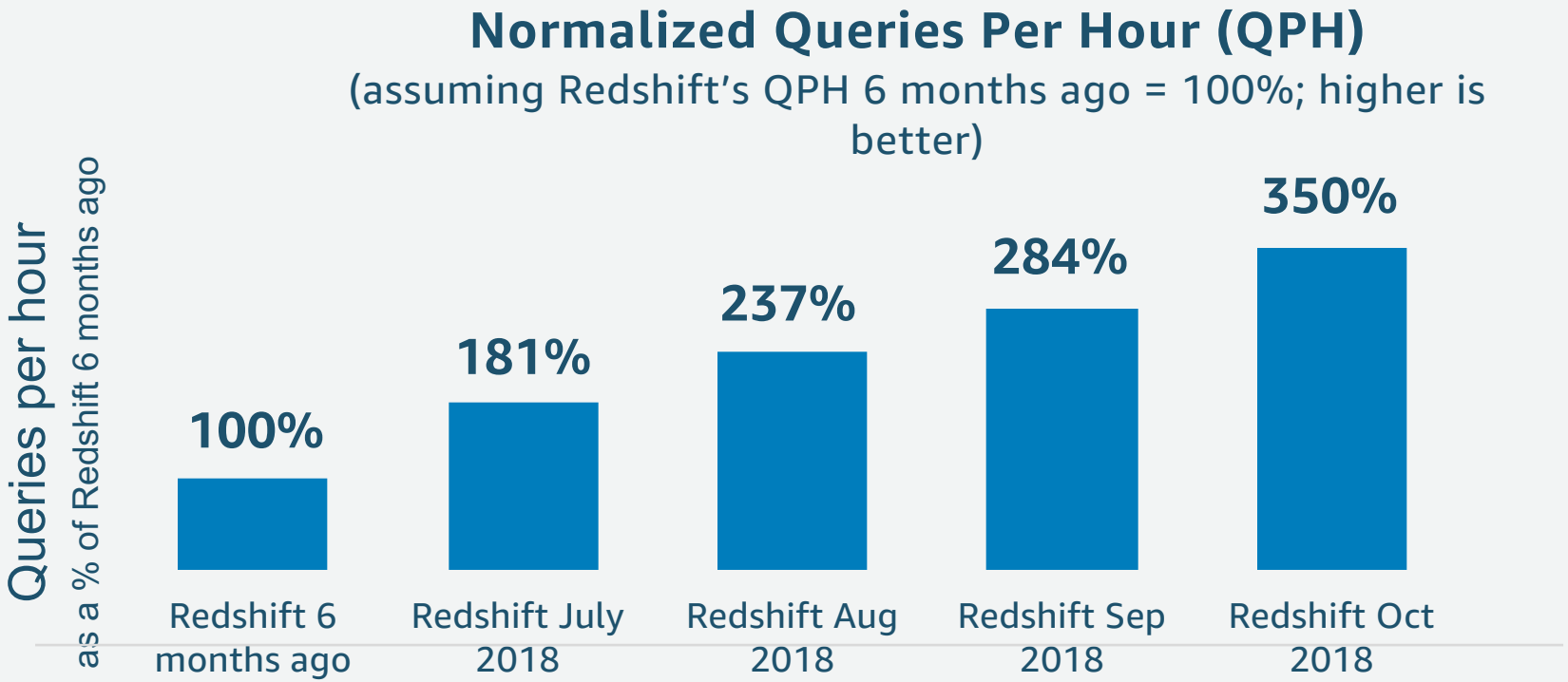
Quickly scale up or down to increase performance on-demand



Add/remove additional nodes to cluster in minutes

Available today

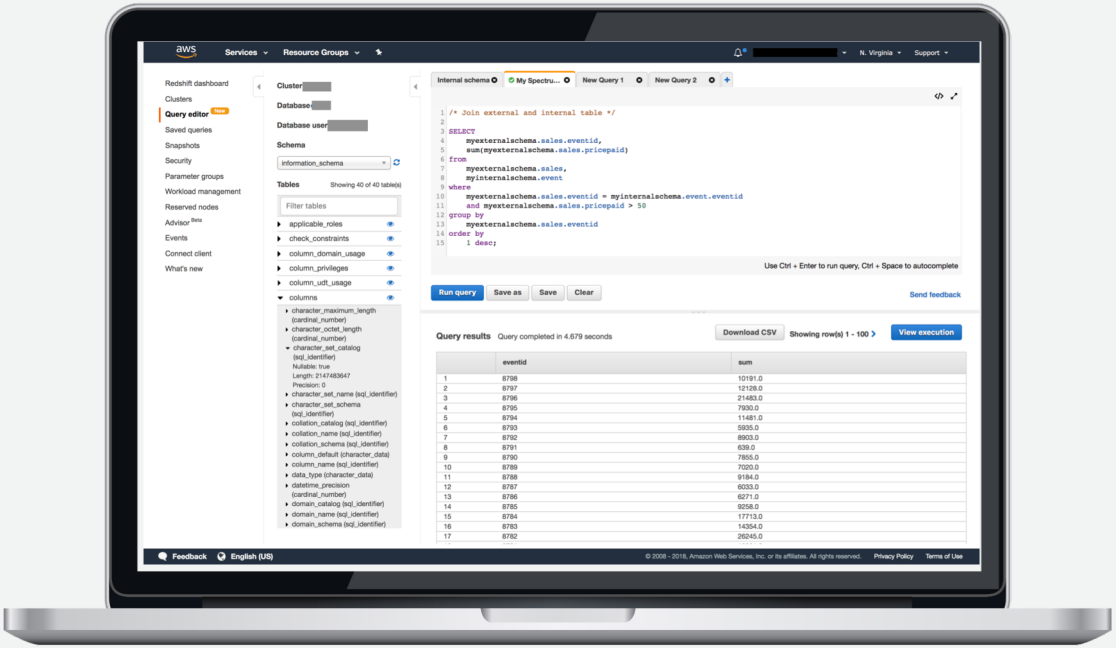
Amazon Redshift is >3x faster than 6 months ago



aws

Redshift Query Editor

Launched in October!



Query data directly from the AWS console

Results are instantly visible within the console

No need to install and setup an external JDBC/ODBC client

Amazon Redshift intelligent administration

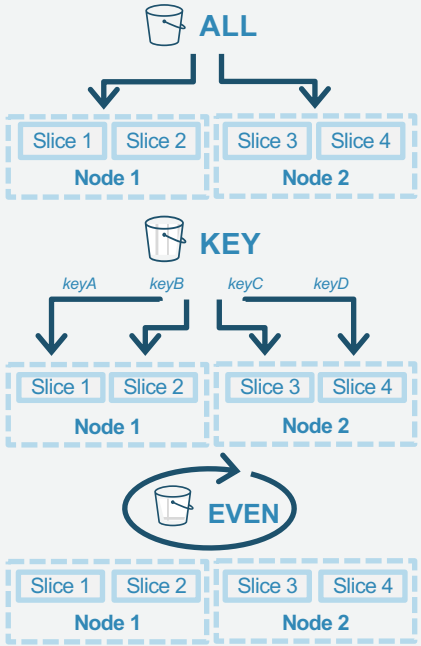
Coming Soon!

Automates data distribution in tables for improved performance and disk space utilization.

No more messing with distkeys!

Advise

Amazon Redshift recommended distribution key



Provides intelligent recommendations for tuning based on continuous workload analysis.

aws

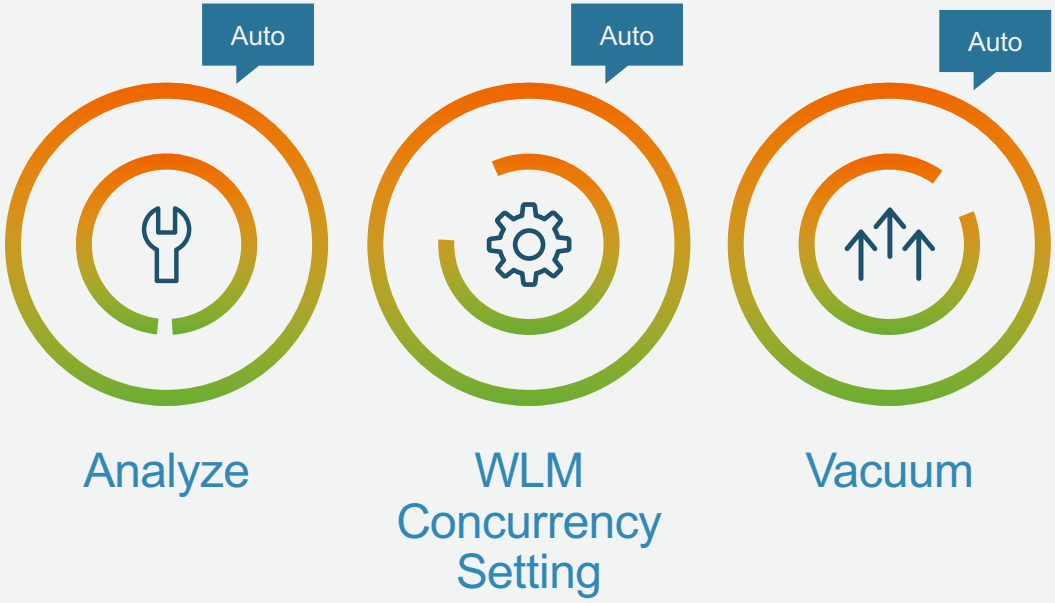
Amazon Redshift intelligent maintenanc

Coming Soon!

Maintenance processes like vacuum and analyze will automatically run in the background.

Moving towards zero-maintenance.

Amazon Redshift will automatically adjust the WLM concurrency setting to deliver optimal throughput.



aws

Run stored procedures in Amazon Redshift

Coming Soon!

Bring your existing **Stored Procedure** and run in Redshift.

Migrating to Redshift is even easier!

Amazon Redshift will support Stored Procedure in PL/pgSQL format, enabling you to bring your existing Stored Procedure to Amazon Redshift.

Support for stored procedure provides the ability to run code where the data is to efficiently run ETL, data validation, and custom business logic.

aws

Amazon Redshift Spectrum

Extend the data warehouse to exabytes of data in S3 Data Lake

No loading required

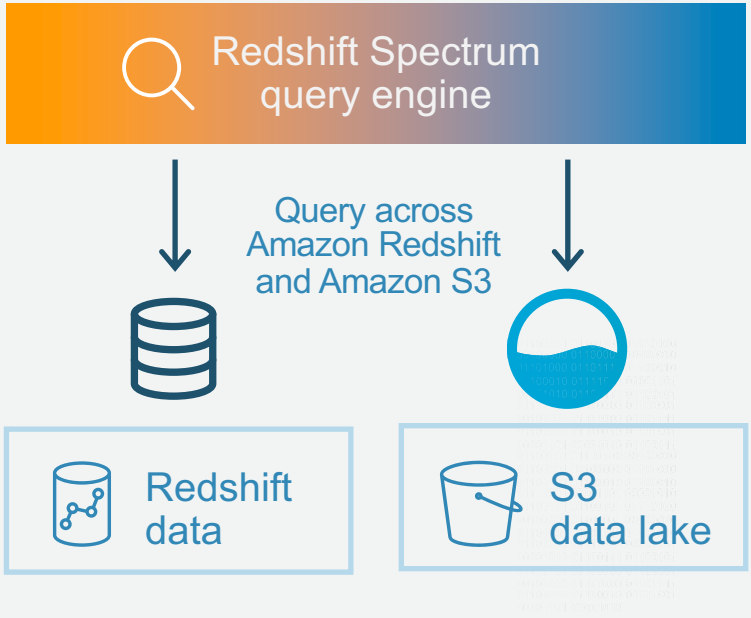
Scale compute and storage separately

Directly query data stored in Amazon S3

Parquet, ORC, Avro, Grok, and CSV data formats

→ Unload to Parquet
→ Spectrum Request Accelerator

Coming Soon!



aws

Security is **built-in**

End-to-end encryption

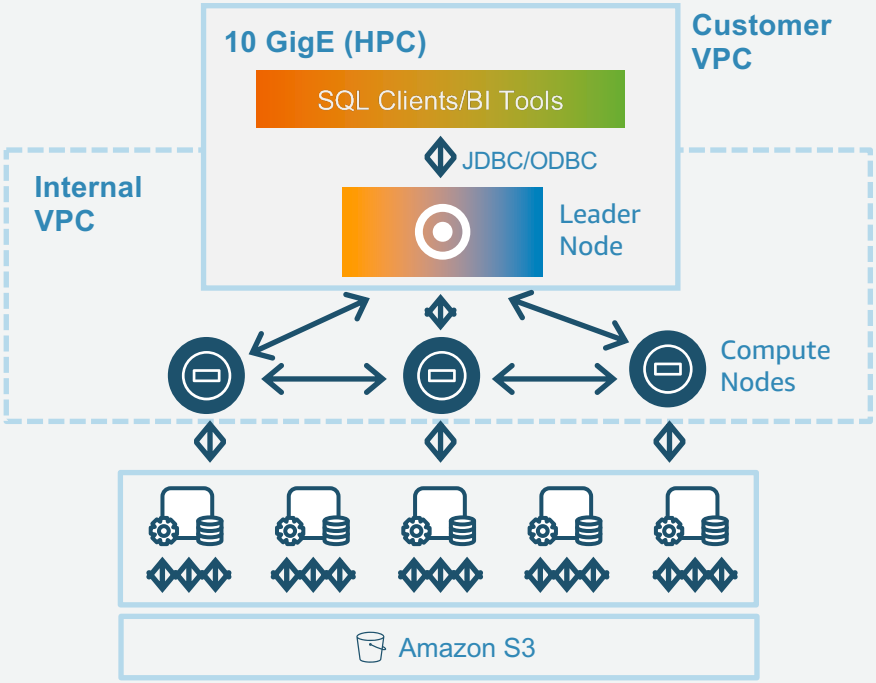
Integration with AWS Key Management Service

Select compliance certifications*

AICPA SOC · FedRAMP · DEPARTMENT OF DEFENSE · PCI DSS COMPLIANT · HIPAA COMPLIANT · ISO

*Full list of compliance certifications is available here: https://aws.amazon.com/compliance

Network Isolation



aws

참고 URLs

- **Amazon Redshift 를 위한 10가지 성능 튜닝 기법**

https://aws.amazon.com/ko/blogs/korea/top-10-performance-tuning-techniques-for-amazon-redshift/

- **Redshift 운영 시 참고 스크립트**
https://github.com/awslabs/amazon-redshift-utils