Curso de

AWS Redshift para Manejo de Big Data

Carlos Alarcón

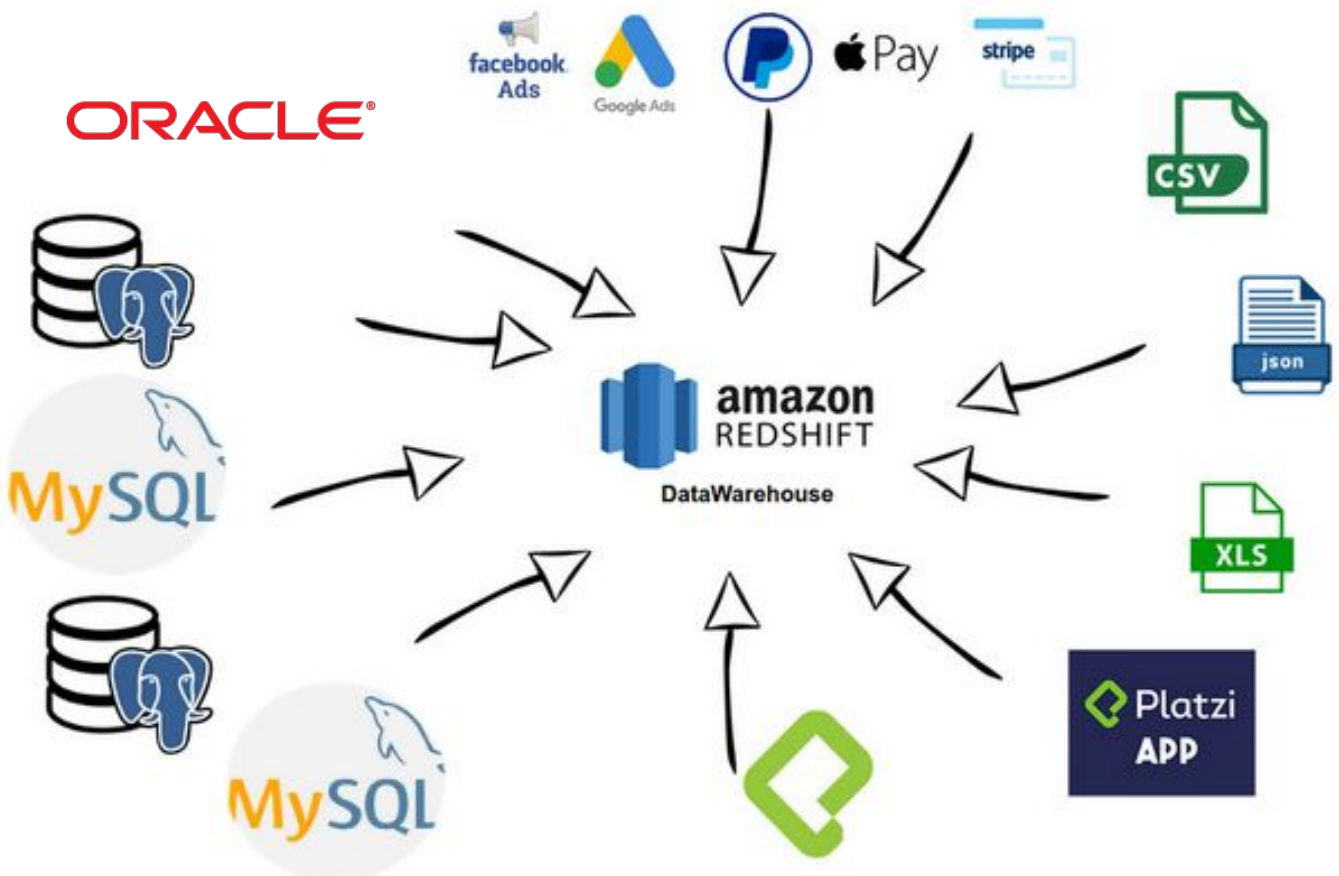
¿Qué es un Data Warehouse?

¿Qué es un Data Warehouse?

Un repositorio unificado de datos que almacena información de distintas fuentes de datos en la organización.

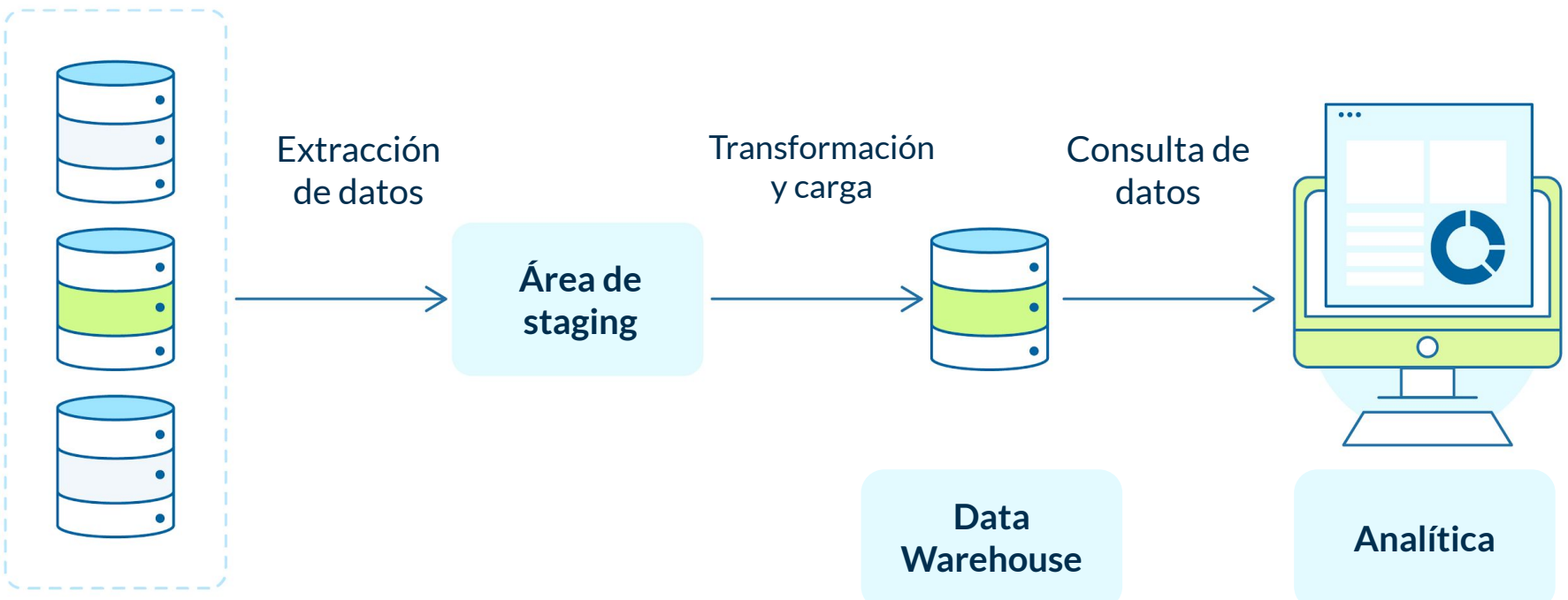
Estos repositorios unificados tienen un fin analítico, de manera que su estructura debe responder a estas necesidades.

¿Qué es un Data Warehouse?



ETL

Por sus siglas en inglés representa: extracción, transformación y carga.

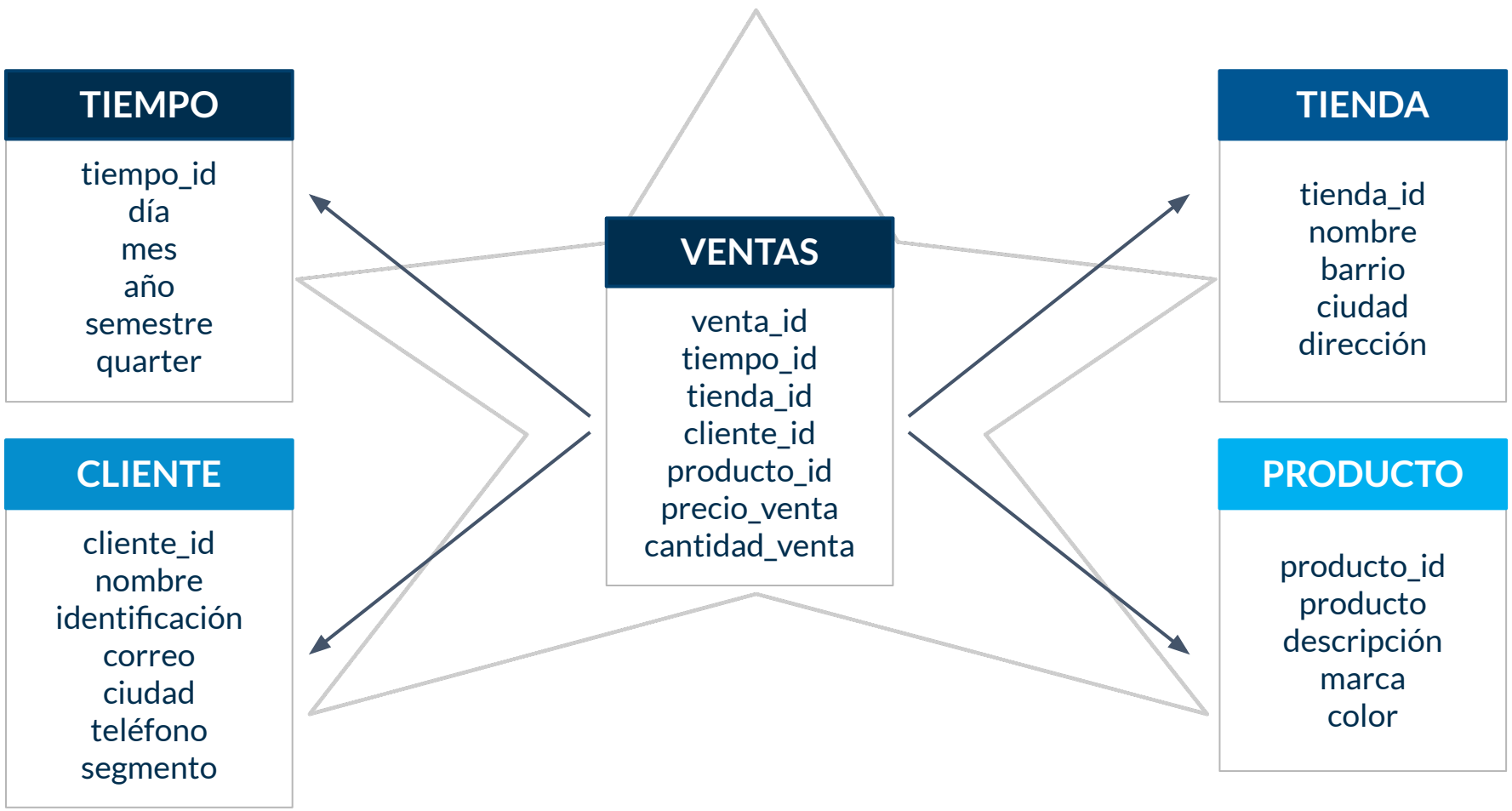


Arquitectura del data warehouse

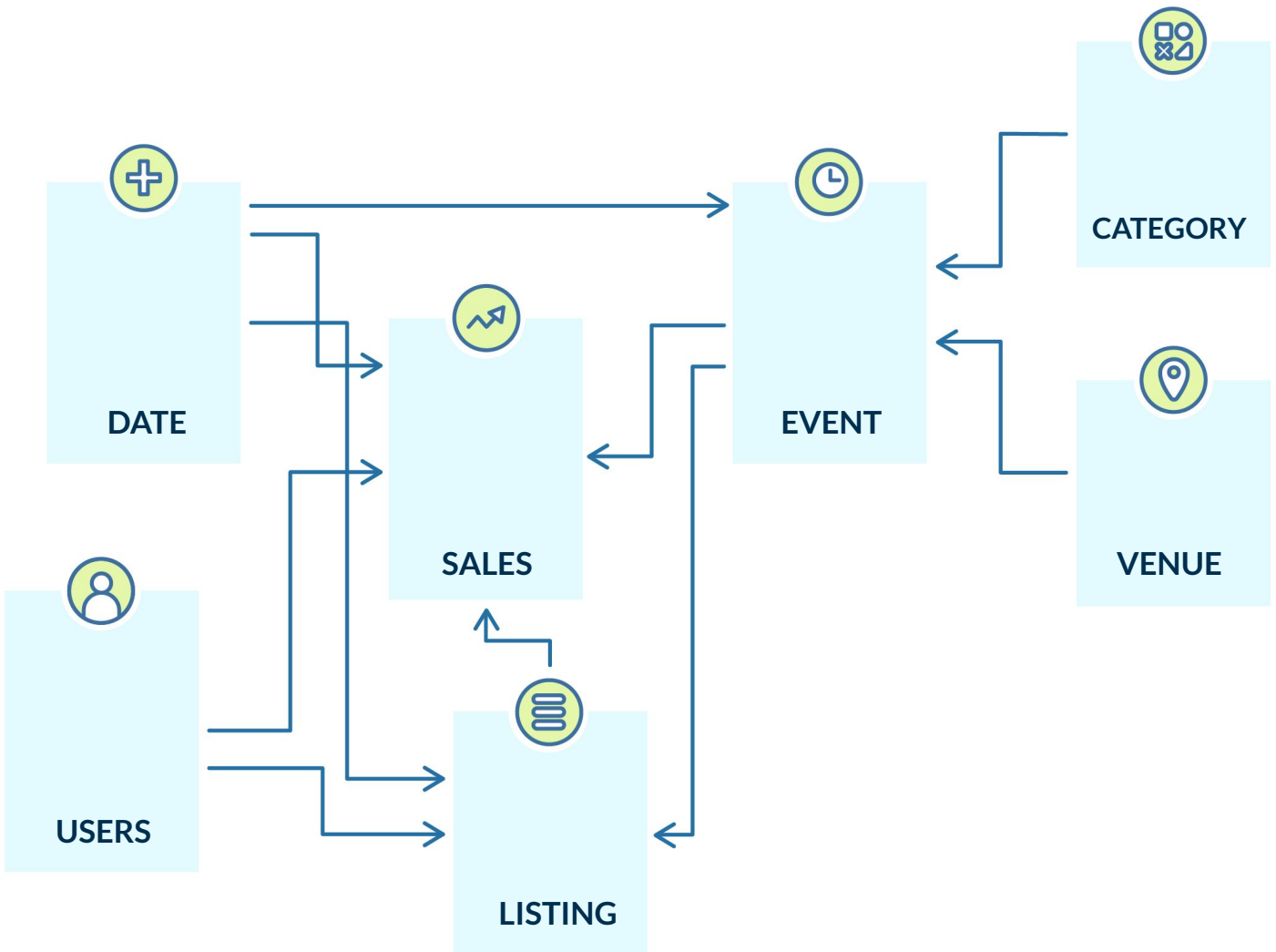
- **Tablas de hechos**
  Contienen la información que queremos medir / analizar.
- **Tablas de dimensiones**
  Contienen la información del “cómo” lo quiero medir.

Arquitectura del data warehouse

Arquitectura del data warehouse



Nuestro modelo de datos



Bases de datos columnares

¿Qué es una base de datos columnar?

Es una base de datos optimizada para lograr una recuperación rápida de columnas de datos, normalmente en aplicaciones analíticas, esto permite procesar queries complejos de una manera óptima.

¿Qué es una base de datos columnar?

- **Bases de datos basadas en filas**
  Enfocados en la transaccionalidad y en la lectura y escritura rápida de filas únicas.
- **Bases de datos basadas en columnas**
  Procesan grandes cantidades de información, se encuentran optimizadas para procesos de analítica.

Bases de datos basadas en filas

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL

Bases de datos basadas en filas

- Excelentes para aplicaciones OLTP.
- Un registro a cada bloque de datos (pueden ser mas).
- Una consulta a una tabla de 10 columnas de las cuales solo requiero 2, leería las 8 innecesarias también

Bases de datos columnares

SSN	Name	Age	Addr	City	St
101259797	SMITH	88	899 FIRST ST	JUNO	AL
892375862	CHIN	37	16137 MAIN ST	POMONA	CA
318370701	HANDU	12	42 JUNE ST	CHICAGO	IL

101259797 | 892375862 | 318370701 | 468248180 | 378568310 | 231346875 | 317346551 | 770336528 | 277332171 | 455124598 | 735885647 | 387586301

Block 1

Bases de datos columnares

- Excelentes para aplicaciones de analítica.
- En redshift cada columna se almacena en bloques de 1 MB.
- Una consulta a una tabla de 10 columnas de las cuales solo requiero 2, leería únicamente las 2 necesarias.

Bases de datos columnares

amazon
REDSHIFT

Google
BigQuery

APACHE
HBASE

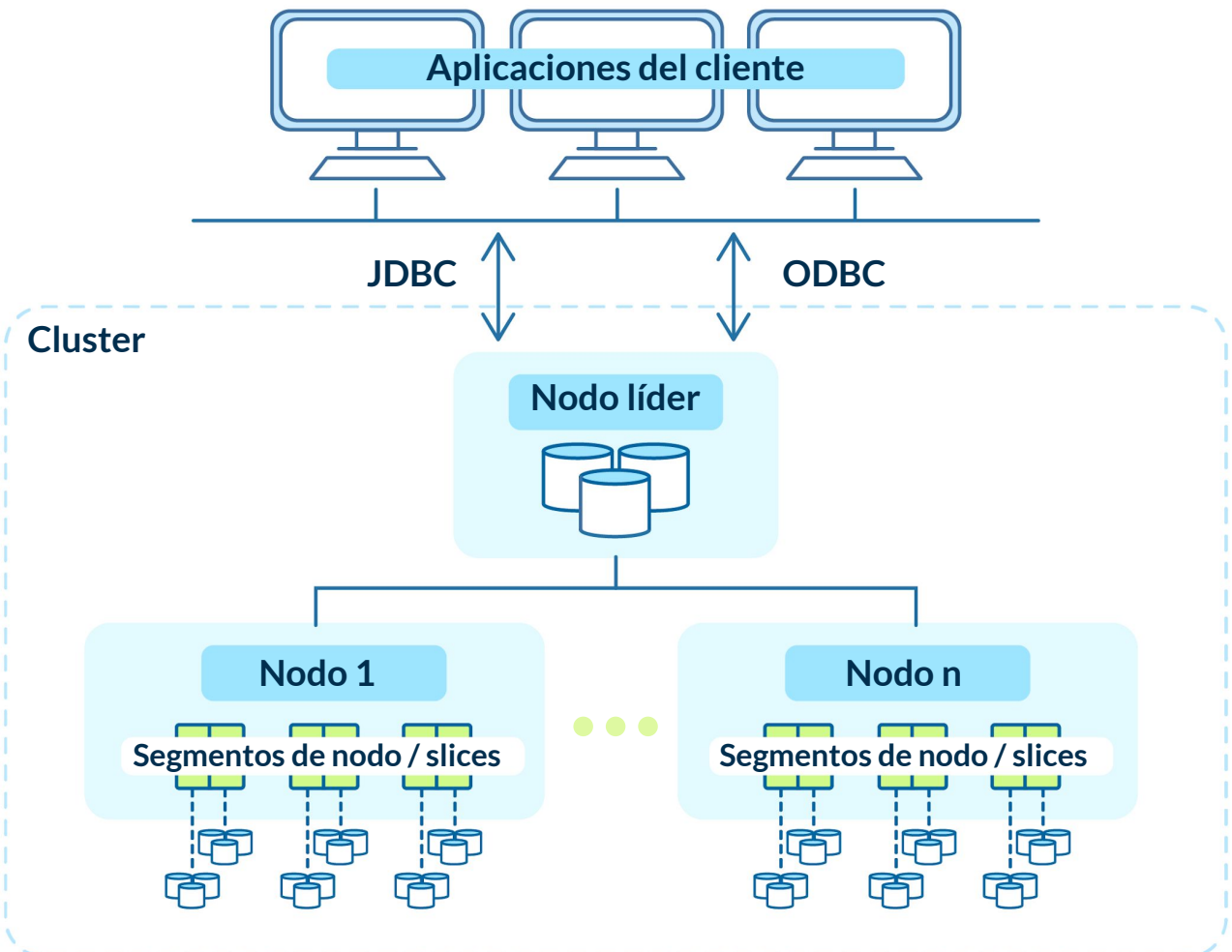
snowflake®

Beneficios de usar Redshift

- Integración total con AWS.
- La base de datos más rápida en la nube.
- Los costos más bajos en la nube.
- Alta escalabilidad.
- Clientes SQL.

¿Cómo funciona Redshift?

¿Cómo funciona Redshift?



¿Cómo funciona Redshift?

¿Cómo funciona Redshift?

- Bloques de datos de 1MB

slice	col	tbl	blocknum	num_values	extended_limits	minvalue	maxvalue
0	9	108.619	0	130.994	0	2	261.990
1	9	108.619	0	130.994	0	3	262.639

- Data distribuida

table	slice	rows
cartesian_venue	0	17.709.724
cartesian_venue	1	21.174.670
encodingvenue	0	19.442.197
encodingvenue	1	19.442.197
listing	0	95.967
listing	1	96.530
users	0	24.958
users	1	25.032
venue	0	92
venue	1	110

¿Qué es la compresión en Redshift?

Compresión con Redshift

- Es una operación dirigida a las columnas.
- Reduce el tamaño en almacenamiento de los datos reduciendo el I/O.
- Las consultas son más veloces en una columna comprimida.

Sentencia

CREATE TABLE table_name (column_name data_type ENCODE encoding-type)

Ejemplo:
CREATE TABLE test_compresion (nombre varchar(30) ENCODE TEXT255)

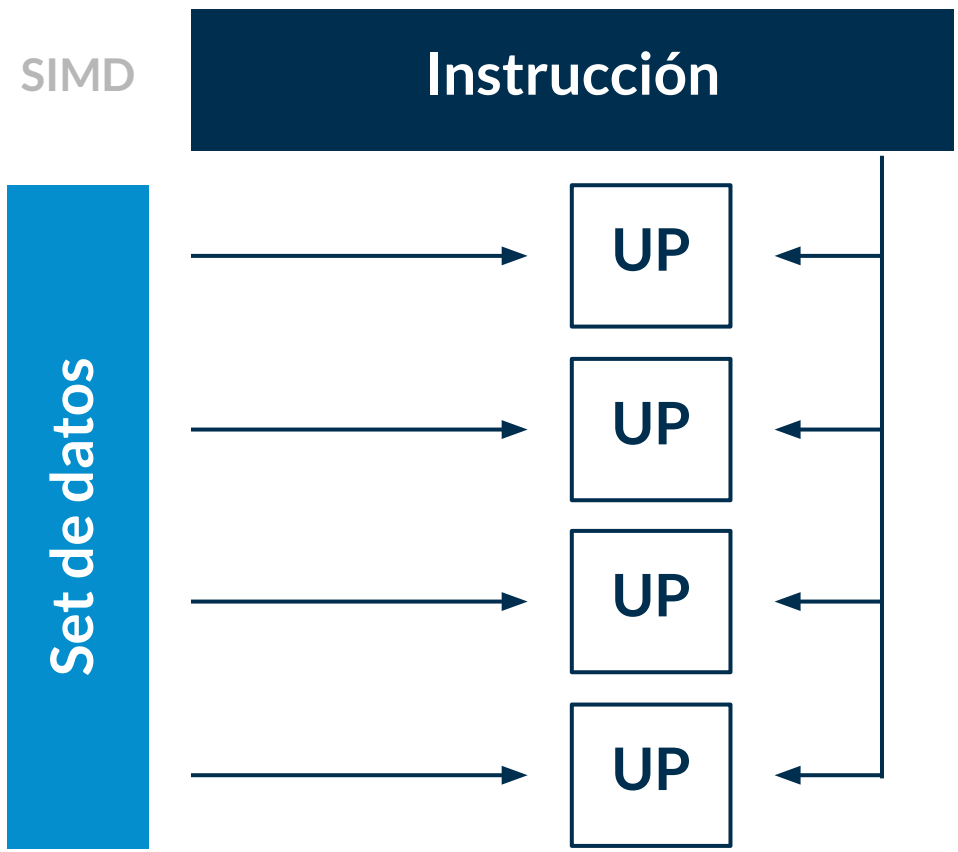
Codificación Raw

- Los datos se almacenan descomprimidos y sin formato.
- Tipos de dato: todos.
- Tipo de codificación por defecto.

Codificación AZ64

- Codificación propia de amazon (AWS).
- Tipos de dato: smallint, integer, bigint, decimal, date, timestamp, timestamptz.
- Utiliza SIMD (Single Instruction, Multiple Data) para procesamiento paralelo.

Codificación AZ64



Codificación por diccionario de bytes

- Muy eficaz cuando una columna tiene una cantidad limitada de valores únicos (menos de 256).
- Crea un diccionario en un bloque de redshift de 1MB.
- Tipos de dato: smallint, integer, bigint,decimal, real, double, precision, char, varchar,date, timestamp, timestamptz.

Codificación por diccionario de bytes

Diccionario de la columna “País” varchar(30):

País	Índice (diccionario)	Tamaño (bytes)
Colombia	0	30
México	1	30
Venezuela	2	30
Argentina	3	30
Perú	4	30
Total		**150**

Codificación por diccionario de bytes

Comparación, compresión por diccionario de bytes vs ninguna compresión:

País	Tamaño original (bytes)	Índice	Nuevo tamaño
México	30	1	1
México	30	1	1
Colombia	30	0	1
Perú	30	4	1
Colombia	30	0	1
Venezuela	30	2	1
Argentina	30	3	1
Total	**210**		**7**

Codificación Delta

- Muy útiles para las columnas con formato fecha y hora.
- Guarda la diferencia entre un registro y el siguiente.
- Tipos de dato: smallint, int, bigint, date, timestamp, decimal.

Codificación Delta

- Existe Delta de un byte y de dos bytes 8 y 16 bits respectivamente.
- No se pueden superar estos bytes en la diferencia, de ser así la codificación no se aplica.
- El rango de 1 byte abarca desde -127 hasta 127 y el rango de 2 bytes desde -32K hasta 32K.

Codificación Delta

Valor original	Tamaño original (bytes)	Delta (Diferencia)	Nuevo valor	Nuevo tamaño	Descripción
1	4		1	1 + 4 = 5	El valor inicial no tiene delta
10	4	9	9	1	
15	4	5	5	1	
9	4	-6	-6	1	
150	4	141	141	1 + 4 = 5	El delta sobrepasa los 127 del rango para 1 byte
220	4	70	70	1	
221	4	1	1	1	
Total	**28**			**15**	

Codificación LZO

- Muy útil para largas cadenas de texto.
- Funciona para texto libre.
- Tipos de dato: smallint, integer, bigint, decimal, char, varchar, date, timestamp, timestamptz.

Algoritmos de compresión con Redshift

Codificación Mostly

- Se utiliza cuando la mayoría de los datos de una columna son mejores en bits al valor de la columna misma.
- Existe mostly de 8, 16 y 32 bits.
- Tipos de dato: smallint, int, bigint, decimal.

Codificación Mostly 8 bits

Valor original	Tamaño original (bytes)	Nuevo tamaño
1	4	1
10	4	1
100	4	1
1.000	4	4
10.000	4	4
50.000	8	8
10.0000	8	8
20.0000.000	8	8
Totales	44	35

Codificación Mostly 16 bits

Valor original	Tamaño original (bytes)	Nuevo tamaño
1	4	2
10	4	2
100	4	2
1.000	4	2
10.000	4	2
50.000	8	8
10.0000	8	8
20.0000.000	8	8
Totales	44	34

Codificación Mostly 32 bits

Valor original	Tamaño original (bytes)	Nuevo tamaño
1	4	4
10	4	4
100	4	4
1.000	4	4
10.000	4	4
50.000	8	4
10.0000	8	4
20.0000.000	8	4
Totales	44	32

Codificación Runlength

- Reemplaza un valor que se repite de manera consecutiva por un valor y por un recuento de la cantidad de números consecutivos.
- Idea para una tabla en la que los valores de datos suelen repetirse de manera consecutiva.

Codificación Runlength

- Tipos de dato: smallint, integer, bigint, decimal, real, double precision, boolean, char, varchar, date, timestamp, timestamptz.

Codificación Runlength

Valor original	Tamaño original (bytes)	Valor comprimido (token)	Nuevo tamaño
Expert plus	11	{2,Expert plus}	12
Expert plus	11		0
Expert	6	{1,Expert}	7
Basic	5	{3,Basic}	6
Basic	5		0
Basic	5		0
Expert	6	{2,Expert}	7
Expert	6		0
Totales	**50**		**32**

Codificaciones Text255 y Text32k

- Son útiles para comprimir columnas VARCHAR en las que se repiten con frecuencia las mismas palabras.
- Text255 usa las 245 palabras más frecuentes de la columna y crea un diccionario.
- Text32k hace lo mismo pero indexa palabras hasta tener un diccionario de 32k.

Codificación Zstandard

- La codificación ZSTD funciona especialmente bien para las columnas CHAR y VARCHAR.
- Es muy poco probable que ZSTD aumente el uso del disco.
- Tipos de dato: smallint, integer, bigint, decimal, real, double, precision, boolean, char, varchar, date, timestamp, timestamptz.

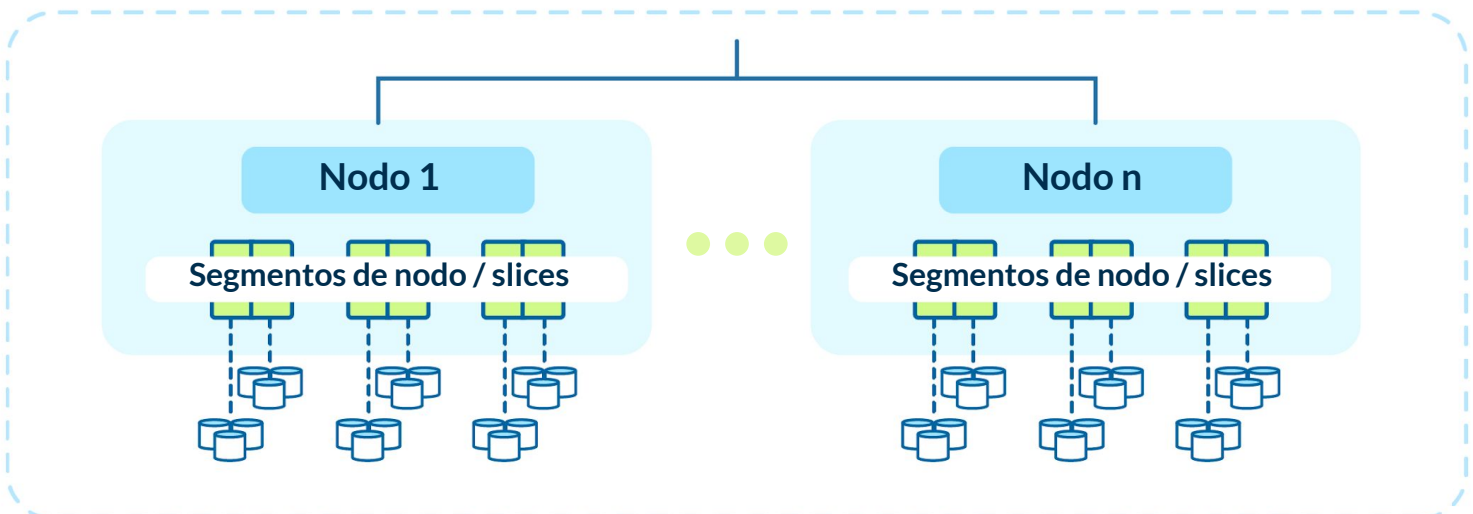
Piensa en la compresión

Columna	Tipo de dato	Compresión
ID	int	delta
Nombre	varchar(50)	raw
Genero	varchar(10)	bytedict - text255
Pais	varchar(20)	bytedict - text255
Ciudad	varchar(50)	text255
Direccion	varchar(20)	text255
Suscripcion_promo	varchar(60)	runlength
Fecha_creacion	timestamp	delta32k

Estilos de distribución con Redshift

Distribución con Redshift

Redshift distribuye las filas de la tabla a cada uno de los sectores del nodo, en función del estilo de distribución de la tabla.



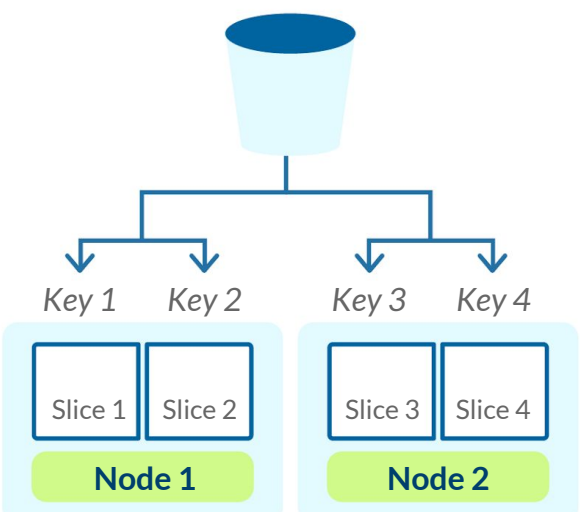
Distribución con Redshift

Distribuir carga significa compartir esa carga de trabajo de una tabla de manera equitativa en los nodos, si no está distribuida correctamente unos nodos trabajan más que otros, y eso se traduce en consultas más lentas.

Distribución con Redshift

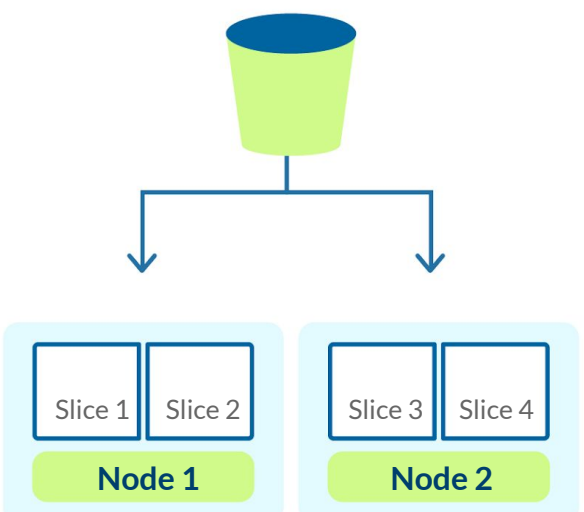
Distribución (Key)

Determinada columna a mismas locaciones



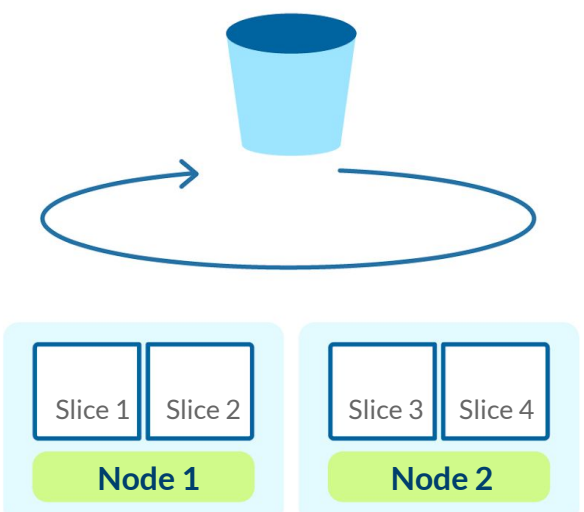
Distribución (All)

Todos los datos se replican en cada nodo



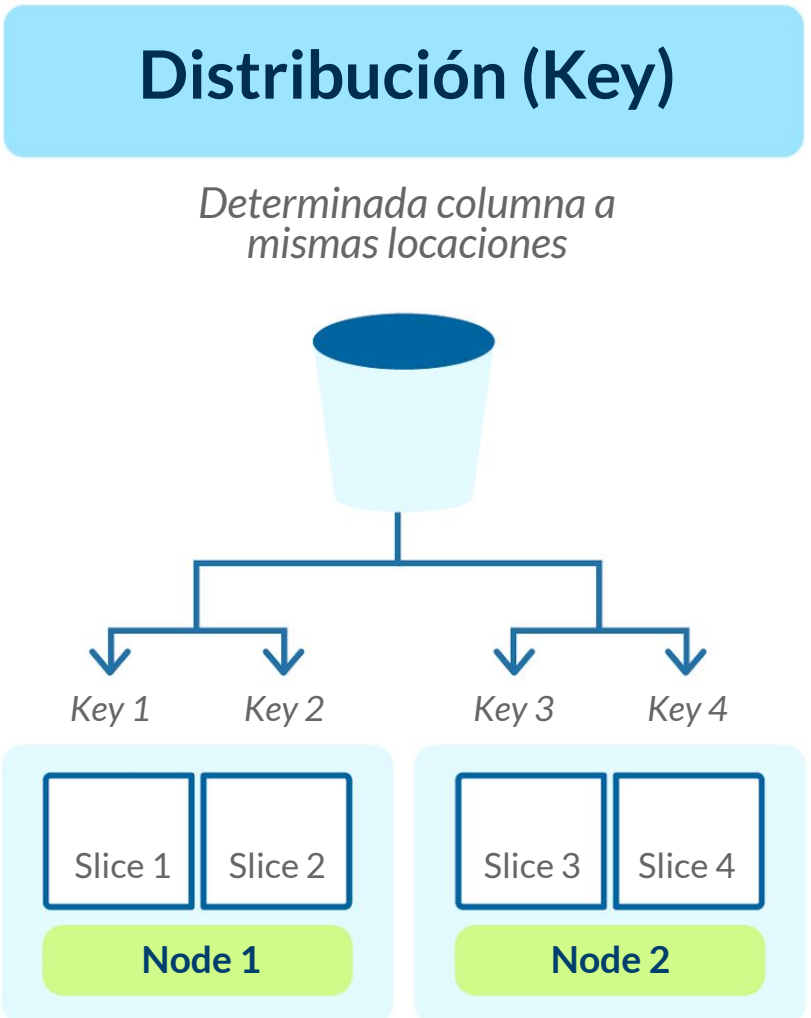
Distribución (Even)

Distribución de round-robin



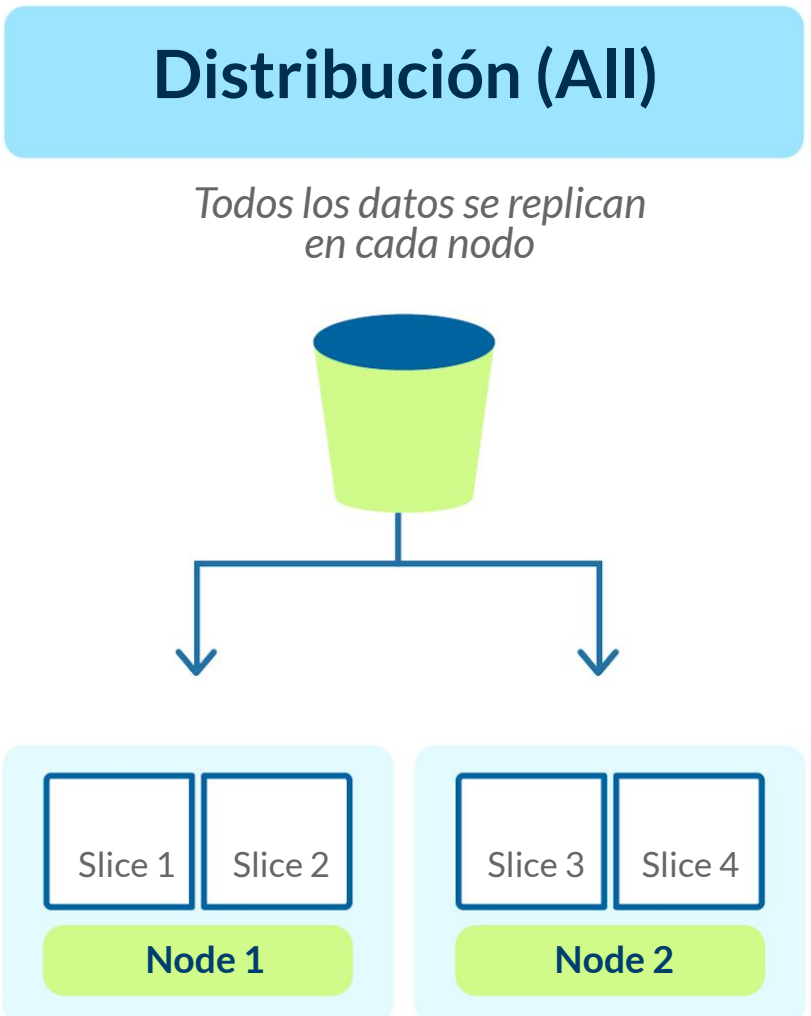
Key

- Mejora los joins y **group by**.
- Los valores llave se almacenan juntos físicamente en cada nodo.



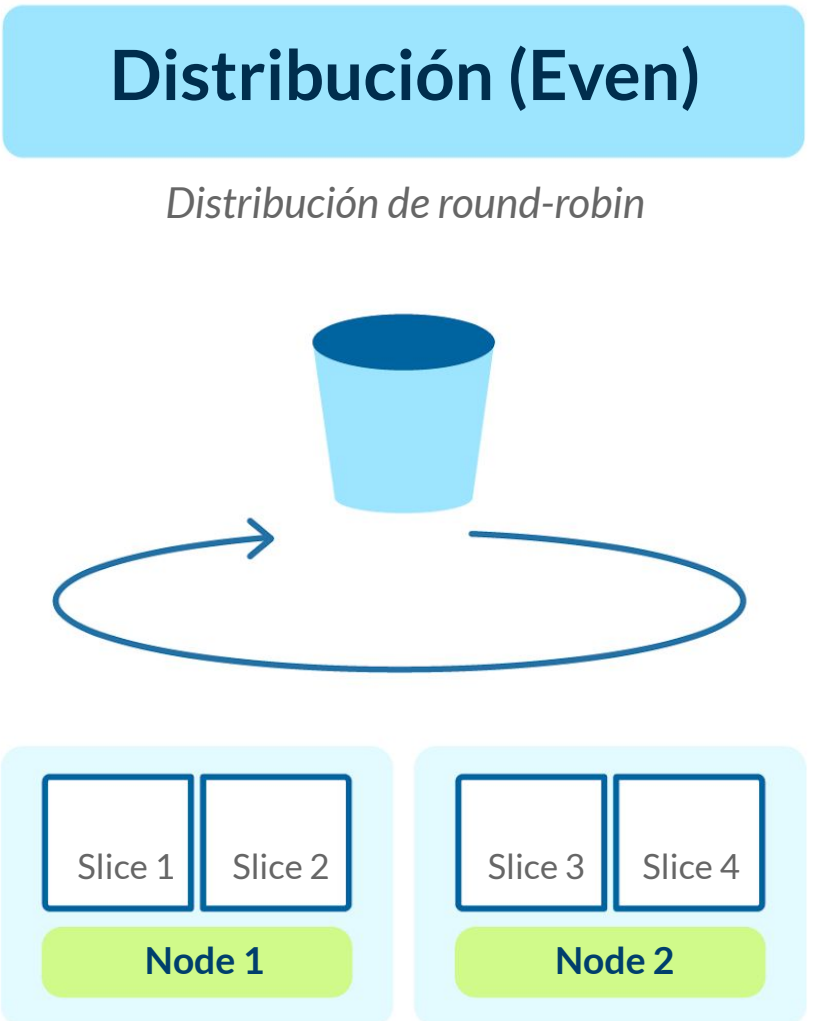
All

- Se usa en tablas pequeñas.
- Se distribuyen todos los datos de la tabla en cada nodo.
- Ocupa más espacio en disco y requiere más tiempo para actualizar, eliminar e insertar.



Even

- Úsala si no es posible una partición tipo key.
- La data se distribuye en todos los nodos y slides de manera rotativa.



Llaves de ordenamiento con Redshift

Llaves de ordenamiento con Redshift

Guardar los datos ordenadamente ofrece muchos beneficios, pues al tener valores mínimos y máximos en los bloques de datos de 1MB es posible descartar lotes innecesarios rápidamente para procesar una consulta.

Llaves de ordenamiento con Redshift

FECHA	TIPO	STATUS
2019-01-01	Expert plus	True
2019-01-02	Expert plus	True
2019-03-05	Basic	True
2020-02-01	Expert	False

Tipos de ordenamiento

- Simple
- Compuesto (COMPOUND)
- Intercalado (INTERLEAVED)

--Compuesto (COMPOUND)

```
Create table empresa_compound(
id int2,
nombre varchar(50),
region varchar(15),
ciudad varchar(20),
segmento varchar(15),
fecha_creacion date,
total_empleados int4)
compound sortkey (region, ciudad,
segmento, fecha_creacion);
```

Compuesto (COMPOUND)

- Ordenamiento por default.
- No funciona bien si se filtran solo por llaves secundarias.
- Mejora operaciones de **group by** y order by.

--Intercalado (INTERLEAVED)

```
Create table empresa_interleaved(
id int2,
nombre varchar(50),
region varchar(15),
ciudad varchar(20),
segmento varchar(15),
fecha_creacion date,
total_empleados int4)
interleaved sortkey (region, ciudad,
segmento, fecha_creacion);
```

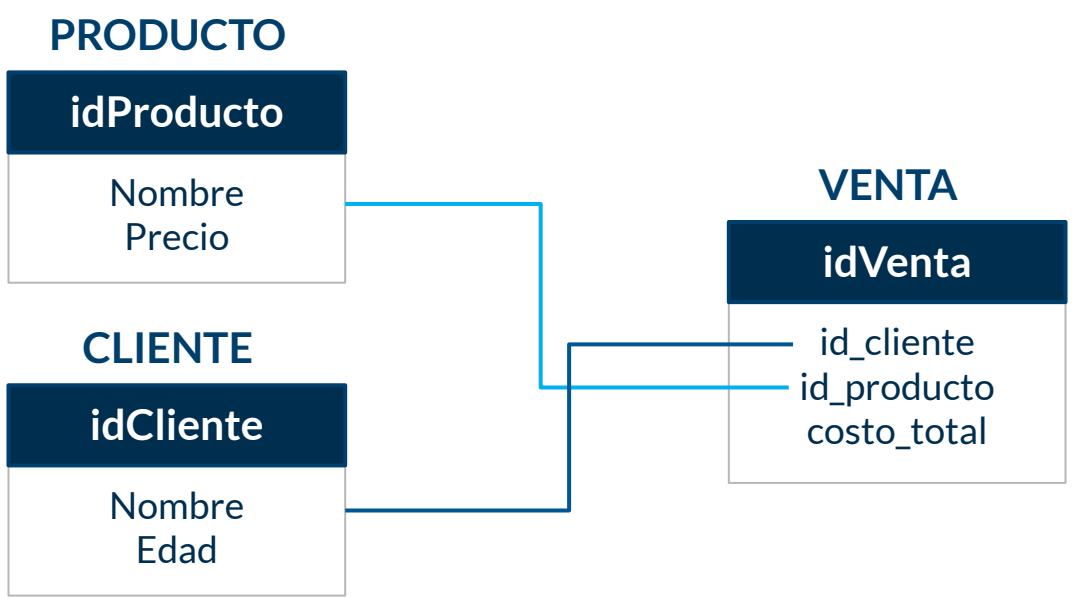
Intercalado (INTERLEAVED)

- No aplicar a columnas de crecimiento masivo como llaves primarias.
- Funciona igual de bien por cada filtro en las columnas por las que está creado.
- La carga es más lenta en estas tablas.

Buenas prácticas para diseñar tablas en Redshift

Constraints

- PRIMARY KEY
- FOREIGN KEY



Columnas

- Usa siempre tipos de datos **date** o **timestamp** para las fechas, si es necesario ajusta tus procesos ETL.
- Tamaños muy elevados degradan el uso de algoritmos de compresión y requieren más almacenamiento en disco.
- Comenta en qué consisten tus columnas en el diccionario de datos.

¿Cómo mantener el performance de tu base de datos?

Analyze

- La operación ANALYZE actualiza los metadatos estadísticos que el planificador de consultas usa para seleccionar planes óptimos.
- La operación de Analyze consume muchos recursos del sistema, por eso si no han cambiado determinado porcentaje de filas en la tabla no se ejecuta.

Analyze

- El siguiente comando define el porcentaje de cambio en las tablas para analyze.

  set analyze_threshold_percent to 10;

- El análisis de estadísticas puede ser por toda la base de datos, tablas en específico o columnas específicas.

Vacuum (Limpieza)

- Reordena las filas y recupera espacio en una tabla especificada o en todas las tablas de la base de datos actual.
- Tipo de Vacuum:
  - FULL
  - SORT ONLY
  - DELETE ONLY
  - REINDEX

Vacuum (Limpieza)

- Vacuum indispone las tablas de modo que hay que ejecutarlo en una ventana de cero transacciones.
- VACUUM omite la fase de ordenación para cualquier tabla que tenga más del 95 por ciento de las filas de la tabla ordenadas.

  vacuum sort only sales to 85 percent;

Vacuum (Sort)

Tabla original

Id	Fecha
10	2019-01-01
20	2019-01-02
30	2019-01-03

Copy

Id	Fecha
10	2020-06-01
20	2020-06-01
30	2020-06-01

Tabla después del copy

Id	Fecha
10	2019-01-01
20	2019-01-02
30	2019-01-03
10	2020-06-01
20	2020-06-01
30	2020-06-01

Vacuum (Sort)

Tabla original

Id	Fecha
10	2019-01-01
20	2019-01-02
30	2019-01-03
10	2020-06-01
20	2020-06-01
30	2020-06-01

Copy 2

Id	Fecha
10	2020-08-01
20	2020-08-02
30	2020-08-03

Tabla después del copy 2

Id	Fecha
10	2019-01-01
20	2019-01-02
30	2019-01-03
10	2020-06-01
20	2020-06-021
30	2020-06-01
10	2020-08-01
20	2020-08-02
30	2020-08-03

Vacuum (Sort)

Tabla original

Id	Fecha
10	2019-01-01
20	2019-01-02
30	2019-01-03
10	2020-06-01
20	2020-06-021
30	2020-06-01
10	2020-08-01
20	2020-08-02
30	2020-08-03

Vacuum sort

Id	Fecha
10	2019-01-01
20	2019-01-02
30	2019-01-03
10	2020-06-01
10	2020-08-01
20	2020-06-021
20	2020-08-02
30	2020-06-01
30	2020-08-03

Vacuum merged

Id	Fecha
10	2019-01-01
10	2020-06-01
10	2020-08-01
20	2019-01-02
20	2020-06-021
20	2020-08-02
30	2019-01-03
30	2020-06-01
30	2020-08-03

Buenas prácticas para diseñar consultas

Buenas prácticas para diseñar consultas

- Evita usar select * from …
- Usar joins por llaves siempre que sea posible.
- Tantas condiciones en el where como sea posible.
- Evita usar funciones en el select.

Buenas prácticas para diseñar consultas

- Usa las columnas “sort” en el **group by** (mismo orden).
- Usa subconsultas con menos de 200 registros.

```
select sum(sales.qtysold)
from sales
where salesid in (select listid from listing where
listtime > '2008-12-26');
```

Buenas prácticas para diseñar consultas

- Si usas **group by** y order by asegúrate que estén en el mismo orden.

```
select  id, nombre, fecha,
sum(total)
group by id, nombre, fecha
order by id, nombre, fecha
```

Buenas prácticas para diseñar consultas

- Filtra lo mismo cuantas veces se pueda en las distintas tablas.

```
select listing.sellerid, sum(sales.qtysold)
from sales, listing
where sales.salesid = listing.listid
and listing.listtime > '2008-12-01'
and sales.saletime > '2008-12-01'
group by 1 order by 1;
```

Comando Copy

Comando Copy

- Procesamiento masivo en paralelo (MPP).
- Un solo llamado para múltiples archivos.
- Compresión de archivos.
- Compatible con s3.

Carga de archivos

- Otorgar permisos al recurso.
- Validar formato de archivos (UTF-8).
- Verificar longitudes de las columnas.
- ¿Existe un delimitador?.
- Comprobar el formato de fechas.
- Particionar los datos en distintos archivos.

¿En cuantos archivos debo dividir mis datos?

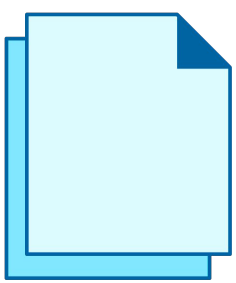
Un múltiplo de mis sectores (slides)

2 sectores por nodo → 4 servirá

data.txt.1
data.txt.2
data.txt.3
data.txt.4

Tamaño

Entre 1 y 125 mb después de compresión es un tamaño óptimo



110 mb

Conclusiones