

# Cloud Data Warehouses

Course 2 | Cloud Data Warehouses

- ▼ Introduction to DWH (Business vs Technical Perspective, CUBE ROLAP, Columnar storage vs Row)

## Business vs Technical Perspective

---

### Operational vs Analytical Business Processes

---



Operational Processes  
*Make it work!*

- Find goods & make orders (for customers)
- Stock and find goods (for inventory staff)
- Pick up & deliver goods (for delivery staff)

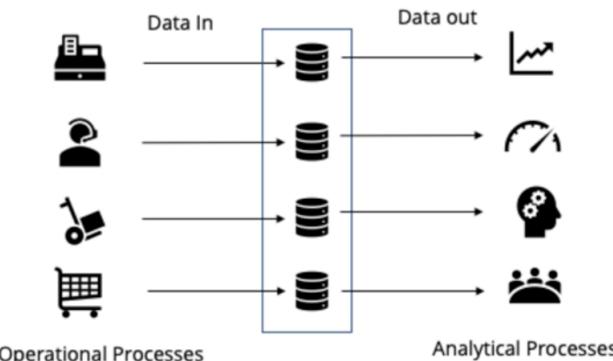


Analytical Processes  
*What is going on?*

- Assess the performance of sales staff (for HR)
- See the effect of different sales channels (for marketing)
- Monitor sales growth (for management)

### Same data source for operational & analytical processes?

---



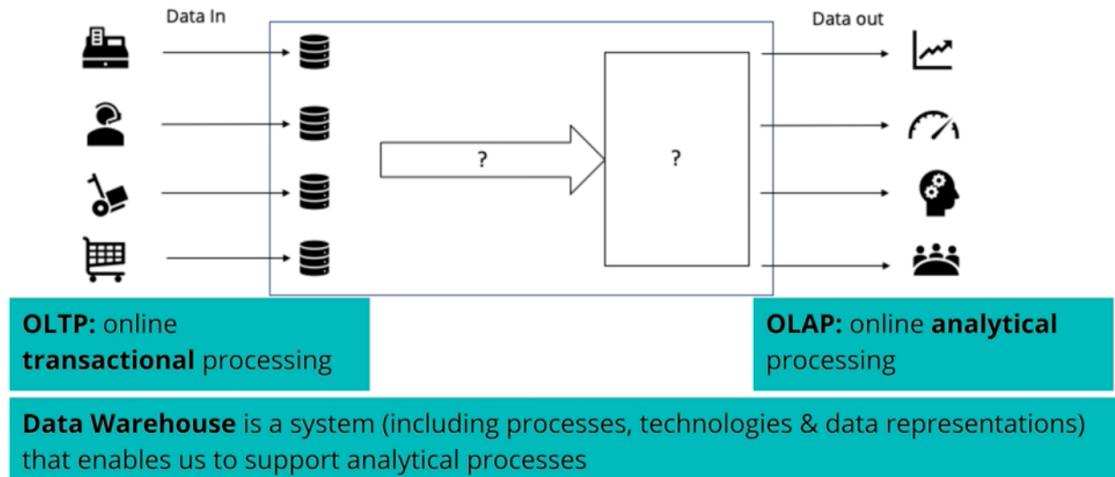
#### Operational Databases

- Excellent for operations
- No redundancy, high integrity

#### Operational Databases

- Too slow for analytics, too many joins
- Too hard to understand

## Solution: Create 2 processing modes, Create a system for them to co-exist



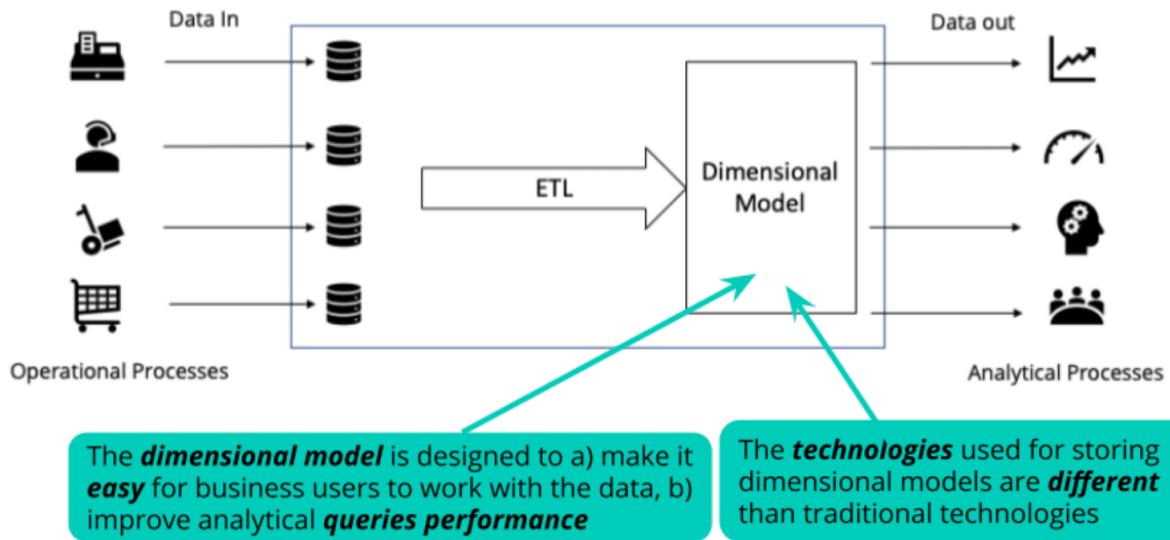
### Data Warehouse Definitions

- A data warehouse is a copy of transaction data specifically structured for query and analysis. - *Kimball*
- A data warehouse is a **subject-oriented, integrated, nonvolatile**, and **time-variant** collection of data in support of management's decisions. - *Inmon*
- A data warehouse is a system that retrieves and consolidates data periodically from the source systems into a **dimensional** or **normalized** data store. It usually keeps years of history and is queried for business intelligence or other analytical activities. It is typically updated in batches, not every time a transaction happens in the source system. - *Rainard*

### Data Warehouse: Technical Perspective

Extract the data from the source systems used for operations, transform the data, and load it into a dimensional model

### DWH: Tech Perspective



## Dimensional Model Review

### Goals of the Star Schema

- Easy to understand
- Fast analytical query performance

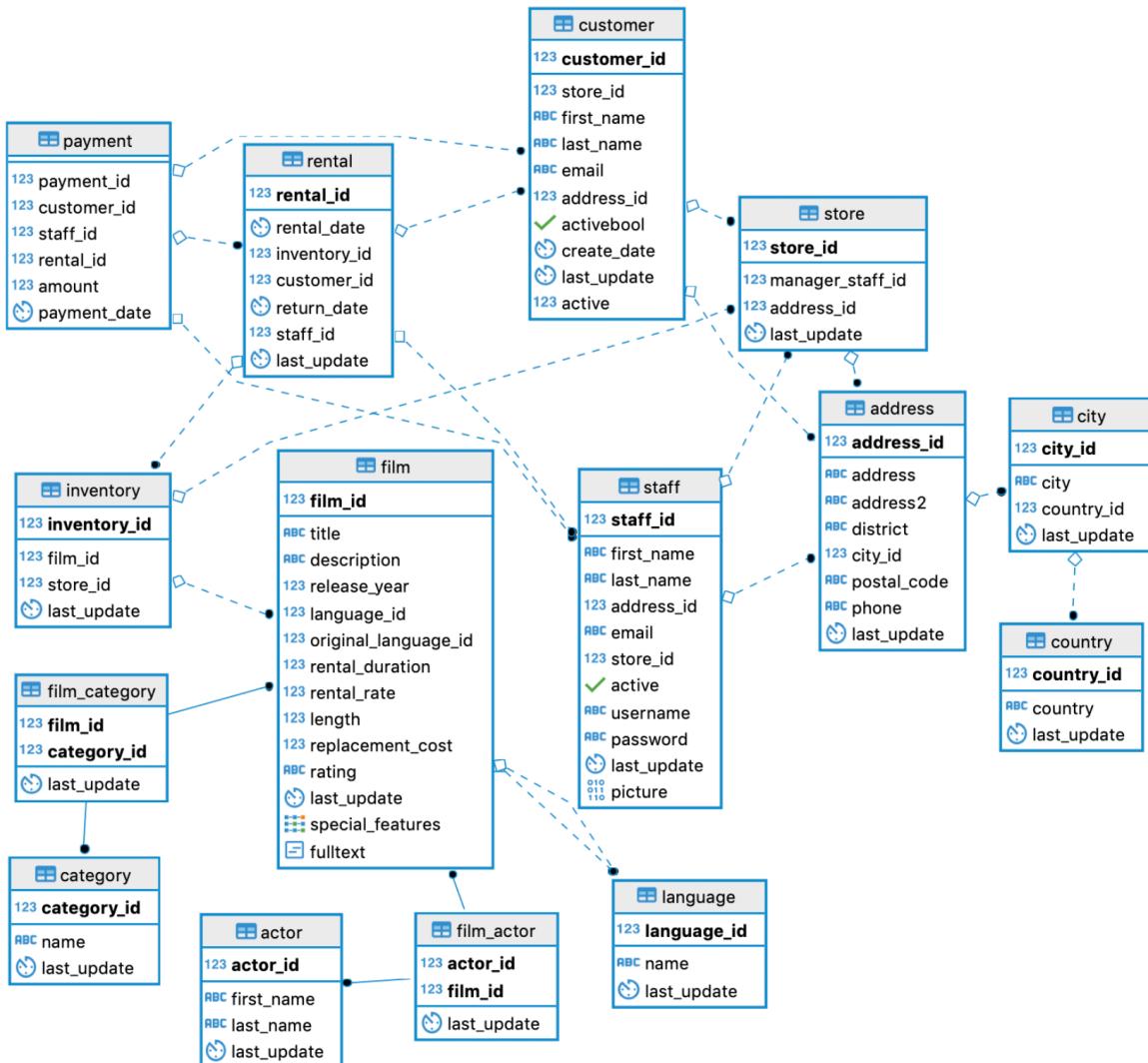
### Fact Tables

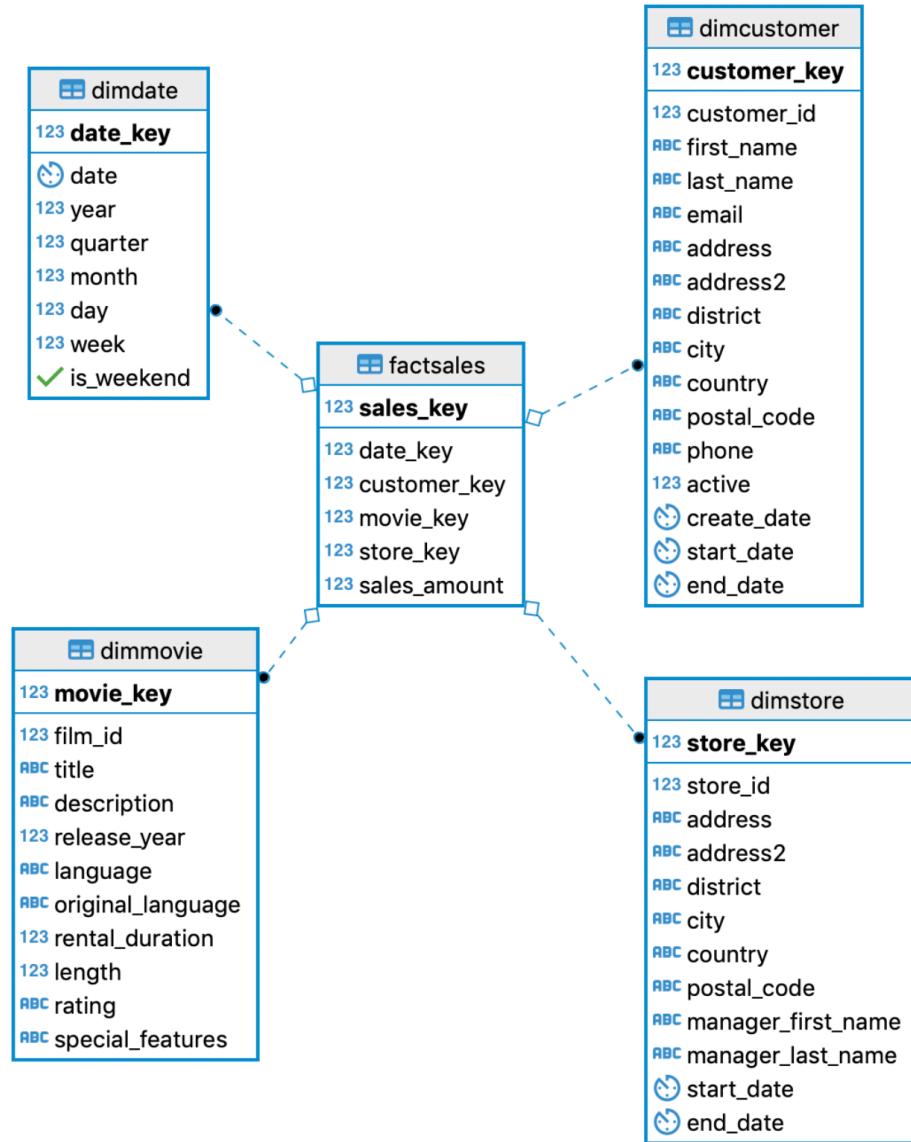
- Record business events, like an order, a phone call, a book review
- Fact tables columns record events recorded in quantifiable metrics like quantity of an item, duration of a call, a book rating

### Dimension Tables

- Record the context of the business events, e.g. who, what, where, why, etc..
- Dimension tables columns contain attributes like the store at which an item is purchased or the customer who made the call, etc.

### ▼ DEMO Snowflake to Star (Fact & Dimension) Schema





```

!PGPASSWORD=student createdb -h 127.0.0.1 -U student pagila
!PGPASSWORD=student psql -q -h 127.0.0.1 -U student -d pagila -f Data/pagila-schema.sql
!PGPASSWORD=student psql -q -h 127.0.0.1 -U student -d pagila -f Data/pagila-data.sql

%load_ext sql

DB_ENDPOINT = "127.0.0.1"
DB = 'pagila'
DB_USER = 'student'
DB_PASSWORD = 'student'
DB_PORT = '5432'

# postgresql://username:password@host:port/database
conn_string = "postgresql://{}:{}@{}:{}/{}" \
    .format(DB_USER, DB_PASSWORD, DB_ENDPOINT, DB_PORT, DB)

print(conn_string)
%sql $conn_string

## Explore the (snowflake) 3NF Schema first
nStores = %sql select count(*) from store;
nFilms = %sql select count(*) from film;
nCustomers = %sql select count(*) from customer;
nRentals = %sql select count(*) from rental;

```

```

nPayment = %sql select count(*) from payment;
nStaff = %sql select count(*) from staff;
nCity = %sql select count(*) from city;
nCountry = %sql select count(*) from country;

print("nFilms\t\t=", nFilms[0][0])
print("nCustomers\t=", nCustomers[0][0])
print("nRentals\t=", nRentals[0][0])
print("nPayment\t=", nPayment[0][0])
print("nStaff\t\t=", nStaff[0][0])
print("nStores\t\t=", nStores[0][0])
print("nCities\t\t=", nCity[0][0])
print("nCountry\t\t=", nCountry[0][0])

## Build the Star Schema now
%%sql
CREATE TABLE dimDate
(
    date_key integer NOT NULL PRIMARY KEY,
    date date NOT NULL,
    year smallint NOT NULL,
    quarter smallint NOT NULL,
    month smallint NOT NULL,
    day smallint NOT NULL,
    week smallint NOT NULL,
    is_weekend boolean
);

CREATE TABLE dimCustomer
(
    customer_key SERIAL PRIMARY KEY,
    customer_id smallint NOT NULL,
    first_name varchar(45) NOT NULL,
    last_name varchar(45) NOT NULL,
    email varchar(50),
    address varchar(50) NOT NULL,
    address2 varchar(50),
    district varchar(20) NOT NULL,
    city varchar(50) NOT NULL,
    country varchar(50) NOT NULL,
    postal_code varchar(10),
    phone varchar(20) NOT NULL,
    active smallint NOT NULL,
    create_date timestamp NOT NULL,
    start_date date NOT NULL,
    end_date date NOT NULL
);

CREATE TABLE dimMovie
(
    movie_key      SERIAL PRIMARY KEY,
    film_id        smallint NOT NULL,
    title          varchar(255) NOT NULL,
    description     text,
    release_year   year,
    language        varchar(20) NOT NULL,
    original_language varchar(20),
    rental_duration smallint NOT NULL,
    length          smallint NOT NULL,
    rating          varchar(5) NOT NULL,
    special_features varchar(60) NOT NULL
);
CREATE TABLE dimStore
(
    store_key      SERIAL PRIMARY KEY,
    store_id        smallint NOT NULL,
    address         varchar(50) NOT NULL,
    address2        varchar(50),
    district        varchar(20) NOT NULL,
    city            varchar(50) NOT NULL,
    country         varchar(50) NOT NULL,
    postal_code     varchar(10),
    manager_first_name varchar(45) NOT NULL,
    manager_last_name varchar(45) NOT NULL,
    start_date      date NOT NULL,
    end_date        date NOT NULL
);
CREATE TABLE factSales
(
    sales_key      SERIAL PRIMARY KEY,
    date_key        INT NOT NULL REFERENCES dimDate(date_key),

```

```

customer_key      INT NOT NULL REFERENCES dimCustomer(customer_key),
movie_key        INT NOT NULL REFERENCES dimMovie(movie_key),
store_key        INT NOT NULL REFERENCES dimStore(store_key),
sales_amount     decimal(5,2) NOT NULL
);

##ETL from the 3NF into Fact & Dimension tables
%%sql
INSERT INTO dimDate (date_key, date, year, quarter, month, day, week, is_weekend)
SELECT DISTINCT(TO_CHAR(payment_date :: DATE, 'yyyyMMDD')::integer) AS date_key,
       date(payment_date)                               AS date,
       EXTRACT(year FROM payment_date)                AS year,
       EXTRACT(quarter FROM payment_date)             AS quarter,
       EXTRACT(month FROM payment_date)               AS month,
       EXTRACT(day FROM payment_date)                 AS day,
       EXTRACT(week FROM payment_date)                AS week,
       CASE WHEN EXTRACT(ISODOW FROM payment_date) IN (6, 7) THEN true ELSE false END AS is_weekend
FROM payment;

INSERT INTO dimCustomer (customer_key, customer_id, first_name, last_name, email, address, address2, district, city, country, postal_code, active, create_date, start_date, end_date)
SELECT c.customer_id AS customer_key,
       c.customer_id,
       c.first_name,
       c.last_name,
       c.email,
       a.address,
       a.address2,
       a.district,
       ci.city,
       co.country,
       a.postal_code,
       a.phone,
       c.active,
       c.create_date,
       now()          AS start_date,
       now()          AS end_date
FROM customer c
JOIN address a  ON (c.address_id = a.address_id)
JOIN city ci   ON (a.city_id = ci.city_id)
JOIN country co ON (ci.country_id = co.country_id);

INSERT INTO dimMovie (movie_key, film_id, title, description, release_year, language, original_language, rental_duration, length, rating, special_features)
SELECT f.film_id      AS movie_key,
       f.film_id,
       f.title,
       f.description,
       f.release_year,
       l.name          AS language,
       orig_lang.name AS original_language,
       f.rental_duration,
       f.length,
       f.rating,
       f.special_features
FROM film f
JOIN language l           ON (f.language_id=l.language_id)
LEFT JOIN language orig_lang ON (f.original_language_id = orig_lang.language_id);

INSERT INTO dimStore (store_key, store_id, address, address2, district, city, country, postal_code, manager_first_name, manager_last_name, start_date, end_date)
SELECT s.store_id     AS store_key,
       s.store_id,
       a.address,
       a.address2,
       a.district,
       c.city,
       co.country,
       a.postal_code,
       st.first_name AS manager_first_name,
       st.last_name  AS manager_last_name,
       now()          AS start_date,
       now()          AS end_date
FROM store s
JOIN staff st  ON (s.manager_staff_id = st.staff_id)
JOIN address a  ON (s.address_id = a.address_id)
JOIN city c    ON (a.city_id = c.city_id)
JOIN country co ON (c.country_id = co.country_id);

INSERT INTO factSales (date_key, customer_key, movie_key, store_key, sales_amount)
SELECT TO_CHAR(p.payment_date :: DATE, 'yyyyMMDD')::integer AS date_key ,
       p.customer_id                                AS customer_key,
       i.film_id                                    AS movie_key,

```

```

    i.store_id                                AS store_key,
    p.amount                                    AS sales_amount
  FROM payment p
  JOIN rental r   ON ( p.rental_id = r.rental_id )
  JOIN inventory i ON ( r.inventory_id = i.inventory_id );

##queries are not much faster and easy to write
##the one below on the new schema
%%time
%%sql
SELECT dimMovie.title, dimDate.month, dimCustomer.city, sum(sales_amount) as revenue
FROM factSales
JOIN dimMovie  on (dimMovie.movie_key      = factSales.movie_key)
JOIN dimDate   on (dimDate.date_key        = factSales.date_key)
JOIN dimCustomer on (dimCustomer.customer_key = factSales.customer_key)
group by (dimMovie.title, dimDate.month, dimCustomer.city)
order by dimMovie.title, dimDate.month, dimCustomer.city, revenue desc;

##vs the old schema one
%%time
%%sql
SELECT f.title, EXTRACT(month FROM p.payment_date) as month, ci.city, sum(p.amount) as revenue
FROM payment p
JOIN rental r   ON ( p.rental_id = r.rental_id )
JOIN inventory i ON ( r.inventory_id = i.inventory_id )
JOIN film f ON ( i.film_id = f.film_id )
JOIN customer c ON ( p.customer_id = c.customer_id )
JOIN address a ON ( c.address_id = a.address_id )
JOIN city ci ON ( a.city_id = ci.city_id )
group by (f.title, month, ci.city)
order by f.title, month, ci.city, revenue desc;

!PGPASSWORD=student pg_dump -h 127.0.0.1 -U student pagila > Data/pagila-star.sql

```

## ▼ DEMO Excercise 2 CUBE

```

## CUBE
## Group by CUBE (dim1, dim2, ..) , produces all combinations of different lenghts in one go.
## This view could be materialized in a view and queried which would save lots repetitive aggregations

##this cube below
SELECT dimDate.month, dimStore.country, sum(sales_amount) as revenue
FROM factSales
JOIN dimDate   on (dimDate.date_key        = factSales.date_key)
JOIN dimStore on (dimStore.store_key = factSales.store_key)
GROUP by cube(dimDate.month, dimStore.country);

##is equivalent to the code below but runs it half the time
SELECT NULL as month, NULL as country, sum(sales_amount) as revenue
FROM factSales
UNION all
SELECT NULL, dimStore.country, sum(sales_amount) as revenue
FROM factSales
JOIN dimStore on (dimStore.store_key = factSales.store_key)
GROUP by dimStore.country
UNION all
SELECT cast(dimDate.month as text) , NULL, sum(sales_amount) as revenue
FROM factSales
JOIN dimDate on (dimDate.date_key = factSales.date_key)
GROUP by dimDate.month
UNION all
SELECT cast(dimDate.month as text), dimStore.country, sum(sales_amount) as revenue
FROM factSales
JOIN dimDate   on (dimDate.date_key        = factSales.date_key)
JOIN dimStore on (dimStore.store_key = factSales.store_key)
GROUP by (dimDate.month, dimStore.country)

```

## ▼ Column Format in ROLAP

```

...
ROLAP means doing the OLAP cubes on the fly from a
relational database instead of buying a specific OLAP database for handling these (MOLAP).

A feature of ROLAP is that you can retrieve data column-wise from the disk
instead of row-wise - so you don't have to load for the whole row to get maybe just one value.

```

```

Performance increases a lot. Demo below
```
%load_ext sql
##create database
!sudo -u postgres psql -c 'CREATE DATABASE reviews;'

!wget http://examples.citusdata.com/customer_reviews_1998.csv.gz
!wget http://examples.citusdata.com/customer_reviews_1999.csv.gz

!gzip -d customer_reviews_1998.csv.gz
!gzip -d customer_reviews_1999.csv.gz

!mv customer_reviews_1998.csv /tmp/customer_reviews_1998.csv
!mv customer_reviews_1999.csv /tmp/customer_reviews_1999.csv
##connect to db
DB_ENDPOINT = "127.0.0.1"
DB = 'reviews'
DB_USER = 'student'
DB_PASSWORD = 'student'
DB_PORT = '5432'

# postgresql://username:password@host:port/database
conn_string = "postgresql://{}:{}@{}:{}/{}" \
    .format(DB_USER, DB_PASSWORD, DB_ENDPOINT, DB_PORT, DB)

print(conn_string)
%sql $conn_string

#create a table with normal row storage and copy data inside
%%sql
DROP TABLE IF EXISTS customer_reviews_row;
CREATE TABLE customer_reviews_row
(
    customer_id TEXT,
    review_date DATE,
    review_rating INTEGER,
    review_votes INTEGER,
    review_helpful_votes INTEGER,
    product_id CHAR(10),
    product_title TEXT,
    product_sales_rank BIGINT,
    product_group TEXT,
    product_category TEXT,
    product_subcategory TEXT,
    similar_product_ids CHAR(10)[]
)
%%sql
COPY customer_reviews_row FROM '/tmp/customer_reviews_1998.csv' WITH CSV;
COPY customer_reviews_row FROM '/tmp/customer_reviews_1999.csv' WITH CSV;

##create table with columnar storage now
%%sql

-- load extension first time after install
CREATE EXTENSION cstore_fdw;

-- create server object
CREATE SERVER cstore_server FOREIGN DATA WRAPPER cstore_fdw;

%%sql
-- create foreign table
DROP FOREIGN TABLE IF EXISTS customer_reviews_col;

CREATE FOREIGN TABLE customer_reviews_col
(
    customer_id TEXT,
    review_date DATE,
    review_rating INTEGER,
    review_votes INTEGER,
    review_helpful_votes INTEGER,
    product_id CHAR(10),
    product_title TEXT,
    product_sales_rank BIGINT,
    product_group TEXT,
    product_category TEXT,
    product_subcategory TEXT,
    similar_product_ids CHAR(10)[]
)
SERVER cstore_server
OPTIONS(compression 'pglz');

```

```

%%sql
COPY customer_reviews_col FROM '/tmp/customer_reviews_1998.csv' WITH CSV;
COPY customer_reviews_col FROM '/tmp/customer_reviews_1999.csv' WITH CSV;

##compare performance
%%time
%%sql
SELECT
    customer_id, review_date, review_rating, product_id, product_title
FROM
    customer_reviews_row
WHERE
    customer_id = 'A27T7HVDXA3K2A' AND
    product_title LIKE '%Dune%' AND
    review_date >= '1998-01-01' AND
    review_date <= '1998-12-31';

%%time
%%sql
SELECT
    customer_id, review_date, review_rating, product_id, product_title
FROM
    customer_reviews_col
WHERE
    customer_id = 'A27T7HVDXA3K2A' AND
    product_title LIKE '%Dune%' AND
    review_date >= '1998-01-01' AND
    review_date <= '1998-12-31';

#another
%%time
%%sql
SELECT product_title, avg(review_rating)
FROM customer_reviews_col
WHERE review_date >= '1995-01-01'
    AND review_date <= '1998-12-31'
GROUP BY product_title
ORDER by product_title
LIMIT 20;

%%time
%%sql
SELECT product_title, avg(review_rating)
FROM customer_reviews_row
WHERE review_date >= '1995-01-01'
    AND review_date <= '1998-12-31'
GROUP BY product_title
ORDER by product_title
LIMIT 20;

```

▼ **Intro to Cloud Computing and AWS** (Creating IAM Roles / IAM Users / Security Groups, Launching/Deleting Redshift Clusters, Creating S3 buckets)

---

## What is Cloud Computing?

Using a **network of remote servers** hosted on the internet to **store, manage, and process data**, rather than a local server or a personal computer.



### Cloud computing:

the practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer.

## Amazon Web Services

Amazon Web Services is one of the largest providers in the cloud computing industry, with over 140 services in compute, storage, databases, networking, developer tools, security, and more. In this lesson, we'll learn about a few essential tools and services in AWS and practice using them. These services can be accessed in three different ways: the AWS Management Console, the Command Line Interface (CLI), or Software Development Kits (SDKs), which can be used in combination.

We'll start with the AWS Management Console, which is the web user interface. The AWS CLI is a useful way to control and automate your services with code, and SDKs allow you to easily integrate services with your applications through APIs built around specific languages and platforms.

### Dimensional Model Storage on AWS

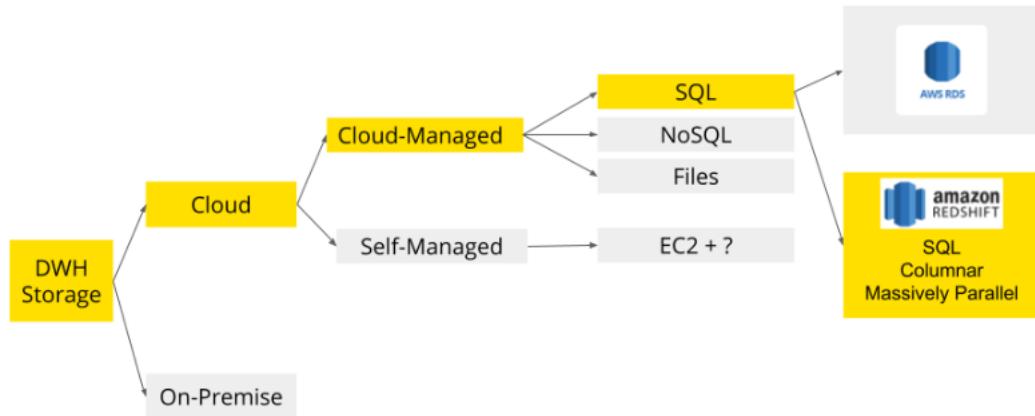
#### Cloud-Managed

- Amazon RDS, Amazon DynamoDB, Amazon S3
- Re-use of expertise; way less IT Staff for security, upgrades, etc. and way less OpEx
- Deal with complexity with techniques like: "Infrastructure as code"

#### Self-Managed

- EC2 + Postgresql, EC2 + Cassandra
- EC2 + Unix FS
- Always "catch-all" option if needed

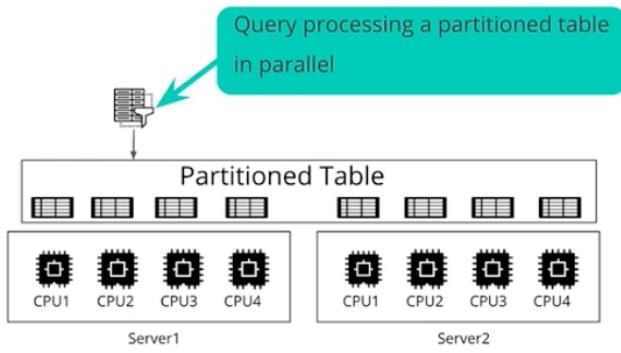
## DWH Dimensional Model Storage on AWS



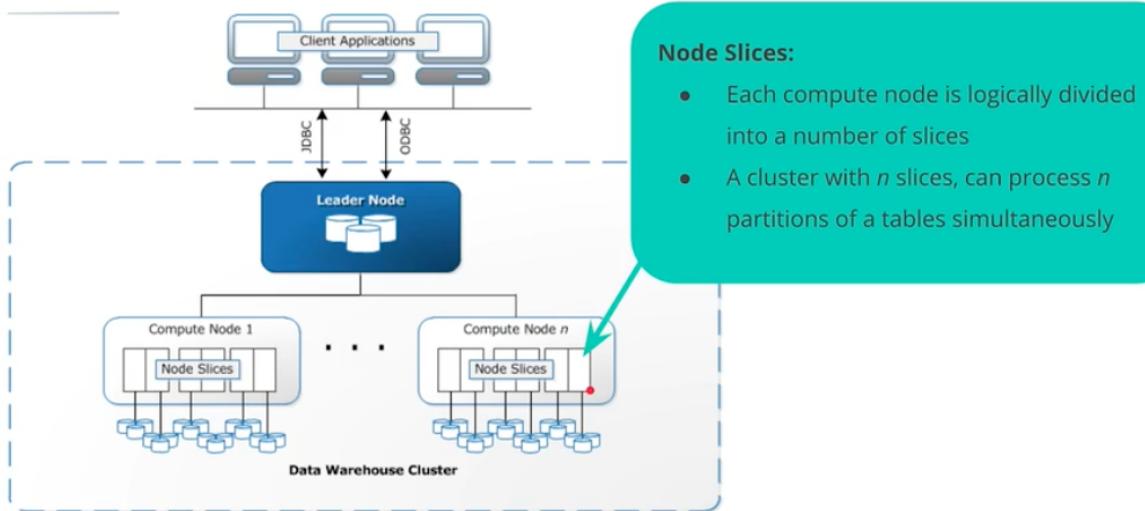
## Amazon Redshift Technology



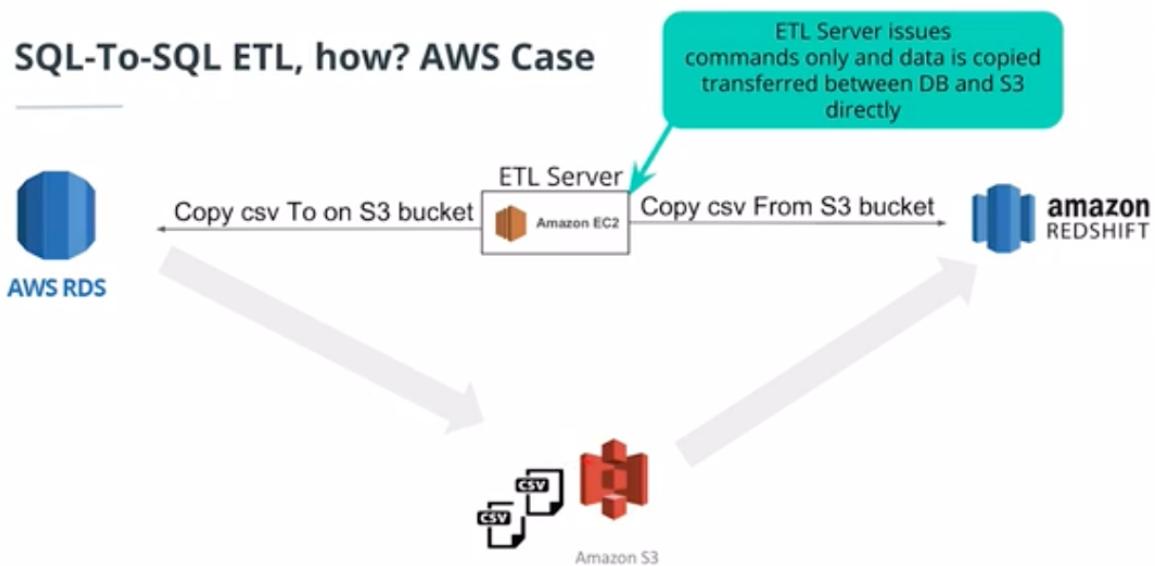
- Massively Parallel Processing (MPP) databases **parallelize the execution of one query on multiples CPUs/machines**
- **How? A table is partitioned and partitions are processed in parallel**
- Amazon Redshift is a *cloud-managed*, column-oriented, MPP database
- Other examples include Teradata Aster, Oracle ExaData and Azure SQL



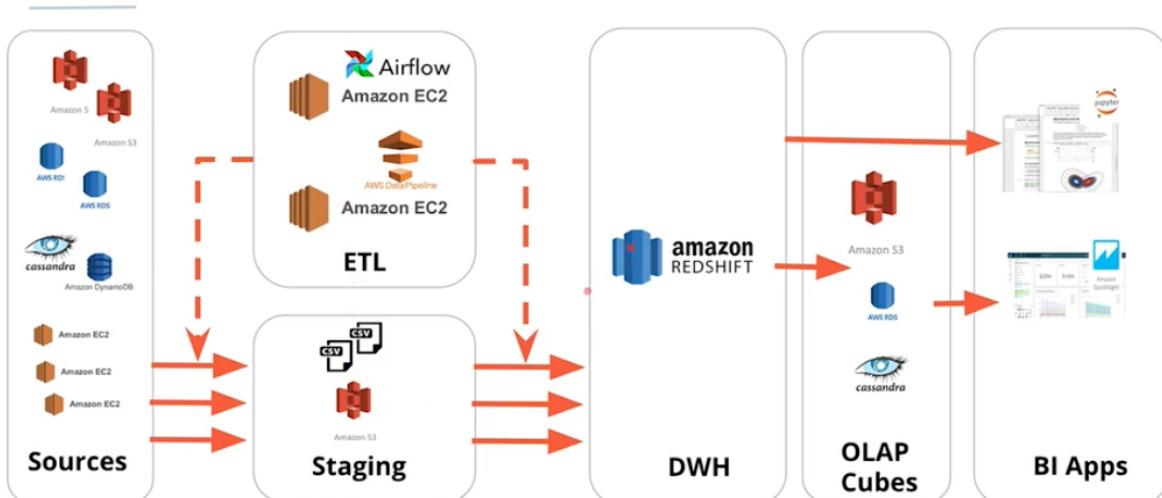
## Redshift Architecture: Slices



## SQL-To-SQL ETL, how? AWS Case



## Redshift & ETL in Context



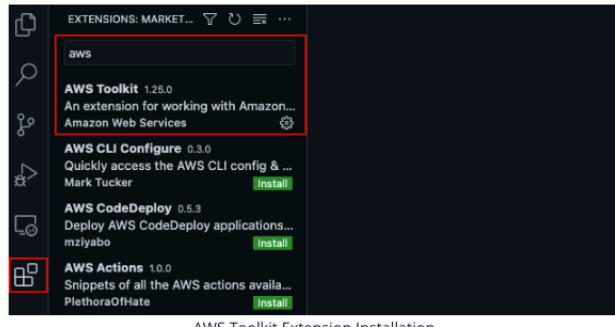
### ▼ Creating AWS Resources

#### ▼ CONNECT VSCode with AWS

Connect VSCode with AWS

Install AWS Toolkit Extension in VS Code

- Open **Extensions** tab
- Search **AWS**
- Install **AWS Toolkit**



AWS Toolkit Extension Installation

**Step 1:** Create a new file called "credentials" with the following details:

```
[default]
aws_access_key_id=COPY_FROM_CLASSROOM
aws_secret_access_key=COPY_FROM_CLASSROOM
aws_session_token=COPY_FROM_CLASSROOM
```

Replace `COPY_FROM_CLASSROOM` with the values you see when you click on the "Launch AWS Gateway" button in the classroom as shown below.

 OPEN AWS CONSOLE

**Step 2:** Create a new file called "config" with the following details:

```
[default]
region=us-west-2
output=json
```

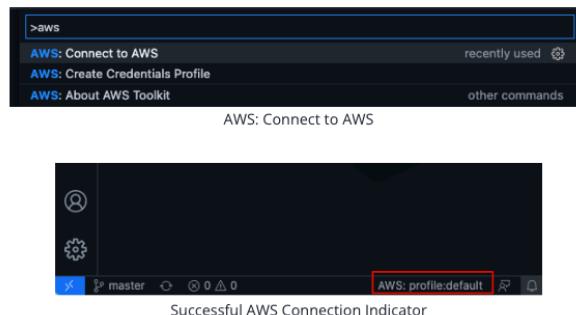
**Step 3:** Save the "credentials" and "config" file at the following location

| Operating system | Location of files                           |
|------------------|---------------------------------------------|
| Linux and macOS  | <code>~/.aws/config</code>                  |
|                  | <code>~/.aws/credentials</code>             |
| Windows          | <code>%USERPROFILE%\.aws\config</code>      |
|                  | <code>%USERPROFILE%\.aws\credentials</code> |

**Note** - If the `.aws` directory is not present, please create it first, and then save the files at the location specified in the table above.

#### Connect to AWS

- Once the credentials are saved, open the VS Code's command palette by pressing `Ctrl+Shift+P` for Windows/Linux OR `Cmd+Shift+P` for Mac, and select `AWS: Connect to AWS`
- On a successful connection `AWS: profile:default` at the bottom left of VS Code window.

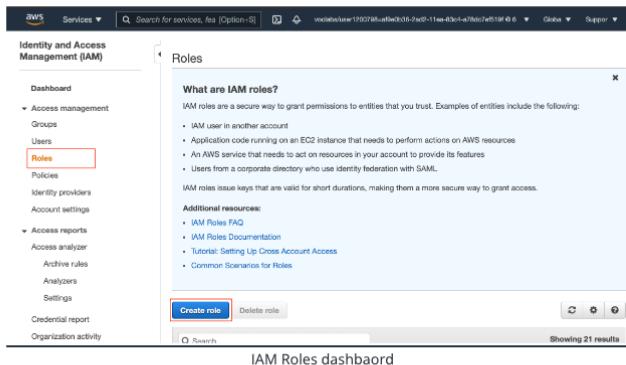


#### ▼ CREATE IAM Role

## Create an IAM Role

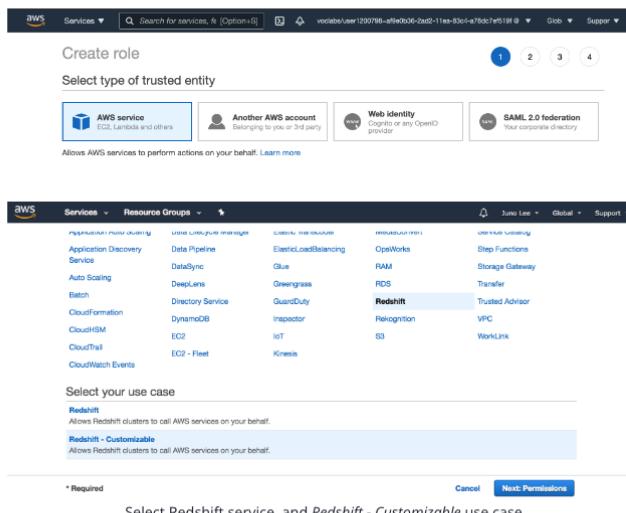
Here, you'll create an IAM role that you will later attach to your Redshift cluster to enable your cluster to load data from Amazon S3 buckets. Read more about IAM roles and Redshift [here](#).

1. Once you have signed into the AWS management console, navigate to the [IAM service dashboard](#).
2. In the left navigation pane, choose **Roles**.
3. Choose **Create role**.



4. In the **AWS Service** group as the trusted entity, and choose **Redshift** service.
5. Under **Select your use case**, choose **Redshift - Customizable**, and then **Next: Permissions**.

4. In the **AWS Service** group as the trusted entity, and choose **Redshift** service.
5. Under **Select your use case**, choose **Redshift - Customizable**, and then **Next: Permissions**.



6. On the **Attach permissions policies** page, search for and select the **AmazonS3ReadOnlyAccess** policy, and then click on the **Next: Tags** button.
7. Tags are optional. Click on the **Next: Review** button.

6. On the **Attach permissions policies** page, search for and select the **AmazonS3ReadOnlyAccess** policy, and then click on the **Next: Tags** button.  
 7. Tags are optional. Click on the **Next: Review** button.

The screenshot shows the 'Attach permissions policies' step of the 'Create role' wizard. A search bar at the top right is set to 's3'. Below it is a table titled 'Showing 4 results' with columns 'Policy name', 'Used as', and 'Description'. The 'AmazonS3ReadOnlyAccess' policy is selected (indicated by a checked checkbox) and highlighted in blue. Other policies listed are 'AmazonS3FullAccess', 'AmazonS3ReadWriteAccess', and 'QuickSightAccessForS3StorageManagement...'. At the bottom right of the table are buttons for 'Cancel', 'Previous', and 'Next: Tags'.

8. For **Role name**, enter `myRedshiftRole`, and then choose **Create Role**.

The screenshot shows the 'Review' step of the 'Create role' wizard. It displays the role configuration: 'Role name' is 'myRedshiftRole', 'Role description' is 'Allows Redshift clusters to call AWS services on your behalf.', and the 'Policies' section shows 'AmazonS3ReadOnlyAccess' attached. At the bottom right are buttons for 'Cancel', 'Previous', and 'Create role'.

9. You will see a success message when the new role will be created.

The screenshot shows the success message 'The role myRedshiftRole has been created.' followed by a table of roles. The table has columns 'Role name', 'Trusted entities', and 'Last activity'. A single row is shown for 'myRedshiftRole', which is highlighted with a red border. Below the table is the message 'Role created successfully'.

*That's great! On the next page, you'll learn to **attach this role to a new/existing cluster***

## ▼ CREAT Security Group EC2

## Create Security Group

Here, you'll create a security group you will later use to authorize access to your Redshift cluster.

*A security group will act as firewall rules for your Redshift cluster to control inbound and outbound traffic.*

1. Navigate to the [EC2 service](#)

The screenshot shows the AWS Services dashboard. The 'Compute' section is expanded, and 'EC2' is highlighted with a red box. Other services like Lightsail, Lambda, and Batch are also listed. The 'Recently visited' section includes EC2, IAM, and CloudFormation. The 'All services' grid contains various AWS products such as Machine Learning, Storage, Blockchain, Satellite, AR & VR, and Application Integration. A red box highlights the 'EC2' link in the 'Recently visited' list.

Navigate to any service

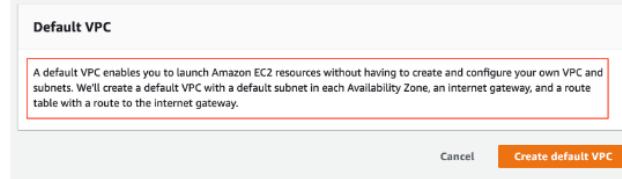
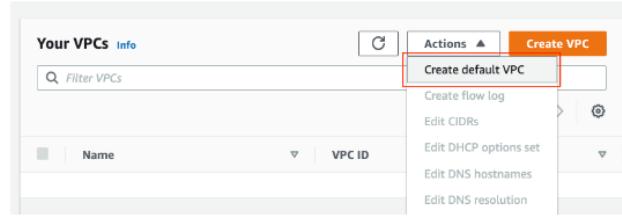
2. Under **Network and Security** in the left navigation pane, select **Security Groups**. Click the **Create Security Group** button to launch a wizard.

The screenshot shows the 'Network & Security' section of the AWS console. The 'Security Groups' section is highlighted with a red box. It displays a list of existing security groups with columns for Name, Security group ID, Security group name, and VPC ID. A red box highlights the 'Create security group' button at the top right of the table.

| Name | Security group ID     | Security group name | VPC ID            |
|------|-----------------------|---------------------|-------------------|
| -    | sg-0377c416a9e9eb5    | launch-wizard-2     | vpc-cb298dbe      |
| -    | sg-046d7262b788c79c   | default             | vpc-049ae40d16fc6 |
| -    | sg-04ed0c8373bb83c79  | launch-wizard-3     | vpc-cb298dbe      |
| -    | sg-09617cb13bb8311606 | launch-wizard-4     | vpc-cb298dbe      |
| -    | sg-0b05374e50bed3f29  | launch-wizard-1     | vpc-cb298dbe      |
| -    | sg-653b516c           | default             | vpc-cb298dbe      |

3. In the *Create security group* wizard, enter the basic details.

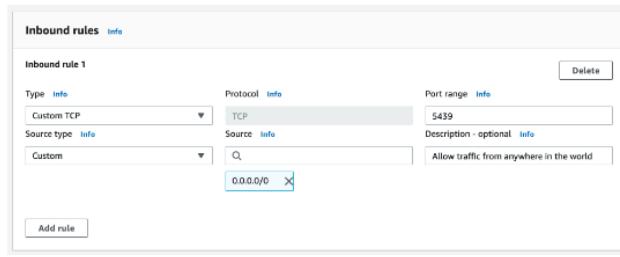
| Section       | Field               | Value                                                                                                                                                                                      |
|---------------|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Basic details | Security group name | redshift_security_group                                                                                                                                                                    |
|               | Description         | Authorise redshift cluster access                                                                                                                                                          |
| VPC           |                     | Choose the default VPC<br>It is a VPC in a default region,<br>and has a public subnet in each Availability Zone.<br>If a default VPC doesn't show up, <a href="#">create a default VPC</a> |



- 
4. In the *Inbound rules* section, click on **Add Rule** and enter the following values:

| Section       | Field       | Value                                                                                   |
|---------------|-------------|-----------------------------------------------------------------------------------------|
| Inbound rules | Type        | Custom TCP Rule                                                                         |
|               | Protocol    | TCP                                                                                     |
|               | Port range  | 5439<br>The default port for Amazon Redshift is 5439, but your port might be different. |
|               | Source type | Custom                                                                                  |
|               | Source      | <b>0.0.0.0/0</b><br>(Anywhere in the world)                                             |

**Important:** Using **0.0.0.0/0** is not recommended for anything other than demonstration purposes because it allows access from any computer on the internet. In a real environment, you would create inbound rules based on your own network settings.



The screenshot shows the 'Inbound rules' configuration page. It displays a single rule named 'Inbound rule 1'. The rule details are as follows:

- Type: Custom TCP
- Protocol: TCP
- Port range: 5439
- Source type: Custom
- Source: 0.0.0.0/0 (Anywhere in the world)
- Description: optional - Allow traffic from anywhere in the world

An 'Add rule' button is visible at the bottom left of the form.

5. Outbound rules allow traffic to anywhere by default.



The screenshot shows the 'Outbound rules' configuration page, which is currently empty.

## Inbound rules

5. Outbound rules allow traffic to anywhere by default.

Outbound rules

6. Click on the *Create security group* button at the bottom. You will see a success message.

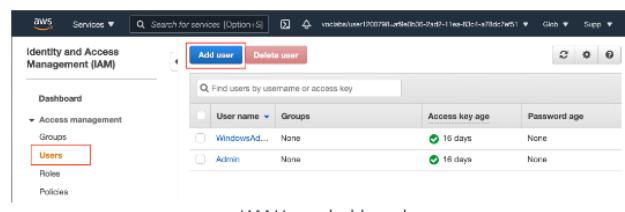
Details of a security group

## ▼ CREATE IAM User

## Create an IAM User

Here, you'll create an IAM user that you will use to access your Redshift cluster.

1. Navigate to the [IAM console](#). In the left navigation pane, choose **Users**, and click on the **Add User** button. It will launch a new wizard.



IAM Users dashboard

### 2. Set user details

Enter a name for your user , say `airflow_redshift_user`, and choose **Programmatic access**. Then click on the **Next: Permissions** button.

The screenshot shows the 'Add user' wizard, Step 1: Set user details. It has five steps numbered 1 to 5 at the top right. The first step is active. The form includes fields for 'User name\*' (set to 'airflow\_redshift\_user') and a link to 'Add another user'. Below that is a section titled 'Select AWS access type' with instructions: 'Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step.' Two options are available: 'Programmatic access' (selected with a checked checkbox) and 'AWS Management Console access' (unchecked). A note for 'Programmatic access' says: 'Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.' A note for 'AWS Management Console access' says: 'Enables a password that allows users to sign-in to the AWS Management Console.' At the bottom are 'Required' and 'Cancel' buttons, and a 'Next: Permissions' button.

Create IAM users → Set user details

### 3. Set permissions

Choose **Attach existing policies directly** option.

### 3. Set permissions

Choose **Attach existing policies directly** option.

- Search for redshift and select **AmazonRedshiftFullAccess**.
- Then, search for S3 and select **AmazonS3ReadOnlyAccess**.

After selecting both policies, choose **Next: Tags**. Skip this page and choose **Next: Review**.

Set permissions

Add user to group Copy permissions from existing user Attach existing policies directly

Create policy

Filter policies Q: redshift Showing 6 results

| Policy name                  | Type        | Used as                | Description                                    |
|------------------------------|-------------|------------------------|------------------------------------------------|
| AmazonOMSRedshift...         | AWS managed | None                   | Provides access to manage S3 settings for ...  |
| AmazonMachineLearn...        | AWS managed | None                   | Allows Machine Learning to configure and u...  |
| <b>AmazonRedshiftFull...</b> | AWS managed | Permissions policy (1) | Provides full access to Amazon Redshift via... |
| AmazonRedshiftQuo...         | AWS managed | Permissions policy (1) | Provides full access to the Amazon Redshift... |
| AmazonRedshiftRe...          | AWS managed | Permissions policy (1) | Provides read only access to Amazon Reds...    |
| AWSQuickSightDes...          | AWS managed | None                   | Allow QuickSight to describe Redshift resou... |

Cancel Previous Next: Tags

Create IAM user → Set permissions → Select **AmazonRedshiftFullAccess**

Set permissions

Add user to group Copy permissions from existing user Attach existing policies directly

Create policy

Filter policies Q: s3 Showing 4 results

| Policy name                | Type        | Used as                | Description                                      |
|----------------------------|-------------|------------------------|--------------------------------------------------|
| AmazonOMSRedshift...       | AWS managed | None                   | Provides access to manage S3 settings for ...    |
| AmazonS3FullAccess         | AWS managed | Permissions policy (1) | Provides full access to all buckets via the A... |
| <b>AmazonS3ReadOnly...</b> | AWS managed | Permissions policy (1) | Provides read only access to all buckets via...  |
| QuickSightAccessFor...     | AWS managed | None                   | Policy used by QuickSight team to access c...    |

Cancel Previous Next: Tags

Create IAM user → Set permissions → Select **AmazonS3ReadOnlyAccess**

4. **Review** your choices and finally click on the **Create user** button.

**Review**

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

**User details**

|                      |                                          |
|----------------------|------------------------------------------|
| User name            | airflow_redshift_user                    |
| AWS access type      | Programmatic access - with an access key |
| Permissions boundary | Permissions boundary is not set          |

**Permissions summary**

The following policies will be attached to the user shown above.

| Type           | Name                     |
|----------------|--------------------------|
| Managed policy | AmazonRedshiftFullAccess |
| Managed policy | AmazonS3ReadOnlyAccess   |

**Tags**

No tags were added.

**Create user**

Review the new IAM user details

#### 5. Save your credentials!

This is the only time you can view or download these credentials on AWS. Choose **Download .csv** to download these credentials and then save them to copy and paste this **Access key ID** and **Secret access key** in the next step.

We **strongly advise** you to keep this **Access key ID** and **Secret access key** closely guarded, including not putting them in a GitHub public repo, etc.

**Add user**

1 2 3 4 5

**Success**  
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.  
Users with AWS Management Console access can sign-in at: <https://junoless.sigin.aws.amazon.com/console>

**Download .csv**

| User                  | Access key ID        | Secret access key |
|-----------------------|----------------------|-------------------|
| airflow_redshift_user | AKIAJBFNV3NT4NVHNDKA | Show              |

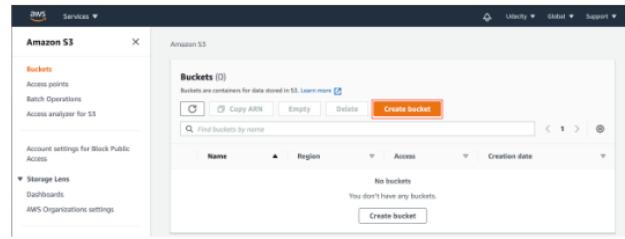
User created successfully.  
**Copy the Access key ID and Secret access key.**

**Close**

#### ▼ CREATE A Bucket S3

## Create a Bucket

1. Navigate to the [S3 dashboard](#), and click on the **Create bucket** button. It will launch a new wizard.



S3 service → Buckets dashboard.

View all of the S3 buckets in your account

(S3 is a global service, not a region-specific).

We create a bucket first, and later we upload files and folders to it.

### 2. General configuration

Provide the bucket-name and the region where you want to locate the bucket. The bucket name must be unique worldwide, and must not contain

Amazon S3 > Create bucket

### Create bucket

Buckets are containers for data stored in S3. Learn more [\[?\]](#)

#### General configuration

Bucket name  Bucket name must be unique and must not contain spaces or uppercase letters. See rules for bucket naming [\[?\]](#)

Region

Copy settings from existing bucket - optional  
Only the bucket settings in the following configuration are copied.

Create a bucket - Provide general details

### 3. Public Access settings

You can choose public visibility. Let's uncheck the *Block all public access* option.

**Block all public access**  
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- Block public access to buckets and objects granted through new access control lists (ACLs)**  
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- Block public access to buckets and objects granted through any access control lists (ACLs)**  
S3 will ignore all ACLs that grant public access to buckets and objects.
- Block public access to buckets and objects granted through new public bucket or access point policies**  
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- Block public and cross-account access to buckets and objects through any public bucket or access point policies**  
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

**⚠️ Turning off block all public access might result in this bucket and the objects within becoming public**  
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Create a bucket - Make it public

#### 4. Bucket Versioning and Encryption

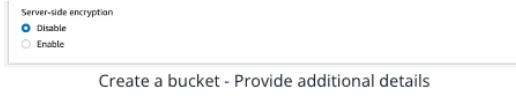
- Bucket Versioning - Keep it disabled.
- Encryption - If enabled, it will encrypt the files being stored in the bucket.
- Object Lock - If enables, it will prevent the files in the bucket from being deleted or modified.

**Bucket Versioning**  
Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

**Disable**  
 **Enable**

**Tags (0) - optional**  
Track storage cost or other criteria by tagging your bucket. [Learn more](#)

No tags associated with this bucket.  
[Add tag](#)



In the snapshots above, we have created a public bucket. Let's see **how to upload files and folders to the bucket**, and configure additional settings.

### Upload File/Folders to the Bucket

From the [S3 dashboard](#), click on the name of the bucket you have created in the step above.

Amazon S3 > mtvbucket

## mtvbucket

| Bucket overview          |                                 |                                      |                       |
|--------------------------|---------------------------------|--------------------------------------|-----------------------|
| Region                   | Amazon resource name            | Creation date                        | Access                |
| US East (Ohio) us-east-2 | (ARN)<br>arn:aws:s3:::mtvbucket | November 25, 2020, 16:11 (UTC+05:30) | Objects can be public |

**Objects** Properties Permissions Metrics Management Access points

Drag and drop files and folders you want to upload here, or choose Upload.

**Objects (0)**  
Objects are the fundamental entities stored in Amazon S3. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Delete  Actions  Create folder Upload

Find objects by prefix

< 1 >

| Name                                                     | Type | Last modified | Size | Storage class |
|----------------------------------------------------------|------|---------------|------|---------------|
| No objects<br>You don't have any objects in this bucket. |      |               |      |               |

Upload

Details of an existing bucket. Upload files/folders to this bucket.



In the snapshot above, it shows that the bucket is in the Region: **US East (Ohio) us-east-2**, and it has a unique Amazon resource name (ARN): **arn:aws:s3:::mtvbucket**. You can view more details of the bucket, in the tabs next to the bucket overview: **Objects, Properties, Permissions, Metrics, Management, and Access Control**. We have uploaded a sample file to the bucket:

1. Click on the **Upload** button to upload files and folders into the current bucket. In the snapshot below, we have uploaded a **Sample.txt** file.

The screenshot shows the AWS S3 'Objects' list interface. It displays one object: 'Sample.txt'. The details are as follows:

| Name       | Type | Last modified                        | Size   | Storage class |
|------------|------|--------------------------------------|--------|---------------|
| Sample.txt | txt  | November 25, 2020, 16:51 (UTC+05:30) | 2.0 KB | Standard      |

A sample file in the bucket

2. Click on the file name to view the file-specific details, as shown below.

The screenshot shows the AWS S3 'Object Details' page for 'Sample.txt'. The 'Details' tab is selected. The object overview section includes the following details:

- Owner:** AWS Region
- S3 URI:** <s3://mtvbucket/Sample.txt>
- AWS Region:** US East (Ohio) us-east-2
- Amazon resource name (ARN):** <arn:aws:s3:::mtvbucket/Sample.txt>
- Last modified:** November 25, 2020, 16:51 (UTC+05:30)
- Entity tag (Etag):** [19e4248fd5e3034c9b2d675d905725c0](#)
- Size:** 2.0 KB
- Type:** txt
- Object URL:** <https://mtvbucket.s3.us-east-2.amazonaws.com/Sample.txt>
- Key:** Sample.txt

Details of an individual file (object)

## Details of an Existing Bucket

### 1. Properties

There are several properties that you can set for S3 buckets, such as:

- Bucket Versioning - Allows you to keep multiple versions of an object in the same bucket.
- Static website hosting - Mark if the bucket is used to host a website. S3 is a very cost-effective and cheap solution for serving up static web content.
- Requester pays - Make the requester pays for requests and data transfer costs.
- Server access logging - Log requests for access to your bucket.
- Permissions

It shows who has access to the S3 bucket, and who has access to the data within the bucket. In the example snapshots above, the bucket is public, meaning anyone can access it. Here, we can write an access policy (in JSON format) to provide access to the objects stored in the bucket.

### 2. Metrics

View the metrics for usage, request, and data transfer activity within your bucket, such as, total bucket size, total number of objects, and storage class analysis.

### 3. Management

It allows you to create life cycle rules to help manage your objects. It includes rules such as transitioning objects to another storage class, archiving them, or deleting them after a specified period of time.

### 4. Access points

Here, you can create access endpoints for sharing the bucket at scale. Using an endpoint, you can perform all regular operations on the bucket.

## ▼ Launching Redshift Cluster

## Launching a Redshift Cluster in the AWS Console

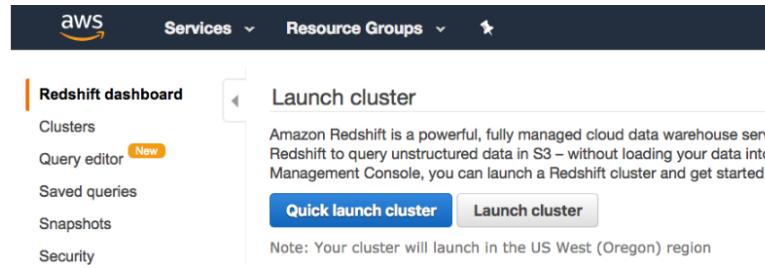
- Follow the instructions below to create a Redshift cluster
- Use the query editor to create a table and insert data
- Delete the cluster

Note: The steps below were introduced in lesson 2. You can use the IAM role and security group created in the last lesson.

### Launch a Redshift Cluster

**WARNING:** The cluster that you are about to launch will be live, and you will be charged the standard Amazon Redshift usage fees for the cluster until you delete it. **Make sure to delete your cluster each time you're finished working to avoid large, unexpected costs.** Instructions on deleting your cluster are included on the last page. You can always launch a new cluster, so don't leave your Redshift cluster running overnight or throughout the week if you don't need to.

- Open AWS Console by clicking on the [Launch AWS Gateway](#) button followed by [Open AWS Console](#).
- Search and select [Redshift](#) in the AWS Services search bar. This will open the Amazon Redshift Dashboard.
- On the Amazon Redshift Dashboard, choose [Create cluster](#).



- On the Cluster details page, enter the following values and then choose Continue:

- Cluster identifier: Enter `redshift-cluster`.

3. On the Cluster details page, enter the following values and then choose Continue:

- Cluster identifier: Enter `redshift-cluster`.
- Database name: Enter `dev`.
- Database port: Enter `5439`.
- Master user name: Enter `awsuser`.
- Master user password and Confirm password: Enter a password for the master user account.

Launch your Amazon Redshift cluster - Advanced settings | [Switch to quick launch](#)

CLUSTER DETAILS   NODE CONFIGURATION   ADDITIONAL CONFIGURATION   REVIEW

Provide the details of your cluster. Fields marked with \* are required.

|                       |                                               |                                                                                                                                                             |
|-----------------------|-----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Cluster identifier*   | <input type="text" value="redshift-cluster"/> | This is the unique key that identifies a cluster. This parameter is stored as a lowercase string. (e.g. my-dw-instance)                                     |
| Database name         | <input type="text" value="dev"/>              | Optional. A default database named dev is created for the cluster. Optionally, specify a custom database name (e.g. mydb) to create an additional database. |
| Database port*        | <input type="text" value="5439"/>             | Port number on which the database accepts connections.                                                                                                      |
| Master user name*     | <input type="text" value="awsuser"/>          | Name of master user for your cluster. (e.g. awsuser)                                                                                                        |
| Master user password* | <input type="password" value="*****"/>        | Password must contain 8 to 64 printable ASCII characters excluding: /, *, \, and @. It must contain 1 uppercase letter, 1 lowercase letter, and 1 number.   |
| Confirm password*     | <input type="password" value="*****"/>        | Confirm master user password                                                                                                                                |

[Cancel](#) [Continue](#)

4. On the Node Configuration page, accept the default values and choose Continue.

Launch your Amazon Redshift cluster - Advanced settings | [Switch to quick launch](#)



Launch your Amazon Redshift cluster - Advanced settings | [Switch to quick launch](#)

[CLUSTER DETAILS](#) [NODE CONFIGURATION](#) [ADDITIONAL CONFIGURATION](#) [REVIEW](#)

Choose a number of nodes and node type below. Number of Compute Nodes is required for multi-node clusters.

 The ds2 and dc2 node types replace the ds1 and dc1 node types, respectively. The newer ds2 and dc2 node types provide higher performance than ds1 and dc1 at no extra cost. [Learn more](#).

**Node type** **dc2.large** 

Specifies the compute, memory, storage, and I/O capacity of the cluster's nodes.

**CPU** 7 EC2 Compute Units (2 virtual cores) per node

**Memory** 15.25 GiB per node

**Storage** 160GB SSD storage per node

**I/O performance** Moderate

**Cluster type** **Single Node** 

**Number of compute nodes\***

Single Node clusters consist of a single node which performs both leader and compute functions.

**Maximum** 1

**Minimum** 1

[Cancel](#)

[Previous](#) [Continue](#)

5. On the Additional Configuration page, enter the following values:

- **VPC security groups:** redshift\_security\_group
- **Available IAM roles:** myRedshiftRole

Choose **Continue**.

**Publicly accessible**   

Select Yes if you want the cluster to be accessible from the public internet. Select No if you want it to be accessible only from within your private VPC network.

5. On the Additional Configuration page, enter the following values:

- **VPC security groups:** redshift\_security\_group
- **Available IAM roles:** myRedshiftRole

Choose **Continue**.

Publicly accessible  Yes  No Select Yes if you want the cluster to be accessible from the public internet. Select No if you want it to be accessible only from within your private VPC network.

Choose a public IP address  Yes  No Select Yes if you want to select your own public IP address from a list of elastic IP (EIP) addresses that are already configured for your cluster's VPC. Select No if you want Amazon Redshift to provide an EIP for you instead.

Enhanced VPC Routing  Yes  No Select Yes if you want to enable Enhanced VPC Routing. [Learn more](#)

Availability zone  The EC2 Availability Zone that the cluster will be created in.

---

Associate your cluster with one or more security groups.

VPC security groups  List of VPC security groups to associate with this cluster.

---

Optional, create a basic alarm for this cluster.

Create CloudWatch Alarm  Yes  No Create a CloudWatch alarm to monitor the disk usage of your cluster.

---

Optional, select your maintenance track for this cluster.

Maintenance Track  Current  Trailing Select Current to apply the latest certified maintenance release including features and bug-fixes. Select Trailing to apply the previously certified maintenance release.

---

Optional, associate up to 10 IAM roles with this cluster.

Available IAM roles    
 myRedshiftRole

6. Review your Cluster configuration and choose **Launch cluster**.

Launch your Amazon Redshift cluster - Advanced settings | [Switch to quick launch](#)

CLUSTER DETAILS NODE CONFIGURATION ADDITIONAL CONFIGURATION REVIEW

You are about to launch a cluster with following the following specifications:

**Cluster properties**

These attributes specify the name of your cluster, what type of virtual hardware it will run on, how many nodes it will contain, and the availability zone in which it will be located.

**Cluster identifier:** redshift-cluster  
**Node type:** dc2.large  
**Number of compute nodes:** 1 (leader and compute run on a single node)  
**Availability zone:** No preference

**Database configuration**

These properties specify the database name, port, and username you will use to connect to the database. The parameter group contains configuration values used by the database.

**Database name:** dev  
**Database port:** 5439  
**Master user name:** awsuser  
**Cluster parameter group:** default.redshift-1.0

---

**Security, access, and encryption**

These settings control whether your cluster will be created in an existing VPC to allow for simpler integration with other AWS Services, and the security groups which define access rules to your cluster.

**Virtual private cloud:** vpc-3924fa41  
**Cluster subnet group:**  
Publicly accessible: Yes  
Elastic IP: Not used  
**VPC security groups** redshift\_security\_group (sg-0eafe91b9bd584f51)  
Enhanced VPC Routing: No  
Encrypt database: No

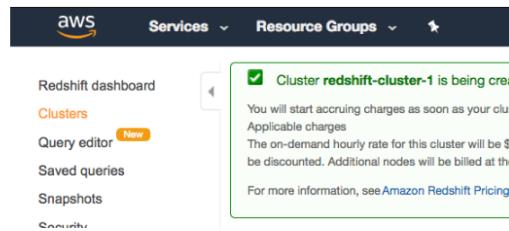
**CloudWatch alarms**

CloudWatch alarms are used to notify if metrics for your cluster are within a certain threshold. All recipients under the SNS topic specified for your alarm will receive notifications once an alarm is triggered.

Basic alarms will not be created for this cluster

[Cancel](#) [Previous](#) [Launch cluster](#)

7. A confirmation page will appear and the cluster will take a few minutes to finish. Choose **Clusters** in the left navigation pane to return to the list of clusters.



8. On the Clusters page, look at the cluster that you just launched and review the **Cluster Status** information. Make sure that the **Cluster Status** is **available** and the **Database Health** is **healthy** before you try to connect to the database later. You can expect this to take 5-10 minutes.

| Cluster            | Cluster Status | DB Health | Release Status | In Maintenance | Recent Events | View timeline                 |
|--------------------|----------------|-----------|----------------|----------------|---------------|-------------------------------|
| redshift-cluster-1 | creating       | unknown   | Not found      | unknown        | 8             | <a href="#">View timeline</a> |

| Cluster            | Cluster Status | DB Health | Release Status | In Maintenance | Recent Events | View timeline                 |
|--------------------|----------------|-----------|----------------|----------------|---------------|-------------------------------|
| redshift-cluster-1 | available      | healthy   | Up to date     | no             | 9             | <a href="#">View timeline</a> |

### Delete a Redshift Cluster

Make sure to delete your cluster each time you're finished working to avoid large, unexpected costs. You can always launch a new cluster, so don't leave it running overnight or longer than you need to.

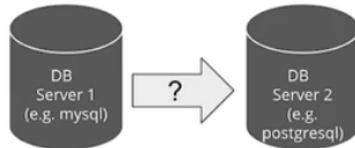
## ▼ Implementing Data Warehouses on AWS

### ▼ Theory

## SQL-To-SQL ETL, how? Different Database Server

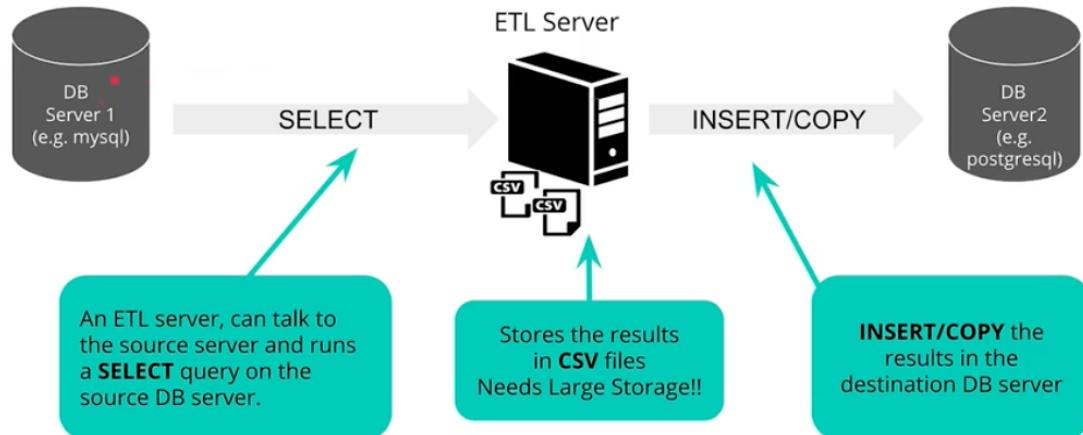
- But what do we do if we want to copy the results of a query to another table on a totally different database server?

```
SELECT fact1, fact2  
INTO OtherServer,newFactTable  
FROM table X,Y  
WHERE X.id = Y.id AND x.v <> null  
Group by Y.d
```

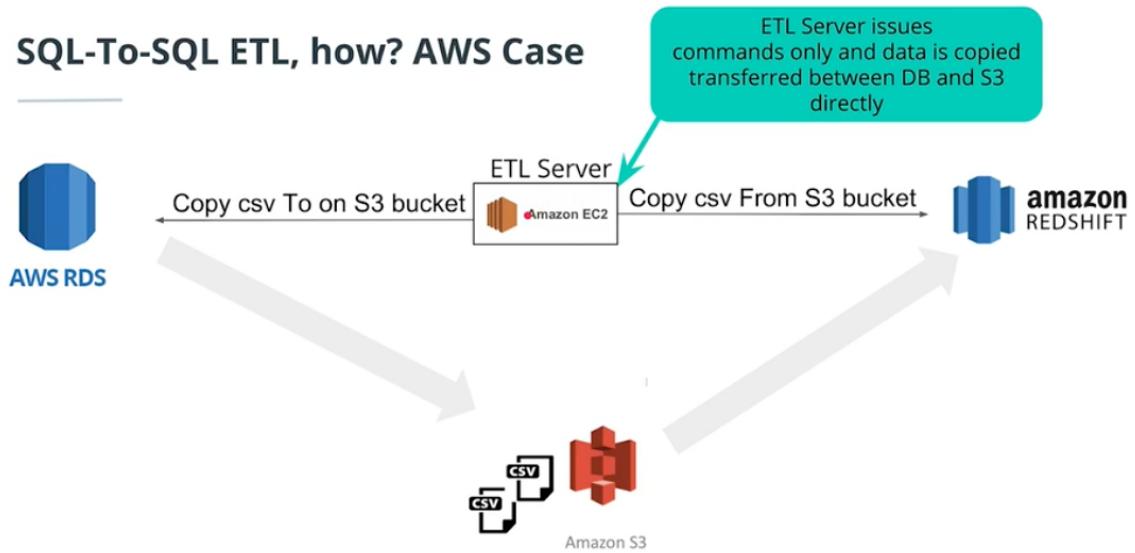


- If both servers are running the same RDBMS, that might be possible, but harder between two completely different RDBMSs.
- And even if we can, we probably need to do some transformations, cleaning, governance, etc..

## SQL-To-SQL ETL, how? A more general solution



## SQL-To-SQL ETL, how? AWS Case



## Ingesting at Scale: Use COPY

- To transfer data from an S3 staging area to redshift use the **COPY** command
- Inserting data row by using **INSERT** will be very slow
- If the file is large:
  - It is better to break it up to **multiple files**
  - Ingest in **Parallel**
    - Either using a **common prefix**
    - Or a **manifest file**
- Other considerations:
  - Better to ingest from the same AWS region
  - Better to compress the all the csv files
- One can also specify the delimiter to be used

### Common Prefix Example

```
COPY sporting_event_ticket FROM 's3://udacity-labs/tickets/split/part'  
CREDENTIALS 'aws_iam_role=arn:aws:iam::464956546:role/dwhRole'  
gzip DELIMITER ';' REGION 'us-west-2';
```

```
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00000.csv.gz')  
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00001.csv.gz')  
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00002.csv.gz')  
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00003.csv.gz')  
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00004.csv.gz')  
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00005.csv.gz')  
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00006.csv.gz')  
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00007.csv.gz')  
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00008.csv.gz')  
s3.ObjectSummary(bucket_name='udacity-labs', key='tickets/split/part-00009.csv.gz')
```

## Manifest File Example

```
{  
  "entries": [  
    {"url":"s3://mybucket-alpha/2013-10-04-custdata", "mandatory":true},  
    {"url":"s3://mybucket-alpha/2013-10-05-custdata", "mandatory":true},  
    {"url":"s3://mybucket-beta/2013-10-04-custdata", "mandatory":true},  
    {"url":"s3://mybucket-beta/2013-10-05-custdata", "mandatory":true}  
  ]  
}
```

```
COPY customer  
FROM 's3://mybucket/cust.manifest'  
IAM_ROLE 'arn:aws:iam::0123456789012:role/MyRedshiftRole'  
manifest;
```

### ▼ Infrastructure as a Code (IaC) Exercise

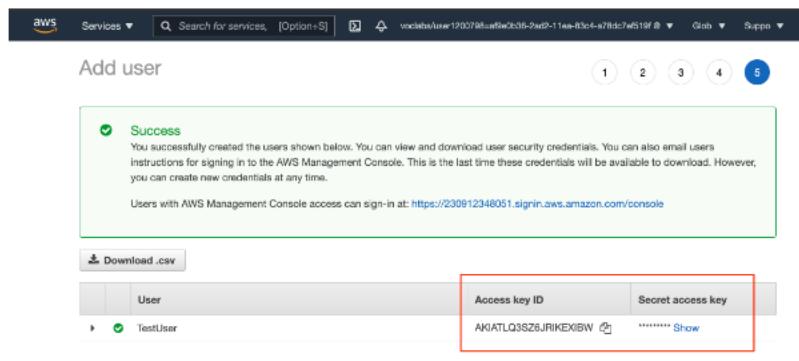
```
import pandas as pd  
import boto3  
import json
```

## STEP 0: (Prerequisite) Save the AWS Access key

### 1. Create a new IAM user

IAM service is a global service, meaning newly created IAM users are not restricted to a specific region by default.

- Go to [AWS IAM service](#) and click on the "Add user" button to create a new IAM user in your AWS account.
- Choose a name of your choice.
- Select "Programmatic access" as the access type. Click Next.
- Choose the *Attach existing policies directly* tab, and select the "**AdministratorAccess**". Click Next.
- Skip adding any tags. Click Next.
- Review and create the user. It will show you a pair of access key ID and secret.
- Take note of the pair of access key ID and secret. This pair is collectively known as **Access key**.



The screenshot shows the AWS IAM 'Add user' success page. A green box at the top indicates success: 'You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.' Below this, a 'Download .csv' button is visible. A table lists a single user, 'TestUser', with columns for 'User' and 'Access key ID' (containing 'AKIATLQ3S26JRIKEXIBW') and 'Secret access key' (containing '\*\*\*\*\* Show'). The 'Secret access key' column is highlighted with a red box.

Snapshot of a pair of an Access key

### 2. Save the access key and secret

Edit the file `dwh.cfg` in the same folder as this notebook and save the access key and secret against the following variables:

```
KEY= <YOUR_AWS_KEY>
SECRET= <YOUR_AWS_SECRET>
```

For example:

```
KEY=6JW3ATLQ34PH3AKI
SECRET=wnoBHA+qUBFgwCRHJqqqrLU0i
```

### 3. Troubleshoot

If your keys are not working, such as getting an `InvalidAccessKeyId` error, then you cannot retrieve them again. You have either of the following two options:

#### 1. Option 1 - Create a new pair of access keys for the existing user

- Go to the [IAM dashboard](#) and view the details of the existing (Admin) user.
- Select on the **Security credentials** tab, and click the **Create access key** button. It will generate a new pair of access key ID and secret.
- Save the new access key ID and secret in your `dwh.cfg` file

| Access key ID      | Created                   | Last used | Status                                                 |
|--------------------|---------------------------|-----------|--------------------------------------------------------|
| AKIAJLQ9S2JURKEIBN | 2021-04-20 10:59 UTC+0530 | N/A       | <a href="#">Active</a>   <a href="#">Make Inactive</a> |

Snapshot of creating a new Access keys for the existing user

#### 2. Option 2 - Create a new IAM user with Admin access - Refer to the instructions at the top.

```
### LOAD DWH PARAMS from a file

import configparser
config = configparser.ConfigParser()
config.read_file(open('dwh.cfg'))

KEY = config.get('AWS', 'KEY')
SECRET = config.get('AWS', 'SECRET')

DWH_CLUSTER_TYPE = config.get("DWH", "DWH_CLUSTER_TYPE")
DWH_NUM_NODES = config.get("DWH", "DWH_NUM_NODES")
DWH_NODE_TYPE = config.get("DWH", "DWH_NODE_TYPE")

DWH_CLUSTER_IDENTIFIER = config.get("DWH", "DWH_CLUSTER_IDENTIFIER")
DWH_DB = config.get("DWH", "DWH_DB")
DWH_DB_USER = config.get("DWH", "DWH_DB_USER")
DWH_DB_PASSWORD = config.get("DWH", "DWH_DB_PASSWORD")
DWH_PORT = config.get("DWH", "DWH_PORT")

DWH_IAM_ROLE_NAME = config.get("DWH", "DWH_IAM_ROLE_NAME")

(DWH_DB_USER, DWH_DB_PASSWORD, DWH_DB)

pd.DataFrame({"Param": [
    "DWH_CLUSTER_TYPE", "DWH_NUM_NODES", "DWH_NODE_TYPE", "DWH_CLUSTER_IDENTIFIER", "DWH_DB", "DWH_DB_USER", "DWH_DB_PASSWORD", "DWH_PORT", "DWH_IAM_ROLE_NAME"
    ],
    "Value": [
        DWH_CLUSTER_TYPE, DWH_NUM_NODES, DWH_NODE_TYPE, DWH_CLUSTER_IDENTIFIER, DWH_DB, DWH_DB_USER, DWH_DB_PASSWORD, DWH_PORT, DWH_IAM_ROLE_NAME
    ]})
```

### Create clients for IAM, EC2, S3 and Redshift

**Note:** We are creating these resources in the the **us-west-2** region. Choose the same region in your AWS web console to see these resources.

```

import boto3

ec2 = boto3.resource('ec2',
                     region_name="us-west-2",
                     aws_access_key_id=KEY,
                     aws_secret_access_key=SECRET
                     )

s3 = boto3.resource('s3',
                    region_name="us-west-2",
                    aws_access_key_id=KEY,
                    aws_secret_access_key=SECRET
                    )

iam = boto3.client('iam',aws_access_key_id=KEY,
                   aws_secret_access_key=SECRET,
                   region_name='us-west-2'
                   )

redshift = boto3.client('redshift',
                       region_name="us-west-2",
                       aws_access_key_id=KEY,
                       aws_secret_access_key=SECRET
                       )

```

### Check out the sample data sources on S3

```

sampleDbBucket = s3.Bucket("awssampledbuswest2")
for obj in sampleDbBucket.objects.filter(Prefix="ssbgz"):
    print(obj)
# for obj in sampleDbBucket.objects.all():
#     print(obj)

```

### STEP 1: IAM ROLE

- Create an IAM Role that makes Redshift able to access S3 bucket (ReadOnly)

```

from botocore.exceptions import ClientError

#1.1 Create the role,
try:
    print("1.1 Creating a new IAM Role")
    dwhRole = iam.create_role(
        Path='/',
        RoleName=DWH_IAM_ROLE_NAME,
        Description = "Allows Redshift clusters to call AWS services on your behalf.",
        AssumeRolePolicyDocument=json.dumps(
            {'Statement': [{'Action': 'sts:AssumeRole',
               'Effect': 'Allow',
               'Principal': {'Service': 'redshift.amazonaws.com'}}],
            'Version': '2012-10-17'})
except Exception as e:
    print(e)

print("1.2 Attaching Policy")

iam.attach_role_policy(RoleName=DWH_IAM_ROLE_NAME,
                      PolicyArn="arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess"
                      )['ResponseMetadata']['HTTPStatusCode']

print("1.3 Get the IAM role ARN")
roleArn = iam.get_role(RoleName=DWH_IAM_ROLE_NAME)['Role']['Arn']

print(roleArn)

```

### STEP 2: Redshift Cluster

- Create a [RedShift Cluster](#)
- For complete arguments to `create_cluster`, see [docs](#)

```

try:
    response = redshift.create_cluster(
        #HW
        ClusterType=DWH_CLUSTER_TYPE,
        NodeType=DWH_NODE_TYPE,
        NumberOfNodes=int(DWH_NUM_NODES),

        #Identifiers & Credentials
        DBName=DWH_DB,
        ClusterIdentifier=DWH_CLUSTER_IDENTIFIER,
        MasterUsername=DWH_DB_USER,
        MasterUserPassword=DWH_DB_PASSWORD,

        #Roles (for s3 access)
        IamRoles=[roleArn]
    )
except Exception as e:
    print(e)

```

## 2.1 Describe the cluster to see its status

- run this block several times until the cluster status becomes Available

```

def prettyRedshiftProps(props):
    pd.set_option('display.max_colwidth', -1)
    keysToShow = ["ClusterIdentifier", "NodeType", "ClusterStatus", "MasterUsername", "DBName", "Endpoint", "NumberOfNodes", 'VpcId']
    x = [(k, v) for k,v in props.items() if k in keysToShow]
    return pd.DataFrame(data=x, columns=["Key", "Value"])

myClusterProps = redshift.describe_clusters(ClusterIdentifier=DWH_CLUSTER_IDENTIFIER)['Clusters'][0]
prettyRedshiftProps(myClusterProps)

```

## 2.2 Take note of the cluster endpoint and role ARN

DO NOT RUN THIS unless the cluster status becomes "Available". Make ure you are checking your Amazon Redshift cluster in the **us-west-2** region.

```

DWH_ENDPOINT = myClusterProps['Endpoint']['Address']
DWH_ROLE_ARN = myClusterProps['IamRoles'][0]['IamRoleArn']
print("DWH_ENDPOINT :: ", DWH_ENDPOINT)
print("DWH_ROLE_ARN :: ", DWH_ROLE_ARN)

```

## STEP 3: Open an incoming TCP port to access the cluster ednpoint

```

try:
    vpc = ec2.Vpc(id=myClusterProps['VpcId'])
    defaultSg = list(vpc.security_groups.all())[0]
    print(defaultSg)
    defaultSg.authorize_ingress(
       GroupName=defaultSg.group_name,
        CidrIp='0.0.0.0/0',
        IpProtocol='TCP',
        FromPort=int(DWH_PORT),
        ToPort=int(DWH_PORT)
    )
except Exception as e:
    print(e)

```

## STEP 4: Make sure you can connect to the cluster

```

%load_ext sql

conn_string="postgresql://{}:{}@{}:{}/{}".format(DWH_DB_USER, DWH_DB_PASSWORD, DWH_ENDPOINT, DWH_PORT,DWH_DB)
print(conn_string)
%sql $conn_string

```

## STEP 5: Clean up your resources

**DO NOT RUN THIS UNLESS YOU ARE SURE** We will be using these resources in the next exercises

```
#### CAREFUL!!
-- Uncomment & run to delete the created resources
redshift.delete_cluster( ClusterIdentifier=DWH_CLUSTER_IDENTIFIER, SkipFinalClusterSnapshot=True)
#### CAREFUL!!

myClusterProps = redshift.describe_clusters(ClusterIdentifier=DWH_CLUSTER_IDENTIFIER)['Clusters'][0]
prettyRedshiftProps(myClusterProps)

#### CAREFUL!!
-- Uncomment & run to delete the created resources
iam.detach_role_policy(RoleName=DWH_IAM_ROLE_NAME, PolicyArn="arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess")
iam.delete_role(RoleName=DWH_IAM_ROLE_NAME)
#### CAREFUL!!
```

## ▼ Parallel ETL exercise

The scope here is to show that parallel copying is faster than uploading the whole csv

```
%load_ext sql
from time import time
import configparser
import matplotlib.pyplot as plt
import pandas as pd
```

## STEP 1: Get the params of the created redshift cluster

- We need:
  - The redshift cluster endpoint
  - The that give access to Redshift to read from S3 IAM role ARN

```
config = configparser.ConfigParser()
config.read_file(open('dwh.cfg'))
KEY=config.get('AWS','key')
SECRET= config.get('AWS','secret')

DWH_DB= config.get("DWH","DWH_DB")
DWH_DB_USER= config.get("DWH","DWH_DB_USER")
DWH_DB_PASSWORD= config.get("DWH","DWH_DB_PASSWORD")
DWH_PORT = config.get("DWH","DWH_PORT")

# FILL IN THE REDSHIFT ENPOINT HERE
# e.g. DWH_ENDPOINT="redshift-cluster-1.csmamz5zxmle.us-west-2.redshift.amazonaws.com"
DWH_ENDPOINT="dwhcluster.csslugxhlivog.us-east-2.redshift.amazonaws.com"

#FILL IN THE IAM ROLE ARN you got in step 2.2 of the previous exercise
#e.g DWH_ROLE_ARN="arn:aws:iam::988332130976:role/dwhRole"
DWH_ROLE_ARN="arn:aws:iam::149551910519:role/dwhRole"
```

## STEP 2: Connect to the Redshift Cluster

```
conn_string="postgresql://{}:{}@{}:{}/{}".format(DWH_DB_USER, DWH_DB_PASSWORD, DWH_ENDPOINT, DWH_PORT,DWH_DB)
print(conn_string)
%sql $conn_string

import boto3

s3 = boto3.resource('s3',
                    region_name="us-west-2",
                    aws_access_key_id=KEY,
                    aws_secret_access_key=SECRET
```

```

        )
sampleDbBucket = s3.Bucket("udacity-labs")

for obj in sampleDbBucket.objects.filter(Prefix="tickets"):
    print(obj)

```

### STEP 3: Create Tables

```

%%sql
DROP TABLE IF EXISTS "sporting_event_ticket";
CREATE TABLE "sporting_event_ticket" (
    "id" double precision DEFAULT nextval('sporting_event_ticket_seq') NOT NULL,
    "sporting_event_id" double precision NOT NULL,
    "sport_location_id" double precision NOT NULL,
    "seat_level" numeric(1,0) NOT NULL,
    "seat_section" character varying(15) NOT NULL,
    "seat_row" character varying(10) NOT NULL,
    "seat" character varying(10) NOT NULL,
    "ticketholder_id" double precision,
    "ticket_price" numeric(8,2) NOT NULL
);

```

### STEP 4: Load Partitioned data into the cluster

```

%%time
qry = """
    copy sporting_event_ticket from 's3://udacity-labs/tickets/split/part'
    credentials 'aws_iam_role={}
    gzip delimiter ';' compupdate off region 'us-west-2';
""".format(DWH_ROLE_ARN)

%sql $qry

```

### STEP 4: Create Tables for the non-partitioned data

```

%%sql
DROP TABLE IF EXISTS "sporting_event_ticket_full";
CREATE TABLE "sporting_event_ticket_full" (
    "id" double precision DEFAULT nextval('sporting_event_ticket_seq') NOT NULL,
    "sporting_event_id" double precision NOT NULL,
    "sport_location_id" double precision NOT NULL,
    "seat_level" numeric(1,0) NOT NULL,
    "seat_section" character varying(15) NOT NULL,
    "seat_row" character varying(10) NOT NULL,
    "seat" character varying(10) NOT NULL,
    "ticketholder_id" double precision,
    "ticket_price" numeric(8,2) NOT NULL
);

```

### STEP 5: Load non-partitioned data into the cluster

Note how it's slower than loading partitioned data

```

%%time
qry = """
    copy sporting_event_ticket_full from 's3://udacity-labs/tickets/full/full.csv.gz'
    credentials 'aws_iam_role={}
    gzip delimiter ';' compupdate off region 'us-west-2';
""".format(DWH_ROLE_ARN)

%sql $qry

```

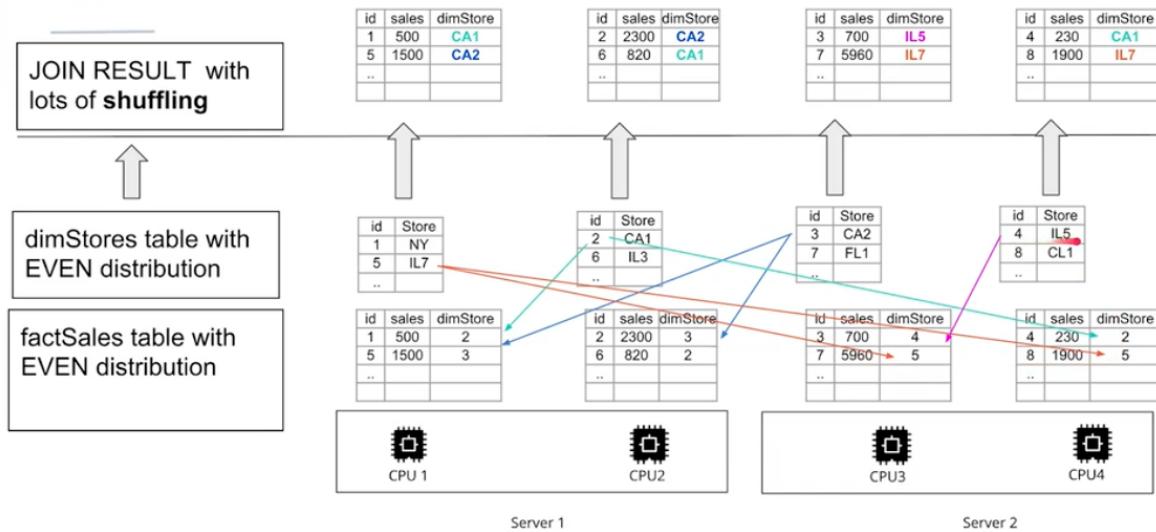
### ▼ Distributions Styles

# Distribution Styles

- **EVEN distribution**
- **ALL distribution**
- **AUTO distribution**
- **KEY distribution**

**EVEN Distribution** can be fast as it split the tables equally between CPU. But it has a high cost of join because different CPUs need to talk as in the picture below.

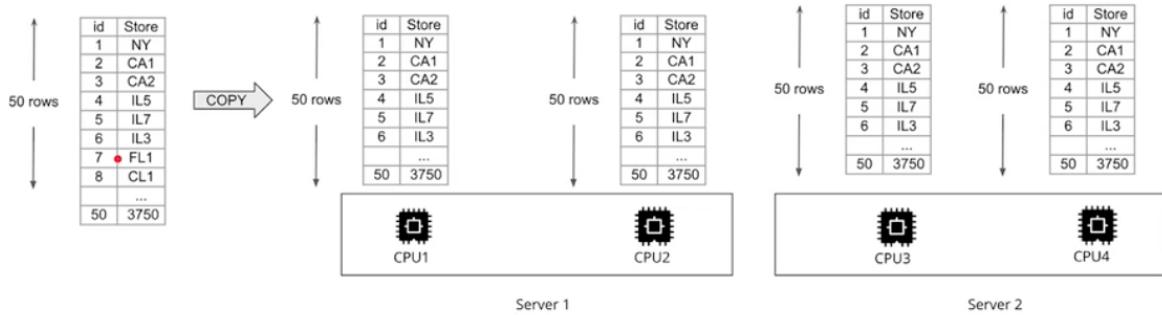
## High Cost of Join with EVEN Distribution (Shuffling)



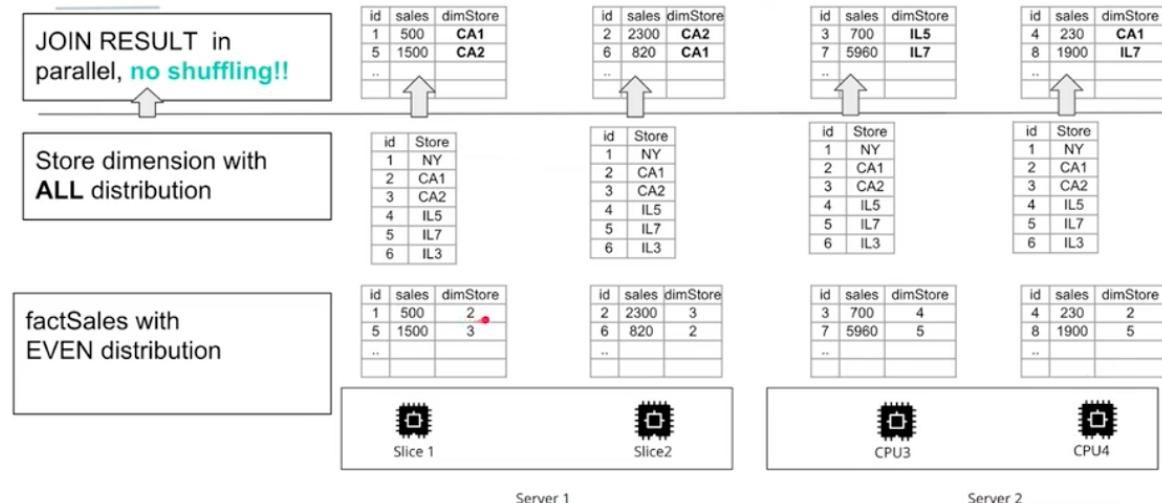
## ALL Distribution

## Distribution Style: ALL

- Small tables could be replicated on all slices to speed up joins
- Used frequently for dimension tables
- AKA “broadcasting”



## Distribution Style: Distributing facts with EVEN and Dimensions with ALL eliminates shuffling



## AUTO Distribution

## Distribution Style : AUTO

- Leave decision to Redshift
- "Small enough" tables are distributed with an ALL strategy
- Large tables are distributed with EVEN strategy

KEY Distribution

## Distribution Style: KEY

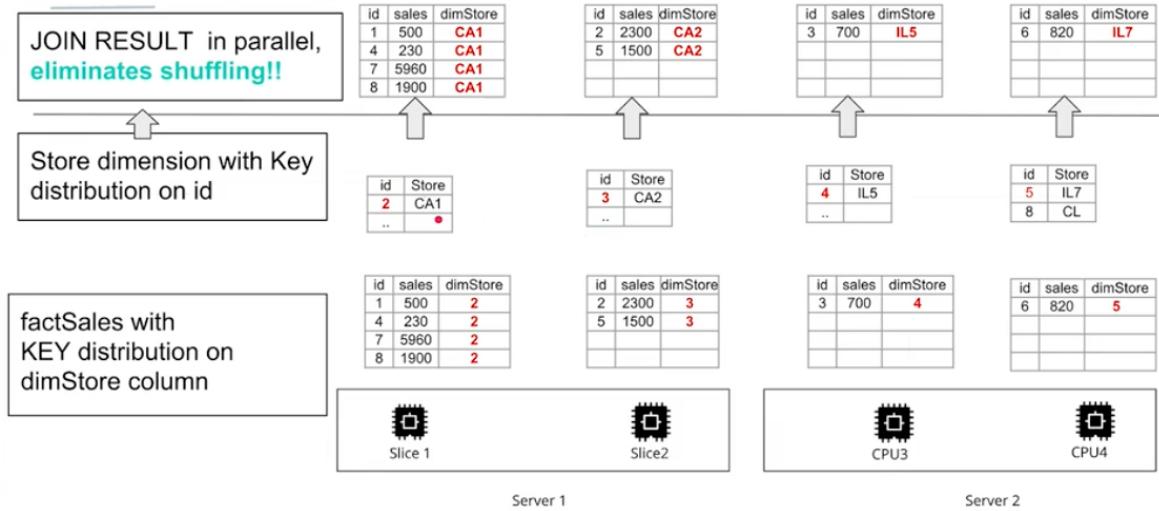
- Rows having similar values are placed in the same slice



## Distribution Style : KEY - cont'd

- Rows having similar values are placed in the same slice
- This can lead to a skewed distribution if some values of the dist key are more frequent than others
- However, very useful when a dimension table is too big to be distributed with ALL strategy. In that case, we distribute both the fact table and the dimension table using the same dist key.
- If two tables are distributed on the joining keys, redshift collocates the rows from both tables on the same slices

## Distribution Style: Distributing both facts and Dimensions on the joining KEYS eliminates shuffling



## Distribution Style KEY : Syntax

```

CREATE TABLE lineorder (
    lo_orderkey      integer      not null,
    lo_linenumber    integer      not null,
    lo_custkey       integer      not null,
    lo_partkey       integer      not null distkey,
    lo_suppkey       integer      not null,
    lo_orderdate     integer      not null,
    lo_orderpriority varchar(15)  not null,
    lo_shippriority  varchar(1)   not null,
    lo_quantity      integer      not null,
    lo_extendedprice integer      not null,
    lo_ordertotalprice integer     not null,
    lo_discount      integer      not null,
    lo_revenue        integer      not null,
    lo_supplycost    integer      not null,
    lo_tax           integer      not null,
    lo_commitdate    integer      not null,
    lo_shipmode      varchar(10)  not null
);

CREATE TABLE part (
    p_partkey       integer      not null distkey,
    p_name          varchar(22)  not null,
    p_mfgr          varchar(6)   not null,
    p_category      varchar(7)   not null,
    p_brand1        varchar(9)   not null,
    p_color         varchar(11)  not null,
    p_type          varchar(25)  not null,
    p_size          integer      not null,
    p_container     varchar(10)  not null
);

```

### ▼ Table Design

```

%load_ext sql

from time import time
import configparser
import matplotlib.pyplot as plt
import pandas as pd

config = configparser.ConfigParser()
config.read_file(open('dwh.cfg'))
KEY=config.get('AWS','key')
SECRET= config.get('AWS','secret')

DWH_DB= config.get("DWH","DWH_DB")

```

```

DWH_DB_USER= config.get("DWH","DWH_DB_USER")
DWH_DB_PASSWORD= config.get("DWH", "DWH_DB_PASSWORD")
DWH_PORT = config.get("DWH","DWH_PORT")

```

### STEP 1: Get the params of the created redshift cluster

We need:

- The redshift cluster endpoint
- The IAM role ARN that give access to Redshift to read from S3

IAM role ARN

```

# FILL IN THE REDSHIFT ENDPOINT HERE
# e.g. DWH_ENDPOINT="redshift-cluster-1.csmamz5zxmle.us-west-2.redshift.amazonaws.com"
DWH_ENDPOINT="dwhcluster.cslugxhlivog.us-east-2.redshift.amazonaws.com"

#FILL IN THE IAM ROLE ARN you got in step 2.2 of the previous exercise
#e.g DWH_ROLE_ARN="arn:aws:iam::988332130976:role/dwhRole"
DWH_ROLE_ARN="arn:aws:iam::149551910519:role/dwhRole"

```

### STEP 2: Connect to the Redshift Cluster

```

import os
conn_string="postgresql://{}:{}@{}:{}/{}".format(DWH_DB_USER, DWH_DB_PASSWORD, DWH_ENDPOINT, DWH_PORT,DWH_DB)
print(conn_string)
%%sql $conn_string

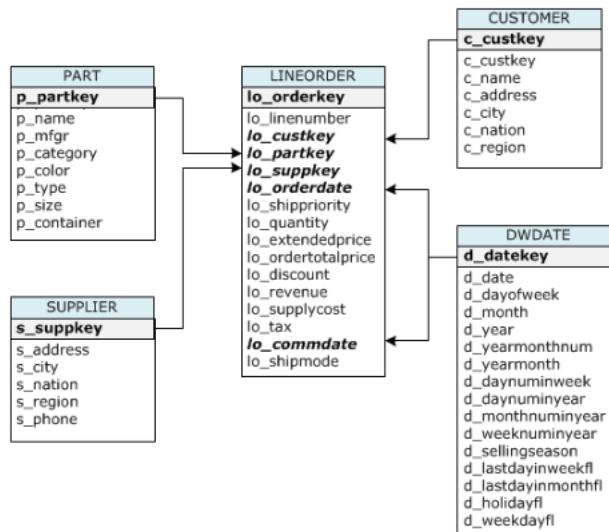
```

### STEP 3: Create Tables

We are going to use a benchmarking data set common for benchmarking star schemas in data warehouses.

The data is pre-loaded in a public bucket on the [us-west-2](#) region

Our examples will be based on the Amazon Redshift tutorial but in a scripted environment in our workspace



#### 3.1 Create tables (no distribution strategy) in the [nodist](#) schema

```

%%sql
CREATE SCHEMA IF NOT EXISTS nodist;
SET search_path TO nodist;

```

```

DROP TABLE IF EXISTS part cascade;
DROP TABLE IF EXISTS supplier;
DROP TABLE IF EXISTS supplier;
DROP TABLE IF EXISTS customer;
DROP TABLE IF EXISTS dwdate;
DROP TABLE IF EXISTS lineorder;

CREATE TABLE part
(
    p_partkey      INTEGER NOT NULL,
    p_name         VARCHAR(22) NOT NULL,
    p_mfgr         VARCHAR(6) NOT NULL,
    p_category     VARCHAR(7) NOT NULL,
    p_brand1       VARCHAR(9) NOT NULL,
    p_color        VARCHAR(11) NOT NULL,
    p_type         VARCHAR(25) NOT NULL,
    p_size          INTEGER NOT NULL,
    p_container    VARCHAR(10) NOT NULL
);

CREATE TABLE supplier
(
    s_suppkey      INTEGER NOT NULL,
    s_name         VARCHAR(25) NOT NULL,
    s_address      VARCHAR(25) NOT NULL,
    s_city          VARCHAR(10) NOT NULL,
    s_nation        VARCHAR(15) NOT NULL,
    s_region        VARCHAR(12) NOT NULL,
    s_phone         VARCHAR(15) NOT NULL
);

CREATE TABLE customer
(
    c_custkey      INTEGER NOT NULL,
    c_name         VARCHAR(25) NOT NULL,
    c_address      VARCHAR(25) NOT NULL,
    c_city          VARCHAR(10) NOT NULL,
    c_nation        VARCHAR(15) NOT NULL,
    c_region        VARCHAR(12) NOT NULL,
    c_phone         VARCHAR(15) NOT NULL,
    c_mktsegment   VARCHAR(10) NOT NULL
);

CREATE TABLE dwdate
(
    d_datekey      INTEGER NOT NULL,
    d_date         VARCHAR(19) NOT NULL,
    d_dayofweek    VARCHAR(10) NOT NULL,
    d_month        VARCHAR(10) NOT NULL,
    d_year         INTEGER NOT NULL,
    d_yearmonthnum INTEGER NOT NULL,
    d_yearmonth    VARCHAR(8) NOT NULL,
    d_daynuminweek INTEGER NOT NULL,
    d_daynuminmonth INTEGER NOT NULL,
    d_daynuminyear INTEGER NOT NULL,
    d_monthnuminyear INTEGER NOT NULL,
    d_weeknuminyear INTEGER NOT NULL,
    d_sellingseason VARCHAR(13) NOT NULL,
    d_lastdayinweekfl VARCHAR(1) NOT NULL,
    d_lastdayinmonthfl VARCHAR(1) NOT NULL,
    d_holidayfl    VARCHAR(1) NOT NULL,
    d_weekdayfl    VARCHAR(1) NOT NULL
);

CREATE TABLE lineorder
(
    lo_orderkey    INTEGER NOT NULL,
    lo_linenumber  INTEGER NOT NULL,
    lo_custkey     INTEGER NOT NULL,
    lo_partkey     INTEGER NOT NULL,
    lo_suppkey     INTEGER NOT NULL,
    lo_orderdate   INTEGER NOT NULL,
    lo_orderpriority VARCHAR(15) NOT NULL,
    lo_shipppriority VARCHAR(1) NOT NULL,
    lo_quantity    INTEGER NOT NULL,
    lo_extendedprice INTEGER NOT NULL,
    lo_ordertotalprice INTEGER NOT NULL,
    lo_discount    INTEGER NOT NULL,
    lo_revenue     INTEGER NOT NULL,
    lo_supplycost  INTEGER NOT NULL,
    lo_tax          INTEGER NOT NULL,
);

```

```

    lo_commitdate      INTEGER NOT NULL,
    lo_shipmode        VARCHAR(10) NOT NULL
);

```

### 3.1 Create tables (with a distribution strategy) in the `dist` schema

```

%%sql

CREATE SCHEMA IF NOT EXISTS dist;
SET search_path TO dist;

DROP TABLE IF EXISTS part cascade;
DROP TABLE IF EXISTS supplier;
DROP TABLE IF EXISTS supplier;
DROP TABLE IF EXISTS customer;
DROP TABLE IF EXISTS dwdate;
DROP TABLE IF EXISTS lineorder;

CREATE TABLE part (
    p_partkey      integer      not null sortkey distkey,
    p_name         varchar(22)   not null,
    p_mfgr         varchar(6)    not null,
    p_category     varchar(7)    not null,
    p_brand1       varchar(9)    not null,
    p_color        varchar(11)   not null,
    p_type         varchar(25)   not null,
    p_size         integer      not null,
    p_container    varchar(10)   not null
);

CREATE TABLE supplier (
    s_suppkey      integer      not null sortkey,
    s_name         varchar(25)   not null,
    s_address      varchar(25)   not null,
    s_city          varchar(10)   not null,
    s_nation        varchar(15)   not null,
    s_region        varchar(12)   not null,
    s_phone         varchar(15)   not null
)
diststyle all;

CREATE TABLE customer (
    c_custkey      integer      not null sortkey,
    c_name         varchar(25)   not null,
    c_address      varchar(25)   not null,
    c_city          varchar(10)   not null,
    c_nation        varchar(15)   not null,
    c_region        varchar(12)   not null,
    c_phone         varchar(15)   not null,
    c_mktsegment   varchar(10)   not null
)
diststyle all;

CREATE TABLE dwdate (
    d_datekey      integer      not null sortkey,
    d_date          varchar(19)   not null,
    d_dayofweek     varchar(10)   not null,
    d_month         varchar(10)   not null,
    d_year          integer      not null,
    d_yeарmonthnum integer      not null,
    d_yeарmonth    varchar(8)    not null,
    d_daynuminweek integer      not null,
    d_daynuminmonth integer      not null,
    d_daynuminyear integer      not null,
    d_monthnuminyear integer      not null,
    d_weeknuminyear integer      not null,
    d_sellingseason varchar(13)   not null,
    d_lastdayinweekfl varchar(1)  not null,
    d_lastdayinmonthfl varchar(1)  not null,
    d_holidayfl    varchar(1)  not null,
    d_weekdayfl    varchar(1)  not null
)
diststyle all;

CREATE TABLE lineorder (
    lo_orderkey     integer      not null,
    lo_linenumber   integer      not null,
    lo_custkey      integer      not null,
    lo_partkey      integer      not null distkey,
    lo_suppkey      integer      not null,
    lo_orderdate    integer      not null sortkey,

```

```

lo_orderpriority    varchar(15)      not null,
lo_shippriority     varchar(1)       not null,
lo_quantity          integer         not null,
lo_extendedprice    integer         not null,
lo_orderottotalprice integer         not null,
lo_discount          integer         not null,
lo_revenue           integer         not null,
lo_supplycost        integer         not null,
lo_tax               integer         not null,
lo_commitdate        integer         not null,
lo_shipmode          varchar(10)      not null
);

```

## STEP 4: Copying tables

Our intent here is to run 5 COPY operations for the 5 tables respectively as shown below.

However, we want to do accomplish the following:

- Make sure that the `DWH_ROLE_ARN` is substituted with the correct value in each query
- Perform the data loading twice once for each schema (dist and nodist)
- Collect timing statistics to compare the insertion times. Thus, we have scripted the insertion as found below in the function `loadTables` which returns a pandas dataframe containing timing statistics for the copy operations

```

copy customerfrom 's3://awssampledbuswest2/ssbgz/customer'
credentials 'aws_iam_role=<DWH_ROLE_ARN>'
gzip region 'us-west-2';

copy dwdatefrom 's3://awssampledbuswest2/ssbgz/dwdate'
credentials 'aws_iam_role=<DWH_ROLE_ARN>'
gzip region 'us-west-2';

copy lineorderfrom 's3://awssampledbuswest2/ssbgz/lineorder'
credentials 'aws_iam_role=<DWH_ROLE_ARN>'
gzip region 'us-west-2';

copy partfrom 's3://awssampledbuswest2/ssbgz/part'
credentials 'aws_iam_role=<DWH_ROLE_ARN>'
gzip region 'us-west-2';

copy supplierfrom 's3://awssampledbuswest2/ssbgz/supplier'
credentials 'aws_iam_role=<DWH_ROLE_ARN>'
gzip region 'us-west-2';

```

### 4.1 Automate the copying

```

def loadTables(schema, tables):
    loadTimes = []
    SQL_SET_SCEMA = "SET search_path TO {};" .format(schema)
    %sql $SQL_SET_SCEMA

    for table in tables:
        SQL_COPY = """
copy {} from 's3://awssampledbuswest2/ssbgz/{}'
credentials 'aws_iam_role={}'
gzip region 'us-west-2';
        """.format(table, table, DWH_ROLE_ARN)

        print("===== LOADING TABLE: ** {} ** IN SCHEMA ==> {} =====".format(table, schema))
        print(SQL_COPY)

        t0 = time()
        %sql $SQL_COPY
        loadTime = time() - t0
        loadTimes.append(loadTime)

    print("== DONE IN: {:.2f} sec\n".format(loadTime))
    return pd.DataFrame({"table":tables, "loadtime_" + schema:loadTimes}).set_index('table')

```

```

-- List of the tables to be loaded
tables = ["customer", "dwdate", "supplier", "part", "lineorder"]

-- Insertion twice for each schema (WARNING!! EACH CAN TAKE MORE THAN 10 MINUTES!!!)
nodistStats = loadTables("nodist", tables)
distStats = loadTables("dist", tables)

```

#### 4.1 Compare the load performance results

```

-- Plotting of the timing results
stats = distStats.join(nodistStats)
stats.plot.bar()
plt.show()

```

#### STEP 5: Compare Query Performance

```

oneDim_SQL ="""
set enable_result_cache_for_session to off;
SET search_path TO {};

select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, dwdate
where lo_orderdate = d_datekey
and d_year = 1997
and lo_discount between 1 and 3
and lo_quantity < 24;
"""

twoDim_SQL="""
set enable_result_cache_for_session to off;
SET search_path TO {};

select sum(lo_revenue), d_year, p_brand1
from lineorder, dwdate, part, supplier
where lo_orderdate = d_datekey
and lo_partkey = p_partkey
and lo_suppkey = s_suppkey
and p_category = 'MFGR#12'
and s_region = 'AMERICA'
group by d_year, p_brand1
"""

drill_SQL = """
set enable_result_cache_for_session to off;
SET search_path TO {};

select c_city, s_city, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, dwdate
where lo_custkey = c_custkey
and lo_suppkey = s_suppkey
and lo_orderdate = d_datekey
and (c_city='UNITED KI1' or
c_city='UNITED KI5')
and (s_city='UNITED KI1' or
s_city='UNITED KI5')
and d_yeарmonth = 'Dec1997'
group by c_city, s_city, d_year
order by d_year asc, revenue desc;
"""

oneDimSameDist_SQL ="""
set enable_result_cache_for_session to off;
SET search_path TO {};

select lo_orderdate, sum(lo_extendedprice*lo_discount) as revenue
from lineorder, part
where lo_partkey = p_partkey
group by lo_orderdate
order by lo_orderdate
"""

def compareQueryTimes(schema):
    queryTimes = []

```

```
for i,query in enumerate([oneDim_SQL, twoDim_SQL, drill_SQL, oneDimSameDist_SQL]):  
    t0 = time()  
    q = query.format(schema)  
    %sql $q  
    queryTime = time()-t0  
    queryTimes.append(queryTime)  
return pd.DataFrame({"query":["oneDim", "twoDim", "drill", "oneDimSameDist"], "queryTime_" + schema:queryTimes}).set_index("query")
```

```
noDistQueryTimes = compareQueryTimes("nodist")  
distQueryTimes = compareQueryTimes("dist")
```

```
queryTimeDF = noDistQueryTimes.join(distQueryTimes)  
queryTimeDF.plot.bar()  
plt.show()
```

```
improvementDF = queryTimeDF["distImprovement"] = 100.0 * (queryTimeDF['queryTime_nodist'] - queryTimeDF['queryTime_dist']) / queryTimeDF['queryTime_nodist']  
improvementDF.plot.bar(title="% dist Improvement by query")  
plt.show()
```