

Snowflake Partner Boot Camp

Hands-on Workbook

Snowflake Korea

본 워크북은 파트너사의 Snowflake 데이터 플랫폼 설계자, 관리자를 위해 핵심적인 기능들을 직접 실행시켜 보면서 중요한 컨셉과 피처를 이해하도록 하는 가이드입니다.

Features Deep Dive 세션을 통해서 Snowflake의 핵심적인 컨셉과 기능을 이해한 다음에 직접 핸즈온을 진행하면서 해당 토픽에 대한 이해도를 높이는 것을 목표로 하고 있습니다.

[Snowflake 30일 무료 평가판에 등록](#)해서 실습이 진행되며, 핸즈온이 진행되는 동안 [Snowflake Documentation\(한국어\)](#)을 참조하면서 각 토픽에 대한 이해도를 높이도록 권장 드립니다.

1. Course Setup

Trial 계정 등록 및 랩 환경 준비

[Snowflake 30일 무료 평가판](#)에서 Trial 계정을 등록합니다.

워크샵이 진행되는 동안 [Snowflake Documentation\(한국어\)](#)을 함께 참조하도록 합니다.

- Email : 비즈니스 이메일 계정을 등록하기 바랍니다.
- 클라우드 공급자, 리전 및 에디션: AWS의 Seoul 리전과 Enterprise 에디션을 선택하도록 권장합니다.

등록 후, 활성화 링크와 Snowflake 계정 URL이 담긴 이메일을 받게 되므로 스팸 메일로 분류되지 않았는지 확인합니다.

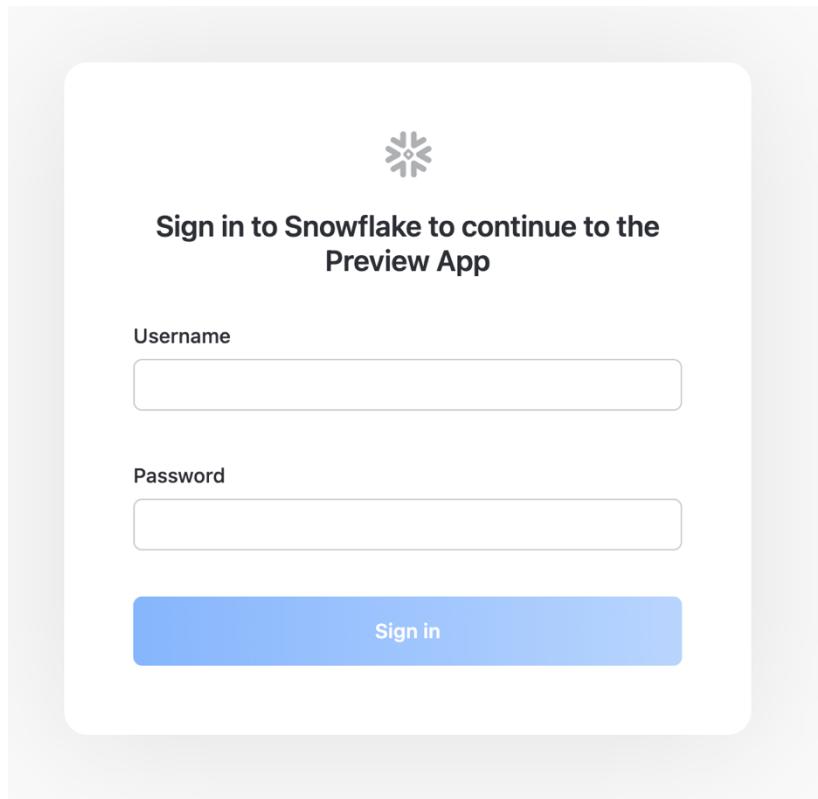
(3 ~ 5분 후에 링크가 활성화되는 경우도 있으므로 이 경우에는 조금 기다린 후에 활성화 링크를 클릭하고 로그인하시기 바랍니다.)

2. Snowflake User Interface (Snowsight)

Snowflake 사용자 인터페이스(UI)에 로그인

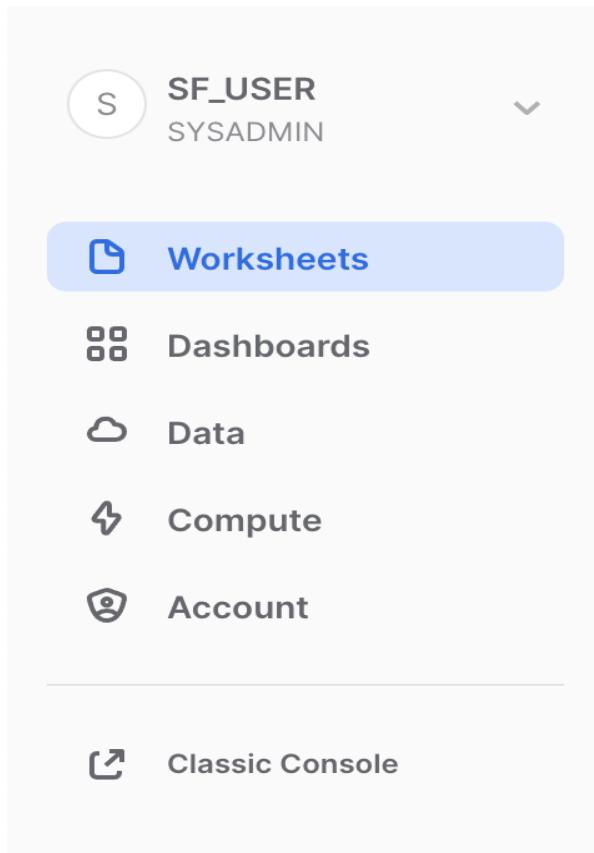
브라우저 창을 열고 등록 이메일에서 받은 Snowflake 30일 평가판 환경의 URL을 입력합니다.

아래 로그인 화면이 나타납니다. 등록에 사용한 사용자 이름 및 암호를 입력하세요.



Snowflake UI 확인

로그인 이후에 사용자 인터페이스를 확인합니다. 왼쪽의 대메뉴부터 위에서 아래로 이동하면서 알아 보도록 하겠습니다.

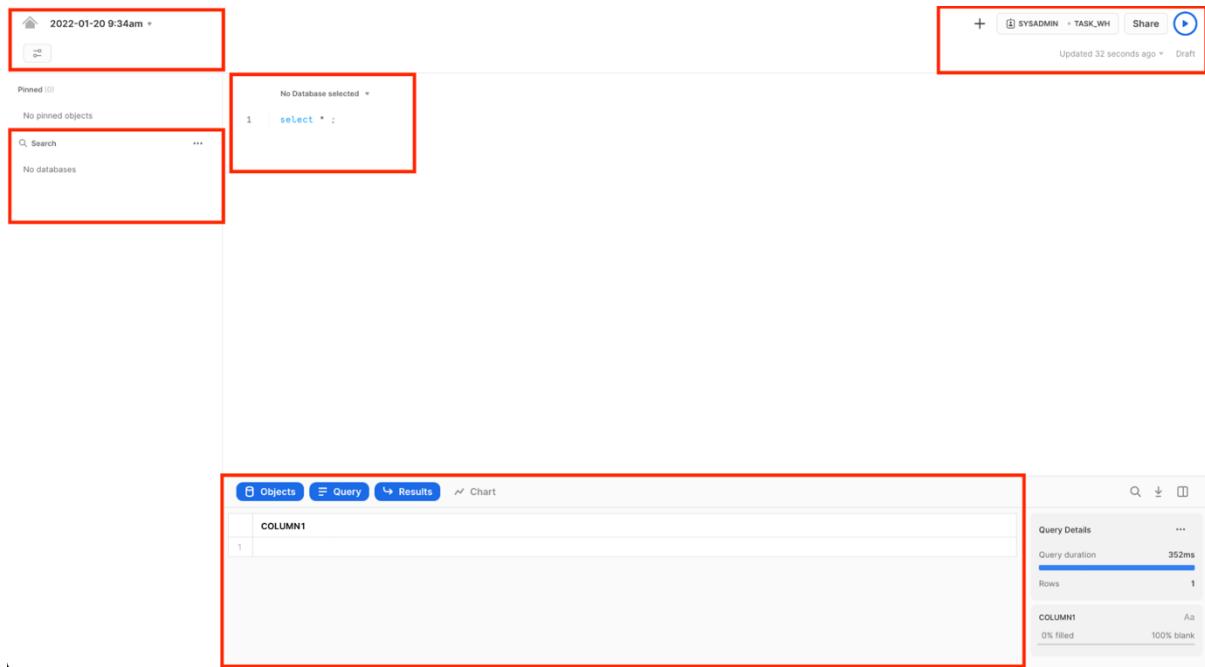


왼쪽의 대메뉴를 통해서 각 구성요소들을 살펴보도록 하겠습니다.

Worksheets

A screenshot of the 'Worksheets' page. The left sidebar is identical to the one above. The main area has a title 'Worksheets' with a sub-header 'Welcome to Worksheets'. It features a large blue button labeled '+ Worksheet' with a red rectangle drawn around it. Below the button is a small icon of a speech bubble with a line graph. A message at the bottom reads: 'Welcome to Worksheets. Use Worksheets to write and run queries. New features like autocomplete, schema browsing, automatic statistics, and sharing help you get the most out of your queries.' There are also 'Search' and 'New folder' buttons.

Worksheets 탭은 SQL 쿼리 실행, DDL 및 DML 작업 수행 그리고 쿼리 또는 작업 완료 시 결과 확인을 위한 인터페이스를 제공합니다. + **Worksheet** 버튼을 클릭하여 새로운 워크시트를 생성 할 수 있습니다.



왼쪽 상단 모서리에는 다음이 포함됩니다.

- **Home** 아이콘: 기본 콘솔로 돌아가거나 워크시트를 닫을 때 사용합니다.
- **Worksheet 이름** 드롭다운: 기본 이름은 워크시트가 생성된 타임스탬프입니다. 타임스탬프를 클릭하여 워크시트 이름을 편집합니다. 드롭다운에는 워크시트에 대해 수행할 수 있는 추가 작업도 표시됩니다.
- **필터 관리자** 버튼: 맞춤 필터는 하위 쿼리 또는 값 목록으로 해석되는 특수 키워드입니다.

오른쪽 상단 모서리에는 다음이 포함됩니다.

- + 버튼: 새 워크시트를 생성합니다.
- **Context** 상자: 해당 세션 동안 사용할 역할과 웨어하우스를 선택합니다. UI 또는 SQL 명령을 통해 변경할 수 있습니다.
- **Share** 버튼: 공유 메뉴를 열어 다른 사용자와 공유하거나 워크시트 링크를 복사합니다.
- **Play/Run** 버튼: 현재 커서가 있는 SQL 문 또는 선택한 여러 문을 실행합니다.

중간 창에는 다음이 포함됩니다.

- 워크시트에 대한 데이터베이스/스키마/개체 컨텍스트를 설정하기 위한 상단의 드롭다운.

- 쿼리 및 기타 SQL 문을 입력하고 실행하는 일반 작업 영역입니다.

가운데 왼쪽 패널에는 현재 워크시트에 사용 중인 역할이 액세스할 수 있는 모든 데이터베이스, 스키마, 테이블 및 보기의 탐색을 위한 데이터베이스 개체 브라우저가 있습니다.

아래쪽 창에는 쿼리 및 기타 작업의 결과가 표시됩니다. 또한 UI에서 해당 패널을 열고 닫는 4가지 옵션(**Object, Query, Result, Chart**)이 포함됩니다. **Chart**는 반환된 결과에 대한 시각화 패널을 열 수 있습니다.

Dashboards

The screenshot shows the Snowflake Data Studio interface. On the left, there's a sidebar with a tree view of database objects: SF_USER (DBA_CITIBIKE). Below the tree are categories: Worksheets, Dashboards (which is selected and highlighted in blue), Data, Compute, Account, and a link to the Classic Console. At the bottom of the sidebar, it says DEMO265 | PREVIEW APP. The main content area is titled "Dashboards". It features a search bar with a magnifying glass icon and a "Search" button, followed by a three-dot menu and a blue button labeled "+ Dashboard". A large red box highlights the "+ Dashboard" button. Below this, there's a small icon representing a dashboard with a chart and a plus sign. The text "Welcome to Dashboards" is displayed, along with the instruction: "Use Dashboards to create and share beautiful and interactive visualizations of your data."

Dashboards 탭을 사용하면 하나 이상의 차트(재배열할 수 있는 태일 형태의 그래프)를 유연하게 표시할 수 있습니다. 대시보드에 표현되는 각 태일 및 위젯은 워크시트에 결과를 반환하는 SQL 쿼리를 실행하여 생성됩니다.

Databases

The screenshot shows the Snowflake UI with the left sidebar expanded. Under the 'Data' section, the 'Databases' tab is selected. The main area displays a table titled 'Databases' with one entry: 'SNOWFLAKE'. The table includes columns for NAME, SOURCE, OWNER, and CREATED. A search bar and filter buttons are at the top right.

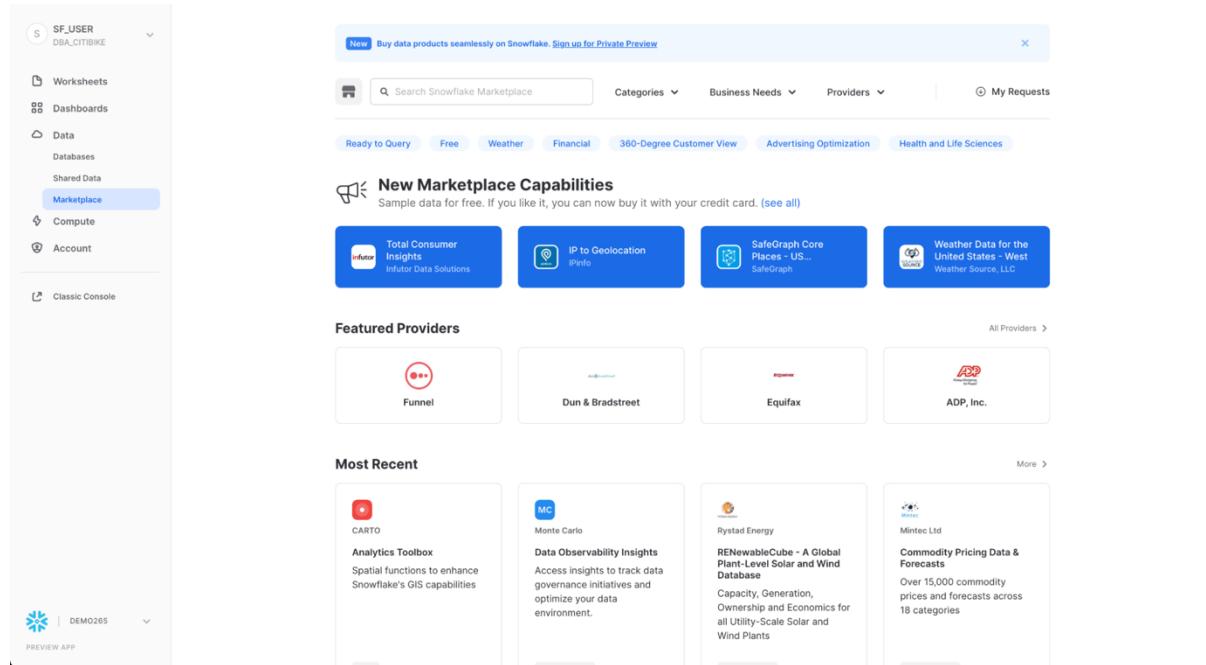
Data 아래의 **Database** 탭에는 사용자가 생성했거나 액세스 권한이 있는 데이터베이스에 대한 정보가 표시됩니다. 데이터베이스 생성, 복제, 삭제 또는 소유권(Ownership)을 이전하고 UI에서 데이터를 로드할 수 있습니다.

Shared Data

The screenshot shows the Snowflake UI with the left sidebar expanded. Under the 'Data' section, the 'Shared Data' tab is selected. The main area displays a table titled 'Ready to Get' with one entry: 'SAMPLE_DATA'. Below this, there is a section titled 'Snowflake Demo Resources' featuring several demo datasets. At the bottom, there is a 'SE Sandbox' section.

또한 Data 아래의 **Shared Data** 탭에서는 데이터 복사본을 만들지 않고도 별도의 Snowflake 계정 또는 외부 사용자 간에 Snowflake 테이블을 쉽고 안전하게 공유하도록 데이터 공유를 구성할 수 있습니다.

Marketplace



The screenshot shows the Snowflake Marketplace interface. On the left, a sidebar menu includes options like Worksheets, Dashboards, Data, Shared Data, **Marketplace**, Compute, Account, and Classic Console. The main area features a search bar and navigation tabs for Business Needs and Providers. A prominent section titled "New Marketplace Capabilities" offers sample data for free, with links to Total Consumer Insights, IP to Geolocation, SafeGraph Core Places, and Weather Data for the United States - West. Below this are sections for Featured Providers (Funnel, Dun & Bradstreet, Equifax, ADP, Inc.) and Most Recent products (CARTO Analytics Toolbox, Monte Carlo Data Observability Insights, RENewableCube, and Mintec Ltd Commodity Pricing Data & Forecasts).

Data 아래의 마지막 탭인 **Marketplace**는 모든 Snowflake 고객이 공급자가 제공한 데이터 세트를 검색하고 사용할 수 있는 곳입니다. 공유 데이터에는 공개 및 개인화의 두 가지 유형이 있습니다. 공개 데이터는 즉시 쿼리할 수 있는 무료 데이터 세트입니다. 개인화된 데이터는 데이터 공유 승인을 위해 데이터 제공자에게 연락해야 합니다.

History

The screenshot shows the Snowflake History page. The left sidebar has a tree structure with 'SF_USER DBA_CITIBIKE' at the top, followed by 'Worksheets', 'Dashboards', 'Data', 'Compute' (which is expanded to show 'History' which is selected, 'Warehouses', and 'Resource Monitors'), and 'Account'. Below the sidebar is a 'Classic Console' link. The main area is titled 'History' with tabs for 'Queries' (selected) and 'Copies'. It shows '0 Queries' and includes a search icon and filter buttons for 'Status All', 'User ADMIN', 'Filters', 'Columns', and a refresh button. Below these is a message: 'No Queries' followed by the subtext 'There are no queries matching your filters.'

Compute 아래의 History 탭에 다음이 표시됩니다.

- **Queries**는 결과(사용자, 웨어하우스, 상태, 쿼리 태그 등)를 다듬는 데 사용할 수 있는 필터와 함께 이전 쿼리가 표시되는 곳입니다. Snowflake 계정에서 지난 14일 동안 실행된 모든 쿼리의 세부 정보를 봅니다. 자세한 내용을 보려면 쿼리 ID를 클릭하십시오.
- **Copies**은 Snowflake로 데이터를 수집하기 위해 실행되는 복사 명령의 상태를 보여줍니다.

Warehouses

The screenshot shows the Snowflake interface for managing Warehouses. On the left, a sidebar menu is open under the 'Compute' section, with 'Warehouses' selected. The main content area is titled 'Warehouses' and shows a table with one row:

NAME	STATUS	SIZE	CLUSTERS	RUNNING	QUEUED	OWNER	CREATED
COMPUTE_WH	Suspended	X-Large	1-1	0	0	SYSADMIN	just now

At the top right of the table, there is a blue button labeled '+ Warehouse'. Above the table, there are search and filter options: 'Search', 'Status All', 'Size All', and a clear button 'C'.

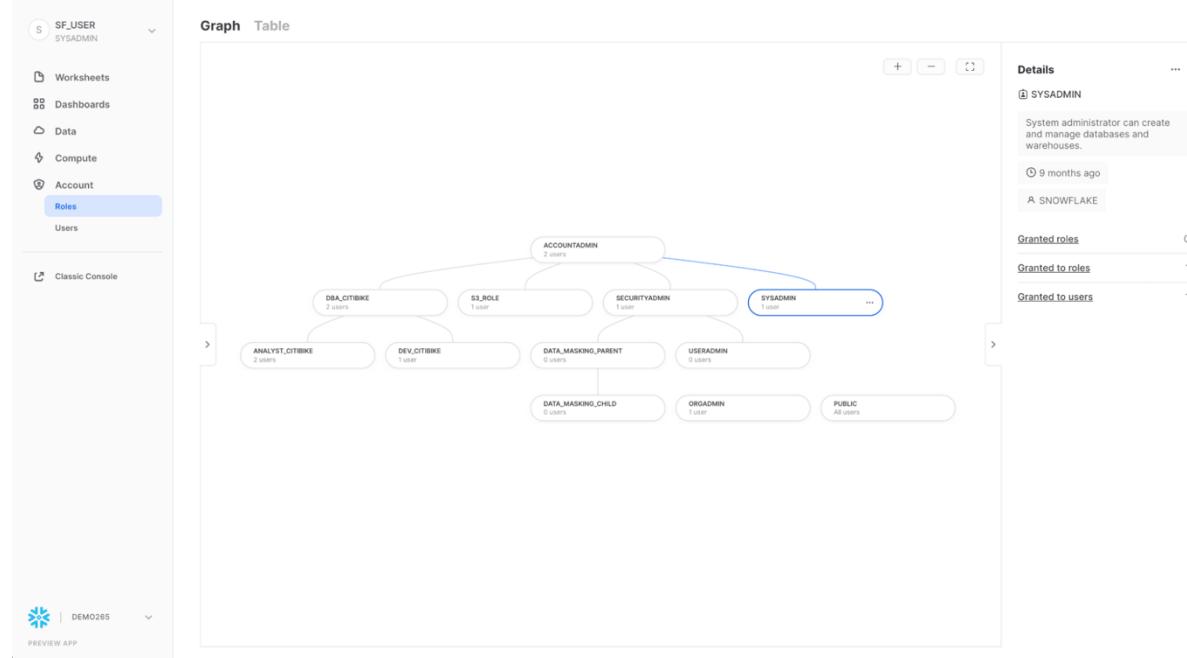
또한 Compute 아래의 Warehouses 탭은 Snowflake에서 데이터를 로드하거나 쿼리하기 위해 가상 웨어하우스로 알려진 컴퓨팅 리소스를 설정하고 관리하는 곳입니다. COMPUTE_WH(XS)라는 웨어하우스가 이미 사용자 환경에 있습니다.

Resource Monitors

The screenshot shows the Snowflake interface for managing Resource Monitors. On the left, a sidebar menu is open under the 'Compute' section, with 'Resource Monitors' selected. The main content area is titled 'Resource Monitors' and shows a message: 'No Resource Monitors'. Below this message, there is a small icon of a person with a gear and a brief description: 'There are no resource monitors associated with this role. Switch roles to view resource monitors available to that role.'

Compute 아래의 마지막 탭인 **Resource Monitors**에는 가상 웨어하우스(Virtual Warehouse)가 소비하는 크레딧(Credit)를 제어하기 위해 생성된 모든 리소스 모니터가 표시됩니다. 각 리소스 모니터에 대해 크레딧 할당량, 모니터링 유형, 일정 및 가상 웨어하우스가 크레딧 한도에 도달했을 때 수행되는 조치를 보여줍니다.

Roles



Account 아래의 **Roles** 탭에는 역할 목록과 계층(Hierarchy)이 표시됩니다. 이 탭에서 역할을 생성, 재구성 및 사용자에게 부여할 수 있습니다. 역할은 페이지 상단의 **Table**을 클릭하여 표/목록 형식으로 표시할 수도 있습니다.

Users

The screenshot shows the Snowflake UI with the following details:

- Top Left:** SF_USER, SYSADMIN.
- Left Sidebar (Account Tab):** Worksheets, Dashboards, Data, Compute, Account, Roles, **Users** (selected), Classic Console.
- Center:** **Users** page, 0 Users. Search, Owner All, Status All buttons.
- Bottom Left:** DEMO265, PREVIEW APP.

A message at the bottom center states: "Data not found. Either this data doesn't exist or your role may not have access to it."

또한 **Account** 탭 아래의 **Users** 탭에는 계정의 사용자 목록, 기본 역할 및 사용자 소유자가 표시됩니다. 새 계정의 경우 추가 역할이 생성되지 않았기 때문에 레코드가 표시되지 않습니다. 탭에서 사용할 수 있는 모든 정보를 보려면 역할을 ACCOUNTADMIN으로 전환(Switch Role)하세요.

The screenshot shows the Snowflake UI with the following details:

- Top Left:** SF_USER, SYSADMIN.
- User Dropdown:** SF_USER, Switch Role (SYSADMIN), Profile, Partner Connect, Documentation, Sign Out.
- Bottom Left:** Classic Console.

UI 오른쪽 상단의 사용자 이름을 클릭하면 비밀번호, 역할 및 기본 설정을 변경할 수 있습니다. Snowflake에는 여러 시스템 정의 역할(System Defined Roles)이 있습니다. 현재 기본 역할인 **SYSADMIN**을 사용하고 있으며 대부분의 실습에서 이 역할을 유지하겠습니다.

SYSADMIN **SYSADMIN**(시스템 관리자라고도 함) 역할은 웨어하우스, 데이터베이스 및 기타 객체를 계정에 생성할 수 있는 권한을 가집니다. 실제 환경에서는 서로 다른 역할에 따라서 적합한 Role을 할당하고 사용자에게 이 Role을 할당하는 형태로 액세스 제어가 이루어 집니다.

- 액세스 제어의 개요: <https://docs.snowflake.com/ko/user-guide/security-access-control-overview.html>
- 액세스 제어 구성하기: <https://docs.snowflake.com/ko/user-guide/security-access-control-configure.html>
- 사용자 지정 역할 만들기: <https://docs.snowflake.com/ko/user-guide/security-access-control-configure.html#creating-custom-roles>

3. Staging and Loading Data

샘플 트랜잭션 정형 데이터를 Snowflake에 로드할 준비부터 시작하겠습니다.

이 섹션은 다음과 같은 단계로 진행됩니다.

- 데이터베이스 및 테이블 생성
- 외부 스테이지(External Stage) 생성
- 데이터에 대한 파일 형식(File Format) 생성

Snowflake 데이터 로딩하는 방법은 아래 매뉴얼에서 확인하기 바랍니다.

Snowflake에 데이터 로딩하기: <https://docs.snowflake.com/ko/user-guide-data-load.html>

Snowflake로 데이터 가져오기

사용할 데이터는 자전거 공유 서비스의 트랜잭션 데이터입니다. 특정 정거장에서 자전거를 빌려서 특정 장소에 반납하면 하나의 트랜잭션(한 번의 Trip)이 완결되고 하나의 레코드로 저장됩니다. 이 데이터는 미국 동부 지역의 Amazon AWS S3 버킷에 미리 저장되어 있으며, 이동 시간, 위치, 사용자 유형, 성별, 나이 등에 관한 관련 정보로 구성됩니다. AWS S3에서 이 데이터는 6,150만 행, 376개의 객체로 표현되고 1.8GB로 압축되어 있습니다.

아래는 Citi Bike CSV 데이터 파일 중 하나의 한 조각입니다.

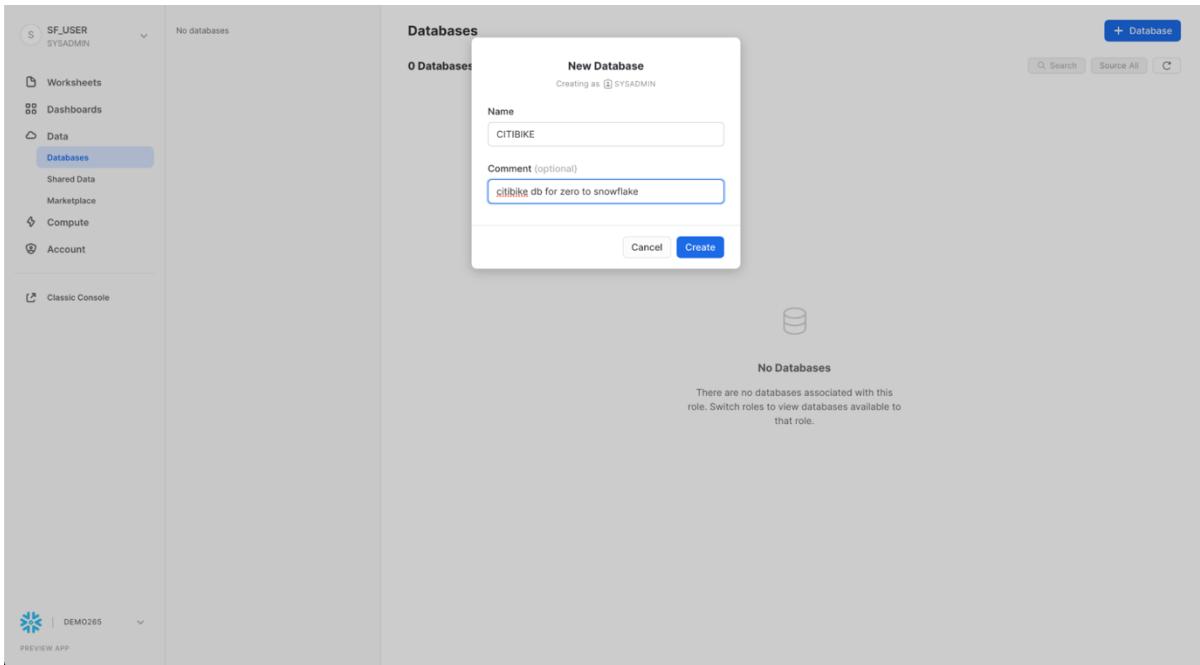
```
"tripduration","starttime","stoptime","start station id","start station name","start station latitude","start station longitude","end station id","end station name","end station latitude","end station longitude","bikeid","name_localizedValue0","usertype","birth year","gender"196,"2018-01-01 00:01:51","2018-01-01 00:05:07",315,"South St & Gouverneur Ln",40.70355377,-74.00670227,259,"South St & Whitehall St",40.70122128,-74.01234218,18534,"Annual Membership","Subscriber",1997,1207,"2018-01-01 00:02:44","2018-01-01 00:06:11",3224,"W 13 St & Hudson St",40.73997354103409,-74.00513872504234,470,"W 20 St & 8 Ave",40.74345335,-74.00004031,19651,"Annual Membership","Subscriber",1978,1613,"2018-01-01 00:03:15","2018-01-01 00:13:28",386,"Centre St & Worth St",40.71494807,-74.00234482,2008,"Little West St & 1 Pl",40.70569254,-74.01677685,21678,"Annual Membership","Subscriber",1982,1
```

맨 위에 헤더가 한 줄이 있고, 각 컬럼은 큰 따옴표로 둘러싸여 있으며 쉼표로 구분된 형식입니다. 이 형식은 이 섹션의 뒷부분에서 이 데이터를 저장할 Snowflake 테이블을 구성할 때 적용됩니다.

데이터베이스 및 테이블 생성

먼저, 정형 데이터를 로딩하는 데 사용할 CITIBIKE라는 이름의 데이터베이스를 생성합니다.

Databases 탭으로 이동합니다. 만들기를 클릭하고, 데이터베이스 이름을 CITIBIKE로 지정한 뒤, 마침을 클릭합니다.



이제 Worksheets 탭으로 이동합니다.

A screenshot of the Snowflake Worksheet interface. The top bar shows the date '2022-01-20 9:34am', user 'SYSADMIN - TASK_WH', and status 'Draft'. The left sidebar shows pinned objects and a search bar. The main area has a query editor with the placeholder 'No Database selected'. Below it is a results table with one row labeled 'COLUMN1'. On the right, there's a 'Query Details' panel showing 'Query duration 790ms', 'Rows 1', and 'COLUMN1 0% filled 100% blank'. Navigation tabs at the bottom include 'Objects', 'Query' (which is active), 'Results', and 'Chart'.

워크시트 내에서 컨텍스트를 적절하게 설정해야 합니다. 워크시트의 오른쪽 상단에 + 옆에 있는 상자를 클릭하여 상황에 맞는 메뉴를 표시합니다. 여기에서 각 워크시트에서 보고 실행할 수 있는 요소를 제어합니다. 여기서 UI를 사용하여 컨텍스트를 설정합니다.

다음과 같이 Role과 Warehouse의 컨텍스트 설정을 선택합니다.

Role: SYSADMIN Warehouse: COMPUTE_WH

The screenshot shows the Snowflake UI interface. At the top right, there is a context menu with 'SYSADMIN' and 'COMPUTE_WH' selected. On the left, a sidebar shows pinned objects and a search bar. The main area displays a query: 'select col from table where created = :daterange'. The bottom right corner shows 'Query Details' with a duration of 790ms and 1 row.

다음으로 데이터베이스 드롭다운에서 Database와 Schema의 컨텍스트 설정을 선택합니다.

Database: CITIBIKE Schema = PUBLIC

The screenshot shows the Snowflake UI interface. At the top right, there is a context menu with 'SYSADMIN' and 'COMPUTE_WH' selected. On the left, a sidebar shows pinned objects and a search bar. The main area displays a query: 'select col from table where created = :daterange'. A dropdown menu for 'No Database selected' is open, showing options: 'CITIBIKE' and 'SNOWFLAKE_SAMPLE_DATA'. The bottom right corner shows 'Query Details' with a duration of 56ms and 1 row.

SQL로 데이터베이스 생성 및 컨텍스트 설정

앞에서 UI로 진행했던 과정은 SQL을 통해서도 진행할 수 있습니다.

```
/* 먼저 어떤 Role과 Warehouse를 사용할지 컨텍스트를 지정합니다.*/
use role sysadmin;
use warehouse compute_wh;

/* citibike 데이터베이스를 생성합니다.*/
create or replace database citibike;

/* 테이블을 생성할 데이터베이스와 스키마를 컨텍스트로 지정합니다.*/
use database citibike;
use schema public;
```

다음으로 샘플 트랜잭션 정형 데이터를 로드하는 데 사용할 TRIPS라는 테이블을 만듭니다. UI를 사용하는 대신 워크시트를 사용하여 테이블을 생성하는 DDL을 실행합니다. 다음 SQL 텍스트를 워크시트에 복사합니다.

```
/* 데이터를 로딩할 테이블을 해당 칼럼으로 생성합니다.*/
create or replace table trips
(tripduration integer,
starttime timestamp,
stoptime timestamp,
start_station_id integer,
start_station_name string,
start_station_latitude float,
start_station_longitude float,
end_station_id integer,
end_station_name string,
end_station_latitude float,
```

```

end_station_longitude float,
bikeid integer,
membership_type string,
usertype string,
birth_year integer,
gender integer);

/* 생성된 테이블과 설정된 parameter를 확인합니다. */
show tables like 'tri%' in citibike.public;

/* alter table 명령으로 parameter를 설정합니다. 예를 들어, Change_tracking을
true로 설정하면 테이블에 일어난 모든 change를 활용해서 CDC 작업을 수행할 수
있습니다. */
alter table trips set change_tracking = true;

/* 테이블의 각 컬럼 정보를 확인합니다. */
desc table trips;

```

- 테이블 관리: <https://docs.snowflake.com/ko/sql-reference/ddl-table.html#table-management>

명령을 실행하는 다양한 옵션.

SQL 명령은 UI를 통해서나, Worksheets 탭을 통해, SnowSQL 명령행 도구를 사용해서, ODBC/JDBC를 통해 선택한 SQL 편집기를 이용해서 또는 Python이나 Spark 커넥터를 통해 실행할 수 있습니다.

- SnowSQL(CLI 클라이언트) : <https://docs.snowflake.com/ko/user-guide/snowsql.html>
- ODBC 드라이버: <https://docs.snowflake.com/ko/user-guide/odbc.html>

커서를 명령 내 어디든 두고 페이지 상단의 파란색 실행 버튼을 클릭하여 쿼리를 실행하십시오. 또는 바로 가기 키 [Ctrl]/[Cmd]+[Enter]를 이용하십시오.

TRIPS 테이블이 생성되었는지 확인합니다. 워크시트 하단에 "테이블 TRIPS가 성공적으로 생성됨" 메시지를 표시하는 결과 섹션이 표시되어야 합니다.

The screenshot shows a Snowflake worksheet titled "CITIBIKE_ZERO_TO_SNOWFLAKE". On the left, there's a sidebar with pinned objects, a search bar, and a connection dropdown set to "CITIBIKE". The main area contains a code editor with the following SQL script:

```
create or replace table trips
(tripduration integer,
starttime timestamp,
endtime timestamp,
start_station_id integer,
start_station_name string,
start_station_latitude float,
start_station_longitude float,
end_station_id integer,
end_station_name string,
end_station_latitude float,
end_station_longitude float,
bikeid integer,
membership_type string,
usertype string,
birth_year integer,
gender integer);
```

Below the code editor, there's a results section with tabs for Objects, Query, Results, and Chart. The Results tab shows a status message: "Table TRIPS successfully created." To the right of the results, there's a "Query Details" panel with metrics: Query duration 277ms, Rows 1, and status 100% filled.

워크시트의 왼쪽 상단에 있는 **HOME** 아이콘을 클릭하여 **Databases** 탭으로 이동합니다. 그런 다음 **Data > Databases**를 클릭합니다. 데이터베이스 목록에서 **CITIBIKE > PUBLIC > TABLES**를 클릭하여 새로 생성된 **TRIPS** 테이블을 확인합니다.

The screenshot shows the Snowflake database interface. On the left, the navigation pane is open with sections for Worksheets, Dashboards, Data (selected), Databases (selected), Shared Data, Marketplace, Compute, and Account. The Databases section shows "CITIBIKE" and "PUBLIC" schemas. In the main content area, the "CITIBIKE / PUBLIC" schema is selected. The "Tables" tab is active, showing a single table named "TRIPS". The table details are as follows:

NAME	TYPE	OWNER	ROWS	BYTES	CREATED
TRIPS	Table	SYSADMIN	0	0.0B	3 minutes ago

방금 생성한 테이블 구조를 보려면 TRIPS 및 Columns 탭을 클릭합니다.

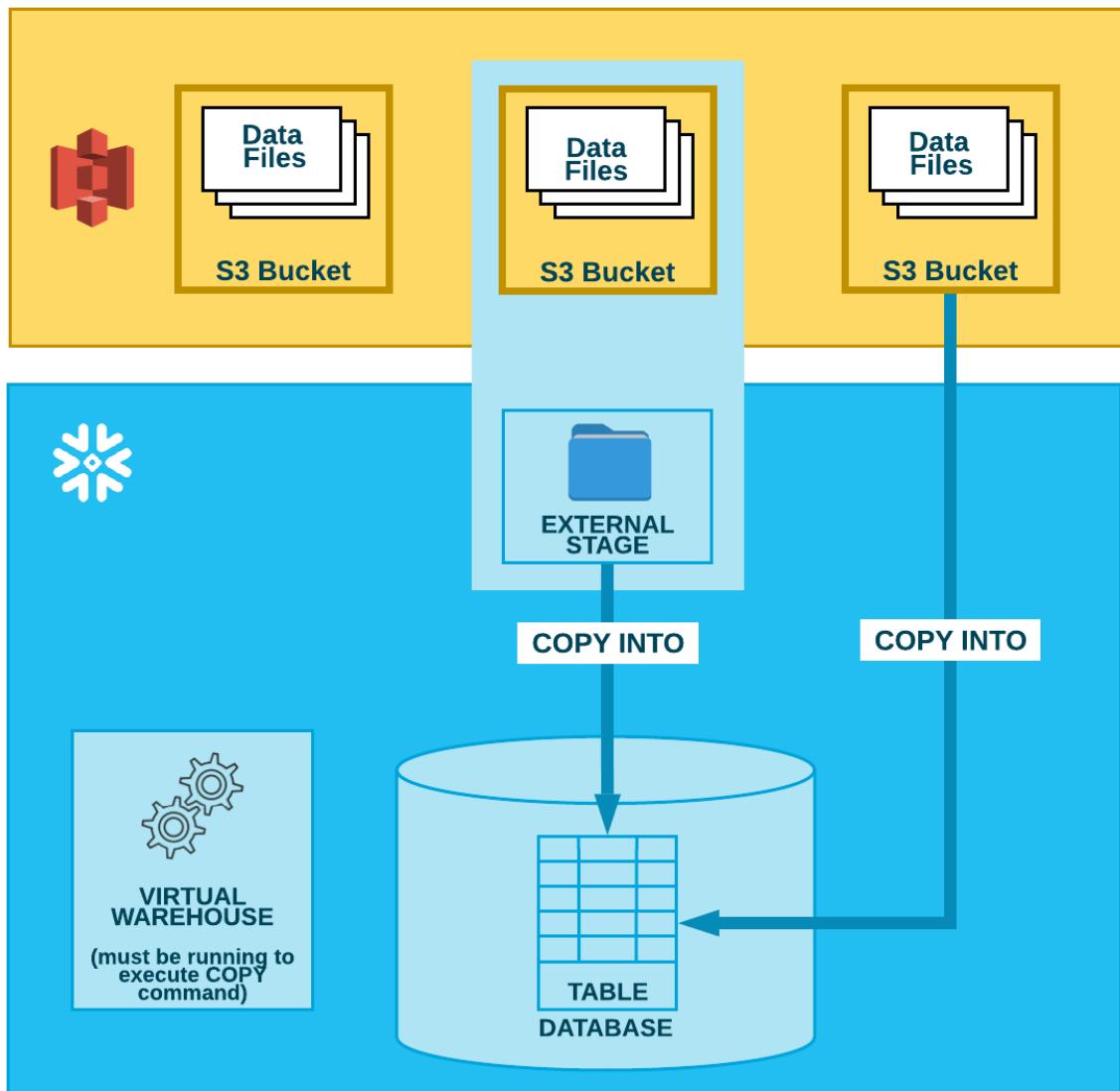
The screenshot shows a database management interface with a sidebar and a main content area. The sidebar on the left includes sections for Worksheets, Dashboards, Data (with Databases selected), Compute, and Account. A preview app icon at the bottom indicates a 'DEMO265' preview. The main content area displays the 'CITIBIKE / PUBLIC / TRIPS' table. It shows 16 columns with their names, types, nullability, and default values. The 'Columns' tab is active. The columns are:

NAME	TYPE	NULLABLE	DEFAULT
BIKEID	NUMBER(38,0)	Yes	NULL
BIRTH_YEAR	NUMBER(38,0)	Yes	NULL
END_STATION_ID	NUMBER(38,0)	Yes	NULL
END_STATION_LATITUDE	FLOAT	Yes	NULL
END_STATION_LONGITUDE	FLOAT	Yes	NULL
END_STATION_NAME	VARCHAR(16777216)	Yes	NULL
GENDER	NUMBER(38,0)	Yes	NULL
MEMBERSHIP_TYPE	VARCHAR(16777216)	Yes	NULL
STARTTIME	TIMESTAMP_NTZ(9)	Yes	NULL
START_STATION_ID	NUMBER(38,0)	Yes	NULL
START_STATION_LATITUDE	FLOAT	Yes	NULL
START_STATION_LONGITUDE	FLOAT	Yes	NULL
START_STATION_NAME	VARCHAR(16777216)	Yes	NULL
STOPTIME	TIMESTAMP_NTZ(9)	Yes	NULL
TRIPDURATION	NUMBER(38,0)	Yes	NULL
USERTYPE	VARCHAR(16777216)	Yes	NULL

Create an External Stage

데이터베이스 테이블로 읽을 데이터는 외부 S3 버킷에 준비되어 있으므로, 이 데이터를 사용하기 전에 먼저 외부 버킷의 위치를 지정하는 단계를 생성해야 합니다.

- 아마존 S3에서 대량 로드: <https://docs.snowflake.com/ko/user-guide/data-load-s3.html>



데이터 송신/전송 비용을 방지하려면 Snowflake 계정과 동일한 클라우드 제공업체 및 지역에서 스테이징 위치를 선택해야 합니다.

Databases 탭에서 CITIBIKE 데이터베이스와 PUBLIC 스키마를 클릭합니다. **Stage** 탭에서 **Create** 버튼을 클릭한 다음 **Stages > Amazon S3**를 클릭합니다.

The screenshot shows the Snowflake interface. On the left, the navigation sidebar has 'SF_USER' selected under 'Databases'. In the main area, 'CITIBIKE' is selected, followed by 'INFORMATION_SCHEMA', 'PUBLIC', 'Tables' (with 'TRIPS' listed), and finally 'Stages'. The 'Stages' tab is active. A context menu is open on the right, with 'Create' selected, showing options like 'Table', 'View', 'Stage', 'Pipe', 'Stream', 'Task', 'Function', and 'Procedure'. Below the stages list, it says 'No stages' and 'This schema contains no stages.' At the bottom, there's a preview section for 'DEMO265' with the message 'PREVIEW APP'.

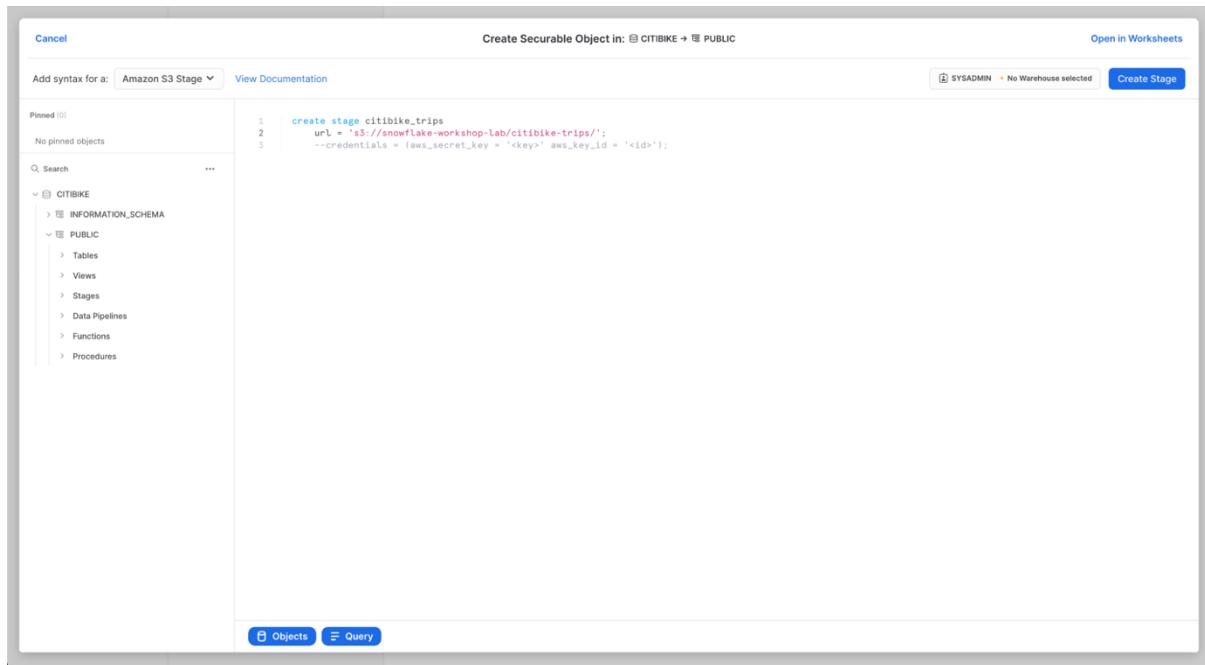
열리는 "Create Securable Object" 대화 상자에서 SQL 문에서 다음 값을 바꿉니다.

stage_name: citibike_trips

url: s3://snowflake-workshop-lab/citibike-trips-csv/

참고: URL 끝에 마지막 슬래시(/)를 포함해야 합니다. 그렇지 않으면 나중에 버킷에서 데이터를 로드할 때 오류가 발생합니다.

이 실습의 S3 버킷은 공개되어 있으므로 자격 증명 옵션을 비워 둘 수 있습니다. 실제 시나리오에서 외부 단계에 사용되는 버킷에는 주요 정보가 필요할 수 있습니다.



SQL로 External Stage 생성하기

앞에서 UI로 진행했던 과정은 SQL을 통해서도 진행할 수 있습니다.

```
/* 외부 스테이지를 생성합니다.
```

```
AWS S3의 버킷을 그대로 데이터 파일을 저장하는 외부 스테이지로 활용할 수 있습니다. */
```

```
create or replace stage citibike_trips
url='s3://snowflake-workshop-lab/citibike-trips-csv/';
```

- Create Stage: <https://docs.snowflake.com/ko/sql-reference/sql/create-stage.html>

- Storage Integration: <https://docs.snowflake.com/ko/sql-reference/sql/create-storage-integration.html>

실환경에서는 보안 Credentials의 자격 증명 옵션들이 필요하며 이 옵션들은 Create Stage에서 직접 지정을 하거나 Storage Integration에서 설정을 해 두고 활용할 수도 있습니다.

이제 `citibike_trips` 스테이지를 살펴보도록 하겠습니다.

```
list @citibike_trips;
```



하단 창의 결과에서 스테이지의 파일 목록을 확인해야 합니다.

The screenshot shows the Snowflake UI with the following details:

- Stage Name:** CITIBIKE_ZERO_TO_SNOWFLAKE
- Owner:** SYSADMIN + COMPUTE_WH
- Last Updated:** Updated 7 seconds ago
- Pinned Objects:** No pinned objects
- Search:** Search bar with placeholder > CITIBIKE
- Code View:** A code editor window showing the creation of the 'trips' table in the 'CITIBIKE.PUBLIC' schema. The code includes columns for tripduration, starttime, endtime, start_station_id, start_station_name, start_station_latitude, start_station_longitude, end_station_id, end_station_name, end_station_latitude, end_station_longitude, biked integer, membership_type string, usertype string, birth_year integer, and gender integer. A note at the bottom says 'List @citibike_trips:'.
- Results View:** A table showing the list of files in the stage. The table has columns: name, size, md5, and last_modified. The data shows 12 files, all of which are 100% filled. The last modified time is Wed, 12 Jan 2022 13:10:38 GMT for most files, except for one which is Wed, 12 Jan 2022 13:10:40 GMT.
- Query Details:** Shows a green bar for Query duration (3.6s), 3.5K rows, and a histogram for size (123) and md5 (4,866 to 9,720,801).

citibike_trips 스테이지에 저장된 데이터 파일의 현황을 검토합니다.

/* 마지막으로 실행시킨 쿼리의 결과를 통해서 전체 사이즈, 평균 파일 크기, 파일 갯수를 확인합니다. */

```
select
    floor(sum($2) / power(1024, 3), 1) total_compressed_storage_gb,
    floor(avg($2) / power(1024, 2), 1) avg_file_size_mb,
    count(*) as num_files
from
    table(result_scan(last_query_id()));
```

Result Scan은 결과가 테이블인 것처럼 이전 명령의 결과 세트를 반환합니다.

- Result Scan: https://docs.snowflake.com/ko/sql-reference/functions/result_scan.html

Create File Format

Snowflake로 데이터를 로드하려면, 먼저 데이터 구조와 일치하는 파일 형식(File Format)을 생성합니다.

워크시트에서 다음 명령을 실행하여 파일 형식을 만듭니다.`create file format`

```
/* 데이터 파일에 저장한 데이터의 구조를 반영하는 File Format을 생성합니다. */

create or replace file format csv type='csv'
    compression = 'auto' field_delimiter = ';' record_delimiter = '\n'
    skip_header = 0 field_optionally_enclosed_by = '\"' trim_space = false
    error_on_column_count_mismatch = false escape = 'none'
    escape_unenclosed_field = '\"'
    date_format = 'auto' timestamp_format = 'auto' null_if = ('') comment = 'file
format for ingesting data to snowflake';
```

- Create File Format: <https://docs.snowflake.com/ko/sql-reference/sql/create-file-format.html>

다음 명령을 실행하여 파일 형식이 올바른 설정으로 생성되었는지 확인합니다.

```
/* 파일 포맷이 생성되었는지 확인합니다. */

show file formats in database citibike;

/* 메타데이터를 통해서 각 파일의 경로 및 이름과 레코드 카운트에 대한 정보를 확인합니다. */

select metadata$filename, metadata$file_row_number
from @citibike_trips (file_format => csv);
```

생성된 파일 형식은 결과에 나열되어야 합니다.

The screenshot shows the Snowflake interface with the following details:

- Query Editor:** A code editor window titled "CITIBIKE_ZERO_TO_SNOWFLAKE". It contains the following SQL code:

```
8   start_station_name string,
9   start_station_longitude float,
10  start_station_latitude float,
11  end_station_id integer,
12  end_station_name string,
13  end_station_latitude float,
14  end_station_longitude float,
15  bike_id integer,
16  membership_type string,
17  user_type string,
18  birth_year integer,
19  gender integer
20 );
21 --created external stage. List files.
22 list @citibike_trips;
23
24 --create file format
25 CREATE_FILE_FORMAT "CITIBIKE"."PUBLIC".CSV TYPE = 'CSV' COMPRESSION = 'AUTO' FIELD_DELIMITER = ',' RECORD_DELIMITER = '\n' SKIP_HEADER = 0
FIELD_OPTIONALLY_ENCLOSED_BY = '\\"' TRIM_SPACE = FALSE ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE ESCAPE = 'NONE' ESCAPE_UNENCLOSED_FIELD = '\\"' DATE_FORMAT = 'AUTO'
TIMESTAMP_FORMAT = 'AUTO' NULL_IF = ('') COMMENT = 'creation of file format for zero to snowflake';
26
27 --verify file format is created
28 | show file formats in database citibike;
```
- Results Tab:** Shows a table with one row of results:

	created_on	name	database_name	schema_name	type	owner	comment	format
1	2022-01-20 11:12:27.666 -0800	CSV	CITIBIKE	PUBLIC	CSV	SYSADMIN	creation of file format for zero to snowflake	{"T":
- Query Details:** On the right, it shows the query duration as 97ms, 1 row processed, and 100% filled for columns created_on, name, database_name, and schema_name.

Copy Into <Table>

이 섹션에서는 데이터 웨어하우스와 COPY 명령을 사용하여 방금 생성한 Snowflake 테이블에 정형 데이터 대량 로드 (bulk loading)를 진행합니다.

- Copy Into <Table>: <https://docs.snowflake.com/ko/sql-reference/sql/copy-into-table.html>

데이터 로드를 위한 웨어하우스 크기 조정

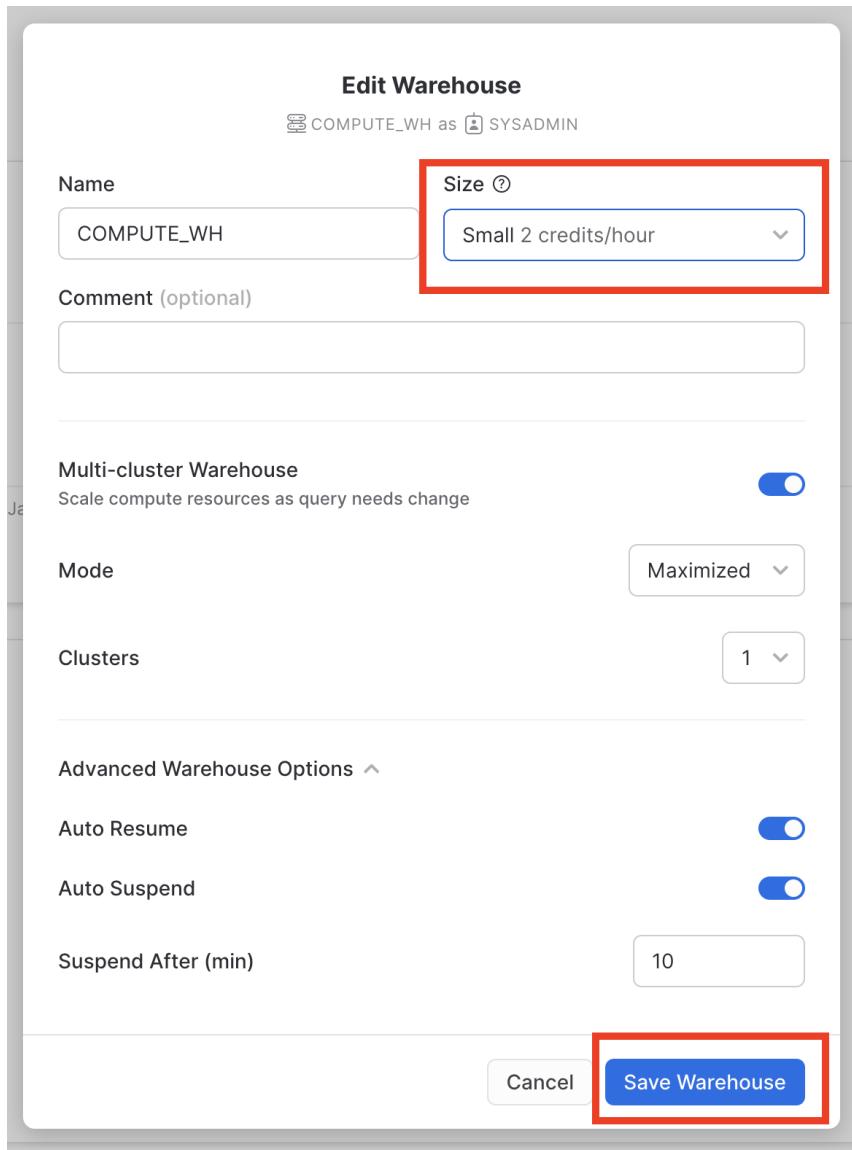
데이터 로딩은 많은 컴퓨팅 리소스를 사용할 수도 있어서 데이터 로딩을 하기 전에 가상 웨어하우스의 성능을 높이도록 하겠습니다.

Snowflake의 컴퓨팅 노드는 가상 웨어하우스 (Virtual Warehouse)라고 하며 워크로드가 데이터 로드, 쿼리 실행 또는 DML 작업을 수행하는지 여부에 따라 워크로드에 맞춰 동적으로 크기를 늘리거나 줄일 수 있습니다. 각 워크로드는 자체 데이터 웨어하우스를 보유할 수 있으므로 리소스 경합이 없습니다.

Home->Admin->Warehouses 탭으로 이동합니다. 여기에서 기존 웨어하우스를 모두 보고 사용 추세를 분석할 수 있습니다.

상단 오른쪽 상단 모서리에 있는 **+ Warehouse** 옵션을 확인하세요. 여기에서 새 웨어하우스를 빠르게 추가할 수 있습니다. 단, 30일 체험판 환경에 포함된 기존 웨어하우스 COMPUTE_WH를 사용합니다.

COMPUTE_WH 웨어하우스 행을 클릭합니다. 그런 다음 위의 오른쪽 상단 모서리에 있는 ...(점 점)을 클릭하고 **Edit**을 선택해서 Size를 조정할 수 있습니다.



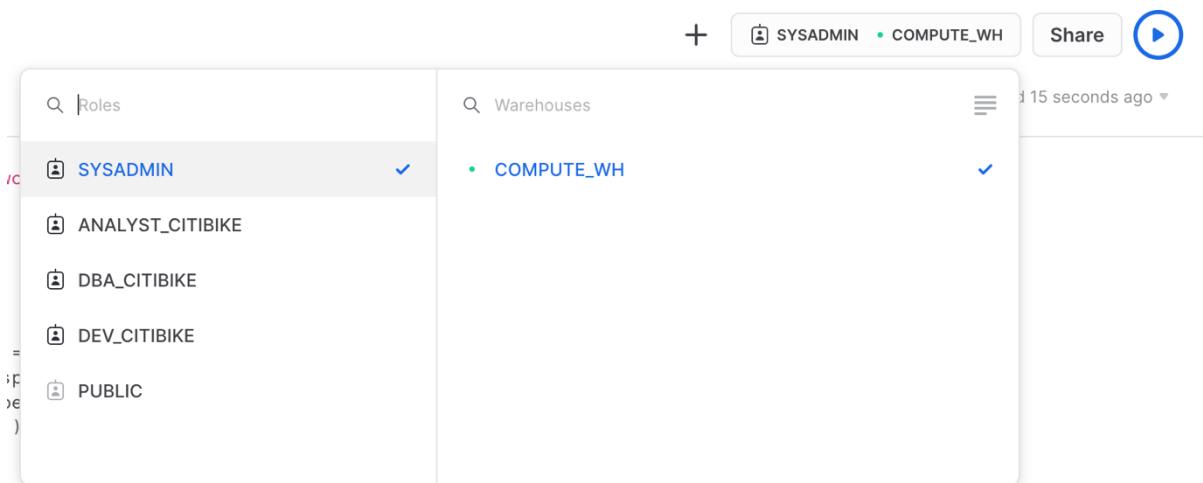
- **Size** 드롭다운은 웨어하우스의 용량을 선택하는 곳입니다. 더 큰 데이터 로드 작업이나 컴퓨팅 집약적인 쿼리의 경우 더 큰 웨어하우스가 권장됩니다. 이 데이터 웨어하우스의 **Size**를 X-Small에서 Small로 변경합니다. **Save Warehouse** 버튼을 클릭합니다.

데이터 로드

이제 COPY 명령을 실행하여 데이터를 앞서 생성한 TRIPS 테이블로 로드할 수 있습니다.

Worksheet 탭에서 컨텍스트가 올바르게 설정되었는지 확인합니다.

Role: SYSADMIN Warehouse: COMPUTE_WH Database: CITIBIKE Schema = PUBLIC



The screenshot shows the Snowflake Worksheet interface. At the top, it displays the context menu with 'SYSADMIN' selected under Roles and 'COMPUTE_WH' selected under Warehouses. The menu also includes options for Share and Run. Below the menu, there are two search fields: 'Roles' and 'Warehouses'. The 'Roles' field shows 'SYSADMIN' as the selected role, with other options like ANALYST_CITIBIKE, DBA_CITIBIKE, DEV_CITIBIKE, and PUBLIC listed. The 'Warehouses' field shows 'COMPUTE_WH' as the selected warehouse. A timestamp '15 seconds ago' is visible at the bottom right of the menu area.

워크시트에서 다음의 문을 실행하여 구성한 데이터를 테이블로 로드하고 실행 시간을 체크합니다.

```
/* citibike_trips 외부 스테이지에 있는 데이터 파일을 csv 파일 포맷에 맞춰서 trips 테이블에 로딩합니다. */
```

```
copy into trips from @citibike_trips file_format=csv pattern='.*CSV.*';
```

```
/* 테이블에 로딩된 데이터를 확인합니다. */
```

```
select * from trips limit 20;
```

결과 창에서 로드된 각 파일의 상태를 확인해야 합니다. 로드가 완료되면 오른쪽 하단의 **Query Details** 창에서 마지막으로 실행된 명령문에 대한 다양한 상태, 오류 통계 및 시각화를 스크롤할 수 있습니다.

CITIBIKE_ZERO_TO_SNOWFLAKE

SYSADMIN COMPUTE_WH Share Updated 1 minute ago

```

22 --create external stage via SQL
23 create or replace stage citibike_trips url = 's3://snowflake-workshop-lab/citibike-trips/';
24
25 --created external stage. List files.
26 list @citibike_trips;
27
28 --Create File Format
29 create or replace file format csv type='csv'
30 compression = 'auto' field_delimiter = ',' record_delimiter = '\n'
31 skip_header = 0 field_optionally_enclosed_by = '\"' trim_space = false
32 error_on_column_count_mismatch = false escape = 'none' escape_unenclosed_field = '\\"'
33 date_format = 'auto' timestamp_format = 'auto' null_if = ('') comment = 'file format for ingesting data for zero to snowflake';
34
35 --verify file format is created
36 show file formats in database citibike;
37
38 --copy data from stage into target table
39 copy into trips from @citibike_trips file_format=csv;
40
41
42
43
44
45
46
47
48
49

```

Objects Query Results Chart

	file	status	rows_parsed	rows_loaded	error_limit	errors_seen	first_error
1	s3://snowflake-workshop-lab/citibike-trips/trips_2013_1_1_0.csv.gz	LOADED	112,123	112,123	1	0	
2	s3://snowflake-workshop-lab/citibike-trips/trips_2013_2_7_0.csv.gz	LOADED	93,143	93,143	1	0	
3	s3://snowflake-workshop-lab/citibike-trips/trips_2013_5_7_0.csv.gz	LOADED	111,042	111,042	1	0	
4	s3://snowflake-workshop-lab/citibike-trips/trips_2014_0_7_0.csv.gz	LOADED	112,334	112,334	1	0	
5	s3://snowflake-workshop-lab/citibike-trips/trips_2014_3_0_0.csv.gz	LOADED	126,789	126,789	1	0	
6	s3://snowflake-workshop-lab/citibike-trips/trips_2014_4_7_0.csv.gz	LOADED	109,635	109,635	1	0	
7	s3://snowflake-workshop-lab/citibike-trips/trips_2014_8_7_0.csv.gz	LOADED	106,608	106,608	1	0	
8	s3://snowflake-workshop-lab/citibike-trips/trips_2015_0_2_0.csv.gz	LOADED	125,062	125,062	1	0	
9	s3://snowflake-workshop-lab/citibike-trips/trips_2015_1_2_0.csv.gz	LOADED	154,940	154,940	1	0	

Query Details

- Query duration: 38s
- Rows: 376
- file: 100% filled
- status: LOADED 376

그런 다음 **Home** 아이콘을 클릭한 다음 **Activity > Query History**를 클릭 합니다. 목록 맨 위에서 마지막으로 실행된 COPY INTO 문을 선택합니다. 쿼리가 실행하기 위해 취한 단계, 쿼리 세부 정보, 가장 비싼 노드 및 추가 통계를 살펴 볼 수 있습니다.



이제 더 큰 웨어하우스로 TRIPS 테이블을 다시 로드하여 추가 컴퓨팅 리소스가 로드 시간에 미치는 영향을 살펴보겠습니다.

워크시트로 돌아가서 TRUNCATE TABLE 명령을 사용하여 모든 데이터와 메타데이터를 지웁니다.

```
/* trips 테이블의 데이터와 메타데이터를 지웁니다. */
truncate table trips;
```

결과에 "Query produced no results"(쿼리에서 결과가 생성되지 않음)이 표시되어야 합니다.

다음 ALTER WAREHOUSE를 사용하여 웨어하우스 크기를 large으로 변경합니다.

```
/* 명령어를 통해서 직접 가상 웨어하우스의 크기를 large로 변경합니다. */
alter warehouse compute_wh set warehouse_size='large';
```

다음 SHOW WAREHOUSES를 사용하여 변경 사항을 확인하십시오.

```
/* 웨어하우스의 변경 사항을 확인 */
show warehouses;
```

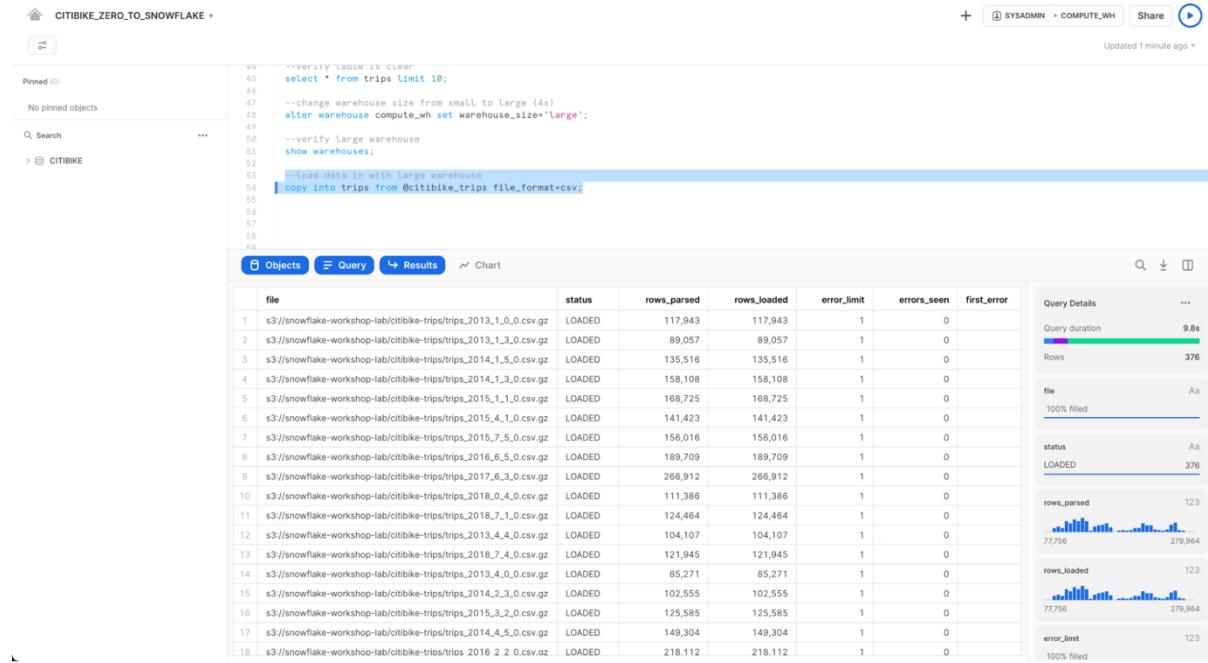
이제 성능의 향상된 가상 웨어하우스를 활용해서 다시 데이터를 로딩합니다

```
/* 데이터를 로딩하기 전에 validation_mode로 데이터 로딩시 발생할 에러를 미리
체크할 수 있습니다. */
copy into trips from @citibike_trips file_format=csv pattern='.*csv.*'
validation_mode=return_all_errors;
```

```
/* citibike_trips 외부 스테이지에 있는 데이터 파일을 csv 파일 포맷에 맞춰서 trips
테이블에 로딩합니다. */
copy into trips from @citibike_trips file_format=csv pattern='.*CSV.*' ;
```

```
/* 테이블에 로딩된 데이터를 확인합니다.*/
select * from trips sample (50 rows);
```

- Validating Staged Files: <https://docs.snowflake.com/en/sql-reference/sql/copy-into-table.html#validating-staged-files>



The screenshot shows the Snowflake Query Editor interface. The query history pane at the top contains the following SQL code:

```
--VERIFY TABLE IS CLEAR
select * from trips limit 10;
--change warehouse size from small to large (4x)
alter warehouse compute_wh set warehouse_size='large';
--verify large warehouse
show warehouses;
--load data in with large warehouse
copy into trips from @citibike_trips file_format=csv;
```

The results pane below displays a table with 18 rows of data, each representing a CSV file being loaded. The columns are: file, status, rows_parsed, rows_loaded, error_limit, errors_seen, and first_error. All rows show a status of 'LOADED' and 0 errors. The 'rows_loaded' column shows values ranging from 85,271 to 218,112. The 'rows_parsed' column shows values ranging from 102,555 to 121,945. The 'error_limit' column is consistently 1. The 'errors_seen' column is consistently 0.

file	status	rows_parsed	rows_loaded	error_limit	errors_seen	first_error
s3://snowflake-workshop-lab/citibike-trips/trips_2013_1_0_0.csv.gz	LOADED	117,943	117,943	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2013_1_3_0.csv.gz	LOADED	89,057	89,057	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2014_1_5_0.csv.gz	LOADED	135,516	135,516	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2014_1_3_0.csv.gz	LOADED	158,108	158,108	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2015_1_1_0.csv.gz	LOADED	168,725	168,725	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2015_4_1_0.csv.gz	LOADED	141,423	141,423	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2015_7_5_0.csv.gz	LOADED	156,016	156,016	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2016_6_5_0.csv.gz	LOADED	189,709	189,709	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2017_6_3_0.csv.gz	LOADED	266,912	266,912	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2018_6_4_0.csv.gz	LOADED	111,386	111,386	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2018_7_1_0.csv.gz	LOADED	124,464	124,464	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2013_4_4_0.csv.gz	LOADED	104,107	104,107	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2018_7_4_0.csv.gz	LOADED	121,945	121,945	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2013_4_0_0.csv.gz	LOADED	85,271	85,271	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2014_2_3_0.csv.gz	LOADED	102,555	102,555	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2015_3_2_0.csv.gz	LOADED	125,585	125,585	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2014_4_5_0.csv.gz	LOADED	149,304	149,304	1	0	
s3://snowflake-workshop-lab/citibike-trips/trips_2016_2_2_0.csv.gz	LOADED	218,112	218,112	1	0	

On the right side of the results pane, there are several data visualization charts for the 'file', 'status', 'rows_parsed', 'rows_loaded', and 'error_limit' columns.

로드가 완료되면 **Queries** 페이지로 다시 이동합니다(**Home** 아이콘 > **Activity** > **Query History**). 두 COPY INTO 명령의 시간을 비교하십시오.

Transforming Data During a Load

Copy into <Table> 명령은 Copy Into <Table> From (SELECT 구문)를 통해서 데이터 파일에서 데이터를 로드하기 전에 변환 작업을 진행할 수 있습니다.

```
/* 데이터 파일에서 일부 데이터를 선별적으로 테이블을 만들기 위해서 새로운 테이블을 생성합니다. 원본 데이터에 없는 tripid라는 새로운 컬럼도 구성합니다. */
create or replace table trips_agg
(tripid number autoincrement,
tripduration integer,
start_station_name string,
end_station_name string,
bikeid integer);

/* 테이블의 각 컬럼 정보를 확인합니다. */
desc table trips_agg;

/* 데이터 파일에서 필요한 컬럼만 추출해서 테이블로 로딩하고 tripid는 자동으로 생성되도록 합니다. */
copy into trips_agg(tripduration, start_station_name, end_station_name, bikeid)
from (select t.$1, t.$5, t.$9, t.$12 from @citibike_trips t)
file_format=csv pattern='.*csv.*';

select * from trips_agg limit 20;

/* 데이터 파일에서 일부 데이터를 전처리해서 로딩하기 위해서 새로운 테이블을 생성합니다. 원본 데이터에 없는 tripid라는 새로운 컬럼도 구성하고 bikeid는 스트링 타입으로 저장하고 membership에는 NULL 값이 없도록 저장하고자 합니다. */
create or replace table trips_cust
(tripid number autoincrement,
tripduration integer,
start_station_name string,
```

```
end_station_name string,  
bikeid string,  
membership_type string);  
  
desc table trips_cust;  
  
/* 데이터 파일에서 필요한 컬럼만 추출하고 해당하는 function을 통해서 데이터 로  
딩 전에 변환 작업을 수행합니다. */  
copy into trips_cust (tripduration, start_station_name, end_station_name, bikeid,  
membership_type)  
from (select t.$1, t.$5, t.$9, to_varchar(t.$12), ifnull(t.$13,'Free Membership')  
from @citibike_trips t)  
file_format=csv pattern='.*csv.*';  
  
select * from trips_cust limit 20;
```

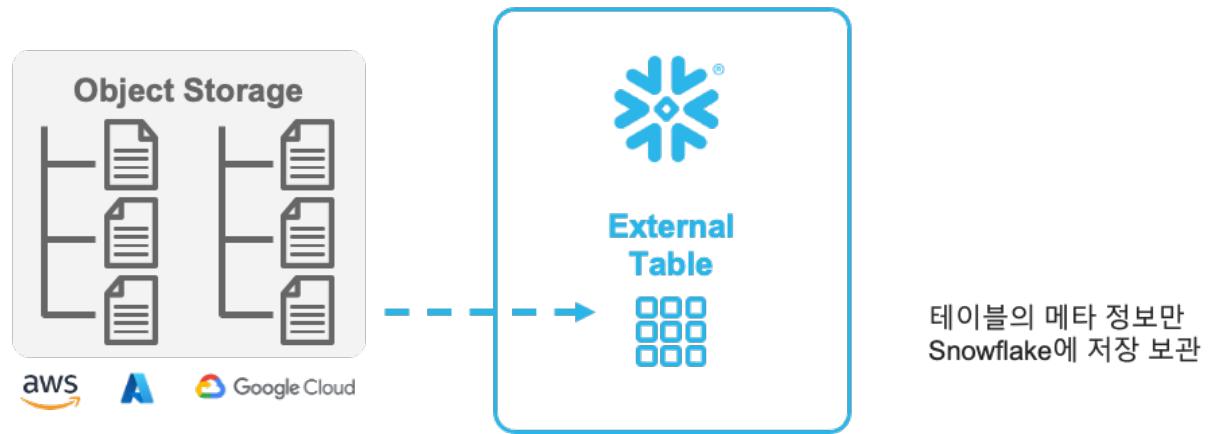
- 로드 중 데이터 변환하기: <https://docs.snowflake.com/ko/user-guide/data-load-transform.html#transforming-csv-data>

Create External Table

외부 테이블은 데이터 레이크를 Snowflake와 통합하기 위한 효과적인 방법입니다. 일반적인 테이블에서 데이터는 데이터베이스에 저장되지만, 외부 테이블에서 데이터는 외부 스테이지(External Stage)의 파일에 저장됩니다. 외부 테이블은 파일 이름, 버전 식별자 및 관련 속성과 같은 데이터 파일에 대한 파일 수준 메타데이터를 저장합니다. 이를 통해 데이터베이스 내부에 있는 것처럼 외부 스테이지에서 파일에 저장된 데이터를 쿼리할 수 있습니다. 외부 테이블은 `COPY INTO <테이블>` 문에서 지원하는 모든 형식으로 저장된 데이터에 액세스할 수 있습니다.

외부 테이블은 읽기 전용이므로 DML 작업을 수행할 수 없지만, 쿼리 및 조인 작업에는 외부 테이블을 사용할 수 있습니다. 외부 테이블에 대해 뷰를 생성할 수 있습니다.

데이터베이스 외부에 저장된 데이터를 쿼리하는 것은 기본 데이터베이스 테이블을 쿼리하는 것보다 느릴 수 있지만, 외부 테이블을 기반으로 하는 Materialized View는 쿼리 성능을 향상시킬 수 있습니다.



- 외부 테이블 소개: <https://docs.snowflake.com/ko/user-guide/tables-external-intro.html>

```
/* 외부 스테이지에 있는 경로와 파일 이름을 확인합니다.*/
select metadata$filename from @citibike_trips;

/* 앞에서 지정한 file_format에 따라 데이터 파일의 컬럼을 해석해서 외부 테이블에
대한 메타데이터만 저장합니다. 기본적으로 데이터는 variant 타입의 value 컬럼에
저장이 되므로 여기서 컬럼을 순서대로 추출해서 형변환을 하는 형태로 외부 테이블
의 컬럼을 정의합니다. */

create or replace external table trips_ext
(tripduration integer as (value:c1::integer),
starttime timestamp as (value:c2::timestamp),
stoptime timestamp as (value:c3::timestamp),
start_station_id integer as (value:c4::integer),
start_station_name string as (value:c5::string),
start_station_latitude float as (value:c6::float),
start_station_longitude float as (value:c7::float),
end_station_id integer as (value:c8::integer),
end_station_name string as (value:c9::string),
end_station_latitude float as (value:c10::float),
end_station_longitude float as (value:c11::float),
bikeid integer as (value:c12::integer),
membership_type string as (value:c13::string),
usertype string as (value:c14::string),
birth_year integer as (value:c15::integer),
gender integer as (value:c16::integer))
location=@citibike_trips/
auto_refresh=false
file_format=csv;

/* 수동으로 refresh를 실행해서 최신 업데이트를 반영합니다.*/
alter external table trips_ext refresh;
```

```
/* 외부 테이블에 쿼리를 바로 실행해서 결과를 확인합니다. 먼저 value 컬럼에 데이터가 어떻게 들어 있는지 확인합니다. */
select value from trips_ext limit 10;

/* 일반 컬럼들을 확인합니다. */
select tripduration, start_station_name, end_station_name from trips_ext limit 10;

/* 성능을 향상시키기 위해서 쿼리 결과를 저장하는 Materialized View를 생성합니다. */
create materialized view trips_mat as
select tripduration, start_station_name, end_station_name, bikeid from trips_ext
where tripduration > 10;

/* trips_mat view에 쿼리를 실행해서 결과를 확인합니다. */
select * from trips_mat limit 100;
```

- Create Materialized View: <https://docs.snowflake.com/ko/sql-reference/sql/create-materialized-view.html>

Continuous Load with Snowpipe

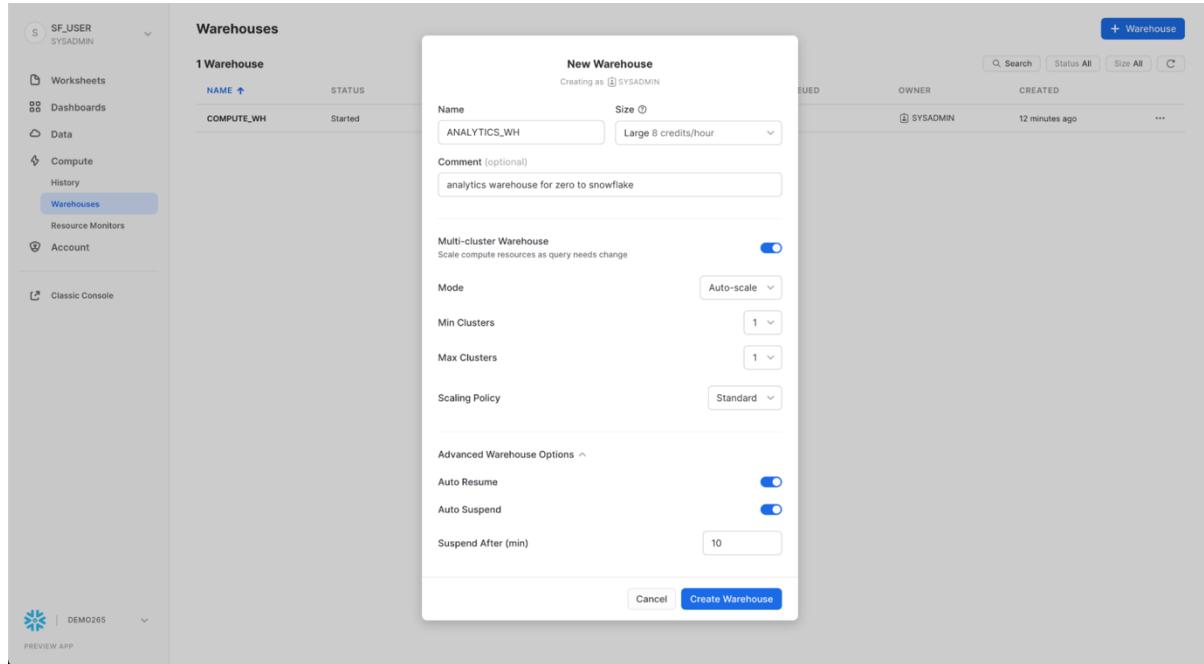
- Snowpipe를 이용한 연속 로드: <https://docs.snowflake.com/ko/user-guide/data-load-snowpipe.html>
- Snowpipe 시작하기 핸즈온 코스:
https://quickstarts.snowflake.com/guide/getting_started_with_snowpipe/index.html#0
- Kafka Connector로 데이터 로딩하기: <https://docs.snowflake.com/ko/user-guide/kafka-connector-overview.html>

4. Database Objects and Working with Queries

데이터 분석을 위한 가상 웨어하우스 생성 및 워크로드 격리

데이터 로드를 위한 가상 웨어하우스를 통해서 데이터 로딩 작업은 진행하고 있기 때문에, 데이터 분석을 실행하기 위한 새로운 가상 웨어하우스를 생성해서 분석 작업의 워크로드를 격리합니다.

Compute > Warehouses 탭으로 이동하여 + Warehouse를 클릭하고 새 웨어하우스의 이름을 ANALYTICS_WH로 지정하고 크기를 Large로 설정합니다.



SQL로 분석용 가상 웨어하우스 생성 및 설정

```
/* 데이터 분석용으로 large 크기의 새로운 가상 웨어하우스를 생성합니다. 5분 이상  
쿼리 요청이 없으면 자동으로 suspend되도록 설정을 하고 쿼리 요청이 들어 오면  
깨어 나서 요청을 처리하는 형태로 비용을 최적화합니다. */
```

```
create or replace warehouse analytics_wh  
warehouse_size='large'  
auto_suspend=300  
auto_resume=true;
```

- Create Warehouse: <https://docs.snowflake.com/ko/sql-reference/sql/create-warehouse.html>

데이터 분석하기

실제로는 분석 사용자가 SYSADMIN이 아닌 다른 역할을 수행할 수도 있습니다. 뒤 세션에서 RBAC(Role Based Access Control)에 대해서 더 알아 보도록 하겠습니다. 그리고 일반적으로 Tableau, Looker, PowerBI 등과 같은 비즈니스 인텔리전스 제품을 통해서 많은 분석이 이루어 집니다. 고급 분석을 위해 Datarobot, Dataiku, AWS Sagemaker와 같은 데이터 과학 도구나 기타 광범위한 파트너 에코시스템의 솔루션들이 Snowflake를 네이티브하게 지원합니다.

JDBC/ODBC, Spark 또는 Python을 활용하는 모든 기술이 Snowflake의 데이터에 대한 분석을 실행할 수 있습니다.

- 커넥터 & 드라이버: <https://docs.snowflake.com/ko/user-guide/conns-drivers.html>

작업 중인 워크시트로 이동하여 마지막 섹션에서 생성한 새 웨어하우스를 사용하도록 웨어하우스를 변경합니다. 워크시트 컨텍스트는 다음과 같아야 합니다.

Role: SYSADMIN Warehouse: ANALYTICS_WH (L) Database: CITIBIKE Schema = PUBLIC

The screenshot shows two panels side-by-side. The left panel is titled 'Roles' and lists several roles: CITIBIKE.PUBLIC (selected), SYSADMIN, ANALYST_CITIBIKE, DBA_CITIBIKE, DEV_CITIBIKE, and PUBLIC. The right panel is titled 'Warehouses' and lists three warehouses: COMPUTE_WH, ANALYST_WH (selected), and PUBLIC. Both panels have a red box highlighting their respective sections.

아래의 쿼리를 실행하여 trips 데이터 샘플을 확인합니다.

```
/* 데이터 분석용 컨텍스트를 확인합니다. 웨어하우스는 분석용 웨어하우스를 사용하도록 하겠습니다. */
```

```
use role sysadmin;
use warehouse analytics_wh;
use database citibike;
use schema public;

select * from trips limit 20;
```

The screenshot shows the Snowflake SQL interface. In the top left, there's a sidebar with pinned objects and a search bar. The main area contains a code editor with the following SQL script:

```

40 --clear table and metadata on table trips for next test
41 truncate table trips;
42
43 --verify table is clear
44 select * from trips limit 10;
45
46 --change warehouse size from small to large (4x)
47 alter warehouse compute_wh set warehouse_size='large';
48
49 --verify large warehouse
50 show warehouses;
51
52 --load data in with large warehouse
53 copy into trips from @citibike_trips file_format=csv;
54
55 --preview trips data
56 select * from trips limit 20;
57
58
59
60

```

The code editor has tabs for Objects, Query, Results, and Chart. Below the code editor is a results table with 14 rows of trip data. To the right of the table are various metrics and charts related to the query.

	TRIPDURATION	STARTTIME	STOPTIME	START_STATION_ID	START_STATION_NAME	START_STATION_LATIT
1	258	2018-04-03 18:31:11.000	2018-04-03 18:35:30.000	3,664	North Moore St & Greenwich St	40.720195
2	659	2018-04-03 18:31:12.000	2018-04-03 18:42:12.000	127	Barrow St & Hudson St	40.731724
3	1,629	2018-04-03 18:31:13.000	2018-04-03 18:58:22.000	3,002	South End Ave & Liberty St	40.7111
4	286	2018-04-03 18:31:18.000	2018-04-03 18:36:05.000	465	Broadway & W 41 St	40.755130
5	1,097	2018-04-03 18:31:19.000	2018-04-03 18:49:36.000	305	E 58 St & 3 Ave	40.76095
6	348	2018-04-03 18:31:19.000	2018-04-03 18:37:06.000	526	E 33 St & 5 Ave	40.747651
7	313	2018-04-03 18:31:22.000	2018-04-03 18:36:35.000	519	Pershing Square North	40.7518
8	421	2018-04-03 18:31:24.000	2018-04-03 18:38:26.000	3,258	W 27 St & 10 Ave	40.750181
9	1,185	2018-04-03 18:31:25.000	2018-04-03 18:51:10.000	3,263	Cooper Square & Astor Pl	40.729514
10	272	2018-04-03 18:31:26.000	2018-04-03 18:35:59.000	3,140	1 Ave & E 78 St	40.771404
11	509	2018-04-03 18:31:27.000	2018-04-03 18:39:56.000	528	2 Ave & E 31 St	40.742901
12	454	2018-04-03 18:31:29.000	2018-04-03 18:39:04.000	248	Laight St & Hudson St	40.72185
13	2,266	2018-04-03 18:31:34.000	2018-04-03 19:09:20.000	334	W 20 St & 7 Ave	40.74238
14	358	2018-04-03 18:31:34.000	2018-04-03 18:37:32.000	418	Front St & Gold St	40.70

Snowflake는 기본적으로 표준 SQL 기반의 쿼리를 제공하므로 Snowflake SQL을 통해서 데이터 분석을 진행할 수 있습니다.

- Snowflake SQL 쿼리 구문: <https://docs.snowflake.com/ko/sql-reference/constructs.html>
- Snowflake SQL 명령 요약: <https://docs.snowflake.com/ko/sql-reference/intro-summary-sql.html>

시간대 별로 이용 현황, 평균 이동 시간 및 평균 이동 거리 등 기본적인 집계는 기존 SQL 구문의 group by를 활용해서 진행할 수 있으며, 다른 분석 쿼리들도 익숙한 SQL 구문에 따라서 다양하게 테스트해 볼 수 있습니다.

/* 자전거를 대여한 시간에서 시간대를 추출해서 각 시간대 별로 이용 현황을 집계 합니다. 데이터 분석용 컨텍스트를 확인합니다.

아래는 하버사인 삼각함수를 통해서 위도 경도 데이터로 대략적인 이동 거리를 계산했지만 그 외에 다양한 지리 정보 함수를 함께 제공합니다. */

```
Select date_trunc('hour', starttime) as "date",
count(*) as "num trips",
avg(tripduration)/60 as "avg duration (mins)",
```

```
avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude,  
end_station_longitude)) as "avg distance (km)"  
from trips  
group by 1 order by 1;
```

/* 자전거 트립이 많고 이동 거리가 길었던 시간대를 집계합니다. 여기서는 지리 공간 함수를 사용해서 평균 이동 거리를 계산했습니다. */

```
select date_trunc('hour', starttime) as "date",  
count(*) as "num trips",  
avg(tripduration)/60 as "avg duration (mins)",  
avg(st_distance(st_makepoint(start_station_latitude, start_station_longitude),  
st_makepoint(end_station_latitude, end_station_longitude)) / 1000) as "avg  
distance (km)"  
from trips  
group by 1  
having "num trips" > 100 and "avg distance (km)" > 1.5  
order by 1;
```

```
/* 2013년 이래로 가장 인기 있는 루트를 집계합니다. */  
select  
start_station_name,  
end_station_name,  
count(*) as "num trips"  
from trips  
where starttime > '2013'  
group by 1, 2 order by 3 desc;
```

- Snowflake집계 함수: <https://docs.snowflake.com/ko/sql-reference/functions-aggregation.html>

- 지리 공간 함수: <https://docs.snowflake.com/ko/sql-reference/functions-geospatial.html>

Result Cache

Snowflake에는 지난 24시간 동안 실행된 모든 쿼리의 결과를 보유하고 있는 결과 캐시가 있습니다. 이는 웨어하우스 전반에 걸쳐 사용할 수 있으므로 기본 데이터가 변경되지 않았다면, 한 사용자에게 반환된 쿼리 결과를 동일한 쿼리를 실행하는 해당 시스템의 다른 사용자가 사용할 수 있습니다. 이러한 반복 쿼리는 매우 빠르게 반환될 뿐만 아니라 컴퓨팅 크레딧도 전혀 사용하지 않습니다.

```
/* 처음 실행되는 쿼리는 시간과 동일한 쿼리를 결과 캐쉬에서 제공하는 시간을 비교해서 확인합니다. */
```

```
select date_trunc('hour', starttime) as "date",
count(*) as "num trips",
avg(tripduration)/60 as "avg duration (mins)",
avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude,
end_station_longitude)) as "avg distance (km)"
from trips
group by 1 order by 1;
```

결과가 캐시되었기 때문에 두 번째 쿼리가 훨씬 더 빠르게 실행되었음을 오른쪽 **Query Details** 창에서 확인합니다.

The screenshot shows the Snowflake UI with a query editor and a results table. The query is:

```

61 count(*) as "num trips",
62 avg(tripduration)/60 as "avg duration (mins)",
63 avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude, end_station_longitude)) as "avg distance (km)"
64
65 from trips
66 group by 1 order by 1;
67
68 --run same query again to see performance improvement due to cache
69 select date_trunc('hour', starttime) as "date",
70 count(*) as "num trips",
71 avg(tripduration)/60 as "avg duration (mins)",
72 avg(haversine(start_station_latitude, start_station_longitude, end_station_latitude, end_station_longitude)) as "avg distance (km)"
73 from trips
74 group by 1 order by 1;
75
76
77
78
79
80

```

The results table has columns: date, num trips, ..., avg duration (mins), avg distance (km). The data shows trip statistics for June 1, 2013, with 152 trips at 26.5251634 mins and 2.127971478 km.

On the right, there are 'Query Details' charts for date, num trips, avg duration (mins), and avg distance (km).

Zero-Copy Clone

Snowflake를 사용하면 "제로 카피 클론"이라고도 하는 테이블, 스키마 및 데이터베이스의 클론을 몇 초 안에 생성할 수 있습니다. 클론을 생성할 때 원본 객체에 있는 데이터의 스냅샷을 찍으며 복제된 객체에서 이를 사용할 수 있습니다. 복제된 객체는 쓰기 가능하고 클론 원본과는 독립적입니다. 따라서 원본 객체 또는 클론 객체 중 하나에 이뤄진 변경은 다른 객체에는 포함되지 않습니다.

제로 카피 클론 생성의 일반적인 사용 사례는 개발 및 테스팅이 사용하는 운영 환경을 복제하여 운영 환경에 부정적인 영향을 미치지 않게 두 개의 별도 환경을 설정하여 관리할 필요가 없도록 테스트하고 실험하는 것입니다.

워크시트에서 다음 명령을 실행하여 trips 테이블의 개발(dev) 테이블 복제본을 만듭니다.

```
/* 운영 환경의 trips 테이블을 클로닝해서 데이터의 복사 없이 새로운 개발 테이블을 생성합니다. */
```

```
create table trips_dev clone trips
```

CITIBIKE 데이터베이스 아래의 개체 트리를 확장하고 trips_dev라는 새 테이블이 표시되는지 확인합니다. 이제 개발 팀은 trips 테이블이나 다른 개체에 영향을 주지 않고 이 테이블을 사용하여 업데이트 또는 삭제를 포함하여 원하는 모든 작업을 수행할 수 있습니다.

The screenshot shows the Snowflake UI interface. On the left, there's a sidebar with pinned objects, a search bar, and a tree view of the database schema under the CITIBIKE database. The schema includes INFORMATION_SCHEMA, PUBLIC, and TRIPS. Under TRIPS, there are two tables: TRIPS and TRIPS_DEV. A context menu is open over the TRIPS_DEV table, with the 'clone' option highlighted in blue. To the right of the table list, there's a code editor window displaying SQL queries for creating a new table based on the TRIPS table. The SQL code is as follows:

```
67 -- run same query again to see
68 select date_trunc('hour', starttime) as hour
69 count(*) as "num trips",
70 avg(tripduration)/60 as "avg trip duration"
71 avg(haversine(start_station_longitude,
72 start_station_latitude, end_station_longitude,
73 end_station_latitude)) as "avg distance"
74 group by 1 order by 1;
75
76 --find busiest months
77 select monthname(starttime) as "month"
78 count(*) as "num trips"
79 from trips
80 group by 1 order by 2 desc;
81
82 --create dev table
```

The table details pane shows the following information for the TRIPS_DEV table:

Number of rows	61.5M
Size	1.9GB
Cluster Key	—
Owner	SYSADMIN
Created	1 minute ago

- Create Clone: <https://docs.snowflake.com/ko/sql-reference/sql/create-clone.html>

Semi-Structured Data and Working with Queries

날씨가 자전거 이용 횟수에 어떻게 영향을 미치는지 확인하고자 하기 위해서 JSON 형식으로 저장된 반정형 데이터를 로딩 합니다.

- 공개된 S3 버킷에 보관된 JSON 형식의 날씨 데이터 로드
- 뷰 생성 및 SQL 점 표기법 (dot notation)을 사용해 반정형 데이터를 쿼리
- JSON 데이터를 이전에 로드된 TRIPS 데이터에 조인하는 쿼리를 실행
- 날씨 및 자전거 이용 횟수 데이터를 분석하여 관계 파악
-

JSON 데이터는 57,900행, 61개 객체 및 2.5MB 압축으로 이루어 진 압축된 JSON 문서로 AWS S3에 저장되어 있습니다.

```
{"city":{"coord":{"lat":43.000351,"lon":-75.499901},"country":"US","findname":"NEW YORK","id":5128638,"name":"New York","zoom":1}, "clouds":{"all":90}, "main":{"humidity":93,"pressure":1008,"temp":293.47,"temp_max":295.37,"temp_min":292.04}, "time":1561467737, "weather":[{"description":"moderate rain","icon":"10d","id":501,"main":"Rain"}, {"wind":{"deg":170,"speed":4.1}}], "city":{"coord":{"lat":40.714272,"lon":-74.005966}, "country":"US", "findname":"NEW YORK", "id":5128581, "name": "New York", "zoom":1}, "clouds":{"all":90}, "main":{"humidity":94,"pressure":1010,"temp":295.16,"temp_max":296.15,"temp_min":294.15}, "time":1561467737, "weather":[{"description":"light rain","icon":"10d","id":500,"main":"Rain"}, {"description":"mist","icon":"50d","id":701,"main":"Mist"}, {"wind":{"deg":0,"speed":2.1}}], "city":{"coord":{"lat":43.000351,"lon":-75.499901}, "country":"US", "findname":"NEW YORK", "id":5128638, "name": "New York", "zoom":1}, "clouds":{"all":90}, "main":{"humidity":94,"pressure":1008,"temp":294.58,"temp_max":297.04,"temp_min":292.04}, "time":1561471336, "weather":[{"description":"overcast clouds","icon":"04d","id":804,"main":"Clouds"}, {"wind":{"deg":270,"speed":3.1}}], "city":{"coord":{"lat":40.714272,"lon":-74.005966}, "country":"US", "findname":"NEW YORK", "id":5128581, "name": "New York", "zoom":1}, "clouds":{"all":90}, "main":{"humidity":100,"pressure":1010,"temp":295.37,"temp_max":296.48,"temp_min":294.26}, "time":1561471336, "weather":[{"description":"mist","icon":"50d","id":701,"main":"Mist"}, {"wind":{"deg":170.797,"speed":0.4}}]}
```

반정형 데이터 처리하기

Snowflake는 변환 없이 JSON, Parquet, Avro, ORC, XML과 같은 반정형 데이터를 쉽게 로드하고 쿼리할 수 있습니다.

- 반정형 데이터 소개: <https://docs.snowflake.com/ko/user-guide/semistructured-intro.html>
- 반정형 데이터에 지원하는 형식: <https://docs.snowflake.com/ko/user-guide/semistructured-data-formats.html>
- 반정형 데이터 타입: <https://docs.snowflake.com/ko/sql-reference/data-types-semistructured.html>

데이터베이스 및 테이블 생성

먼저, 워크시트를 통해, 반정형 데이터를 저장하는 데 사용할 WEATHER라는 이름의 데이터베이스를 생성합니다.

```
/* 새 데이터베이스 생성하기 */  
create or replace database weather;
```

워크시트 내에 컨텍스트를 적절하게 설정합니다.

```
/* 테이블을 생성하기 위한 컨텍스트를 설정합니다. 새로 생성한 weather 데이터베이스의 public 스키마에서 작업을 합니다. */
```

```
use role sysadmin;  
use warehouse compute_wh;
```

```
use database weather;  
use schema public;
```

JSON 데이터를 로딩하는 데 사용할 JSON_WEATHER_DATA라는 이름의 테이블을 생성합니다. Snowflake에는 VARIANT라는 반정형 데이터 타입을 네이티브하게 지원하는 컬럼 유형이 있어 이를 통해 전체 JSON 객체를 저장하고 바로 쿼리할 수 있습니다.

```
/* JSON 데이터를 로딩하려고 하기 때문에 반정형 데이터 탑입을 네이티브하게 지원하는 variant 탑입의 컬럼을 가진 테이블을 생성합니다. */
```

```
create or replace table json_weather_data (v variant);
```

- Variant 탑입: <https://docs.snowflake.com/ko/sql-reference/data-types-semistructured.html#variant>

VARIANT 데이터 유형을 사용하면 Snowflake가 스키마를 미리 정의하지 않고도 반정형 데이터를 수집할 수 있습니다.

워크시트 하단의 결과 창에서 JSON_WEATHER_DATA 테이블이 생성되었는지 확인합니다.

The screenshot shows the Snowflake interface with the following details:

- Left Sidebar:** Shows the schema tree under the database "CITIBIKE_ZERO_TO_SNOWFLAKE". It includes nodes for CITIBIKE, WEATHER, INFORMATION_SCHEMA, PUBLIC, and various tables like JSON_WEATHER_DATA.
- Central Editor Area:** Displays the SQL code for creating the table:

```
--create database for unstructured weather data
create database weather;
--set context for module 7
use role sysadmin;
use warehouse compute_wh;
use database weather;
use schema public;
--create table for weather data using variant to handle semi structured data
create table json_weather_data (v variant);
```
- Bottom Navigation:** Buttons for Objects, Query, Results, and Chart.
- Results Panel:** Shows the status of the query execution:

status
1 Table JSON_WEATHER_DATA successfully created.
- Right Panel:** Shows Query Details:

Query Details
Query duration 127ms
Rows 1
status 100% filled

External Stage 생성

워크시트에서 다음 명령을 사용하여 AWS S3에서 반정형 JSON 데이터가 저장되는 버킷을 가리키는 외부 스테이지를 설정합니다.

```
/* public s3의 버킷을 가리키는 외부 스테이지를 생성합니다. */
```

```
create or replace stage nyc_weather
url = 's3://snowflake-workshop-lab/weather-nyc';
```

LIST 명령을 통해서 파일 목록을 살펴보도록 하겠습니다.

압축된 JSON 파일들이 저장되어 있는 것을 확인할 수 있습니다.

```
list @nyc_weather;
/* 이 결과에서 바로 전체 사이즈와 평균 파일 사이즈를 알아 보도록 하겠습니다. */

select
    floor(sum($2) / power(1024, 2), 1) total_compressed_storage_mb,
    floor(avg($2) / power(1024, 1), 1) avg_file_size_kb,
    count(*) as num_files
from
    table(result_scan(last_query_id()));
```

테이블로 로딩하기 전에 실제로 데이터 파일로 저장되어 있는 데이터의 내용도 바로 살펴 보도록 하겠습니다.

```
/* 외부 스테이지에 있는 데이터 파일에 대해서 바로 쿼리를 실행해서 내용을 살펴  
볼 수 있습니다. JSON 타입으로 해석하는 File Format을 먼저 생성한 다음에 select  
문에서 설정을 해 줍니다. */  
create or replace file format my_json_format type = 'json';  
  
select $1  
From @nyc_weather/ (file_format => my_json_format)  
Limit 10;  
  
/* 메타 데이터도 함께 결과를 확인해 보겠습니다. Parse_json은 문자열을 Variant 형  
식으로 변환하는 유용한 함수입니다. */  
select  
    metadata$filename, metadata$file_row_number, parse_json($1)  
from  
    @nyc_weather/ (file_format => my_json_format);
```

이 섹션에서는 웨어하우스를 사용하여 S3 버킷의 데이터를 이전에 생성한 **JSON_WEATHER_DATA** 테이블로 로드합니다.
워크시트에서 아래 COPY 명령을 실행하여 데이터를 로드합니다.
SQL 명령에서도 **FILE FORMAT** 객체를 인라인으로 지정할 수 있는 방법을 사용할 수도 있습니다.

정형 데이터를 로드했던 이전 섹션에서 파일 형식을 자세하게 정의해야 했습니다.
여기에는 JSON 데이터는 형식이 잘 지정되어 있기 때문에 기본 설정을 사용해 간단하게 JSON 유형을 로딩하고 결과를 살펴 보겠습니다.

```
/* 먼저 Variant 타입의 컬럼에 JSON 데이터를 로딩하겠습니다. */
```

```

copy into json_weather_data
from @nyc_weather
file_format = (type=json);

select * from json_weather_data limit 20;

```

결과에서 한 행을 클릭하여 오른쪽 패널에 형식 JSON을 표시해서 어떤 형태의 데이터인지 살펴 보겠습니다.

The screenshot shows the Snowflake UI interface. On the left, the sidebar displays the database structure under 'CITIBIKE_ZERO_TO_SNOWFLAKE'. A query is running in the center, showing the creation of a stage, loading JSON files into it, and then selecting the first 10 rows from the resulting table. The results pane shows 10 rows of JSON data, which is also displayed in a detailed JSON viewer on the right side of the screen.

V	JSON Data
1	{ "city": { "coord": { "lat": 40.714272, "lon": -74.005966 }, "country": "US", "findname": "NEW YORK", "id": 5128 }
2	{ "city": { "coord": { "lat": 43.000351, "lon": -75.499901 }, "country": "US", "findname": "NEW YORK", "id": 5128 }
3	{ "city": { "coord": { "lat": 40.714272, "lon": -74.005966 }, "country": "US", "findname": "NEW YORK", "id": 5128 }
4	{ "city": { "coord": { "lat": 43.000351, "lon": -75.499901 }, "country": "US", "findname": "NEW YORK", "id": 5128 }
5	{ "city": { "coord": { "lat": 40.714272, "lon": -74.005966 }, "country": "US", "findname": "NEW YORK", "id": 5128 }
6	{ "city": { "coord": { "lat": 43.000351, "lon": -75.499901 }, "country": "US", "findname": "NEW YORK", "id": 5128 }
7	{ "city": { "coord": { "lat": 40.714272, "lon": -74.005966 }, "country": "US", "findname": "NEW YORK", "id": 5128 }
8	{ "city": { "coord": { "lat": 43.000351, "lon": -75.499901 }, "country": "US", "findname": "NEW YORK", "id": 5128 }
9	{ "city": { "coord": { "lat": 40.714272, "lon": -74.005966 }, "country": "US", "findname": "NEW YORK", "id": 5128 }
10	{ "city": { "coord": { "lat": 43.000351, "lon": -75.499901 }, "country": "US", "findname": "NEW YORK", "id": 5128 }

JSON Data (Row 1):

```

{
  "city": {
    "coord": {
      "lat": 40.714272,
      "lon": -74.005966
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

JSON Data (Row 2):

```

{
  "city": {
    "coord": {
      "lat": 43.000351,
      "lon": -75.499901
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

JSON Data (Row 3):

```

{
  "city": {
    "coord": {
      "lat": 40.714272,
      "lon": -74.005966
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

JSON Data (Row 4):

```

{
  "city": {
    "coord": {
      "lat": 43.000351,
      "lon": -75.499901
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

JSON Data (Row 5):

```

{
  "city": {
    "coord": {
      "lat": 40.714272,
      "lon": -74.005966
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

JSON Data (Row 6):

```

{
  "city": {
    "coord": {
      "lat": 43.000351,
      "lon": -75.499901
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

JSON Data (Row 7):

```

{
  "city": {
    "coord": {
      "lat": 40.714272,
      "lon": -74.005966
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

JSON Data (Row 8):

```

{
  "city": {
    "coord": {
      "lat": 43.000351,
      "lon": -75.499901
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

JSON Data (Row 9):

```

{
  "city": {
    "coord": {
      "lat": 40.714272,
      "lon": -74.005966
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

JSON Data (Row 10):

```

{
  "city": {
    "coord": {
      "lat": 43.000351,
      "lon": -75.499901
    },
    "country": "US",
    "findname": "NEW YORK",
    "id": 5128
  }
}

```

View 및 Materialized View

원본 테이블은 JSON 형식의 컬럼이기 때문에 뷰를 사용하면 쿼리 결과에 테이블처럼 액세스할 수 있습니다. Snowflake는 여러 가지 뷰의 형태를 제공하는 데 그 중에 Materialized View는 쿼리의 결과를 저장하기 때문에 저장 용량을 필요로 하지만 성능을 향상 시킬 수 있습니다. Snowflake 엔터프라이즈 이상에서 Materialized View를 사용할 수 있습니다.

- View의 개요: <https://docs.snowflake.com/ko/user-guide/views-introduction.html>
- Materialized View: <https://docs.snowflake.com/ko/user-guide/views-materialized.html>

먼저 반정형 JSON 날씨 데이터에 대한 뷰를 테이블 형식으로 구성해서 분석가들이 익숙한 형태로 쿼리를 할 수 있도록 하겠습니다.

city_id 5128638은 뉴욕시에 해당합니다.

```
/* 원본 테이블이 Variant 타입의 JSON 구조체이므로 분석가들에게 익숙한 테이블 형태의 뷰를 생성해서 분석을 용이하게 합니다.  
JSON 구조체에서 계층에 따라서 .(닷)으로 원하는 컬럼을 구성하고 ::으로 형변환을 해 주는 형식으로 구성합니다. */
```

```
create view json_weather_data_view as  
select  
v:time::timestamp as observation_time,  
v:city.id::int as city_id,  
v:city.name::string as city_name,  
v:city.country::string as country,
```

```
v:city.coord.lat::float as city_lat,  
v:city.coord.lon::float as city_lon,  
v:clouds.all::int as clouds,  
(v:main.temp::float)-273.15 as temp_avg,  
(v:main.temp_min::float)-273.15 as temp_min,  
(v:main.temp_max::float)-273.15 as temp_max,  
v:weather[0].main::string as weather,  
v:weather[0].description::string as weather_desc,  
v:weather[0].icon::string as weather_icon,  
v:wind.deg::float as wind_dir,  
v:wind.speed::float as wind_speed  
from json_weather_data  
where city_id = 5128638;
```

예를 들어서, v:city.coord.lat은 이 명령에서 JSON 계층 구조 내 더 낮은 수준의 값을 가져오는 데 사용됩니다. 이는 각 필드를 관계형 테이블의 열인 것처럼 취급할 수 있도록 합니다.

새로운 뷰가 UI 왼쪽 WEATHER > PUBLIC > Views에 json_weather_data로 나타나도록 브라우저를 새로 고침을 합니다.

```

119  create view json_weather_data_view as
120  select
121    v:time:timestamp as observation_time,
122    v:city_id:int as city_id,
123    v:city_name:string as city_name,
124    v:city_country:string as country,
125    v:city_coord_lat:float as city_lat,
126    v:city_coord_lon:float as city_lon,
127    v:clouds_all:int as clouds,
128    (v:main.temp::float)-273.15 as temp_avg,
129    (v:main.temp_min::float)-273.15 as temp_min,
130    (v:main.temp_max::float)-273.15 as temp_max,
131    v:weather[0].main:string as weather_main,
132    v:weather[0].description:string as weather_desc,
133    v:weather[0].icon:string as weather_icon,
134    v:wind.deg::float as wind_dir,
135    v:wind.speed::float as wind_speed
136  from json_weather_data
137  where city_id = 5128658;

```

JSON_WEATHER_DATA_VIEW

Type: View
Number of columns: 15
Owner: SYSADMIN
Created: just now

successfully created.

Query Details

- Query duration: 256ms
- Rows: 1
- Status: 100% filled

/* 뷰의 내용을 다른 테이블 형식의 데이터처럼 확인을 합니다. */

```

select * from json_weather_data_view
where date_trunc('month',observation_time) = '2018-01-01'
limit 20;

```

The screenshot shows a Snowflake query editor interface. On the left, there's a sidebar with navigation links like 'CITIBIKE_ZERO_TO_SHOWFLAKE', 'Pinned', 'No pinned objects', 'Search', 'CITIBIKE', 'WEATHER', 'INFORMATION_SCHEMA', 'PUBLIC', 'Tables', 'Views', 'Stages', 'Data Pipelines', 'Functions', and 'Procedures'. The main area has tabs for 'Objects', 'Query', 'Results', and 'Chart'. The 'Query' tab contains the following SQL code:

```

126 v:city.coord.lon::float as city_lon,
127 v:clouds.all::int as clouds,
128 v:main.temp::float -273.15 as temp_avg,
129 v:main.temp_min::float -273.15 as temp_min,
130 v:main.temp_max::float -273.15 as temp_max,
131 v:weather[0].main::string as weather,
132 v:weather[0].description::string as weather_desc,
133 v:weather[0].icon::string as weather_icon,
134 v:wind.deg::float as wind_dir,
135 v:wind.speed::float as wind_speed
136 from json_weather_data
137 where city_id = 5128638;
138
139 --validate the view created
140 select * from json_weather_data_view
141 where date_trunc('month',observation_time) = '2018-01-01'
142 limit 20;
143
144

```

The 'Results' tab displays a table with 15 rows of data. The columns are: OBSERVATION_TIME, CITY_ID, CITY_NAME, COUNTRY, CITY_LAT, CITY_LON, CLOUDS, TEMP_AVG, TEMP. The data shows various weather observations for New York City (CITY_ID 5128638) across different months in 2018.

	OBSERVATION_TIME	CITY_ID	CITY_NAME	COUNTRY	CITY_LAT	CITY_LON	CLOUDS	TEMP_AVG	TEMP.
1	2018-01-01 03:05:19.000	5,128,638	New York	US	43.000351	-75.499901	1	-20.85	
2	2018-01-01 04:02:30.000	5,128,638	New York	US	43.000351	-75.499901	1	-21.38	
3	2018-01-08 18:02:41.000	5,128,638	New York	US	43.000351	-75.499901	90	0.1	
4	2018-01-08 19:05:05.000	5,128,638	New York	US	43.000351	-75.499901	90	1.05	
5	2018-01-12 20:03:23.000	5,128,638	New York	US	43.000351	-75.499901	90	14.57	
6	2018-01-12 21:05:14.000	5,128,638	New York	US	43.000351	-75.499901	90	8.85	
7	2018-01-16 01:02:38.000	5,128,638	New York	US	43.000351	-75.499901	90	-8.52	
8	2018-01-16 02:05:08.000	5,128,638	New York	US	43.000351	-75.499901	90	-8	
9	2018-01-25 14:05:13.000	5,128,638	New York	US	43.000351	-75.499901	1	-11.95	
10	2018-01-25 15:02:40.000	5,128,638	New York	US	43.000351	-75.499901	1	-10.95	
11	2018-01-04 15:02:41.000	5,128,638	New York	US	43.000351	-75.499901	90	-9.48	
12	2018-01-04 16:05:25.000	5,128,638	New York	US	43.000351	-75.499901	90	-8.95	
13	2018-01-03 03:04:00.000	5,128,638	New York	US	43.000351	-75.499901	90	-8.53	
14	2018-01-03 04:03:27.000	5,128,638	New York	US	43.000351	-75.499901	90	-8	
15	2018-01-04 23:05:18.000	5,128,638	New York	US	43.000351	-75.499901	90	-7.47	

On the right side of the interface, there are sections for 'Query Details' (Query duration: 1.1s, Rows: 20), 'Observation Time' (histogram from 2018-01-01 to 2018-01-25), and detailed statistics for CITY_ID (123, 100% filled), CITY_NAME (New York, 20), and COUNTRY (US, 20).

JOIN 연산

이제 JSON 날씨 데이터를 CITIBIKE.PUBLIC.TRIPS 데이터에 조인하여 원래 질문인 날씨가 자전거 이용 횟수에 어떻게 영향을 미치는지에 대해 살펴 보겠습니다.

아직 워크사이트에 있기 때문에 WEATHER 데이터베이스가 기본값입니다. 데이터베이스와 스키마 이름을 제공하여 TRIPS 테이블에 대해 완전히 참조할 수 있도록 합니다.

```
/* trips 테이블에 weather view를 left outer join으로 시간대 별로 조인해서 날씨 조건 별로 전체 트립의 개수를 집계하면 각 날씨에 따른 이용 횟수의 관계를 알아 볼 수 있습니다. */
```

```
select weather as conditions
, count(*) as num_trips
from citibike.public.trips
left outer join json_weather_data_view
on date_trunc('hour', observation_time) = date_trunc('hour', starttime)
where conditions is not null
group by 1 order by 2 desc;
```

```
/* 이 연산의 결과를 Secure view로 생성해서 안전하게 공유하는 데 활용할 수도 있습니다. Secure view는 어떤 SQL 구문을 통해서 이 결과가 나왔는지에 대한 DDL 정보를 권한이 있는 사용자(예, 소유자)만 열람할 수 있습니다. */
```

```
create or replace secure view weather_trips_sv as
select weather as conditions
, count(*) as num_trips
from citibike.public.trips
left outer join json_weather_data_view
on date_trunc('hour', observation_time) = date_trunc('hour', starttime)
where conditions is not null
group by 1 order by 2 desc;

select * from weather_trips_sv;
```

- Secure View 관련 작업하기: <https://docs.snowflake.com/ko/user-guide/views-secure.html>

The screenshot shows a Snowflake query editor interface. On the left, there's a sidebar with a tree view of database objects. The main area has a code editor with a highlighted query and a results table.

```

138 --validate the view created
139 select * from json_weather_data_view
140 where date_trunc('month',observation_time) = '2018-01-01'
141
142 limit 20;
143
144 --join weather data and trips data
145 left outer join citibike.public.trips
146 ,count(*) as num_trips
147
148 left outer join json_weather_data_view
149 on date_trunc('hour', observation_time) = date_trunc('hour', starttime)
150 where conditions is not null
151 group by 1 order by 2 desc;
152
153
154
155
156
157
158
159
160
161
162
163
164

```

Results:

CONDITIONS	NUM_TRIPS
1 Clear	9,249,531
2 Clouds	8,934,964
3 Rain	3,206,720
4 Snow	1,380,418
5 Mist	798,487
6 Fog	362,496
7 Thunderstorm	230,539
8 Drizzle	98,779
9 Haze	40,724
10 Smoke	2,871

Query Details:

- Query duration: 1.3s
- Rows: 10

Conditions:

- 100% filled

NUM_TRIPS:

5. Data Protection, RBAC and Managing Account

Time Travel

타임 트래블 기능으로 사전 구성 가능한 기간 내 어느 시점이든 데이터에 액세스할 수 있습니다. 기본 기간은 24시간이며 Snowflake 엔터프라이즈 에디션으로는 90일 까지 가능합니다.

몇 가지 유용한 적용례는 다음과 같습니다.

- 삭제되었을 수도 있는 테이블, 스키마 및 데이터베이스 같은 데이터 관련 객체를 복구
- 과거의 주요 시점으로부터 데이터를 복제하고 백업
- 데이터 사용을 분석하고 특정 기간에 대해 히스토리 분석

Continuous Data Protection Lifecycle



먼저 실수로 또는 의도적으로 삭제한 데이터 객체를 어떻게 복구할 수 있는지 살펴보겠습니다.

워크시트에서 다음의 DROP 명령을 실행하여 `JSON_WEATHER_DATA` 테이블을 제거합니다.

```
drop table json_weather_data;
```

`json_weather_data` 테이블에서 `SELECT` 문을 실행합니다. 기본 테이블이 삭제되었기 때문에 결과 창에 오류가 나타나야 합니다.

```
select * from json_weather_data limit 10;
```

The screenshot shows the Snowflake UI interface. On the left, there's a sidebar with navigation links like CITIBIKE, WEATHER, INFORMATION_SCHEMA, PUBLIC, and others. The main area has a query editor with the following code:

```
140 select outer join json_weather_data_view
141   on date_trunc('hour', observation_time) = date_trunc('hour', starttime)
142   where conditions is not null
143   group by 1 order by 2 desc;
144
145 ///////////////////////////////////////////////////////////////////
146 //MODULE 8//
147 ///////////////////////////////////////////////////////////////////
148 --drop table json_weather_data;
149
150 --verify table is dropped
151 select * from json_weather_data limit 10;
```

The last line of the query, `select * from json_weather_data limit 10;`, is highlighted in blue. Below the query editor, there are tabs for Objects, Query, and Results. The Results tab is selected, showing a warning icon and the error message: "Object 'JSON_WEATHER_DATA' does not exist or not authorized." To the right, there's a "Query Details" panel showing a duration of 32ms and 0 rows.

이제 이 테이블을 다음과 같이 복구합니다.

```
undrop table json_weather_data;
```

json_weather_data 테이블이 복구된 것을 확인합니다.

```
select * from json_weather_data_view limit 10;
```

The screenshot shows the Snowflake UI interface. The sidebar and query editor are identical to the previous screenshot. The Results tab is selected, displaying the following table of data:

	OBSERVATION_TIME	CITY_ID	CITY_NAME	COUNTRY	CITY_LAT	CITY_LON	CLOUDS	TEMP_AVG	TEMP
1	2016-10-23 07:55:33.000	5,128,638	New York	US	43.000351	-75.499901	88	4.723	4
2	2016-10-20 06:51:43.000	5,128,638	New York	US	43.000351	-75.499901	20	8.623	8
3	2016-10-22 06:41:04.000	5,128,638	New York	US	43.000351	-75.499901	92	7.309	7
4	2016-09-13 11:25:35.000	5,128,638	New York	US	43.000351	-75.499901	1	12.12	
5	2016-09-08 23:03:53.000	5,128,638	New York	US	43.000351	-75.499901	40	25.3	2
6	2016-09-05 18:25:39.000	5,128,638	New York	US	43.000351	-75.499901	1	27.82	2
7	2016-09-15 03:04:10.000	5,128,638	New York	US	43.000351	-75.499901	1	13.22	1
8	2016-11-26 21:07:52.000	5,128,638	New York	US	43.000351	-75.499901	90	3.33	
9	2016-08-06 23:31:04.000	5,128,638	New York	US	43.000351	-75.499901	44	25.29	2
10	2016-08-10 01:24:18.000	5,128,638	New York	US	43.000351	-75.499901	0	24.449	24

To the right of the results, there are several analysis cards: "OBSERVATION_TIME" (histogram from 2016-08-06 to 2016-11-26), "CITY_ID" (count of 123), "CITY_NAME" (New York, 100% filled), and "COUNTRY" (US, 100% filled). A "Query Details" panel on the far right shows a duration of 290ms and 10 rows.

Table Rollback

테이블을 이전 상태로 룰백하여 CITIBIKE 데이터베이스의 TRIPS 테이블에 있는 모든 스테이션 이름을 "mistake"라는 단어로 대체하는 의도하지 않은 DML 오류를 수정하겠습니다.

먼저 워크시트의 컨텍스트가 적절한지 다음과 같이 확인합니다.

```
use role sysadmin;  
use warehouse compute_wh;  
  
use database citibike;  
use schema public;
```

다음의 명령을 실행하여 테이블의 모든 스테이션 이름을 "mistake"라는 단어로 대체 합니다.

```
update trips set start_station_name = 'mistake';
```

이제 자전거 이용 횟수별로 상위 20개 스테이션을 반환하는 쿼리를 실행합니다. 스테이션 이름 결과는 단 하나의 행으로 나오는 것을 확인할 수 있습니다.

```
select  
start_station_name as "station",  
count(*) as "rides"  
from trips  
group by 1  
order by 2 desc  
limit 20;
```

```

157   select * from json_weather_data_view limit 10;
158
159   --set context
160   use role sysadmin;
161   use warehouse compute_wh;
162   use database citibike;
163   use schema public;
164
165   --make a mistake by changing the column name
166   update trips set start_station_name='oops';
167
168   --QUERY for top 20 stations by total number of rides
169   select
170     start_station_name as "station",
171     count(*) as "rides"
172   from trips
173   group by 1
174   order by 2 desc
175   limit 20;

```

station	rides
oops	61,468,359

Query Details

- Query duration: 315ms
- Rows: 1
- station: 100% filled
- rides: 100% filled

백업을 하지 않은 상황이라도 Snowflake에서는 아래 처럼 마지막UPDATE 명령의 쿼리 ID를 찾아 \$QUERY_ID라는 변수에 저장하면 됩니다.

```

set query_id =
(select query_id from table(information_schema.query_history_by_session
(result_limit=>5))
where query_text like 'update%' order by start_time limit 1);

```

올바른 스테이션 이름을 가진 테이블을 다음과 같이 다시 만듭니다.

```

create or replace table trips as
(select * from trips before (statement => $query_id));

```

다음과 같이 SELECT 문을 다시 실행하여 스테이션 이름이 복구되었는지 확인합니다.

```

select
start_station_name as "station",
count(*) as "rides"
from trips
group by 1
order by 2 desc
limit 20;

```

```

187 --find the query id that was the last update to the table
188 set query_id =
189 (select query_id from table(information_schema.query_history_by_session (result_limit>=5))
190 where query_text like 'update%' order by start_time limit 1);
191
192 --recreate the table with proper station names
193 create or replace table trips as
194 (select * from trips before (statement => $query_id));
195
196 --select from the restored table to verify
197 select
198   start_station_name as "station",
199   count(*) as "rides"
200   from trips
201   group by 1
202   order by 2 desc
203   limit 20;
204

```

station	rides
Pershing Square North	491,951
E 17 St & Broadway	481,065
W 21 St & 6 Ave	458,626
8 Ave & W 31 St	438,001
West St & Chambers St	432,518
Broadway & E 22 St	421,812
Lafayette St & E 8 St	397,724
Broadway & E 14 St	394,995
8 Ave & W 33 St	379,843
W 41 St & 6 Ave	359,838
Cleveland Pl & Spring St	358,485
W 20 St & 11 Ave	352,099
Carmine St & 6 Ave	348,158
University Pl & E 14 St	332,803
Greenwich Ave & 8 Ave	329,347

- 타임 트래블 이해 및 사용하기: <https://docs.snowflake.com/ko/user-guide/data-time-travel.html>
- Fail-Safe 이해 및 보기: <https://docs.snowflake.com/ko/user-guide/data-failsafe.html>

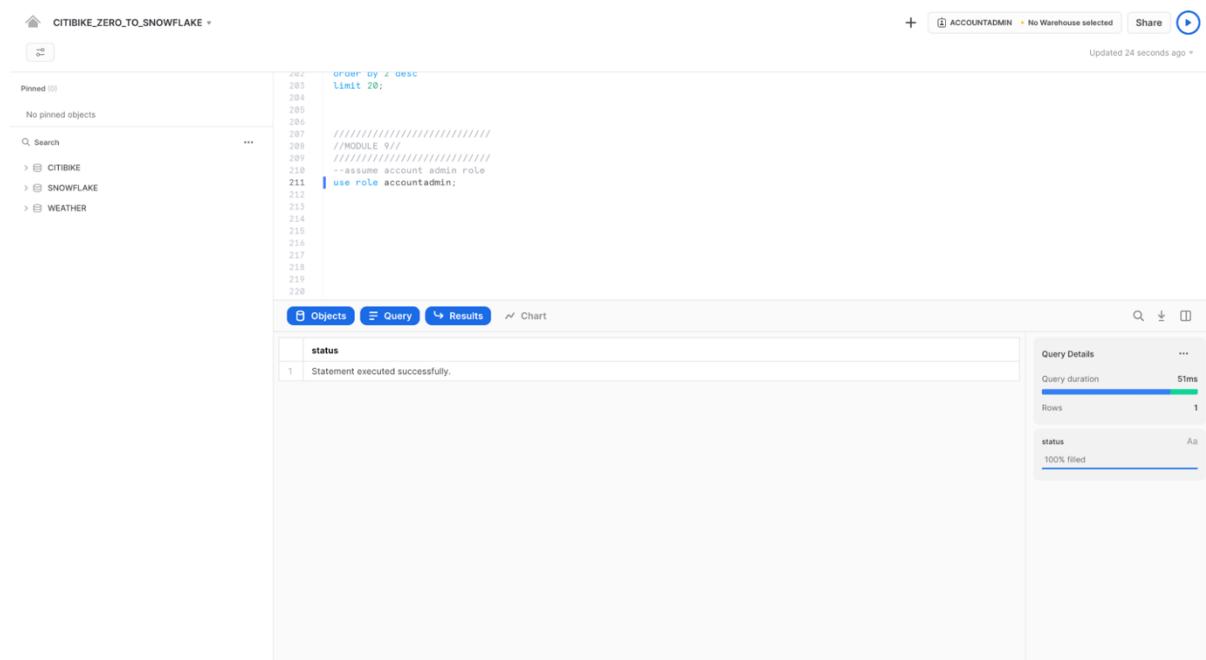
Role Based Access Control

이 섹션에서는 새로운 역할 생성 및 특정 권한 부여와 같은 Snowflake의 역할 기반 액세스 제어(RBAC) 측면을 살펴보고자 합니다. 또한 ACCOUNTADMIN(계정 관리자) 역할도 다뤄볼 것입니다.

주니어 DBA가 Citi Bike에 합류하여 시스템에서 정의한 기본 역할인 SYSADMIN보다 적은 권한을 지닌 새로운 역할을 만들고 싶다고 가정해 보겠습니다.

워크시트에서 ACCOUNTADMIN 역할을 새로운 역할로 전환합니다. ACCOUNTADMIN 은 SYSADMIN 및 SECURITYADMIN 시스템 정의 역할을 함께 가지고 있는 시스템의 최상위 역할이므로 계정에서 제한된 수의 사용자에게만 부여되어야 합니다. 워크시트에서 다음을 실행합니다.

```
use role accountadmin;
```



The screenshot shows the Snowflake UI interface. On the left, there's a sidebar with pinned objects and a search bar. The main area has tabs for Objects, Query, Results, and Chart. A query is being run:

```
202 order by z desc
203 limit 20;
204
205
206
207 /////////////////
208 //MODULE 9//
209 /////////////////
210 --assume account admin role
211 use role accountadmin;
212
213
214
215
216
217
218
219
220
```

The Results tab shows the output:

status
1 Statement executed successfully.

On the right, there are sections for Query Details (Query duration: 51ms, Rows: 1) and a status bar indicating 100% filled.

역할을 액세스 제어에 사용하려면 최소한 한 명의 사용자가 역할에 할당되어야 합니다. 이제 JUNIOR_DBA라는 새 역할을 만들고 Snowflake 사용자에게 할당해 보겠습니다. 이 작업을 완료하려면 UI에 로그인할 때 사용한 이름인 사용자 이름을 알아야 합니다.

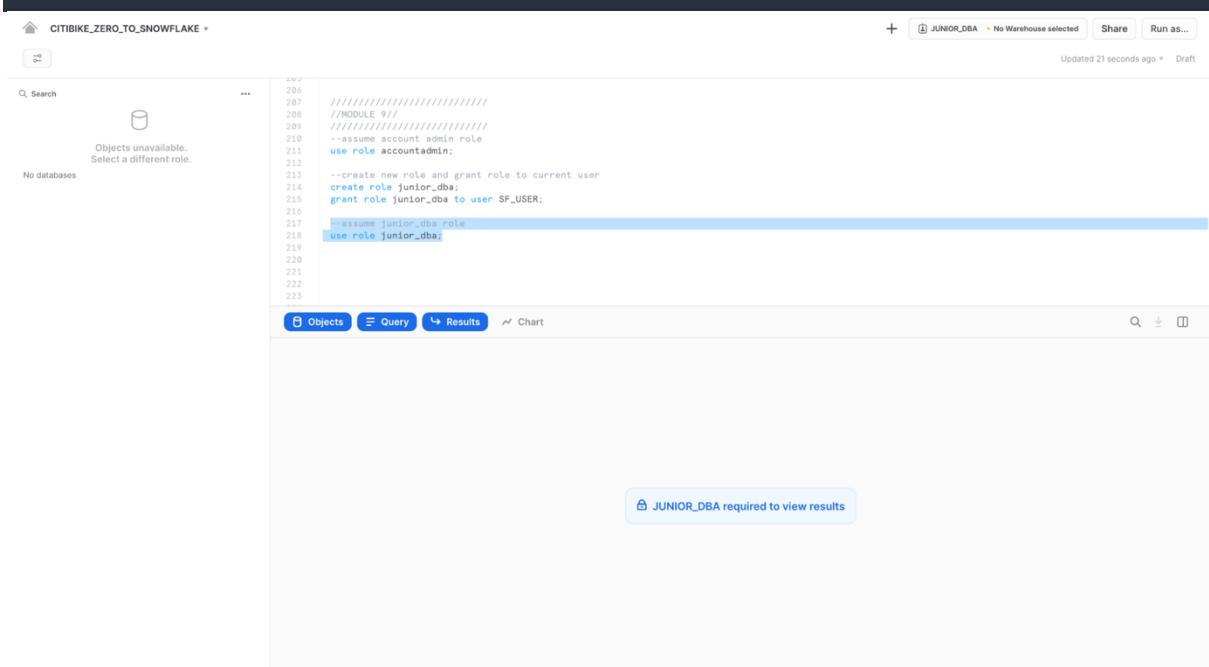
다음 명령을 사용하여 역할을 생성하고 할당합니다. GRANT ROLE 명령을 실행하기 전에 <user>를 사용자 이름으로 바꿉니다.

```
create role junior_dba;
grant role junior_dba to user <user>;
```

SYSADMIN과 같은 역할로 이 작업을 수행하려고 하면, 권한이 부족하여 실패할 것입니다. 기본적으로 SYSADMIN 역할은 새로운 역할이나 사용자를 생성할 수 없습니다.

워크시트 컨텍스트를 다음과 같이 JUNIOR_DBA 역할로 변경합니다.

```
use role junior_dba;
```



```
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
```

```
--assume account admin role
use role accountadmin;
--create new role and grant role to current user
create role junior_dba;
grant role junior_dba to user SF_USER;
--assume junior_dba role
use role junior_dba;
```

JUNIOR_DBA required to view results

또한 새로 생성된 역할에는 웨어하우스에 대한 사용 권한이 없기 때문에 웨어하우스가 선택되지 않습니다. ADMIN 역할로 다시 전환하여 수정하고 COMPUTE_WH 웨어하우스에 사용 권한을 부여해 보겠습니다.

```
use role accountadmin;
```

```
grant usage on warehouse compute_wh to role junior_dba;
```

JUNIOR_DBA 역할로 다시 전환합니다. 이제 COMPUTE_WH를 사용할 수 있습니다.

```
use role junior_dba;  
  
use warehouse compute_wh;
```

마지막으로 왼쪽의 데이터베이스 개체 브라우저 패널에서 CITIBIKE 및 WEATHER 데 이터베이스가 더 이상 나타나지 않는 것을 확인할 수 있습니다. 이는 JUNIOR_DBA 역할에 액세스 권한이 없기 때문입니다.

ACCOUNTADMIN 역할로 다시 전환하고 CITIBIKE 및 WEATHER 데이터베이스를 보고 사용하는 데 필요한 USAGE 권한을 JUNIOR_DBA에 부여합니다.

```
use role accountadmin;  
  
grant usage on database citibike to role junior_dba;  
grant usage on database weather to role junior_dba;
```

JUNIOR_DBA 역할로 다음과 같이 전환합니다.

```
use role junior_dba;
```

이제 CITIBIKE 및 WEATHER 데이터베이스가 나타나는지 확인하십시오. 나타나지 않는다면 새로 고침 아이콘을 클릭하여 시도하십시오.

The screenshot shows the Snowflake SQL interface. In the top right, there are buttons for '+', JUNIOR_DBA, 'No Warehouse selected', 'Share', and 'Run as...'. Below that, it says 'Updated 34 seconds ago'. The left sidebar has a search bar and shows 'CITIBIKE_ZERO_TO_SHOWFLAKE' with a database icon. It also lists 'CITIBIKE' and 'WEATHER'. The main area has tabs for 'Objects', 'Query', 'Results', and 'Chart'. The 'Query' tab is active, displaying the following code:

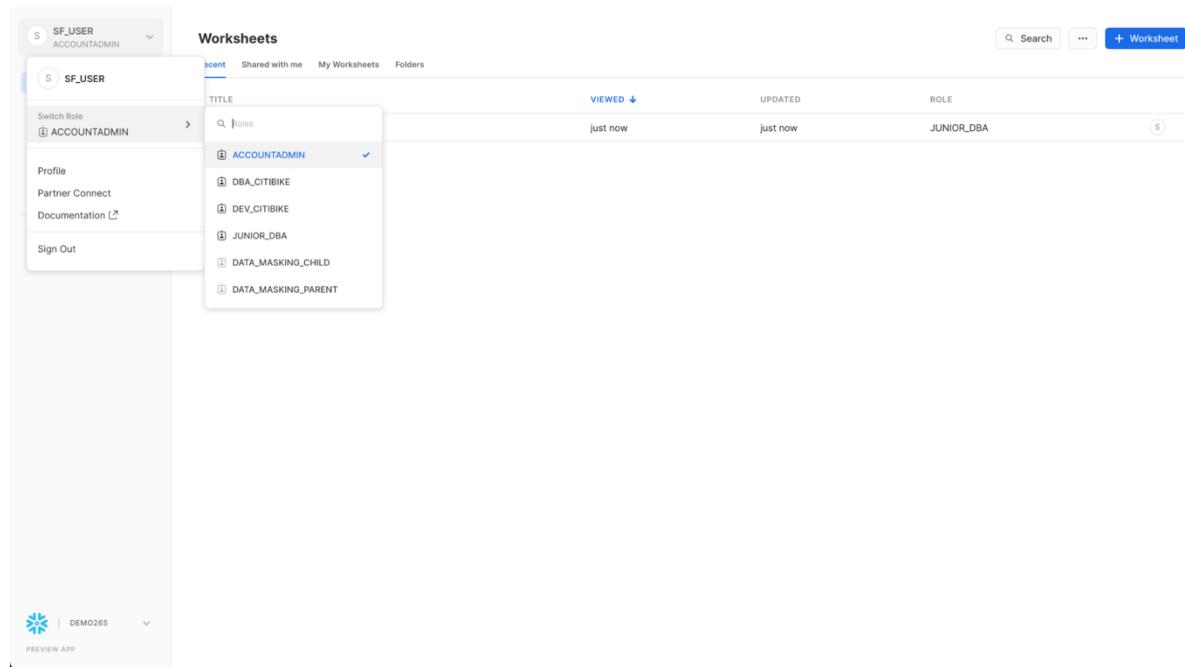
```
218 use role junior_dba;
219
220 --change role to give permissions to junior_dba role
221 use role accountadmin;
222 grant usage on database citibike to junior_dba;
223 grant usage on database weather to junior_dba;
224
225 --switch back to junior_dba
226 use role junior_dba;
227
228
229
230
231
232
233
234
235
236
```

The 'Results' tab shows a message: 'JUNIOR_DBA required to view results'.

ACCOUNTADMIN

이 역할만 액세스할 수 있는 UI의 다른 영역을 보려면 액세스 제어 역할을 다시 ACCOUNTADMIN으로 변경합니다. 그러나 이 작업을 수행하려면 워크시트 대신 UI를 사용하십시오.

먼저 워크시트의 왼쪽 상단 모서리에 있는 **Home** 아이콘을 클릭합니다. 그런 다음 UI의 왼쪽 상단에서 이름을 클릭하여 사용자 기본 설정 메뉴를 표시합니다. 메뉴에서 **Switch Role**으로 이동하여 ACCOUNTADMIN을 선택합니다.



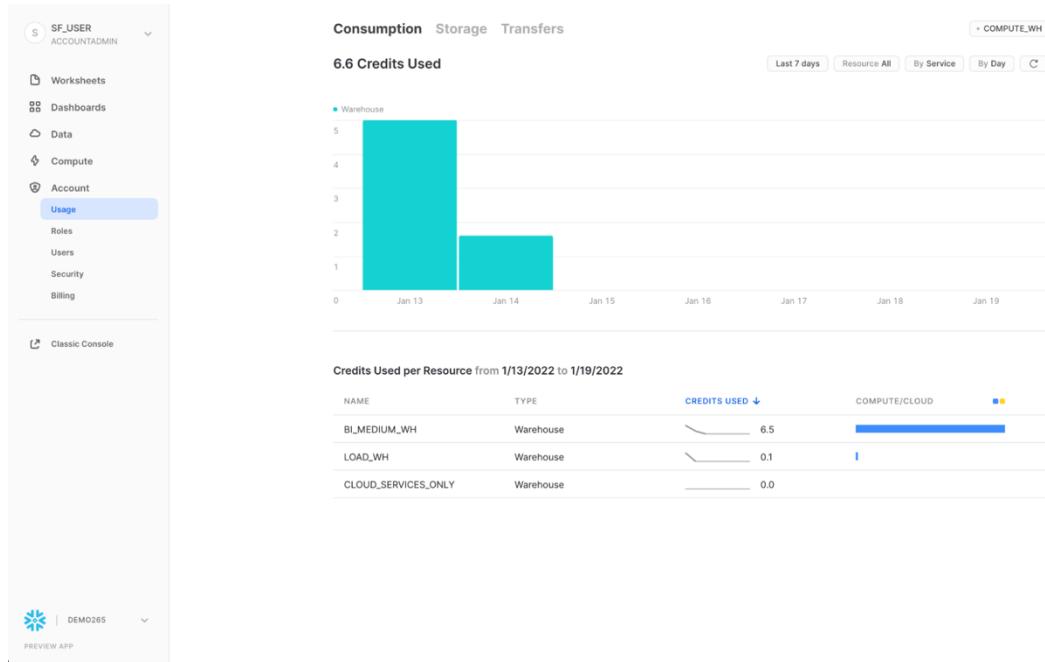
The screenshot shows the Snowflake Worksheets interface. On the left, there's a sidebar with user information (SF_USER, ACCOUNTADMIN), a 'Switch Role' button (ACCOUNTADMIN), and links for Profile, Partner Connect, Documentation, and Sign Out. The main area is titled 'Worksheets' and shows a table of worksheets. The table has columns for TITLE, VIEWED, UPDATED, and ROLE. One row is selected, showing 'ACCOUNTADMIN' as the title, 'just now' for both viewed and updated, and 'JUNIOR_DBA' for the role. A dropdown menu is open over this row, listing several roles: ACCOUNTADMIN (selected), DBA_CITIBIKE, DEV_CITIBIKE, JUNIOR_DBA, DATA_MASKING_CHILD, and DATA_MASKING_PARENT. At the bottom of the interface, it says 'PREVIEW APP' and 'DEMO265'.

User Preference 설정의 역할 vs 워크시트

UI 세션과 각 워크시트에는 고유한 역할이 있습니다. UI 세션 역할은 UI에서 보고 액세스할 수 있는 요소를 제어하는 반면 워크시트 역할은 역할 내에서 액세스할 수 있는 개체 및 작업만 제어합니다.

UI 세션을 ACCOUNTADMIN 역할로 전환하면 **Account**에서 새 탭을 사용할 수 있습니다.

Usage



Usage 탭에는 각각 고유한 페이지가 있는 다음이 표시됩니다.

- **Organization:** 조직의 모든 계정에서 사용된 크레딧입니다.
 - **Consumption:** 현재 계정의 가상 웨어하우스에서 소비한 크레딧입니다.
 - **Storage:** 지난 달 현재 계정의 모든 데이터베이스, 내부 단계 및 Snowflake Failsafe에 저장된 평균 데이터 양입니다.
 - **Transfers:** 지난 한 달 동안 해당 지역(현재 계정의 경우)에서 다른 지역으로 전송된 평균 데이터 양입니다.

각 페이지의 오른쪽 상단 모서리에 있는 필터를 사용하여 사용량/소비량 등을 분류할 수 있습니다.

Security

The screenshot shows the Snowflake interface with the 'Security' tab selected. On the left, a sidebar lists various navigation options: Worksheets, Dashboards, Data, Compute, Account (with sub-options Usage, Roles, Users), Security (which is highlighted in blue), and Billing. Below the sidebar is a 'Classic Console' link. The main content area is titled 'Network Policies' and 'Sessions'. At the top right is a blue button labeled '+ Network Policy'. In the center, there is a small icon of a lock inside a shield. Below the icon, the text 'No network policies' is displayed, followed by a descriptive message: 'Restrict or allow access to your Snowflake account based on IP address.' with a 'Learn More' link.

Security 탭에는 Snowflake 계정에 대해 생성된 네트워크 정책이 포함되어 있습니다. 페이지 오른쪽 상단의 "+ 네트워크 정책"을 선택하여 새 네트워크 정책을 만들 수 있습니다.

Billing

The screenshot shows the Snowflake interface with the 'Billing' tab selected. The left sidebar includes the same navigation options as the Security tab, with 'Billing' also highlighted in blue. The main content area is titled 'Billing' and 'Payment Method'. A blue button at the top right says '+ Credit Card'. Below it, there is a message: 'Add a credit card to continue using Snowflake once your free trial ends.' At the bottom of the main content area, there is a 'PREVIEW APP' link.

Billing 탭에는 계정에 대한 결제 수단이 포함되어 있습니다.

- Snowflake 계약 고객인 경우 탭에 계약 정보와 연결된 이름이 표시됩니다.
- 주문형 Snowflake 고객인 경우 탭에 월별 결제에 사용된 신용 카드가 표시됩니다(입력된 경우). 등록된 신용 카드가 없는 경우 평가판이 종료될 때 Snowflake를 계속 사용하려면 신용 카드를 추가할 수 있습니다.

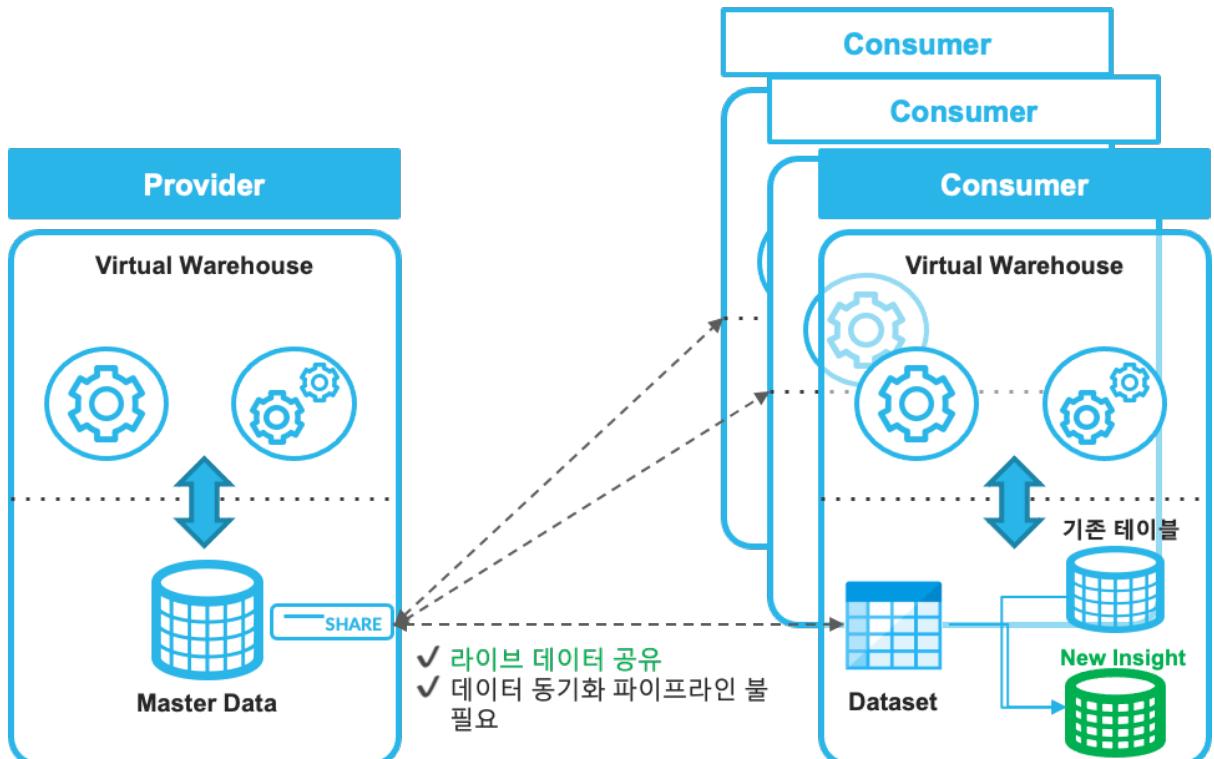
다음 섹션에서도 공유에 대한 중요한 권한 때문에 ACCOUNTADMIN 역할을 유지합니다.

6. Secure Data Sharing and Snowflake Marketplace

Snowflake는 공유를 통해 계정 간 데이터 액세스를 가능하게 합니다. 공유는 데이터 공급자가 생성하고 데이터 소비자가 자신의 Snowflake 계정 또는 프로비저닝된 Snowflake 읽기 전용 계정을 통해 가져옵니다.

다음을 통해 안전하게 데이터를 공유합니다.

- 데이터 사본은 하나뿐이며 데이터 제공자의 계정에 있음
- 공유 데이터는 항상 활성화되어 있고, 실시간이며 소비자가 즉시 사용할 수 있음
- 공급자가 공유에 대한 취소 가능하고 세분화된 액세스 권한을 설정할 수 있음
- 데이터 공유는 특히 인터넷을 통한 대용량 .csv 파일 전송을 포함하는 수동의 안전하지 않은 이전 데이터 공유 방법과 비교하여 간단하고 안전함



- Secure Data Sharing 소개: <https://docs.snowflake.com/ko/user-guide/data-sharing-intro.html>

기존 공유 보기

홈페이지에서 **Data > Databases**로 이동합니다. 데이터베이스 목록에 서 **SOURCE** 열을 확인합니다. 열에 Local이 있는 두 개의 데이터베이스가 표시되어야 합니다. 이들은 이전에 실습에서 만든 두 개의 데이터베이스입니다. 다른 데이터베이스인 SNOWFLAKE는 열에 Share를 표시하여 공급자로부터 공유되었음을 나타냅니다.

The screenshot shows the Snowflake Data Catalog interface. On the left, there is a sidebar with navigation links: SF_USER, ACCOUNTADMIN, Worksheets, Dashboards, Data (with Databases selected), Shared Data, Marketplace, Compute, and Account. Below the sidebar, it says DEMO265 and PREVIEW APP. The main content area is titled 'Databases' and shows a list of 3 Databases:

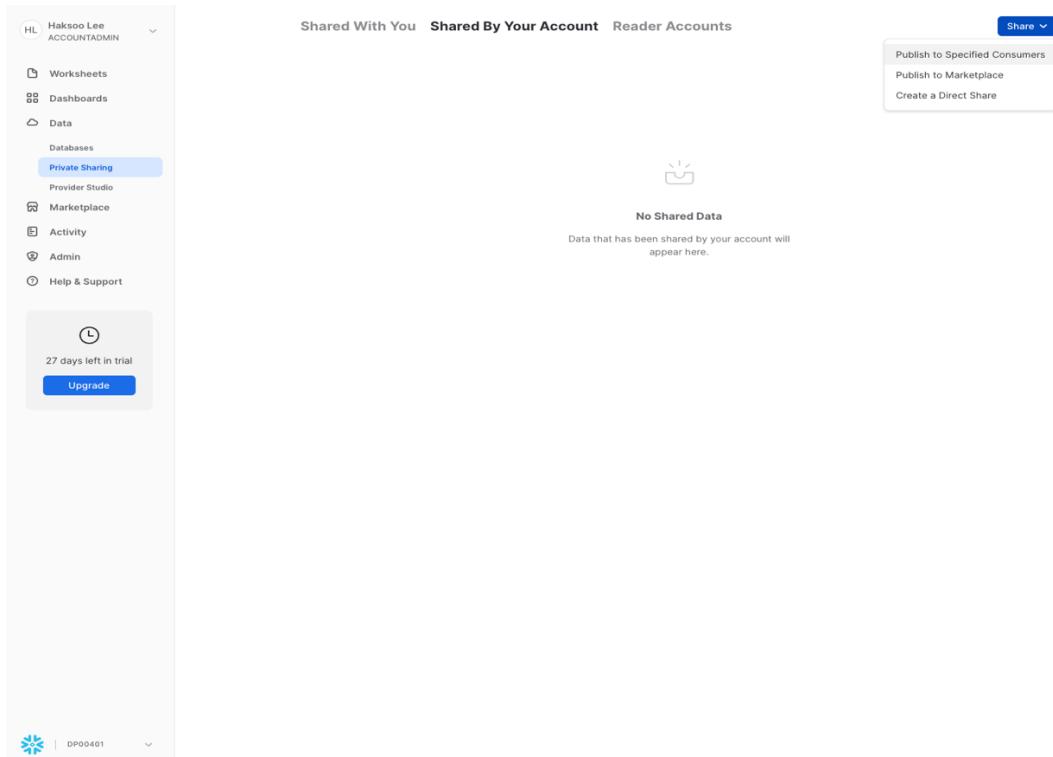
NAME ↑	SOURCE	OWNER	CREATED	...
CITIBIKE	Local	SYSADMIN	13 hours ago	...
SNOWFLAKE	Share	—	9 months ago	...
WEATHER	Local	SYSADMIN	53 minutes ago	...

아웃바운드 공유 생성

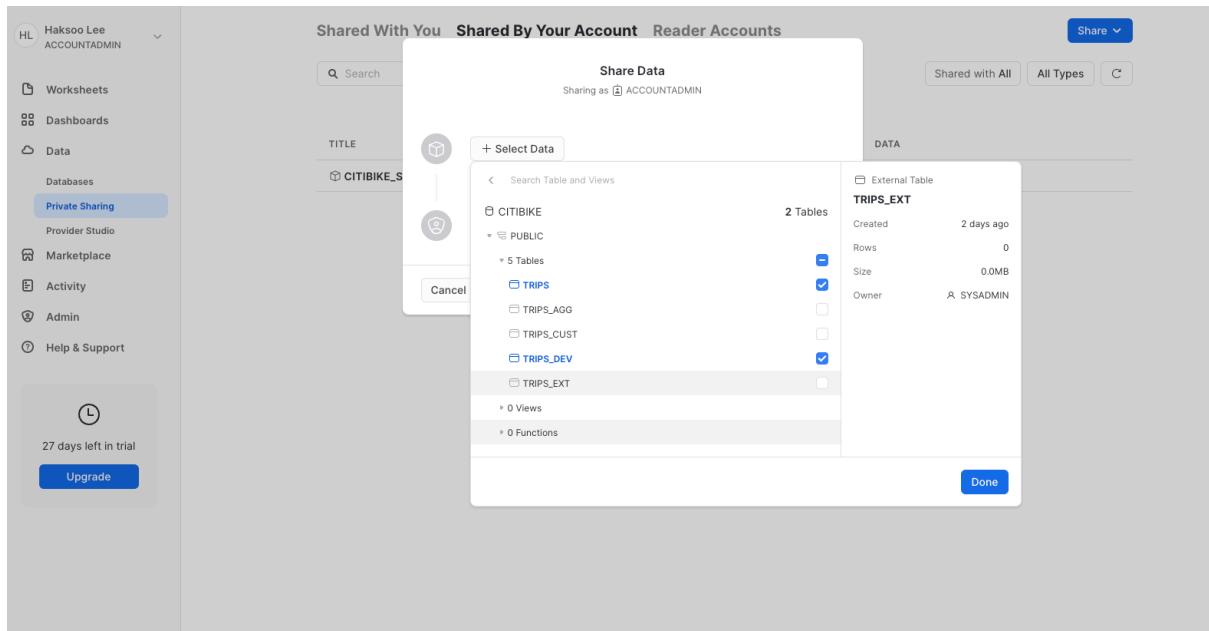
Snowflake 계정 관리자라고 가정하겠습니다. TRIPS 데이터베이스의 데이터를 거의 실시간으로 분석하기 원하는 신뢰할 수 있는 파트너가 있습니다. 이 파트너는 또한 우리 지역(Region)에 속한 고유한 Snowflake 계정도 갖고 있습니다.

따라서 Snowflake 데이터 공유를 이용하여 그들이 이 정보에 액세스할 수 있도록 해 보겠습니다.

Data > Private Sharing으로 이동한 다음 탭 상단의 **Shared by Your Account**를 클릭합니다. 오른쪽 상단의 **Share** 버튼을 클릭하고 **Create a Direct Share**를 선택합니다.

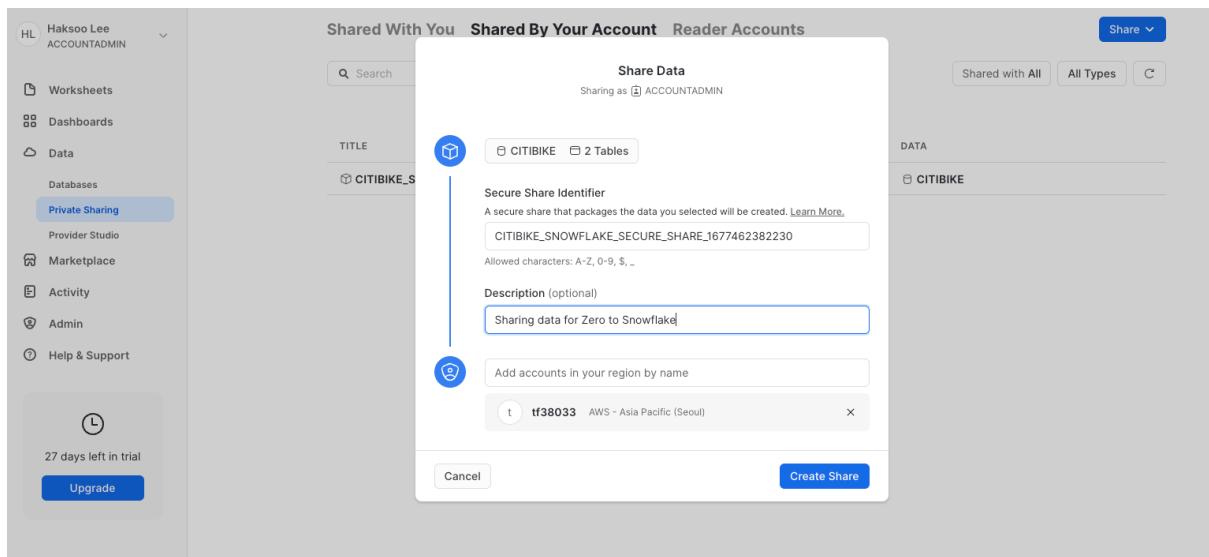


+ **Data**를 클릭하고 CITIBIKE 데이터베이스 및 PUBLIC 스키마로 이동합니다. 스키마에서 생성한 2개의 테이블을 선택하고 **Done** 버튼을 클릭합니다.



공유의 기본 이름은 임의의 숫자 값이 추가된 일반 이름입니다. 나중에 공유를 식별하는 데 도움이 되도록 기본 이름을 수정합니다.

대화 상자 하단의 **Create Share** 버튼을 클릭합니다.



대화 상자가 닫히고 페이지에 생성한 Secure Share가 표시됩니다.

The screenshot shows the 'Shared With' section of a Snowflake secure share. The share is titled 'CITIBIKE_SNOWFLAKE_SECURE_SHARE_1677462382230'. It was created 22 minutes ago by 'ACCOUNTADMIN' and is located in 'AWS - Asia Pacific (Seoul)'. The 'Shared With' section lists one consumer: 'SFSEAPAC.KR_DEMO16' (Account). There is a 'Add Consumers' button. Below it, the 'Description' section contains a placeholder 'A short summary of what this shared data includes and how it can be used.' with an 'Add' button. The final section, 'Data', shows two tables: 'TRIPS' and 'TRIPS_DEV', both of which are tables and belong to the 'PUBLIC' schema. There is an 'Edit' button for this section.

언제든지 소비자를 추가하고 설명을 추가/변경하고 공유의 개체를 편집할 수 있습니다. 페이지에서 공유 이름 옆에 있는 < 버튼을 클릭하여 **Share with Other Accounts** 페이지로 돌아갑니다.

The screenshot shows the 'Shared With You' page. It lists the secure share 'CITIBIKE_SNOWFLAKE_SECURE_SHARE_1677462382230' shared with 'SFSEAPAC.KR_DEMO16' at 23 minutes ago. The page includes filters for 'Shared with All', 'All Types', and a search bar. The main area displays columns for TITLE, SHARED WITH, CREATED, and DATA.

Snowflake는 기밀성을 손상시키지 않고 데이터를 안전하게 공유하는 여러 방법을 제공합니다. 테이블 외에도 Secure View, Secure UDF(사용자 정의 함수) 및 기타 보안 개체를 공유할 수 있습니다.

데이터 수신

The screenshot shows the Snowflake interface for managing data sharing. On the left, the navigation bar includes 'Worksheets', 'Dashboards', 'Data', 'Databases', 'Private Sharing' (which is selected), 'Provider Studio', 'Marketplace', 'Activity', 'Admin', 'Help & Support', and 'Classic Console'. The main area displays a 'Shared With You' card for 'ZBGJAGW.DP00401'. This card shows the database name 'ZBGJAGW_DP00401_CITIBIKE_SNOWFLAKE_SECURE_SHARE_1677462382230' and the roles 'ANALYST_CITIBIKE, HOL_ROLE, PUBLIC'. A 'Get Data' button is at the bottom right. To the right of this card is a sidebar for 'Knoema COVID-19 Data Atlas', which contains a brief description and a link to Marketplace. Below these cards is a section titled 'Direct Shares' containing three items: 'ZBGJAGW.DP00401' (Shared just now), 'SFSALESSHARED.SFC_SAMPLES_AWSNORTHEAST2' (Shared 1 year ago), and 'SNOWFLAKE ACCOUNT_USAGE' (Shared 1 year ago).

Snowflake Marketplace

ACCOUNTADMIN 역할을 사용 중인지 확인하고 Data 아래에서 Marketplace 탭으로 이동합니다.

The screenshot shows the Snowflake Marketplace interface. The left sidebar has a dropdown for 'SF_USER' (ACCOUNTADMIN) and links for 'Worksheets', 'Dashboards', 'Data', 'Shared Data', 'Marketplace' (which is selected and highlighted with a red border), 'Compute', 'Account', and 'Classic Console'. The main content area features a search bar and filters for 'Categories', 'Business Needs', 'Providers', and 'My Requests'. A prominent section titled 'New Marketplace Capabilities' shows four free data products: 'Total Consumer Insights' by Infor, 'IP to Geolocation' by IPInfo, 'SafeGraph Core Places - US...' by SafeGraph, and 'Weather Data for the United States - West' by Weather Source, LLC. Below this are sections for 'Featured Providers' (Funnel, Dun & Bradstreet, Equifax, ADP, Inc.) and 'Most Recent' products (SEC REPORTING ANALYTICS, CARTO, Monte Carlo, Rystad Energy). Each product card includes a brief description and a 'More' link.

Listing 찾기

상단의 검색 상자를 사용하여 목록을 검색할 수 있습니다. 검색 상자 오른쪽에 있는 드롭다운 목록을 사용하면 공급자, 비즈니스 요구 사항 및 범주별로 데이터 목록을 필터링할 수 있습니다.

검색창에 COVID를 입력하고 결과를 스크롤한 후 **COVID-19 Epidemiological Data**(Starschema 제공)를 선택합니다.

The screenshot shows a search results page with a search bar at the top containing the text "covid". Below the search bar is a navigation bar with categories: Browse Providers, Business Needs, Financial, Weather, Commerce, Health and Life Sciences, Demographics, and More Categories. A red arrow points to the search bar. To the right of the search bar is a "My Requests" button. Below the navigation bar, it says "44 results for COVID". On the right side of the results, there is a "Filters" button. The results are displayed in a grid format:

- Demand Data**
UK Covid Cases
COVID Cases and population in the UK by Local Authority including Shape Files
- State of California**
California COVID-19 Datasets
COVID-19 confirmed counts and county information
- Knoema**
COVID-19 Data Atlas
30+ public COVID-19 pandemic-related datasets from the WHO, JHU, ECDC, CDC, and other authoritative...
- Accern**
AI Powered COVID-19 Insights & Analytics
Discover trends and insights where COVID-19 intersects with company issues

A "Personalized" button is located below the Accern card.

- Traject**
COVID-19 Dataset
A collection of SERP data related to COVID-19 search queries across a variety of US states and metro areas
- ThinkData Works**
Covid-19 Canadian Outbreak
A daily refreshed feed of Covid-19 case counts by Province and Regional Health Boundaries
- DemystData**
Zip Code level COVID-19
US COVID-19 cases and death counts by county and ZIP Code
- Starschema**
COVID-19 Epidemiological Data
Analytics-ready data on COVID-19: cases, vaccinations, healthcare resource availability and...

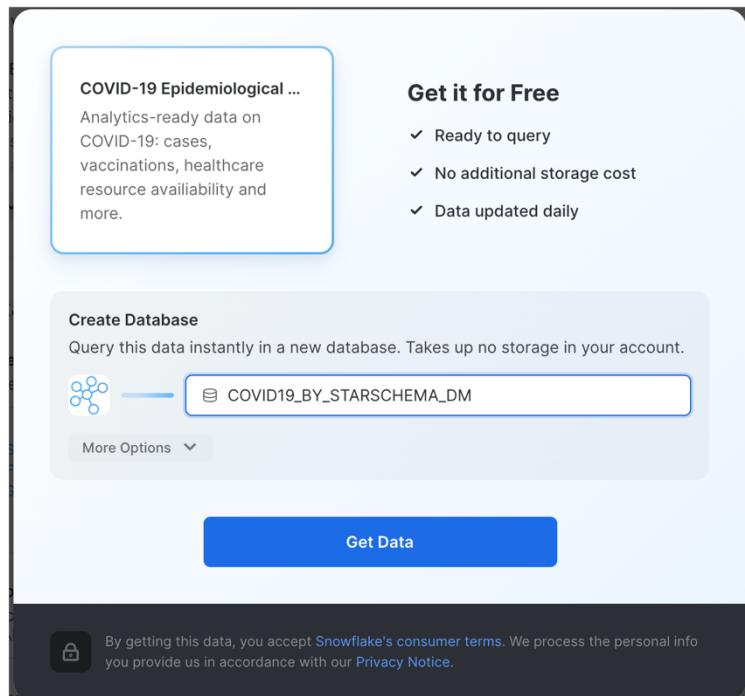
A red box highlights the Starschema card. Below these cards are two more rows of cards:

- Wunderman Thompson Data**
AmeriLINK Insights - Premium
Attitudinal, Buying Behavior, Demographic, Health Related, Lifestyle and Transactional Data
- Wunderman Thompson Data**
AmeriLINK Insights
Marketing information on U.S. consumers.
- Lifesight**
Mobility Data - Custom
High fidelity SDK-based mobility data collected from location-aware apps
- Prevedere**
COVID19 - US Economy Leading Indicators
Leading signals for the economic recovery following this economic downturn

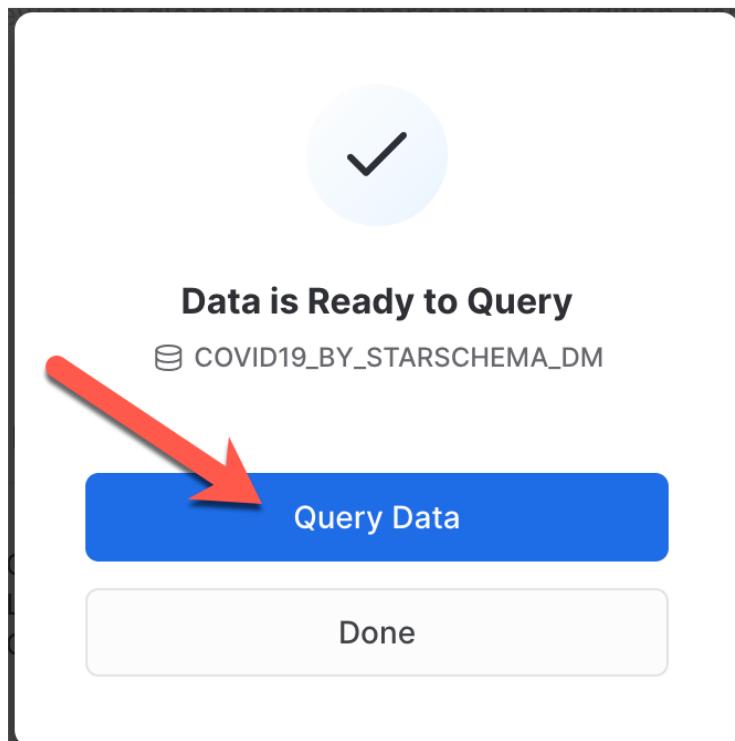
COVID-19 Epidemiological Data 페이지에서 데이터세트에 대해 자세히 알아보고 사용 예시 쿼리를 볼 수 있습니다. 준비가 되면 **Get Data** 버튼을 클릭하여 Snowflake 계정에서 이 정보를 사용할 수 있도록 합니다.

The screenshot shows the 'COVID-19 Epidemiological Data' page from the Snowflake Data Marketplace. At the top, there's a navigation bar with a back arrow and the text 'Snowflake Data Marketplace'. Below it is a header with a blue icon, the title 'COVID-19 Epidemiological Data', the provider 'Starschema', and categories 'Health and Life Sciences' and 'Daily'. A large text block describes the dataset, mentioning its origin in Hubei Province, PRC, in late 2019, and its global coverage since March 2020. It includes information on incidence, mortality, public health measures, and vaccination data. Below this is a section titled 'EXAMPLE USE CASES' with a detailed description of how the data can be used for contingency planning and disaster response. A 'Show More' button is present. To the right, there's a sidebar with a 'Free' plan offer for 'Unlimited queries' and a prominent 'Get Data' button. Further down, there's a 'Starschema' section with a bio about their mission to change the world through data and links to 'Contact', 'Documentation', and 'Terms of Service'.

대화 상자의 정보를 검토하고 **Get Data**를 다시 클릭합니다.



이제 **Done**를 클릭하거나 Starschema에서 제공하는 샘플 쿼리를 실행하도록 선택할 수 있습니다.



Query Data를 선택한 경우 새 브라우저 탭/창에서 새 워크시트가 열립니다.

1. 실행할 쿼리를 선택하거나 쿼리 텍스트에 커서를 놓습니다.
2. **Play/Run** 버튼을 클릭합니다(또는 키보드 단축키 사용).
3. 하단 창에서 데이터 결과를 볼 수 있습니다.
4. 샘플 쿼리 실행이 완료되면 왼쪽 상단의 **Home** 아이콘을 클릭합니다.

The screenshot shows the Snowflake interface with several annotations:

- Step 1:** A red arrow points to the number 1, which is overlaid on the first line of a query in the "Query" tab. The query is:


```

1 // Get total case count by country
2 // Calculates the total number of cases by country, aggregated over time.
3 SELECT COUNTRY_REGION, SUM(CASES) AS Cases
4 FROM ECDC_GLOBAL
5 GROUP BY COUNTRY_REGION
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
      
```
- Step 2:** A red arrow points to the number 2, which is overlaid on the "Share" button in the top right corner.
- Step 3:** A red arrow points to the number 3, which is overlaid on the "Results" tab.
- Step 4:** A red arrow points to the number 4, which is overlaid on the "Home" icon in the top left corner.

The interface includes a sidebar with pinned objects and a list of databases. The results tab displays a table of COVID-19 case data:

COUNTRY_REGION	CASES
Afghanistan	49,273
Albania	48,530
Algeria	92,102
Andorra	7,338
Angola	16,188
Anguilla	10
Antigua and Barbuda	148
Argentina	1,498,160
Armenia	148,682
Aruba	5,049

On the right, there are "Query Details" panels showing duration (1.6s), rows (214), and histograms for COUNTRY_REGION and CASES.

다음:

1. **Data > Databases**를 클릭합니다.
2. 'COVID19_BY_STARSCHHEMA_DM' 데이터베이스를 클릭합니다.
3. 쿼리에 사용할 수 있는 스키마, 테이블 및 뷰에 대한 세부 정보를 볼 수 있습니다.

The screenshot shows the Snowflake Data Marketplace interface. On the left, the user's profile is shown: Jason ACCOUNTADMIN. The navigation bar includes Worksheets, Dashboards, Data (selected), Databases (highlighted with a red circle containing '1'), Compute, Account, Organization, and Classic Console. In the main area, a database named COVID19_BY_STARSCHHEMA_DM is selected, indicated by a red circle containing '2'. The database details page shows the following information:

- Database Details**: Shows the database was created 15 hours ago by ACCOUNTADMIN.
- Schemas**: Shows the INFORMATION_SCHEMA and PUBLIC schemas.
- Source details**: Shows the provider is Starschema, and the title is COVID-19 Epidemiological Data.
- Privileges**: Shows the current role is ACCOUNTADMIN.

이제 글로벌 COVID 데이터로 매일 업데이트되는 Starschema의 COVID-19 데이터 세트를 성공적으로 구독했습니다. 데이터베이스, 테이블, 뷰 또는 ETL 프로세스를 생성할 필요 없이 라이브 업데이트되는 데이터 세트를 분석에 활용할 수 있습니다.