

# SNOWFLAKE용 PYTHON에 대한 데이터 엔지니어의 가이드



챔피언  
가이드

전자책

# TABLE OF CONTENTS

- 3** 소개
- 4** 데이터 엔지니어링용 Snowpark
- 6** Python용 Snowpark
  - Snowpark 클라이언트 API
  - Snowpark 서버 측 런타임
  - Snowflake, Anaconda 및 오픈 소스 생태계
- 11** 모범 사례: Python을 사용하여 Snowpark에서 데이터 엔지니어링
- 12** Snowpark 그 이상: Snowflake 데이터 엔지니어링 생태계의 다른 기능
- 14** Snowpark 시작하기 및 리소스
- 15** Snowflake 소개

## 소개

Python은 지속적으로 가장 널리 사용되는 3대 프로그래밍 언어로 손꼽힙니다. Stack Overflow의 연례 개발자 설문 조사에 따르면 전체 개발자 중 68%가 Python에서 작업하는 것을 ‘매우 선호’한다고 답했습니다.<sup>1</sup>

그러나 수년간 데이터 엔지니어는 데이터 변환을 위해 Python과 다른 프로그래밍 언어에서 별도의 도구를 사용해야 했습니다. 다른 언어에 대한 지식이 있어도 언어별로 별도의 컴퓨팅 환경을 설정하고 관리하는 것은 힘들고 시간이 오래 걸릴 수 있습니다.

Snowflake의 개발자 프레임워크인 Snowpark는 Python, SQL, Java 및 Scala를 기본 지원하여 모든 데이터 사용자가 Snowflake 데이터 클라우드로 작업을 가지고 올 수 있도록 합니다. Snowpark를 통해 데이터 엔지니어는 단일 플랫폼에서 자신이 원하는 언어로 ML 모델 및 애플리케이션과 연결된 파이프라인을 더 빠르고 안전하게 실행할 수 있습니다.

다음 페이지에서는 Snowpark와 Snowflake 데이터 클라우드 내에서 Python을 사용하는 모범 사례를 알아보겠습니다. 여러분은 다음 내용을 배우게 됩니다.

- ▶ Snowpark를 사용한 데이터 엔지니어링을 지원하는 Snowflake, 주요 이점 및 사용 사례
- ▶ SQL과 더불어 Python 및 기타 프로그래밍 언어를 지원하는 Snowpark
- ▶ Snowflake 플랫폼 내에서 효율적이고 효과적이게 Python을 사용할 수 있는 데이터 엔지니어
- ▶ 더 거대한 Snowflake 데이터 엔지니어링 생태계와 맞물리는 Snowpark

또한, 데이터 엔지니어가 Snowflake와 Snowpark를 시작하는 것을 돕기 위해 설계된 리소스를 공유하겠습니다.

# 데이터 엔지니어링용 SNOWPARK

Snowpark는 Snowflake를 위한 최신 개발자 프레임워크로 데이터 엔지니어가 선호하는 프로그래밍 언어로 간단하고 빠른 관리형 파이프라인을 구축할 수 있도록 합니다. Snowpark는 다음과 같은 다양한 이점을 데이터 엔지니어에게 제공합니다.

- SQL, Python, Java 및 Scala를 포함한 여러 언어를 지원하는 단일 플랫폼
- 거버넌스 트레이드오프 없이 모든 워크로드에 걸쳐 일관된 보안
- 더 빠르고, 저렴하고, 탄력적인 파이프라인

## 단일 플랫폼:

각기 다른 팀이 여러 처리 엔진에 걸쳐 각기 다른 언어를 사용하면 아키텍처가 급격하게 복잡해집니다. Snowpark는 그림 1에서 보이는 것과 같이 별도의 처리 엔진 없이 사용자가 선택한 프로그래밍 언어를 기본적으로 지원하여 아키텍처를 간소화합니다. Snowpark는 대신 모든 팀을 단일 플랫폼인 Snowflake에서 동일한 데이터에 대해 협업하도록 모읍니다.

## 고객 주요 사항

### HyperFinity

Snowflake 고객인 HyperFinity는 소매업체 및 소비재를 위한 노코드 의사 결정 인텔리전스 플랫폼으로 자체 ML 및 AI 이니셔티브에 SQL과 Python을 사용합니다. Snowpark를 통해 HyperFinity는 두 언어를 모두 지원하는 단일 플랫폼을 보유하고 있습니다. 따라서 복잡하고 느린 데이터 이동과 여러 서비스에 걸쳐 데이터 이동을 유지하기 위해 개발된 코드가 필요하지 않습니다. 결과적으로 HyperFinity는 더 원활하게 작업합니다. 더 민첩한 전반적인 운영을 위해 하나의 환경에서 Python과 SQL을 개발하고, 테스트하고, 배포합니다.

## SNOWPARK에서 지원되는 언어

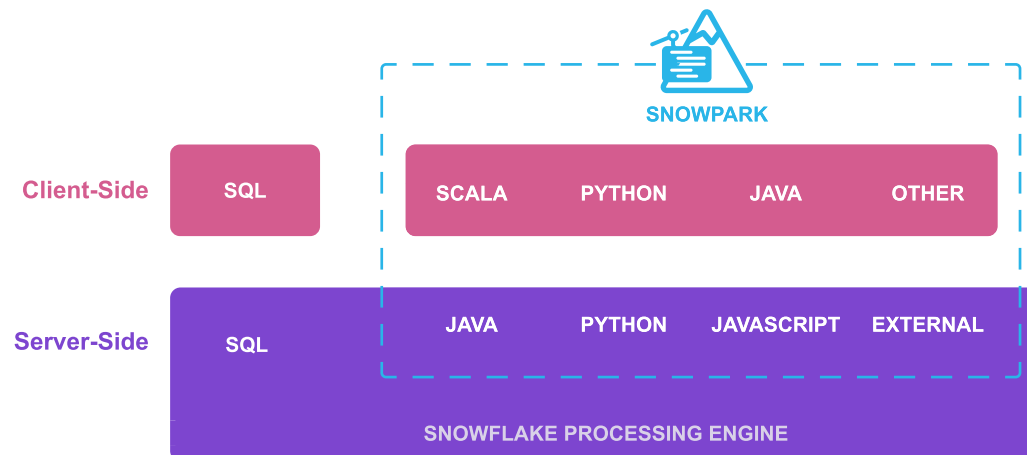


그림 1: Snowpark로 개발자는 Snowflake의 위력을 활용하기 위해 여러 언어로 작업할 수 있습니다.

## 거버넌스 트레이드오프 없음

엔터프라이즈급 거버넌스 제어 및 보안은 Snowflake에 내장되어 있습니다. 예를 들어, Snowpark는 악성 워크로드로부터 네트워크와 호스트를 보호하기 위해 데이터를 격리하는 설계로 보호됩니다. 또한, 관리자에게 개발자가 실행하는 라이브러리에 대한 제어권을 부여합니다. 개발자는 데이터 보안과 규정 준수 조치가 일관되고 내장되어 있음을 알기에 자신 있게 구축할 수 있습니다.

### 고객 주요 사항



가스과 탄소 제로 전기를 영국 가정과 기업에 공급하는 EDF는 데이터 애플리케이션 배포를 돕기 위해 Snowpark를 이용했습니다. EDF는 Snowflake 내에서 작업했기에 해당 프로젝트에는 데이터 접근성을 승인하기 위한 추가적인 서명과 회의가 필요하지 않았습니다. 대신 EDF 팀은 Snowflake가 활성화하는 보안 규칙을 준수하여 원활하게 규모를 조정할 수 있었고 이는 해당 프로젝트에 적용되었습니다.

Snowpark를 자체 데이터 엔지니어링 작업에 통합했기에 EDF는 고객이 직접 사용하는 ML 기반 프로그램의 프로덕션 속도를 몇 개월에서 3~4주로 단축하여 출력을 4배 늘렸습니다.

## 더 빠르고 저렴한 파이프라인

Snowpark는 Snowflake의 고유한 다중 클러스터 공유 데이터 아키텍처 덕분에 파이프라인의 가격 대비 성능을 개선하고, 비용을 투명하게 만들고, 운영 오버헤드를 줄일 수 있습니다. Snowflake는 오늘날 조직에 필요한 성능, 규모, 유연성 및 동시성을 제공하는 단일 통합 플랫폼입니다.

### 고객 주요 사항



생명 과학 산업에서 선도적인 분석, 테크놀로지 솔루션 및 임상 연구 서비스 공급자이자 Snowflake 고객인 IQVIA는 이러한 이점을 경험했습니다. IQVIA는 늘어나는 많은 양의 정형, 반정형 및 비정형 데이터를 처리하고 비즈니스 규모가 확장됨에 따라 늘어나는 복잡성을 해결해야 했습니다.

Snowflake에서 Snowpark를 구현한 후 IQVIA는 행 수준 액세스, 데이터 마스킹, 처리할 데이터와 더 가까운 위치와 같은 일관된 엔터프라이즈급 거버넌스 기능을 통해 데이터 엔지니어링 파이프라인 및 지능형 앱을 더 빠르고 쉽게 개발했습니다. 많은 양의 데이터를 처리하는 자체 파이프라인을 구축하기 위해 Snowpark를 활용하며 IQVIA는 이전 파이프라인 프로세스에 비해 비용이 3배 절약되었음을 깨달았습니다.

## 데이터 엔지니어링용 SNOWFLAKE(및 SNOWPARK)

Snowpark는 데이터 엔지니어링을 위한 강력한 개발자 프레임워크입니다. Snowpark에서 작업하는 데이터 엔지니어에게 중요한 몇몇 사용 사례는 다음과 같습니다.

- **ETL/ELT:** 데이터 팀은 JSON, Parquet, XML 등 유형에 관계없이 원시 데이터를 모델링된 형식으로 변환하는 데 Snowpark를 사용할 수 있습니다. 그런 다음에는 모든 데이터 변환을 Snowpark 저장 프로시저로 패키징하여 Snowflake Tasks 또는 기타 오케스트레이션 도구로 운영하고 작업을 예약할 수 있습니다.
- **맞춤형 논리:** 사용자는 SQL 쿼리 및 변환을 실행하는 바로 그 플랫폼에서 Snowpark의 사용자 정의 함수(UDF)로 복잡한 데이터 처리 및 Python 또는 Java로 작성된 맞춤형 비즈니스 논리가 있는 아키텍처를 간소화할 수 있습니다. 관리, 확장 또는 운영이 필요한 별도의 클러스터는 없습니다.
- **데이터 사이언스 및 ML 파이프라인:** 데이터 팀은 통합 Anaconda 리포지토리 및 패키지 관리자를 통해 ML 데이터 파이프라인을 프로덕션으로 가져오기 위해 협업할 수 있습니다. 학습된 ML 모델을 또한 UDF로 패키징하여 모델 추론을 데이터에 가깝게 실행할 수 있고, 이를 통해 모델 개발에서 프로덕션까지의 경로를 단축할 수 있습니다.



사용자 지정 Python 또는 Java 코드의 경우, SQL로 변환할 필요가 없습니다. 오히려 코드는 직렬화되고 보안 Java 또는 Python 샌드박스 내부에서 처리되도록 Snowflake로 전송됩니다. Python의 경우, 사용자 지정 코드에 통합 Anaconda 패키지 리포지토리에서 제공하는 타사 오픈 소스 라이브러리가 포함되어 있다면 패키지 관리자는 복잡한 환경 관리 없이 코드가 실행되도록 도울 수 있습니다.

### SNOWPARK 클라이언트 API

Snowpark 클라이언트 API는 오픈 소스이며 모든 Python 환경과 작동합니다. 이를 통해 데이터 엔지니어는 SQL 문자열을 생성 및 전달할 필요 없이 자체 Python 코드 내에서 바로 데이터프레임을 사용하여 쿼리를 구축할 수 있습니다.

### Snowpark 데이터프레임

Snowpark는 심층적으로 통합된 데이터프레임 스타일의 프로그래밍을 데이터 엔지니어가 선호하는 언어로 가져옵니다. 데이터 엔지니어는 자신이 원하는 IDE 또는 개발 도구를 사용하여 Python에서 데이터프레임 스타일의 프로그래밍으로 Snowpark에서 쿼리를 구축할 수 있습니다. 백그라운드에서는 모든 데이터프레임 작업이 SQL 쿼리로 투명하게 변환되고 Snowflake의 확장 가능한 처리 엔진으로 푸시다운 됩니다. 데이터프레임은 1급 언어 구성소를 사용하기 때문에 엔지니어는 자신의 개발 환경에서 형식 검사, IntelliSense 및 오류 보고에 대한 지원의 이점을 누릴 수도 있습니다.

### SNOWPARK 서버 측 런타임

Snowflake는 클라우드에서 데이터 플랫폼으로 구축되었습니다. 이는 스토리지와 컴퓨팅을 아키텍처상으로 분리하지만, 논리적으로는 통합하며 이러한 리소스를 거의 무제한으로 제공하기 위해 최적화되어 있습니다. 유연한 규모 조정, 다국어 처리 및 통합 거버넌스는 또한 Snowflake의 아키텍처를 뒷받침합니다.

지능형 인프라는 모든 것이 제대로 작동하게 합니다. 컴퓨팅 클러스터를 자동 또는 수동으로 시작하거나, 중지하거나, 크기를 조정할 수 있습니다. 이를 통해 언제든지 더 많거나 적은 컴퓨팅 리소스가 필요하면 이에 맞춰 조정할 수 있습니다. 유연성과 더불어 Snowflake는 각 워크로드의 전용 컴퓨팅 클러스터에 대한 거의 즉각적인 액세스를 보장하여 속도를 우선시합니다. 따라서 사용자는 성능을 저하시키지 않고 거의 무제한인 동시성의 이점을 누릴 수 있습니다. Snowflake 단일 플랫폼 내에서 통합되는 3개의 아키텍처 계층은 그림 3에 나와 있습니다.

Snowpark Python 서버 측 런타임으로 Snowflake의 보안 Python 샌드박스에 배포되는 Python UDF와 저장 프로시저를 작성하는 것이 가능해집니다. UDF와 저장 프로시저는 데이터 엔지니어가 사용자 지정 Python 로직을 Snowflake 컴퓨팅 엔진으로 가져올 수 있게 하는 동시에 Snowpark에 사전에 설치된 오픈 소스 패키지의 이점을 활용하게 하는 Snowpark의 다른 2가지 핵심 구성 요소입니다.

### SNOWFLAKE 플랫폼 아키텍처

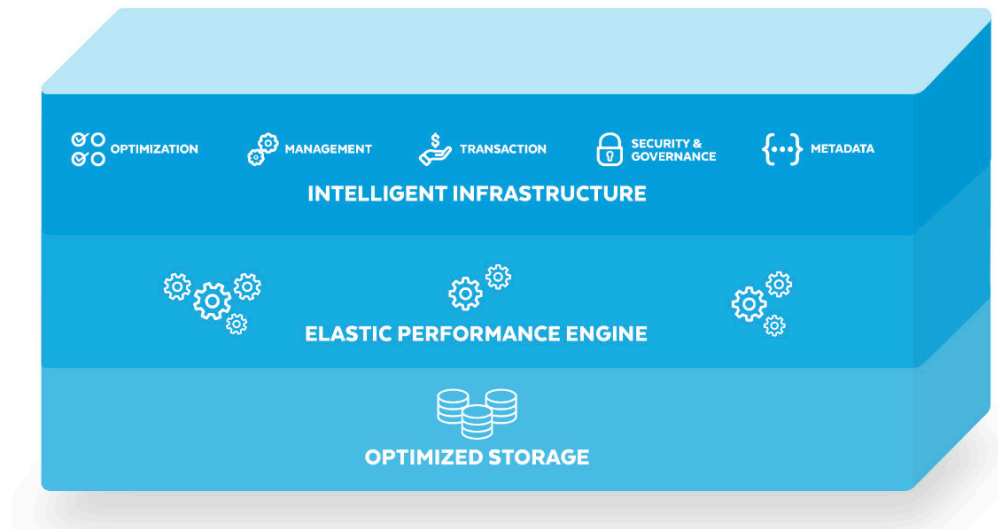


그림 3: Snowflake 단일 플랫폼은 3개의 고유한 아키텍처 계층을 통합합니다.

## Snowpark 사용자 정의 함수(UDF)

Python에서 작성된 사용자 지정 로직은 UDF를 사용하여 Snowflake에서 바로 실행됩니다. 함수는 독립형이거나 데이터를 처리하기 위한 데이터프레임 작업의 일부로 호출될 수 있습니다. Snowpark는 사용자 지정 코드를 Python 바이트 코드로 직렬화하는 작업을 처리하고 데이터 옆에서 실행될 수 있도록 로직을 Snowflake로 모두 푸시합니다. 코드를 호스팅하기 위해 Snowpark는 보안 샌드박스에 존재하는 Python 런타임을 Snowflake 엔진에 바로 구축했습니다. Python UDF는 기본 Python 코드와 관련된 처리를 확장합니다. 이는 모든 스레드와 노드에 걸쳐 동시에 발생하며 함수가 실행되는 가상 웨어하우스를 구성합니다.

다음은 데이터 엔지니어가 Snowpark에서 사용할 수 있는 다양한 UDF 유형입니다.

- **스칼라 UDF:** 각 행에서 독립적으로 작동하고 하나의 결과를 생성합니다
- **벡터화된 UDF:** 입력 행 배치를 Pandas 데이터프레임으로 수신하고, 결과 배치를 Pandas 배열 또는 시리즈로 반환합니다
- **사용자 정의 테이블 함수:** 각 입력 행을 위해 여러 행을 반환하거나, 행 그룹을 위해 하나의 결과를 반환하거나, 여러 행에 걸쳐 상태를 유지합니다

오른쪽에는 유통 센터와 배송 위치 사이의 거리를 계산하는 데 사용된 Snowpark UDF의 예입니다.

#주어진 지리 좌표로 유통 센터와 배송 위치 사이의 거리를 계산하는 UDF

```
from snowflake.snowpark.functions import udf
import geopandas as gpd
from shapely.geometry import Point

@udf/packages=['geopandas']
def calculate_distance(lat1: float, long1: float, lat2: float, long2: float)-> float:
    points_df = gpd.GeoDataFrame({'geometry': [Point(long1, lat1), Point(long2, lat2)], crs='EPSG:4326'}).to_crs('EPSG:3310')
    return points_df.distance(points_df.shift()).iloc[1]
```

# 위치 좌표를 포함하는 데이터프레임에 대해 함수 호출

```
distance_df = loc_df.select(loc_df.sale_id, loc_df.distribution_center_address, loc_df.shipping_address, \
    calculate_distance(loc_df.distribution_center_lat, loc_df.distribution_center_lng, loc_df.shipping_lat, loc_df.shipping_lng) \
    .alias('distribution_center_to_shipping_distance'))
```

## 저장 프로시저

Snowpark 저장 프로시저는 데이터 엔지니어가 자신의 Python 코드를 운영하고 자신의 파이프라인을 실행, 오케스트레이션 및 예약하는 것을 돕습니다. 저장 프로시저를 한 번 생성하면 오케스트레이션 또는 자동화 도구에서 간단한 CALL 문으로 여러 번 실행할 수 있습니다. Snowflake는 데이터 엔지니어가 다국어 파이프라인을 쉽게 생성할 수 있도록 SQL, Python, Java, Javascript 및 Scala에서 저장 프로시저를 지원합니다.

저장 프로시저를 사용하기 위해 개발자는 클라이언트 API의 Snowpark에서 `sproc()` 함수를 사용하여 Python 함수를 묶고 Snowpark가 이를 서버 측에서 배포하게 할 수 있습니다. Snowpark는 Python 코드와 종속성을 바이트 코드로 직렬화하고 이를 Snowflake 스테이지에 자동으로 저장합니다. 이는 임시(세션 수준) 또는 영구 객체로 Snowflake에서 생성될 수 있습니다.

저장 프로시저는 단일 노드입니다. 이는 저장 프로시저 내에서 원하는 규모로 데이터를 변환 또는 분석하려면 컴퓨팅 클러스터의 모든 노드에 걸쳐 컴퓨팅 규모를 조정하기 위해 클라이언트 API 또는 기타 배포된 UDF를 활용해야 함을 의미합니다.

오른쪽에는 회사의 매출 보너스를 매일 계산하고 적용하는 Python용 Snowpark 파이프라인을 운영하는 방법의 간단한 예가 있습니다.

```
-- 보너스를 계산하고 적용하기 위해 Snowpark 파이프라인을 호스팅하고 실행할 Python 저장 프로시저 생성
create or replace procedure apply_bonuses(sales_table string, bonus_table
string)
    returns string
    language python
    runtime_version = '3.8'
    packages = ('snowflake-snowpark-python')
    handler = 'apply_bonuses'
AS
$$
from snowflake.snowpark.functions import udf, col
from snowflake.snowpark.types import *

def apply_bonuses(session, sales_table, bonus_table):
    session.table(sales_table).select(col("rep_id"), col("sales_amount")*0.1).
write.save_as_table(bonus_table)
    return "SUCCESS"
$$;

--보너스를 적용하기 위해 저장 프로시저 호출
call apply_bonuses('wholesale_sales','bonuses');

- Query bonuses table to see newly applied bonuses
select * from bonuses;

- Create a task to run the pipeline on a daily basis
create or replace task bonus_task
warehouse = 'xs'
schedule = '1440 minute'
as
call apply_bonuses('wholesale_sales','bonuses');
```

## SNOWFLAKE, ANACONDA 및 오픈 소스 생태계

Python의 이점 중 하나는 오픈 소스 패키지와 라이브러리로 구성된 풍부한 생태계입니다. 최근 오픈 소스 패키지는 더 빠르고 쉬운 데이터 엔지니어링을 가능하게 한 가장 큰 요인 중 하나가 되었습니다. 오픈 소스 혁신을 활용하려 Snowpark는 웨어하우스 사용량 외에는 사용자에게 추가 비용 또는 라이선스를 요구하지 않고 제품을 통합하기 위해 Anaconda와 파트너십을 체결했습니다.

Snowflake에서 데이터 엔지니어는 이제 원활한 종속성 관리와 Anaconda에서 제공하는 포괄적이고 엄선된 오픈 소스 패키지 세트의 이점을 활용하여 Python 기반 파이프라인의 속도를 개선할 수 있습니다. 데이터를 이동하거나 복사할 필요는 없습니다. 모든 Snowpark 사용자는 Anaconda 리포지토리에서 사전에 설치된 수천 개의 가장 널리 사용되는 패키지의 이점을 누릴 수 있습니다.

여기에는 문자열 매칭을 위한 fuzzy wuzzy, 지리 공간 분석을 위한 h3, 머신 러닝 및 예측 데이터 분석을 위한 scikit-learn이 포함됩니다. 또한, Snowpark는 Conda 패키지 관리자와 통합되어 있기에 사용자는 누락된 종속성으로 인해 손상된 Python 환경을 피할 수 있습니다.

Snowflake에서 오픈 소스 패키지를 사용하는 것은 사용자가 Snowpark에서 바로 NumPy, XGBoost 및 Pandas와 같은 호출 패키지를 사용하는 방법을 보여주는 아래 코드만큼 단순합니다.

Snowpark는 또한 오늘날 가장 널리 사용되는 데이터 변환 솔루션 중 하나인 dbt를 완벽히 지원합니다. 이는 SQL 우선 변환 워크플로우를 지원하며, 2022년 dbt는 Python에 대한 지원을 도입했습니다. SQL과 Python 모두에 대한 dbt의 지원을 통해 사용자는 가장 친숙하고 목적에 맞는 언어로

변환을 작성할 수 있게 되었습니다. 또한, Snowpark의 dbt는 데이터 엔지니어링 및 데이터 사이언스를 위한 최첨단 패키지를 포함하여 오픈 소스 Python 생태계에서 사용할 수 있는 도구를 이용해 분석할 수 있도록 하며, 이 모든 것이 많은 SQL 사용자에게 친숙한 dbt 프레임워크 내에서 이루어집니다. SQL 우선 워크플로우를 지원하며 2022년에 dbt는 Python을 오픈 소스 Python 생태계에서 제공하는 도구를 사용하여 분석을 수행하기 위해 Snowpark 내부 구조를 실행하는 제2언어로 도입했습니다.

```
-- NumPy, Pandas 및 XGboost 패키지 버전의 배열 반환
create or replace function py_udf()
returns array
language python
runtime_version = 3.8
packages = ('numpy','pandas==1.4.*','xgboost==1.5.0')
handler = 'udf'
as $$
import numpy as np
import pandas as pd
import xgboost as xgb
def udf():
    return [np.__version__, pd.__version__, xgb.__version__]
$$;
```





## SNOWFLAKE를 사용한 데이터 엔지니어링



\* in public preview  
 \*\* in private preview

그림 4: 자동화된 워크플로우가 변환과 제공을 용이하게 하는 동안 Snowflake는 비정형, 반정형 및 정형 데이터 수집을 지원합니다.





# SNOWFLAKE 소개

Snowflake와 함께하면 모든 조직이 Snowflake의 데이터 클라우드를 통해 데이터를 집결할 수 있습니다. 고객은 데이터 클라우드를 사용하여 사일로화 된 데이터를 통합하고, 데이터를 검색하고 안전하게 공유하며, 다양한 분석 워크로드를 실행합니다. 데이터 또는 사용자가 어디에 있든 Snowflake를 통해 여러 클라우드와 지역에 걸쳐 단일 데이터를 경험합니다. 2023년 4월 30일 일요일 기준, 2022년 Forbes Global 2000 기업(G2K) 중 590개사를 비롯한 다양한 산업 분야의 수천여 고객이 Snowflake 데이터 클라우드를 사용하여 비즈니스를 강화하고 있습니다.

자세한 내용은 [snowflake.com](https://snowflake.com)에서 확인할 수 있습니다.



© 2023 Snowflake Inc. All rights reserved. 여기에 언급된 Snowflake, Snowflake 로고 및 기타 모든 Snowflake 제품, 기능 및 서비스 이름은 미국 및 기타 국가에서 Snowflake Inc.의 등록 상표 또는 상표입니다. 여기에 언급되거나 사용된 기타 모든 브랜드 이름 또는 로고는 식별 목적으로만 사용되며 해당 소유자의 상표일 수 있습니다. Snowflake는 그러한 소유자와 연관되거나 후원 또는 보증을 받지 않습니다.

## 인용

<sup>1</sup> <https://insights.stackoverflow.com/survey>