

NEXON

스노우플레이크 기초 교육

MEGAZONECLOUD

Snowflake Architecture

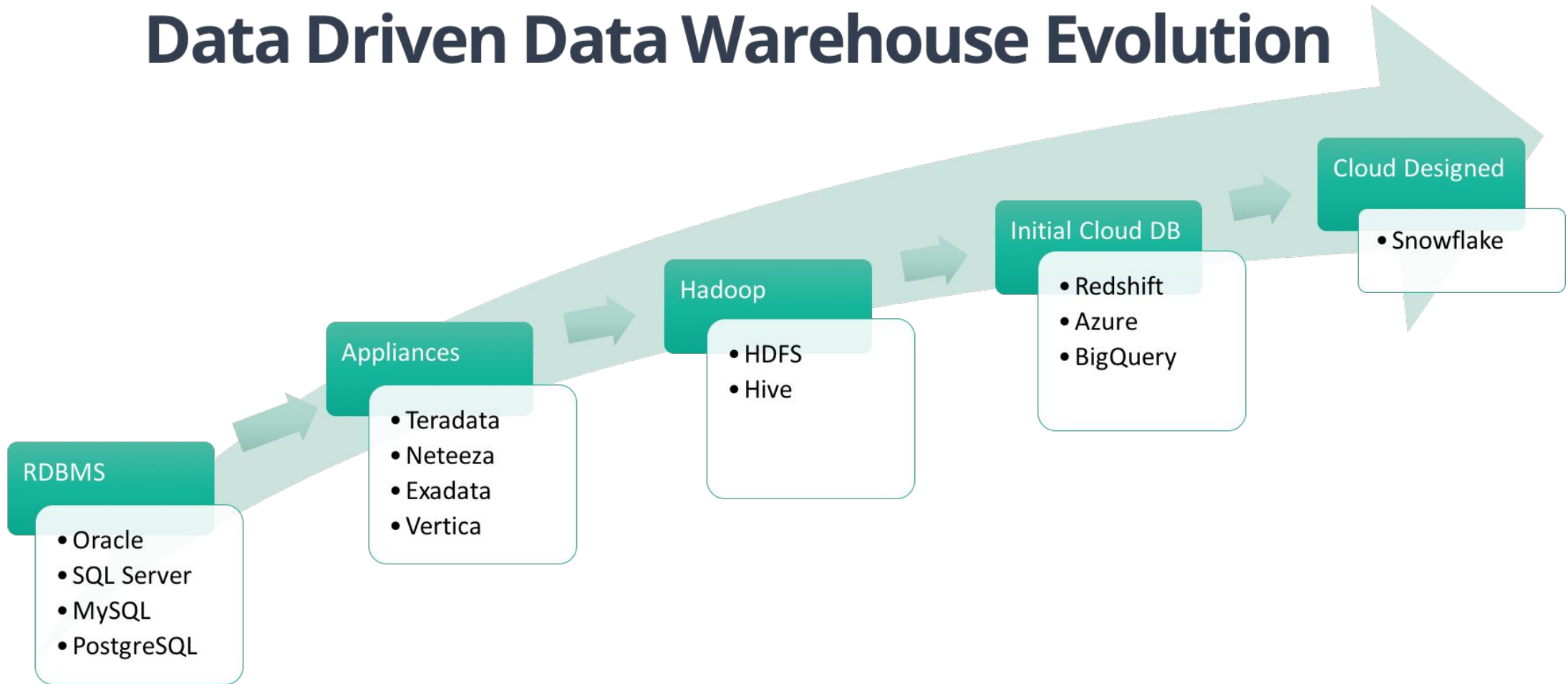
목차

1. Architecture
2. Snowsight
3. SnowSQL

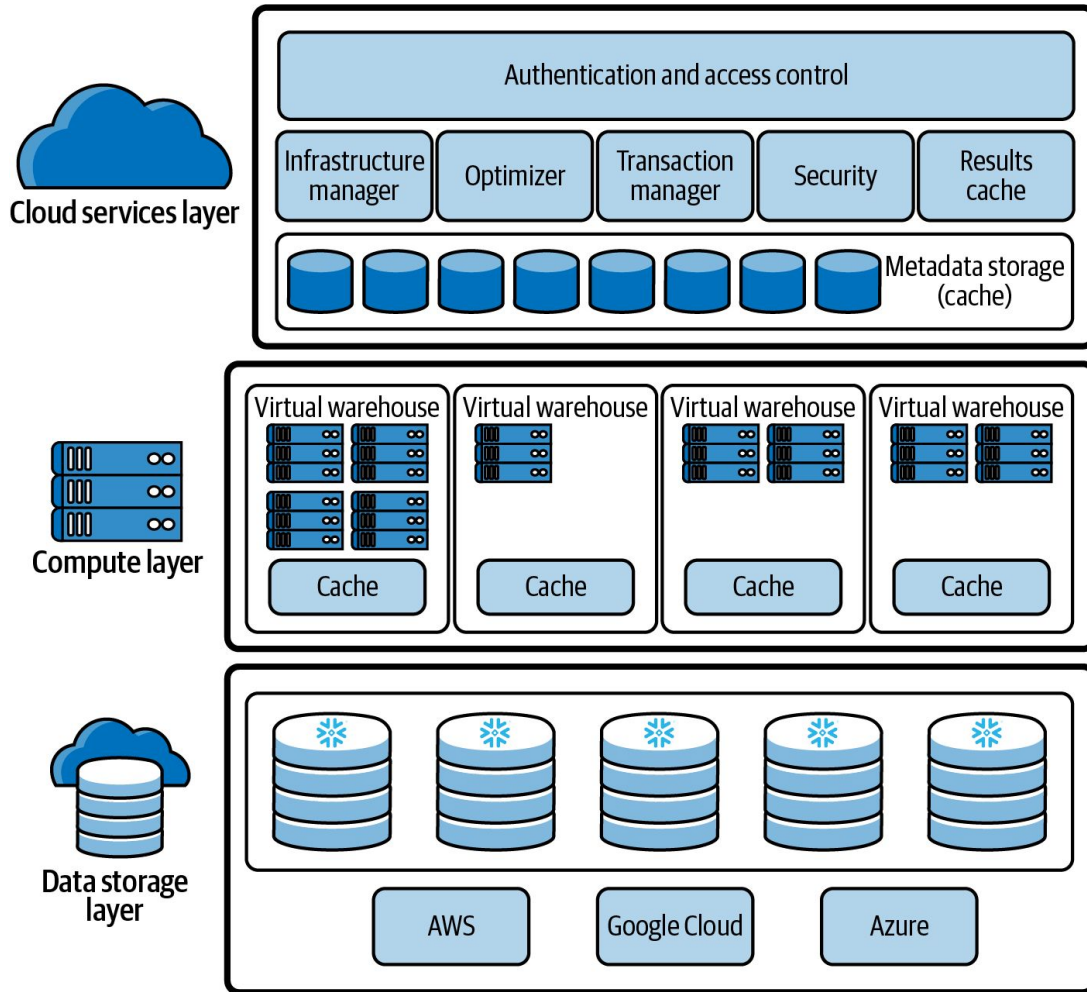
1. 아키텍처 개요

1.1. 데이터 웨어하우스 변화

Data Driven Data Warehouse Evolution

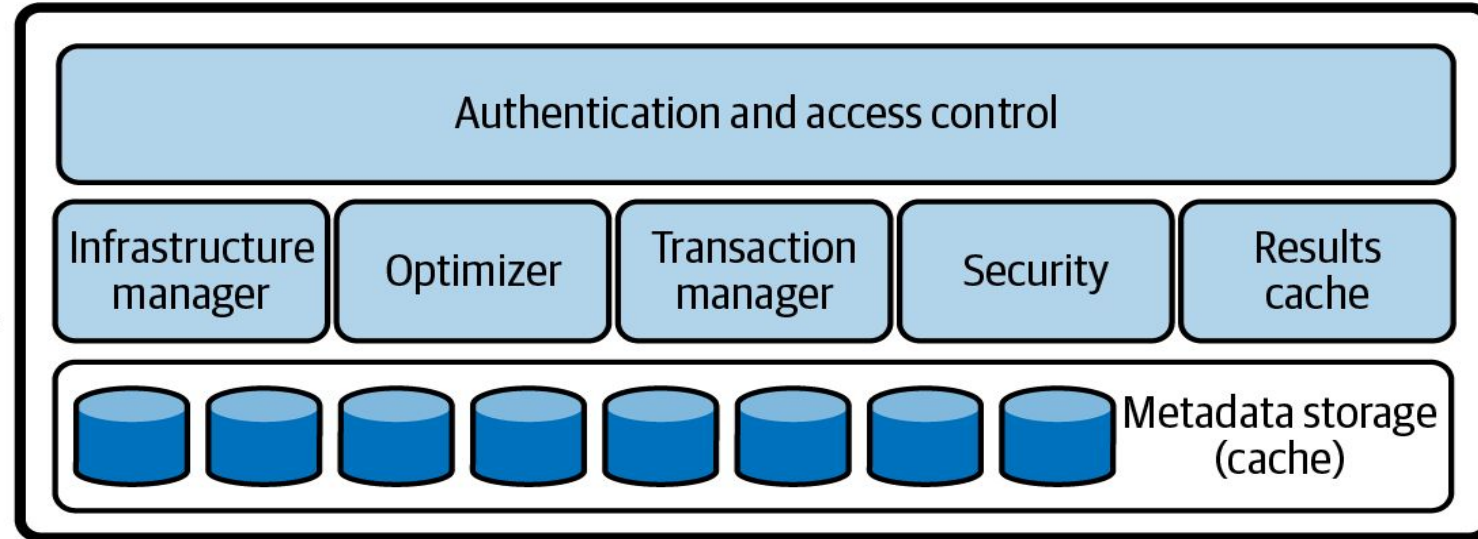


1.2. 스노우플레이크 아키텍처



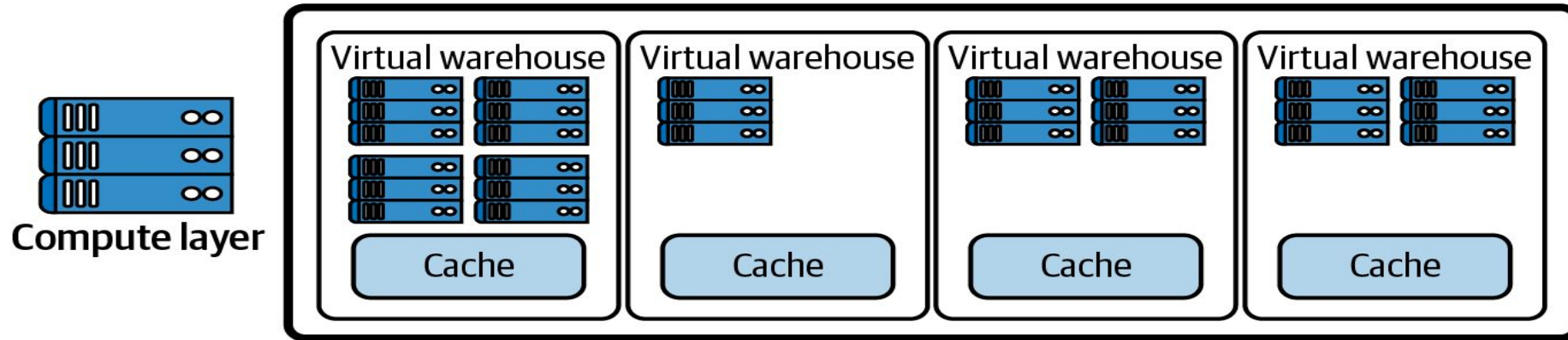
- 개선된 기존 데이터 플랫폼, 특히 온프레미스에 구현되었던 것들조차도 현대의 데이터 문제를 적절히 해결하거나 오래된 확장성 문제를 풀지 못했습니다. 스노우플레이크 팀은 독특한 접근 방식을 취하기로 결정했습니다. 기존 소프트웨어 아키텍처를 점진적으로 개선하거나 변형하는 대신, 그들은 여러 사용자가 라이브 데이터를 동시에 공유할 수 있도록, 오직 클라우드만을 위한 완전히 새롭고 현대적인 데이터 플랫폼을 구축했습니다.
- 스노우플레이크의 독특한 설계는 스토리지와 컴퓨트를 물리적으로 분리하면서도 논리적으로 통합하며, 보안 및 관리와 같은 서비스를 함께 제공합니다. 이어지는 장들에서 스노우플레이크의 많은 고유한 기능들을 살펴보면, 왜 스노우플레이크 아키텍처가 데이터 클라우드를 가능하게 하는 유일한 아키텍처인지 직접 확인할 수 있을 것입니다.
- 스노우플레이크 하이브리드 모델 아키텍처는 세 개의 계층으로 구성됩니다. 클라우드 서비스 계층, 컴퓨트 계층, 그리고 데이터 스토리지 계층입니다. 각 계층과 세 가지 스노우플레이크 캐시에 대해서는 뒤에서 더 자세히 설명합니다.

1.2.1. 클라우드 서비스 레이어



- 스노우플레이크 인스턴스 내 데이터와의 모든 상호작용은 클라우드 서비스 계층에서 시작되며, 이는 전역 서비스 계층이라고도 불립니다. 스노우플레이크 클라우드 서비스 계층은 인증, 접근 제어, 암호화와 같은 활동들을 조정하는 서비스들의 집합체입니다.
- 또한 인프라와 메타데이터를 처리하기 위한 관리 기능뿐만 아니라, 쿼리 파싱 및 최적화 수행과 같은 여러 다른 기능들도 포함합니다. 클라우드 서비스 계층은 때때로 '스노우플레이크의 뇌'라고도 불리는데, 이는 다양한 서비스 계층 구성 요소들이 함께 작동하여 사용자가 로그인을 요청하는 시점부터 시작되는 사용자 요청들을 처리하기 때문입니다.
- 사용자가 로그인을 요청할 때마다, 해당 요청은 클라우드 서비스 계층에서 처리됩니다. 사용자가 스노우플레이크 쿼리를 제출하면, 해당 SQL 쿼리는 처리를 위해 컴퓨트 계층으로 보내지기 전에 먼저 클라우드 서비스 계층의 옵티마이저로 전달됩니다. 클라우드 서비스 계층은 데이터에 대한 데이터 정의 언어(DDL) 및 데이터 조작 언어(DML) 작업을 위한 SQL 클라이언트 인터페이스를 가능하게 하는 역할을 합니다.

1.2.2. 컴퓨터 레이어

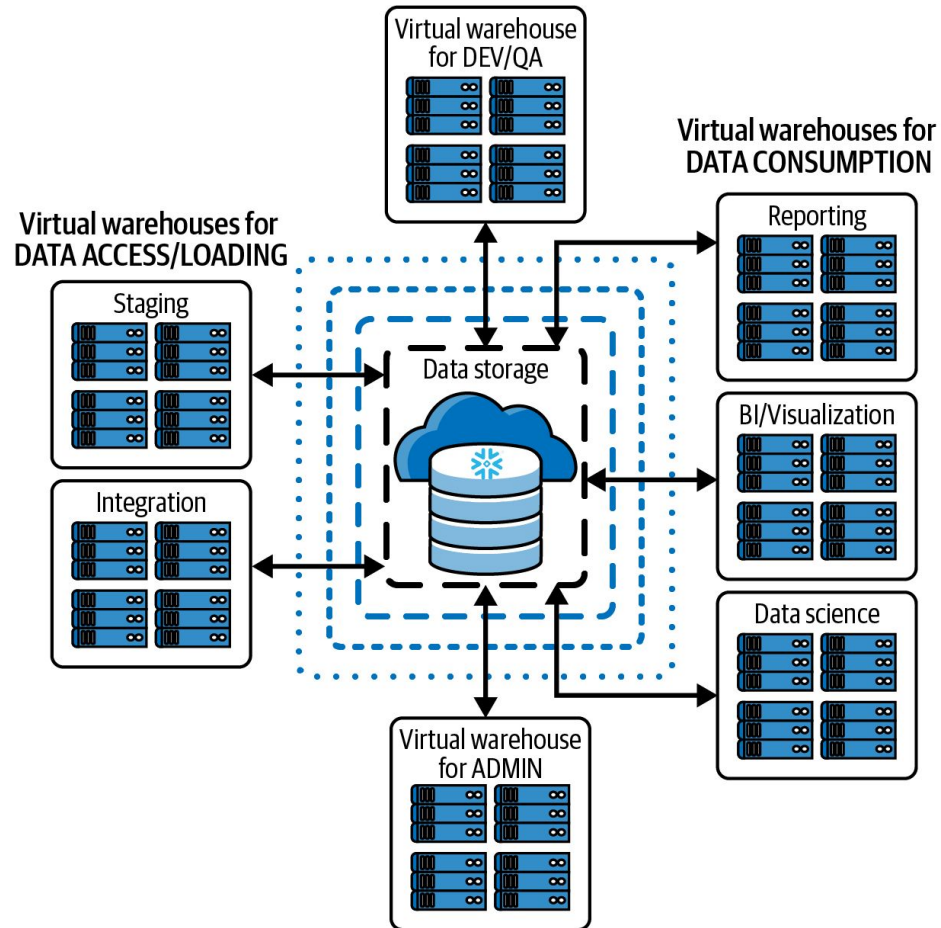


- 가상 웨어하우스는 세션에서 실행 중일 때 항상 크레딧을 소모합니다. 하지만 스노우플레이크 가상 웨어하우스는 언제든지 시작하고 중지할 수 있으며, 실행 중에도 언제든지 크기를 조정할 수 있습니다.
- 스노우플레이크는 웨어하우스를 확장하는 두 가지 다른 방식을 지원합니다. 웨어하우스의 크기를 조정하여 스케일 업 할 수 있고, 웨어하우스에 클러스터를 추가하여 스케일 아웃할 수 있습니다. 하나 또는 두 가지 확장 방식을 동시에 사용하는 것도 가능합니다.
- 스노우플레이크 클라우드 서비스 계층 및 데이터 스토리지 계층과 달리, 스노우플레이크 가상 웨어하우스 계층은 멀티테넌트 아키텍처가 아닙니다.
- 스노우플레이크는 가상 웨어하우스 내 각 노드에 대한 CPU, 메모리, SSD구성을 미리 결정합니다 이러한 정의는 변경될 수 있지만, 구성은 세계의 모든 주요 클라우드 제공업체(AWS, Azure, GCP)에 걸쳐 동일합니다.

X-Small	Small	Medium	Large	X-Large	2X-Large	3X-Large	4X-Large
1	2	4	8	16	32	64	128

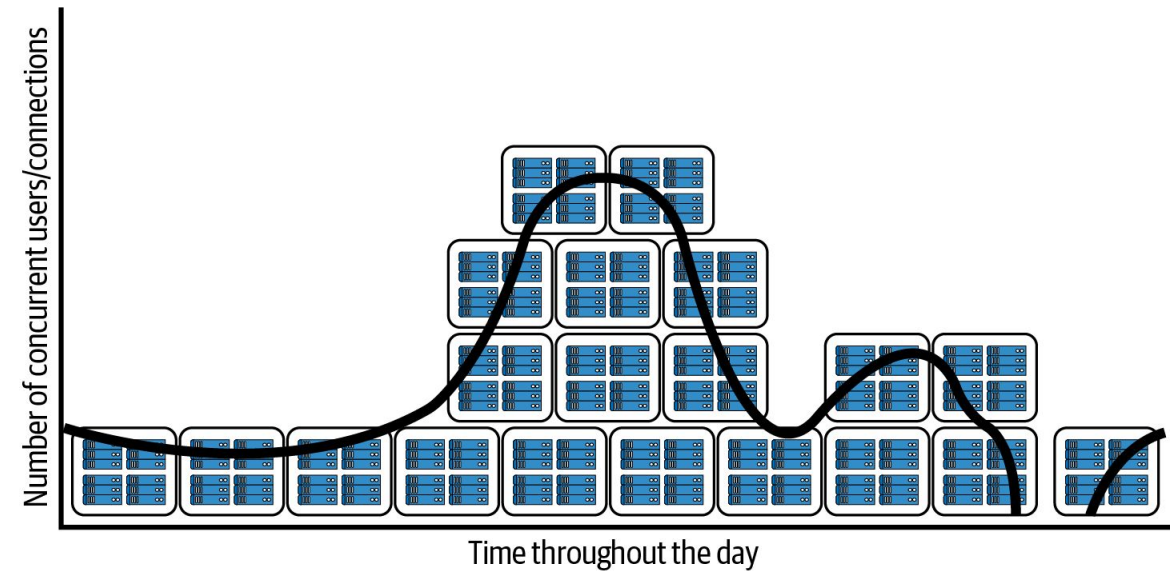
스노우플레이크 가상 웨어하우스 크기와 클러스터별 서버 수

1.2.2. 컴퓨트 레이어

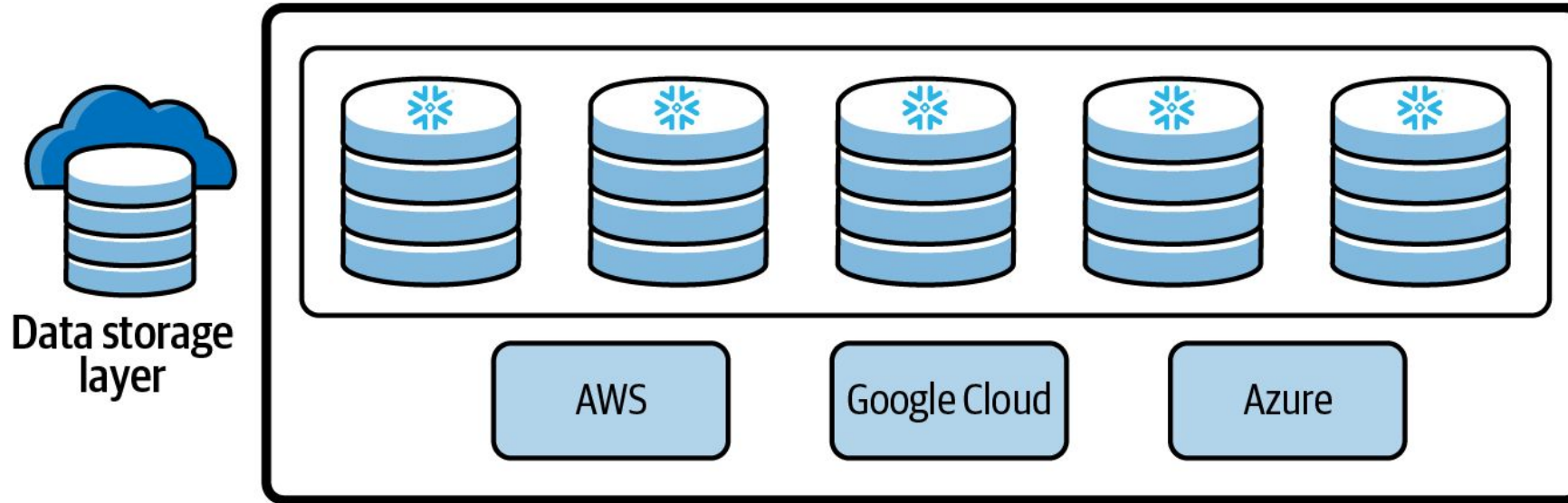


사용자 그룹별로 가상 웨어하우스를 다르게 할당하여
스노우플레이크 워크로드 분리

멀티클러스터 가상 웨어하우스를 활용한 스케일 인/아웃 방식의 스노우플레이크
워크로드 관리

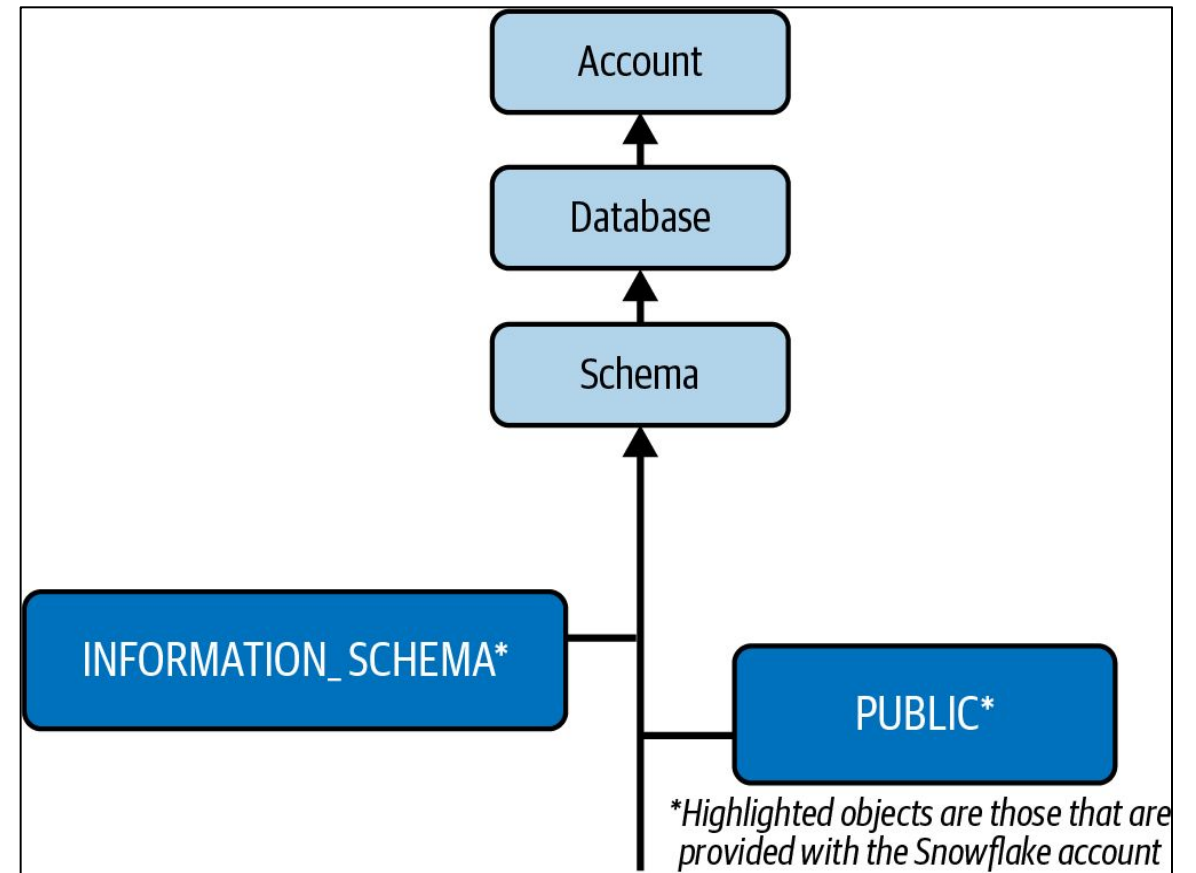
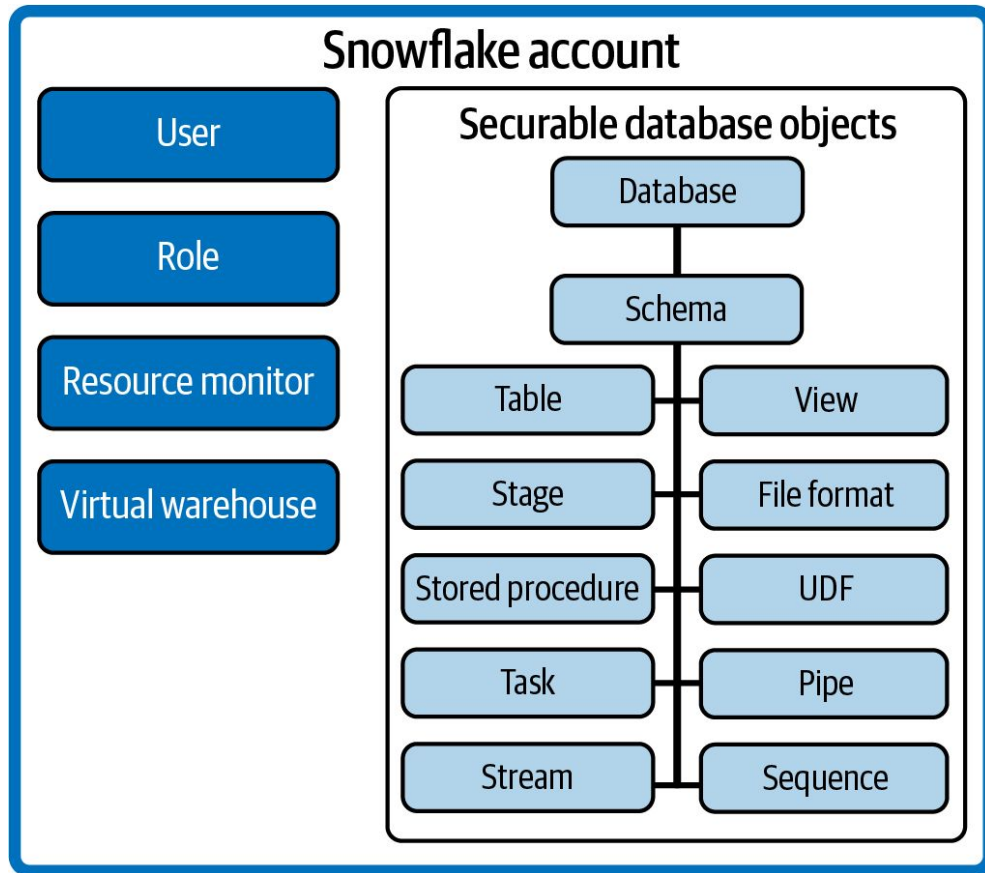


1.2.3. 데이터 스토리지 레이어



- 스노우플레이크의 데이터 스토리지 계층은 때때로 원격 디스크 계층이라고도 불립니다. 기반 파일 시스템은 Amazon, Microsoft 또는 Google Cloud 상에 구현됩니다. 데이터 저장소에 사용되는 특정 제공업체는 스노우플레이크 계정을 생성할 때 선택한 업체입니다. 스노우플레이크는 저장할 수 있는 데이터 양이나 생성할 수 있는 데이터베이스 또는 데이터베이스 객체 수에 제한을 두지 않습니다. 스노우플레이크 테이블은 페타바이트 규모의 데이터도 쉽게 저장할 수 있습니다. 스노우플레이크 계정에서 스토리지 용량이 증가하거나 감소해도 가상 웨어하우스 크기에는 영향이 없습니다. 스토리지와 컴퓨트는 서로 독립적으로, 그리고 클라우드 서비스 계층과도 독립적으로 확장됩니다.
- 스노우플레이크의 중앙 집중식 데이터베이스 스토리지 계층은 정형 및 반정형 데이터를 포함한 모든 데이터를 보유합니다. 데이터가 스노우플레이크에 로드될 때, 최적화되어 압축된 컬럼 형식으로 재구성되고, 스노우플레이크 데이터베이스 내에 저장 및 유지 관리됩니다. 각 스노우플레이크 데이터베이스는 하나 이상의 스키마로 구성되며, 스키마는 테이블이나 뷰와 같은 데이터베이스 객체들의 논리적 그룹입니다.

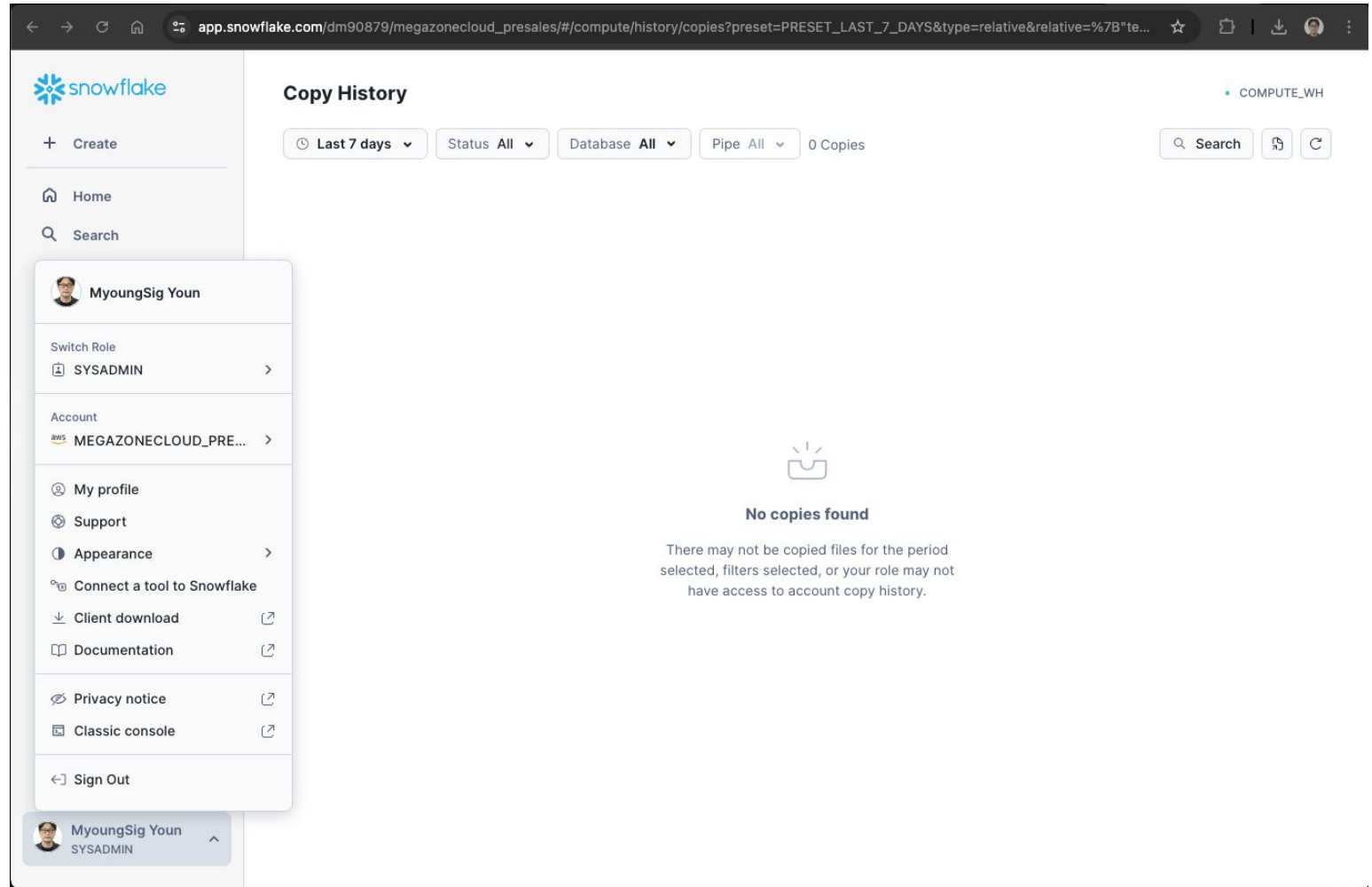
1.3. 스노우플레이크 계정



2. Snowsight

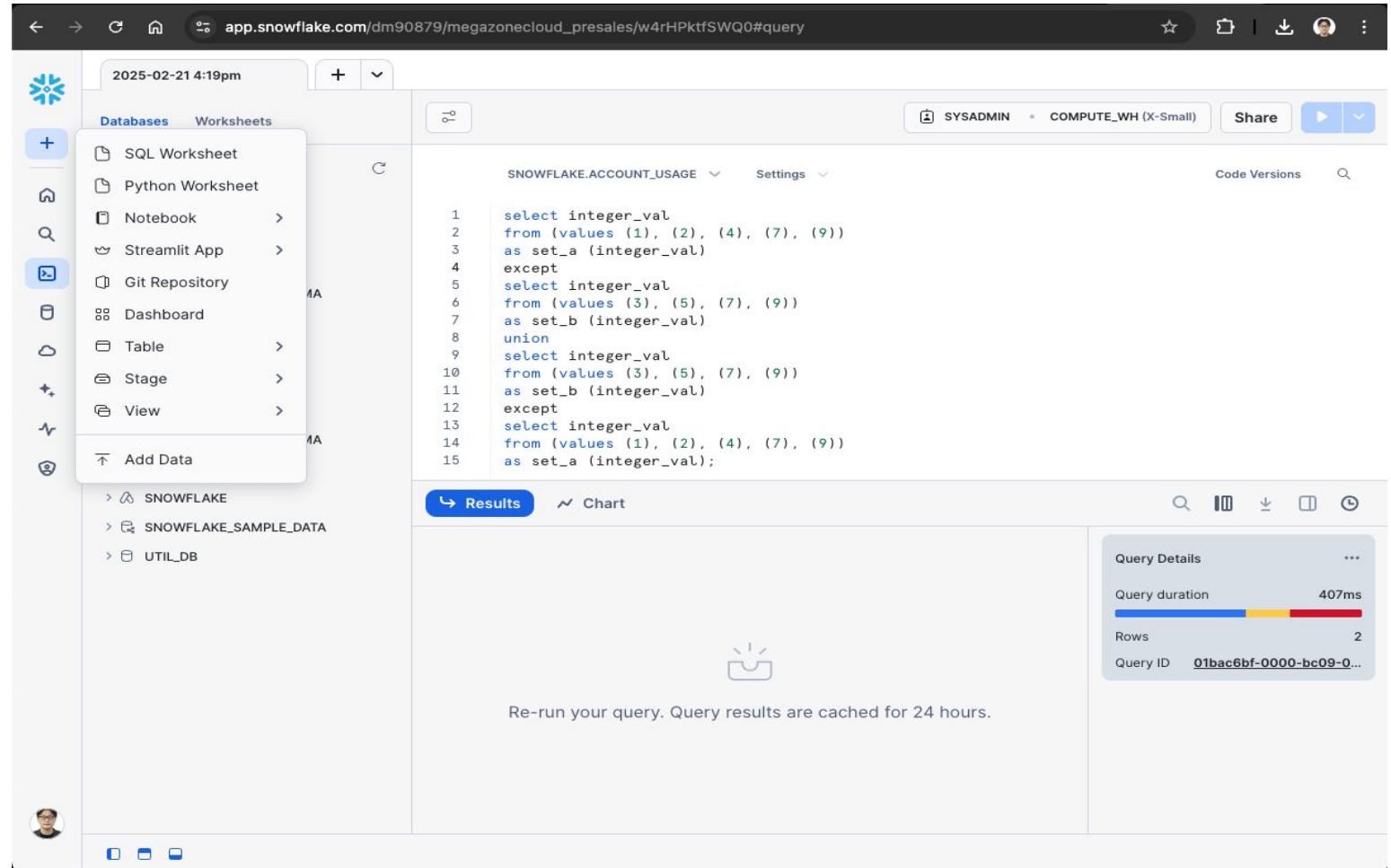
2. 스노우사이트

- 계정 정보 확인



2. 스노우사이트

- 로그인 후 워크시트 선택 화면



2. 스노우사이트

- Admin 메뉴의 Warehouse 화면

The screenshot shows the Snowflake Admin console interface. The left sidebar contains a navigation menu with the following items: Home, Search, Projects, Data, Data Products, AI & ML, Monitoring, Admin, Cost Management, Warehouses (highlighted), Compute Pools, Users & Roles, Accounts, Security, and Contacts. The main content area is titled 'Warehouses' and displays a table of warehouse information. The table has columns: NAME, TYPE, SIZE, STATUS, CLUSTERS, RUNNING, QUEUED, and OWNER. There are two warehouses listed: COMPUTE_WH (Standard, X-Small, Started, 3 clusters) and SYSTEM\$STREAMLIT_NOTEBOOKS (Standard, X-Small, Suspended, 0 clusters). A '+ Warehouse' button is located in the top right corner of the main area. The user profile 'MyoungSig Youn SYSADMIN' is visible in the bottom left corner of the sidebar.

NAME	TYPE	SIZE	STATUS	CLUSTERS	RUNNING	QUEUED	OWNER
COMPUTE_WH	Standard	X-Small	Started	3	3	0	SYSADM
SYSTEM\$STREAMLIT_NOTEBOOKS	Standard	X-Small	Suspended	0	0	0	ACCOUNT

2. 스노우사이트

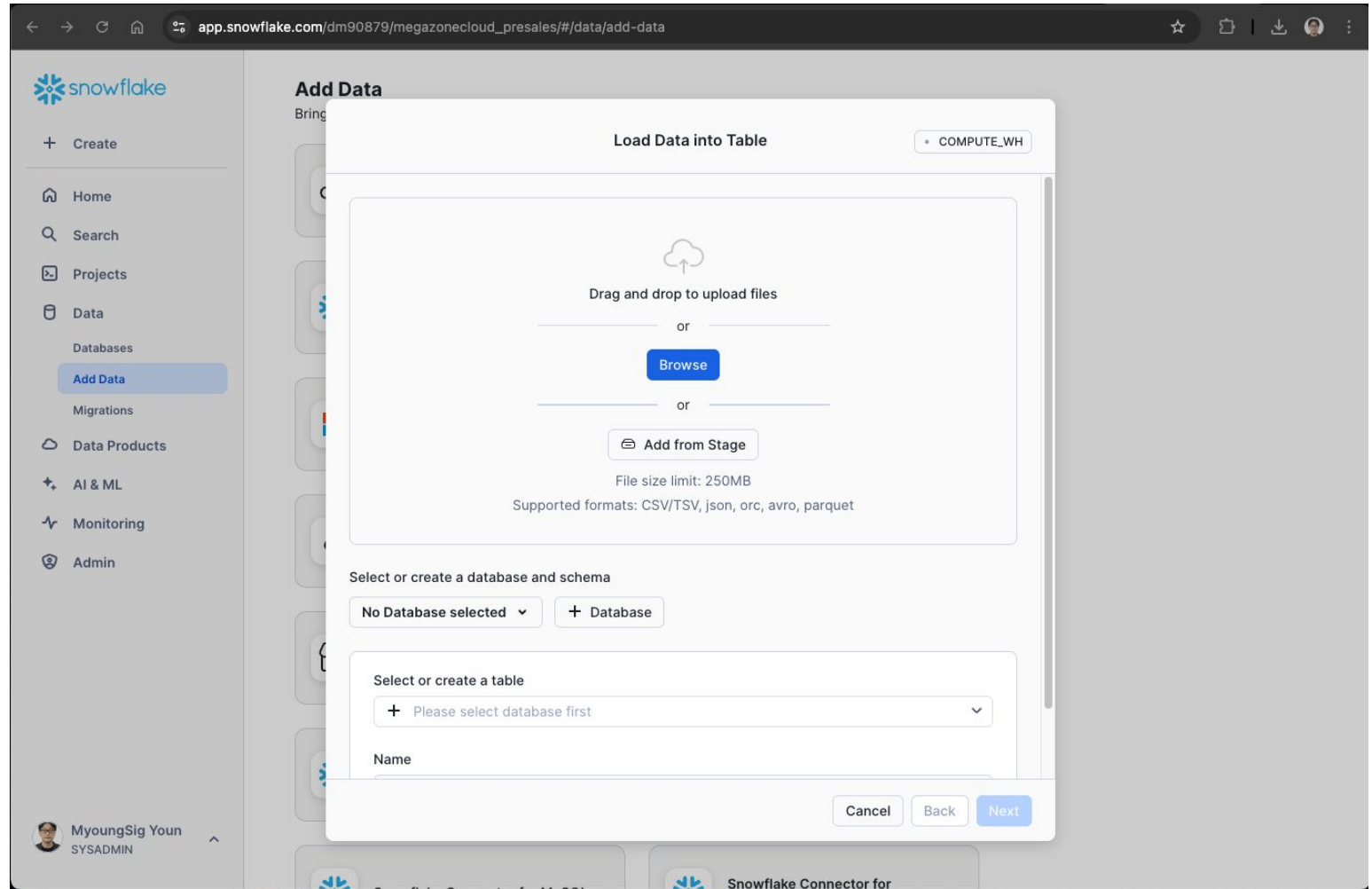
- Data 메뉴의 Database 화면

The screenshot shows the Snowflake web interface. The left sidebar has a navigation menu with options: Home, Search, Projects, Data, Databases (selected), Add Data, Migrations, Data Products, AI & ML, Monitoring, and Admin. The main content area is titled 'Databases' and shows a list of 7 databases. The table has columns: NAME, SOURCE, OWNER, and CREATED. The databases listed are FILE_DB, GARDEN_PLANTS, LEARNING_SQL, MZC_DB, SNOWFLAKE, SNOWFLAKE_SAMPLE_DATA, and UTIL_DB. The user 'MyoungSig Youn' is logged in as SYSADMIN.

NAME	SOURCE	OWNER	CREATED
FILE_DB	Local	SYSADMIN	3 weeks ago
GARDEN_PLANTS	Local	SYSADMIN	3 weeks ago
LEARNING_SQL	Local	SYSADMIN	1 month ago
MZC_DB	Local	SYSADMIN	2 weeks ago
SNOWFLAKE	Share	—	12 months ago
SNOWFLAKE_SA...	Share	ACCOUNTADMIN	9 months ago
UTIL_DB	Local	SYSADMIN	3 weeks ago

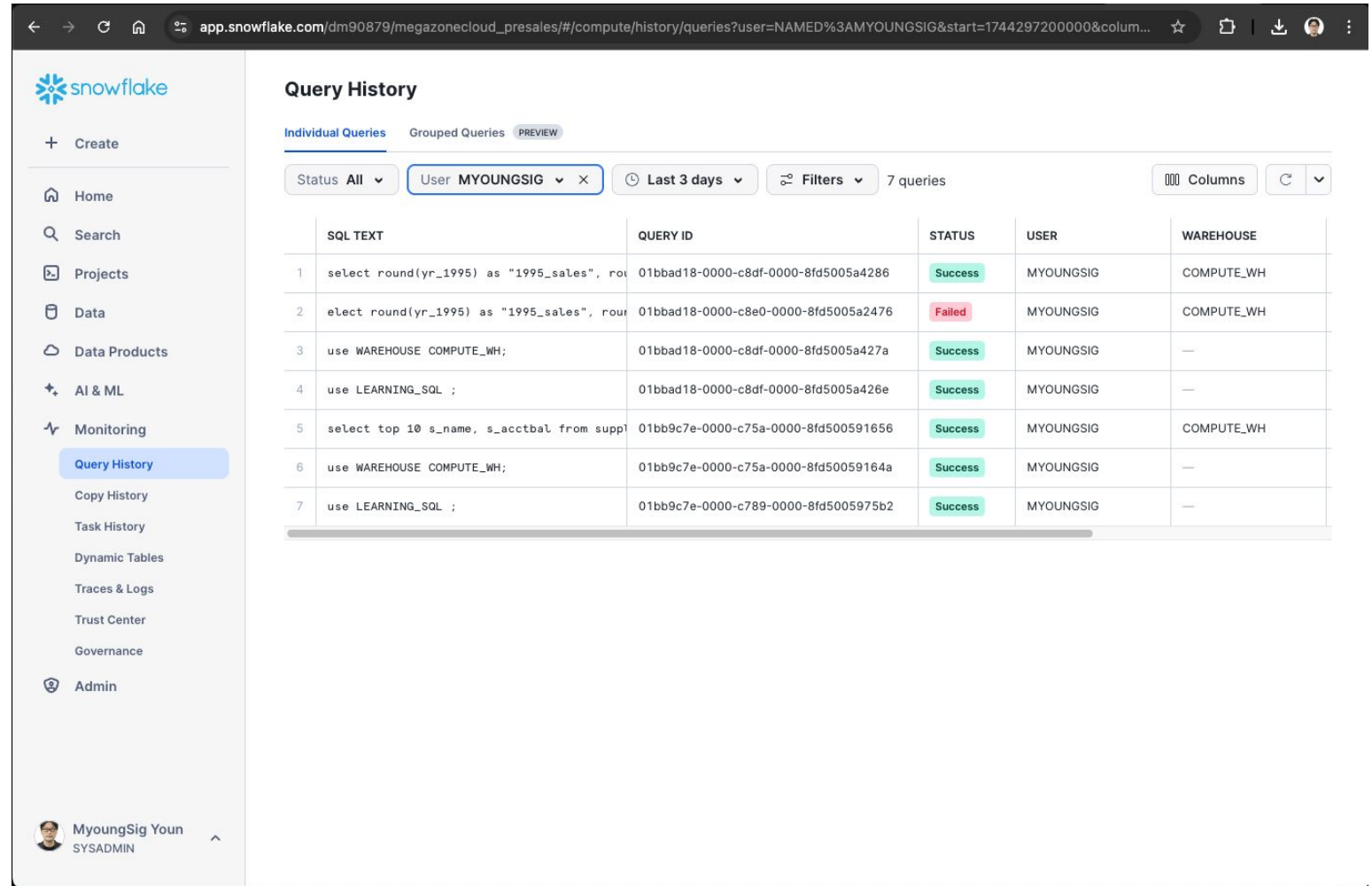
2. 스노우사이트

- Data 메뉴의 Add Data 화면



2. 스노우사이트

- Monitoring 메뉴의 Query History 화면



Query History

Individual Queries | Grouped Queries | PREVIEW

Status: All | User: MYOUNGSIG | Last 3 days | Filters | 7 queries | Columns

	SQL TEXT	QUERY ID	STATUS	USER	WAREHOUSE
1	select round(yr_1995) as "1995_sales", row	01bbad18-0000-c8df-0000-8fd5005a4286	Success	MYOUNGSIG	COMPUTE_WH
2	elect round(yr_1995) as "1995_sales", row	01bbad18-0000-c8e0-0000-8fd5005a2476	Failed	MYOUNGSIG	COMPUTE_WH
3	use WAREHOUSE COMPUTE_WH;	01bbad18-0000-c8df-0000-8fd5005a427a	Success	MYOUNGSIG	—
4	use LEARNING_SQL ;	01bbad18-0000-c8df-0000-8fd5005a426e	Success	MYOUNGSIG	—
5	select top 10 s_name, s_acctbal from suppl	01bb9c7e-0000-c75a-0000-8fd500591656	Success	MYOUNGSIG	COMPUTE_WH
6	use WAREHOUSE COMPUTE_WH;	01bb9c7e-0000-c75a-0000-8fd50059164a	Success	MYOUNGSIG	—
7	use LEARNING_SQL ;	01bb9c7e-0000-c789-0000-8fd5005975b2	Success	MYOUNGSIG	—

MyoungSig Youn
SYSADMIN

2. 스노우사이트

- Query History 화면의 Query Details 탭

The screenshot displays the Snowflake web interface. On the left is a navigation sidebar with the Snowflake logo at the top, followed by a 'Create' button and a list of menu items: Home, Search, Projects, Data, Data Products, AI & ML, Monitoring, Query History (highlighted in blue), Copy History, Task History, Dynamic Tables, Traces & Logs, Trust Center, Governance, and Admin. At the bottom of the sidebar is the user profile for 'MyoungSig Youn SYSADMIN'. The main content area shows the 'Query - 01bbad18-0000-c8df-0000-8fd5005a4286' page. It has tabs for 'Query Details' (selected), 'Query Profile', and 'Query Telemetry'. Below the tabs is a 'Details' section with a grid of query information:

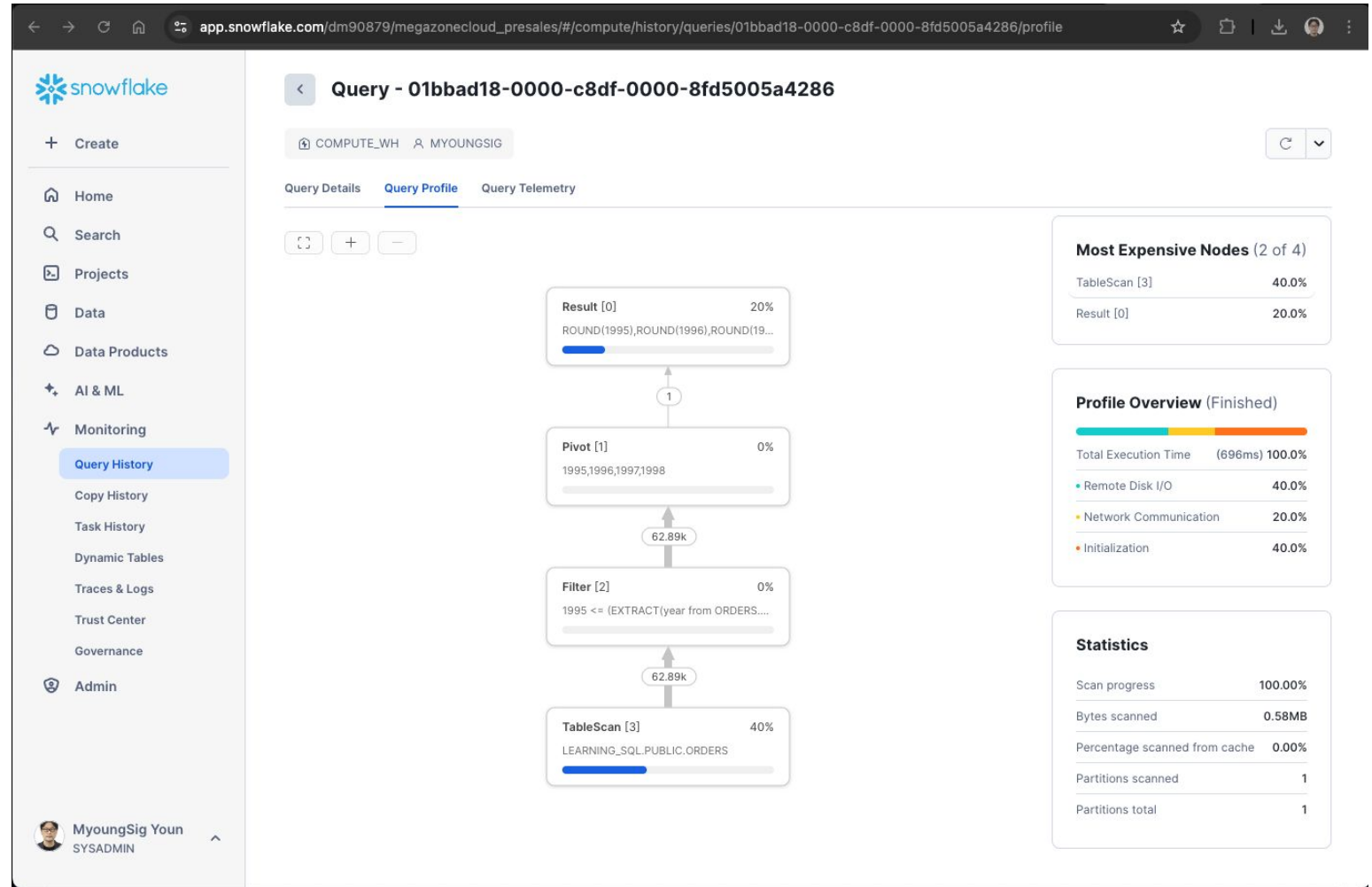
Details		
Status	Duration	Driver Status
Success	912ms	N/A
Start Time	Query ID	Client Driver
4/14/2025, 1:40:34 PM	01bbad18-0000-c8df-0000-8fd500...	SnowSQL 1.3.3
End Time	Query Tag	Session ID
4/14/2025, 1:40:35 PM	—	158144996544534
Warehouse Size	Attributed Compute Credits	Parameterized query hash
X-Small	—	0d56a5cd2436523d762ced1c44541...

Below the details is the 'SQL Text' section, which contains the following SQL query:

```
select round(yr_1995) as "1995_sales",
round(yr_1996) as "1996_sales",
round(yr_1997) as "1997_sales",
round(yr_1998) as "1998_sales"
from (select date_part(year, o_orderdate) as year,
o_totalprice
from orders
where 1995 <= date_part(year, o_orderdate)
)
```

2. 스노우사이트

- Query History 화면의 Query Profile 탭



3. SnowSQL

3. 스노우SQL

SnowSQL은 Snowflake에 연결하여 SQL 쿼리를 실행하고 데이터베이스 테이블의 데이터 로드 및 데이터 언로드 등 모든 DDL 및 DML 작업을 수행하기 위한 명령줄 클라이언트입니다.

<https://docs.snowflake.com/ko/user-guide/snowsql>

```
(base) [~] snowsql -c megazone
* SnowSQL * v1.3.3approval -
Type SQL statements or !help
MYOUNGSIG#MYOUNGSIG_WH@(no database).(no schema)>use LEARNING_SQL
;
+-----+
| status |
+-----+
| Statement executed successfully. |
+-----+
1 Row(s) produced. Time Elapsed: 0.085s
MYOUNGSIG#MYOUNGSIG_WH@LEARNING_SQL.PUBLIC>use WAREHOUSE COMPUTE_WH;
+-----+
| status |
+-----+
| Statement executed successfully. |
+-----+
1 Row(s) produced. Time Elapsed: 0.074s
MYOUNGSIG#COMPUTE_WH@LEARNING_SQL.PUBLIC>
MYOUNGSIG#COMPUTE_WH@LEARNING_SQL.PUBLIC>
```

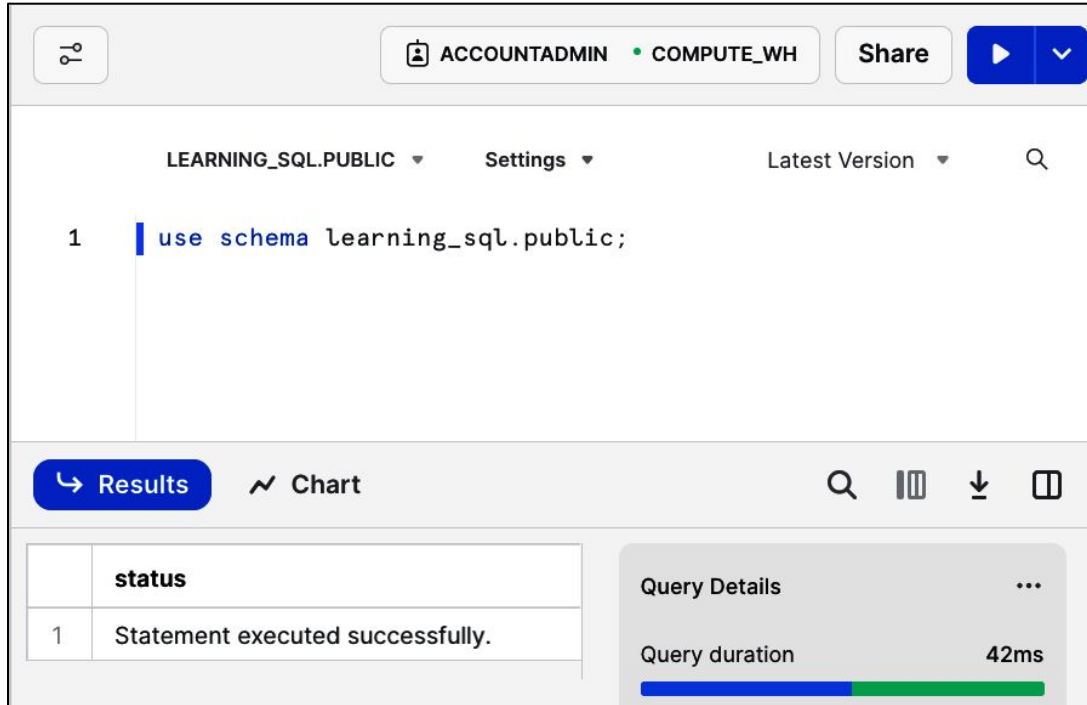
SQL 기초 교육

목차

1. 쿼리 입문
2. 필터링
3. 조인
4. 집합으로 작업하기
5. 데이터 생성과 수정
6. 데이터 생성, 변환, 조작
7. 그룹화와 집계
8. 서브 쿼리
9. 계층적 From 절
10. 조건 논리
11. 뷰
12. 윈도우 함수
13. 반구조화 데이터

1. 쿼리 입문

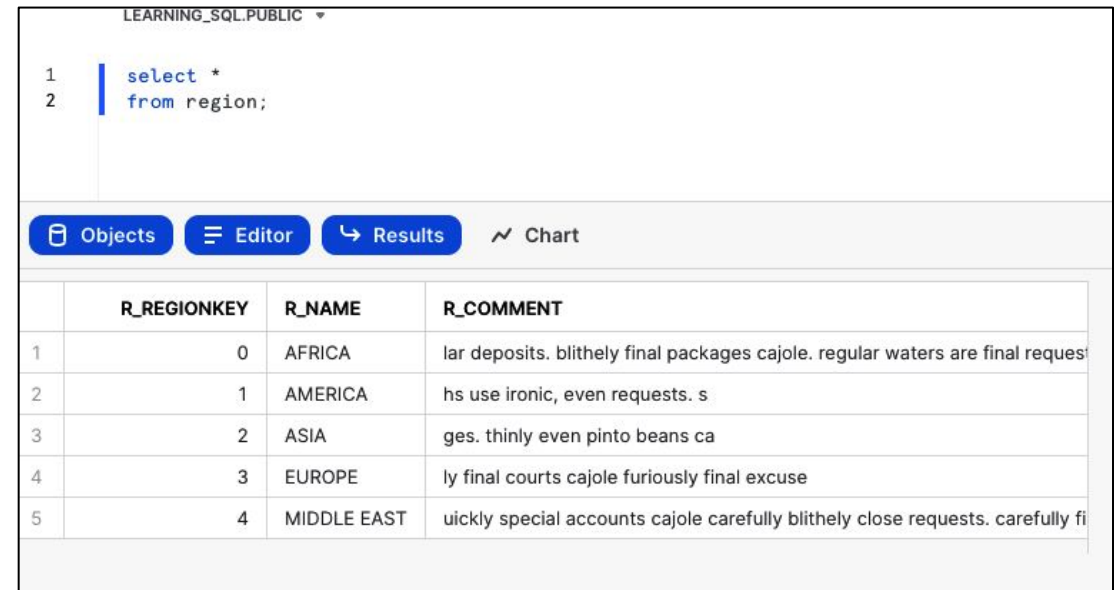
1.1. 쿼리 기본



The screenshot shows the Snowflake web interface. At the top, the user is logged in as ACCOUNTADMIN on the COMPUTE_WH warehouse. The database is set to LEARNING_SQL.PUBLIC. A query is entered in the editor: `use schema learning_sql.public;`. Below the editor, the 'Results' tab is selected, showing a single row with the status 'Statement executed successfully.' The 'Query Details' panel on the right indicates the query duration was 42ms.

스노우플레이크의 웹 인터페이스(스노우사이트)를 사용하여 스키마 설정하기

스노우사이트를 사용하여 Region 테이블 쿼리하기



The screenshot shows the Snowflake web interface with the query `select * from region;` executed. The 'Results' tab is selected, displaying a table with 5 rows and 3 columns: R_REGIONKEY, R_NAME, and R_COMMENT.

	R_REGIONKEY	R_NAME	R_COMMENT
1	0	AFRICA	lar deposits. blithely final packages cajole. regular waters are final request
2	1	AMERICA	hs use ironic, even requests. s
3	2	ASIA	ges. thinly even pinto beans ca
4	3	EUROPE	ly final courts cajole furiously final excuse
5	4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. carefully fi

1.1.1. 기본

다양한 쿼리 형태

```
SELECT CURRENT_DATE;
```

CURRENT_DATE
2025-02-20

```
SHOW TERSE TABLES IN public;
```

created_on	name	kind
2023-02-28 021.3826:55: -0800	CUSTOMER	TABLE
2023-02-28 06:51:41.226 -0800	LINEITEM	TABLE
2023-02-28 06:43:46.739 -0800	NATION	TABLE
2023-02-28 06:53:19.090 -0800	ORDERS	TABLE
2023-02-28 06:44:35.450 -0800	PART	TABLE
2023-02-28 06:45:20.267 -0800	PARTSUPP	TABLE
2023-02-28 06:42:32.322 -0800	REGION	TABLE
2023-02-28 06:49:39.242 -0800	SUPPLIER	TABLE

```
SELECT * FROM region;
```

R_REGIONKEY	R_NAME	R_COMMENT
0	AFRICA	lar deposits. blithely final pac...
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously...
4	MIDDLE EAST	uickly special accounts cajole c...

```
DESCRIBE TABLE region;
```

name	type	kind	null?	default
R_REGIONKEY	NUMBER(38,0)	COLUMN	N	NULL
R_NAME	VARCHAR(25)	COLUMN	N	NULL
R_COMMENT	VARCHAR(152)	COLUMN	Y	NULL

1.2. 쿼리 구문

구문 명	목적
select	결과 집합에 포함할 열을 지정합니다.
from	데이터를 검색할 테이블과 테이블을 조인하는 방법을 식별합니다.
where	결과 집합에서 원하지 않는 행을 제거합니다.
group by	공통 값으로 행을 그룹화 합니다.
having	그룹화를 기반으로 결과 집합에서 원하지 않는 행을 제거합니다.
qualify	윈도우 함수의 결과에 따라 결과 집합에서 원하지 않는 행을 제거합니다.
order by	하나 이상의 열을 기준으로 결과 집합을 정렬합니다.
limit	결과 집합의 행 수를 제한합니다.

표에 나열된 8개의 절 중에서 **select**만 필수이며, 일부는 함께 사용됩니다
(예: **group by** 절을 먼저 지정하지 않고는 **having** 절을 사용할 수 없습니다).

1.2.1. SELECT

SELECT 절은 쿼리에서 유일하게 필수적인 절이지만, 그 자체로는 특별히 유용하지 않습니다.

```
DESCRIBE TABLE nation;
```

name	type	kind	null?	default	primary key	unique key	check	expression	comment	policy name	privacy domain
N_NATIONKEY	NUMBER(38,0)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
N_NAME	VARCHAR(25)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
N_REGIONKEY	NUMBER(38,0)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
N_COMMENT	VARCHAR(152)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL

4 Row(s) produced. Time Elapsed: 0.057s

```
SELECT n_nationkey, n_name, n_regionkey  
FROM nation;
```

N_NATIONKEY	N_NAME	N_REGIONKEY
0	ALGERIA	0
1	ARGENTINA	1
2	BRAZIL	1
3	CANADA	1
4	EGYPT	4
5	ETHIOPIA	0
6	FRANCE	3
7	GERMANY	3

nation 테이블에는 네 개의 열이 있지만, 쿼리는 그 중 세 개만 검색합니다. 따라서 이 쿼리에서 **SELECT** 절의 목적은 가능한 모든 열 중에서 결과 집합에 포함해야 할 열을 지정하는 것입니다.

1.2.1. SELECT

데이터베이스 서버가 열 이름을 자동으로 할당하지만, 열 별칭을 사용하여 사용자 정의 이름을 지정할 수 있습니다. 이는 특히 SELECT 절에 포함된 리터럴, 표현식 또는 함수 호출에 유용합니다.

```
SELECT 'Welcome to Snowflake SQL!' AS welcome_message,  
       5 * 3.1415927 AS circle_circumference,  
       DAYNAME(current_date) AS day_of_week;
```

WELCOME_MESSAGE	CIRCLE_CIRCUMFERENCE	DAY_OF_WEEK
Welcome to Snowflake SQL!	15.7079635	Tue

1 Row(s) produced. Time Elapsed: 0.438s

1.2.1. SELECT

```
SELECT n_regionkey FROM nation;
```

N_REGIONKEY
0
1
1
1
4
0
3
3
2
2
4
4
2
4
0
0
0
1
2
3
4
2
...

25 Row(s) produced. Time Elapsed: 0.473s

```
SELECT DISTINCT n_regionkey FROM nation;
```

N_REGIONKEY
0
1
4
3
2

5 Row(s) produced. Time Elapsed: 0.476s

보시다시피 **nation** 테이블의 25개 행 전체에서 n_regionkey 에 대한 고유한 값은 5개뿐입니다. n_regionkey 열에서 고유한 값 집합만 검색하려면 **DISTINCT** 키워드를 사용할 수 있습니다.

1.2.2. FROM

앞부분에서는 단일 테이블에서 데이터를 검색하는 예시가 나왔지만, FROM 절은 여러 테이블을 참조할 수 있습니다. FROM 절에 두 개 이상의 테이블이 있는 경우, 그 역할은 단순히 테이블 목록을 나열하는 것뿐만 아니라 테이블을 연결하는 방법까지 포함하도록 확장됩니다. 예를 들어, nation 테이블에서 데이터를 검색하되, n_regionkey 열을 검색하는 대신 n_regionkey 값을 사용하여 검색하고 싶다고 가정해 봅시다.

```
SELECT n_nationkey, n_name AS nation_name,  
       r_name AS region_name  
FROM nation JOIN region  
      ON nation.n_regionkey = region.r_regionkey;
```

N_NATIONKEY	NATION_NAME	REGION_NAME
0	ALGERIA	AFRICA
1	ARGENTINA	AMERICA
2	BRAZIL	AMERICA
3	CANADA	AMERICA
4	EGYPT	MIDDLE EAST
5	ETHIOPIA	AFRICA
6	FRANCE	EUROPE
7	GERMANY	EUROPE
8	INDIA	ASIA
9	INDONESIA	ASIA
10	IRAN	MIDDLE EAST
11	IRAQ	MIDDLE EAST
12	JAPAN	ASIA
13	JORDAN	MIDDLE EAST

1.2.2. FROM

```
SELECT *  
FROM (  
    VALUES ('JAN', 1), ('FEB', 2),  
            ('MAR', 3), ('APR', 4),  
            ('MAY', 5), ('JUN', 6),  
            ('JUL', 7), ('AUG', 8),  
            ('SEP', 9), ('OCT', 10),  
            ('NOV', 11), ('DEC', 12)  
    ) AS months (month_name, month_num);
```

MONTH_NAME	MONTH_NUM
JAN	1
FEB	2
MAR	3
APR	4
MAY	5
JUN	6
JUL	7
AUG	8
SEP	9
OCT	10
NOV	11
DEC	12

테이블에서 데이터를 검색하는 것과 함께, 스노우플레이크는 FROM 절의 VALUES 하위 절을 사용하여 즉석에서 데이터를 생성할 수 있도록 합니다. 이는 테이블에서 찾을 수 없는 작은 데이터 세트를 생성하는 데 매우 유용할 수 있습니다.

1.2.3. WHERE

WHERE 절의 역할은 필터링, 즉 원치 않는 행을 제거하는 것입니다.

```
SELECT n_name
FROM nation
WHERE n_name LIKE 'U%';
```

N_NAME
UNITED KINGDOM
UNITED STATES

WHERE 절에는 여러 조건이 있을 수 있습니다.

```
SELECT n_name
FROM nation
WHERE n_name LIKE 'U%'
OR n_name LIKE 'A%';
```

N_NAME
ALGERIA
ARGENTINA
UNITED KINGDOM
UNITED STATES

1.2.4. GROUP BY

이름에서 알 수 있듯이 GROUP BY 절은 행을 그룹화하는 데 사용됩니다. 데이터 행을 그룹화하는 것은 특히 보고 및 데이터 분석에서 매우 일반적인 방법입니다. 다음 예제는 region 테이블의 각 행에 대해 nation 테이블의 국가 수를 계산합니다.

```
SELECT r_name AS region_name,  
       COUNT(*) AS number_of_countries  
FROM nation  
      JOIN region ON nation.n_regionkey = region.r_regionkey  
GROUP BY r_name;
```

REGION_NAME	NUMBER_OF_COUNTRIES
AFRICA	5
AMERICA	5
MIDDLE EAST	5
EUROPE	5
ASIA	5

1.2.5. HAVING

supplier 테이블에는 nation 테이블의 기본 키(n_nationkey)에 대한 링크인 s_nationkey 열이 포함되어 있습니다.

```
SELECT n_name AS nation_name,  
       COUNT(*) AS number_of_suppliers  
FROM supplier  
   JOIN nation ON supplier.s_nationkey = nation.n_nationkey  
GROUP BY n_name;
```

NATION_NAME	NUMBER_OF_SUPPLIERS
PERU	325
MOROCCO	265
UNITED KINGDOM	291
IRAN	306
UNITED STATES	295
CHINA	310
INDIA	301
CANADA	307
RUSSIA	296

동일한 쿼리에 WHERE 및 HAVING 절을 모두 사용할 수 있지만, WHERE 절의 조건은 행을 그룹화하기 전에 평가되는 반면, HAVING 절의 조건은 행이 그룹화된 후에 평가됩니다. 다음은 여러 필터가 있는 예입니다.

```
SELECT n_name AS nation_name,  
       COUNT(*) AS number_of_suppliers  
FROM supplier  
   JOIN nation ON supplier.s_nationkey = nation.n_nationkey  
WHERE n_name LIKE '%A'  
GROUP BY n_name  
HAVING COUNT(*) > 300;
```

NATION_NAME	NUMBER_OF_SUPPLIERS
CHINA	310
INDIA	301
CANADA	307
ALGERIA	318
ARGENTINA	312
SAUDI ARABIA	307

1.2.6. QUALIFY

필터링에 사용되는 세 번째 절은 QUALIFY 이지만, 이는 순위를 할당하는 등 여러 용도로 사용되는 윈도우 함수의 결과를 기반으로 행을 필터링하는 데만 사용되는 특수 목적 절입니다. 다음 예제에서는 내장된 LENGTH() 함수를 사용하여 이름의 문자 수를 기준으로 nation 테이블의 모든 행에 순위를 할당합니다.

```
SELECT n_name,  
       RANK() OVER (ORDER BY LENGTH(n_name) DESC) AS length_rank  
FROM nation;
```

N_NAME	LENGTH_RANK
UNITED KINGDOM	1
UNITED STATES	2
SAUDI ARABIA	3
MOZAMBIQUE	4
ARGENTINA	5
INDONESIA	5
ETHIOPIA	7
MOROCCO	8
ALGERIA	8
VIETNAM	8
ROMANIA	8

```
SELECT n_name,  
       RANK() OVER (ORDER BY LENGTH(n_name) DESC) AS length_rank  
FROM nation  
QUALIFY length_rank <= 5;
```

N_NAME	LENGTH_RANK
UNITED KINGDOM	1
UNITED STATES	2
SAUDI ARABIA	3
MOZAMBIQUE	4
ARGENTINA	5
INDONESIA	5

1.2.7. ORDER BY

일반적으로 쿼리에서 반환되는 결과 집합은 특정 순서로 정렬되지 않습니다. 알파벳순, 숫자순 또는 시간순과 같이 결과를 정렬하려면 쿼리 끝에 ORDER BY 절을 추가할 수 있습니다. ORDER BY 절은 SELECT 절의 요소 중 하나 이상을 포함할 수 있으며 이름 또는 위치로 각 요소를 참조할 수 있습니다.

```
SELECT s_name, s_acctbal
FROM supplier
ORDER BY s_acctbal DESC
LIMIT 10;
```

S_NAME	S_ACCTBAL
Supplier#000006343	9998.20
Supplier#000002522	9997.04
Supplier#000000892	9993.46
Supplier#000002543	9992.70
Supplier#000001833	9992.26
Supplier#000009966	9991.00
Supplier#000002892	9989.02
Supplier#000008875	9984.69
Supplier#000002331	9984.20
Supplier#000007895	9977.32

정렬에 사용되는 열은 SELECT 절의 위치로도 지정할 수 있으며, 이 경우 2가 됩니다.

```
SELECT s_name, s_acctbal
FROM supplier
ORDER BY 2 ASC
LIMIT 10;
```

S_NAME	S_ACCTBAL
Supplier#000009795	-998.22
Supplier#000007259	-997.61
Supplier#000008927	-995.53
Supplier#000005298	-990.16
Supplier#000001764	-990.13
Supplier#000008224	-989.86
Supplier#000001870	-989.05
Supplier#000001654	-988.37
Supplier#000001907	-987.45
Supplier#000003627	-986.14



1.2.8. LIMIT

- 여기서 LIMIT 절이 등장하며, 첫 번째 행 또는 지정된 오프셋에서 시작하여 반환할 행 수를 지정할 수 있습니다.
- 서버는 ORDER BY 절에 지정된 대로 7,400개의 행을 모두 정렬한 다음 처음 10개만 반환합니다. 선택적 OFFSET 하위 절을 사용하여 특정 행에서 시작하도록 서버에 지시할 수도 있습니다. supplier 테이블에 7,400개의 행이 있다는 것을 알고 있으므로 결과 집합의 마지막 10개 행을 보려면 7,390의 오프셋을 지정할 수 있습니다.

```
SELECT s_name, s_acctbal
FROM supplier
ORDER BY s_acctbal DESC
LIMIT 10 OFFSET 7390;
```

S_NAME	S_ACCTBAL
Supplier#000003627	-986.14
Supplier#000001907	-987.45
Supplier#000001654	-988.37
Supplier#000001870	-989.05
Supplier#000008224	-989.86
Supplier#000001764	-990.13
Supplier#000005298	-990.16
Supplier#000008927	-995.53
Supplier#000007259	-997.61
Supplier#000009795	-998.22

```
SELECT s_name, s_acctbal
FROM supplier
ORDER BY s_acctbal ASC
LIMIT 10 OFFSET 7390;
```

S_NAME	S_ACCTBAL
Supplier#000007895	9977.32
Supplier#000002331	9984.20
Supplier#000008875	9984.69
Supplier#000002892	9989.02
Supplier#000009966	9991.00
Supplier#000001833	9992.26
Supplier#000002543	9992.70
Supplier#000000892	9993.46
Supplier#000002522	9997.04
Supplier#000006343	9998.20

1.2.9. TOP

- LIMIT 절과 함께 스노우플레이크는 쿼리가 반환하는 행 수를 제한하기 위해 SELECT 절에 지정할 수 있는 TOP 키워드를 제공합니다. 계정 잔액을 기준으로 상위 10개 공급업체를 보려면 다음 쿼리를 사용할 수 있습니다.
- TOP 10을 사용하는 것은 ORDER BY 뒤에 LIMIT 10 절을 추가하는 것과 동일하지만, TOP 기능은 오프셋을 허용하지 않으므로 유연성이 떨어집니다.

```
SELECT TOP 10 s_name, s_acctbal
FROM supplier
ORDER BY s_acctbal DESC;
```

S_NAME	S_ACCTBAL
Supplier#000006343	9998.20
Supplier#000002522	9997.04
Supplier#000000892	9993.46
Supplier#000002543	9992.70
Supplier#000001833	9992.26
Supplier#000009966	9991.00
Supplier#000002892	9989.02
Supplier#000008875	9984.69
Supplier#000002331	9984.20
Supplier#000007895	9977.32

1.3. 복습하기

? 연습 1-1

nation 테이블에서 n_nationkey 및 n_name 열을 검색하는 쿼리를 작성하고 **region** 테이블에 조인(r_regionkey 열 사용)하여 아프리카 지역(r_name = 'AFRICA')에 속하는 선택 항목만 검색합니다.

? 연습 1-2

supplier 테이블에서 s_name 및 s_acctbal 열을 검색합니다. 내림차순으로 s_acctbal을 정렬하고 처음 10개의 행만 검색합니다(s_acctbal 값이 가장 높은 10개 공급업체가 될 것입니다).

1.3. 복습하기(정답)

100 연습 1-1

```
SELECT n_nationkey, n_name
FROM nation
JOIN region
    ON nation.n_regionkey = region.r_regionkey
WHERE region.r_name = 'AFRICA';
```

100 연습 1-2

```
SELECT s_name, s_acctbal
FROM supplier
ORDER BY s_acctbal DESC
LIMIT 10;
```

2. 필터링

2.1. 조건 평가

오래된 친구를 찾기 위해 디렉토리를 쿼리하고 다음과 같은 **where** 절을 사용한다고 가정해 보겠습니다.

- `WHERE last_name = 'SMITH' AND state = 'CA'`
- `WHERE last_name = 'SMITH' OR state = 'CA'`
- `WHERE (last_name = 'SMITH' OR last_name = 'JACKSON') AND (state = 'CA' OR state = 'WA')`
- `WHERE NOT ((last_name = 'SMITH' OR last_name = 'JACKSON') AND (state = 'CA' OR state = 'WA'))`
- `WHERE last_name <> 'SMITH' AND last_name <> 'JACKSON' AND state <> 'CA' AND state <> 'WA'`

2.2. 조건 구성 요소

1. 조건은 하나 이상의 연산자와 결합된 하나 이상의 표현식으로 구성됩니다. 표현식은 다음 중 하나일 수 있습니다.

- 테이블의 열
- 숫자 또는 날짜
- 'New York City'와 같은 문자열 리터럴
- CONCAT('Snowflake', ' Rules!')와 같은 내장 함수
- 하위 쿼리
- ('New York City', 'Dallas', 'Chicago')와 같은 표현식 목록

2. 조건 내에서 사용되는 연산자는 다음과 같습니다.

- =, <, >, !=, <>, LIKE, IN 및 BETWEEN과 같은 비교 연산자
- +, -, *, /와 같은 산술 연산자

2.2.1. 동등 조건

여러분이 접하게 될 조건의 대부분은 `column = expression` 형태일 것입니다. 예를 들어:

```
c_custkey = 12345
s_name = 'Acme Wholesale'
o_orderdate = to_date('02/14/2022', 'MM/DD/YYYY')
```

이러한 조건은 하나의 표현식을 다른 표현식과 같게 만들기 때문에 동등 조건이라고 합니다. 동등 조건은 쿼리의 `where` 절에서 필터링하는 데 일반적으로 사용되지만, `from` 절에서도 흔히 볼 수 있으며, 이 경우 조인 조건이라고 합니다. 다음은 `from` 및 `where` 절 모두에 등가 조건이 있는 예입니다.

```
SELECT n_name, r_name
FROM nation JOIN region ON nation.n_regionkey = region.r_regionkey
WHERE r_name = 'ASIA';
```

N_NAME	R_NAME
INDIA	ASIA
INDONESIA	ASIA
JAPAN	ASIA
CHINA	ASIA
VIETNAM	ASIA

2.2.2. 부등식 조건

또 다른 일반적인 조건 유형은 두 표현식이 같지 않을 때 참으로 평가되는 부등식 조건입니다. 다음은 where 절에서 부등식 조건을 사용하여 이전 예제가 어떻게 보이는지 보여줍니다.

```
SELECT n_name, r_name
FROM nation JOIN region
      ON nation.n_regionkey = region.r_regionkey
WHERE r_name <> 'ASIA'
LIMIT 10;
```

N_NAME	R_NAME
ALGERIA	AFRICA
ARGENTINA	AMERICA
BRAZIL	AMERICA
CANADA	AMERICA
EGYPT	MIDDLE EAST
ETHIOPIA	AFRICA
FRANCE	EUROPE
GERMANY	EUROPE
IRAN	MIDDLE EAST
IRAQ	MIDDLE EAST

```
SELECT n_name, r_name
FROM nation JOIN region
      ON nation.n_regionkey = region.r_regionkey
WHERE r_name != 'ASIA'
LIMIT 10;
```

N_NAME	R_NAME
ALGERIA	AFRICA
ARGENTINA	AMERICA
BRAZIL	AMERICA
CANADA	AMERICA
EGYPT	MIDDLE EAST
ETHIOPIA	AFRICA
FRANCE	EUROPE
GERMANY	EUROPE
IRAN	MIDDLE EAST
IRAQ	MIDDLE EAST

2.2.3. 범위 조건

경우에 따라 표현식이 지정된 범위 내에 속하는지 확인해야 합니다. 범위 조건이라고 하는 이러한 조건은 일반적으로 숫자 또는 날짜 열과 함께 사용되며 between 연산자를 활용합니다.

```
SELECT s_suppkey, s_name
FROM supplier
WHERE s_suppkey BETWEEN 1 AND 10;
```

S_SUPPKEY	S_NAME
1	Supplier#000000001
4	Supplier#000000004
7	Supplier#000000007
9	Supplier#000000009
10	Supplier#000000010

```
SELECT o_orderkey, o_custkey, o_orderdate
FROM orders
WHERE o_orderdate BETWEEN
      TO_DATE('29-JAN-1998', 'DD-MON-YYYY')
      AND TO_DATE('30-JAN-1998', 'DD-MON-YYYY')
LIMIT 5;
```

O_ORDERKEY	O_CUSTKEY	O_ORDERDATE
5412320	88709	1998-01-29
3604290	10393	1998-01-30
3605158	136969	1998-01-29
1221764	95848	1998-01-29
3035040	111920	1998-01-29

2.2.3. 범위 조건

드물긴 하지만 문자 데이터에 사용되는 범위 조건도 볼 수 있습니다. 다음 쿼리는 이름이 'GA'에서 'IP' 범위에 속하는 모든 국가를 반환합니다.

```
SELECT n_name
FROM nation
WHERE n_name BETWEEN 'GA' AND 'IP';
```

N_NAME
GERMANY
INDIA
INDONESIA

```
SELECT n_name
FROM nation
WHERE n_name BETWEEN 'GA' AND 'IS';
```

N_NAME
GERMANY
INDIA
INDONESIA
IRAN
IRAQ

2.2.4. 회원 조건

경우에 따라, 특정 열의 값이 주어진 여러 값 중 하나와 일치하는 행을 검색해야 할 때가 있습니다. 예를 들어, Customer 테이블에는 'AUTOMOBILE', 'MACHINERY', 'BUILDING', 'HOUSEHOLD', 'FURNITURE' 같은 값들을 포함하는 c_mktsegment 열이 있습니다. 만약 처음 세 개의 시장 세그먼트('AUTOMOBILE', 'MACHINERY', 'BUILDING')에 속하는 고객 정보를 검색하고 싶다면, OR 연산자로 연결된 세 개의 개별 조건을 사용할 수 있습니다.

- `WHERE c_mktsegment = 'AUTOMOBILE' OR c_mktsegment = 'MACHINERY' OR c_mktsegment = 'BUILDING'`
- `WHERE c_mktsegment IN ('AUTOMOBILE', 'MACHINERY', 'BUILDING')`

2.2.5. 매칭 조건

마지막 조건 유형은 부분 문자열 일치에 관한 것입니다. 문자열이 특정 글자로 시작하거나 문자열 내 어느 위치에도 특정 글자들을 포함하는 행을 반환해야 하는 경우가 많습니다. 이런 유형의 조건을 매칭 조건이라고 하며, LIKE 연산자를 사용합니다.

```
SELECT n_name
FROM nation
WHERE n_name LIKE 'M%';
```

N_NAME
MOROCCO
MOZAMBIQUE

```
SELECT n_name
FROM nation
WHERE n_name LIKE 'M%'
      OR n_name LIKE 'U%';
```

N_NAME
MOROCCO
MOZAMBIQUE
UNITED KINGDOM
UNITED STATES

```
SELECT n_name
FROM nation
WHERE regexp_like(n_name, '^[MU].*');
```

N_NAME
MOROCCO
MOZAMBIQUE
UNITED KINGDOM
UNITED STATES

2.2.5. 매칭 조건

- 와일드카드 문자

와일드 카드	매치
%	임의의 문자 수(0,1,...,N)
_	정확히 1개의 문자

- 검색 표현식

검색 표현식	해석
'I__'	I로 시작하는 정확히 4자 길이
'%E'	길이에 상관없이, 끝이 E로 끝남
'__A%'	길이 3자 이상, 세 번째 위치는 A입니다.
'%ND%'	모든 길이, 문자열의 어느 위치에서나 하위 문자열 ND 포함

2.3. NULL 값

데이터베이스에 데이터를 삽입할 때, 특정 열에 값을 제공할 수 없는 여러 상황이 있을 수 있습니다. 여기에는 다음과 같은 몇 가지 경우가 포함될 수 있습니다.

1. 행(row)이 처음 생성될 때는 값이 알려지지 않았고 나중에 제공될 수 있는 경우. (예: 신규 직원의 퇴사일)
2. 해당 열이 특정 행에는 적용되지 않는 경우. (예: 전자적으로 전달되는 전자책(ebook)의 배송 업체 정보)

데이터가 없거나 알 수 없는 경우, 관계형 데이터베이스는 해당 열에 NULL 값을 할당합니다. 필터링 조건을 만들 때는 해당 열이 NULL 값을 허용하는지(이는 테이블 생성 시 지정됩니다) 인지하고, 결과 집합에 NULL 값을 가진 행을 포함시킬지 제외시킬지 고려해야 합니다. 이때 명심해야 할 두 가지 기본 규칙이 있습니다.

1. 표현식은 NULL일 수 있지만, NULL과 동일 비교(=)될 수는 없습니다.
2. 두 NULL 값은 서로 동일하지 않습니다.

```
SELECT 'YES' AS is_valid WHERE NULL = NULL;
```

IS_VALID

```
SELECT 'YES' AS is_valid WHERE NULL IS NULL;
```

IS_VALID
YES

2.3.1 Null 값 예제

```
CREATE TABLE null_example (  
  num_col NUMBER, char_col VARCHAR(10)  
) AS  
SELECT *  
FROM (  
  VALUES (1, 'ABC'), (2, 'JKL'),  
  (NULL, 'QRS'), (3, NULL));
```

```
+-----+  
| status  
+-----+  
| Table NULL_EXAMPLE successfully created.  
+-----+  
1 Row(s) produced. Time Elapsed: 1.094s
```



```
SELECT * FROM null_example;
```

```
+-----+  
| NUM_COL | CHAR_COL |  
+-----+  
| 1 | ABC  
| 2 | JKL  
| NULL | QRS  
| 3 | NULL  
+-----+  
4 Row(s) produced. Time Elapsed: 0.559s
```



```
SELECT num_col, char_col  
FROM null_example  
WHERE num_col < 3  
OR num_col IS NULL;
```

```
+-----+  
| NUM_COL | CHAR_COL |  
+-----+  
| 1 | ABC  
| 2 | JKL  
| NULL | QRS  
+-----+  
3 Row(s) produced. Time Elapsed: 0.435s
```



```
SELECT num_col, char_col  
FROM null_example  
WHERE num_col < 3;
```

```
+-----+  
| NUM_COL | CHAR_COL |  
+-----+  
| 1 | ABC  
| 2 | JKL  
+-----+  
2 Row(s) produced. Time Elapsed: 0.679s
```

2.3.1 Null 값 예제

NULL 값 처리는 필터 조건을 작성할 때 흔히 발생하는 문제이므로, 스노우플레이크를 포함한 모든 주요 데이터베이스는 NULL 값을 다루기 위한 여러 내장 함수를 제공합니다. 그러한 함수 중 하나로 `nvl()`이 있으며, 이 함수는 발견되는 모든 NULL 값을 지정된 값으로 대체하는 데 사용될 수 있습니다.

```
SELECT num_col, char_col
FROM null_example
WHERE NVL(num_col, 0) < 3;
```

NUM_COL	CHAR_COL
1	ABC
2	JKL
NULL	QRS

```
SELECT NVL(num_col, 0) AS num_col, char_col
FROM null_example
WHERE NVL(num_col, 0) < 3;
```

NUM_COL	CHAR_COL
1	ABC
2	JKL
0	QRS

3 Row(s) produced. Time Elapsed: 0.622s

2.5. 복습 하기

? 연습 2-1

customer 테이블에서 c_name, c_mktsegment 및 c_acctbal 열을 가져옵니다. 단, 시장 세그먼트가 Machinery 이고 계정 잔액이 20이거나 시장 세그먼트가 Furniture 이고 계정 잔액이 334인 행에만 해당합니다.

? 연습 2-2

다음 데이터가 있는 balances 라는 테이블이 주어집니다:

acct_num	acct_bal
1234	342.22
3498	9.00
3887	(null)
6277	28.33

acct_bal 열의 값이 9가 아닌 모든 행을 검색하는 쿼리를 작성합니다.
열의 값이 9가 아닌 모든 행을 검색하는 쿼리를 작성합니다.

2.5. 복습 하기(정답)

100 연습 2-1

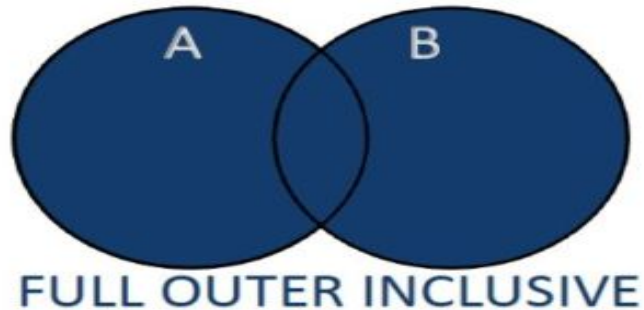
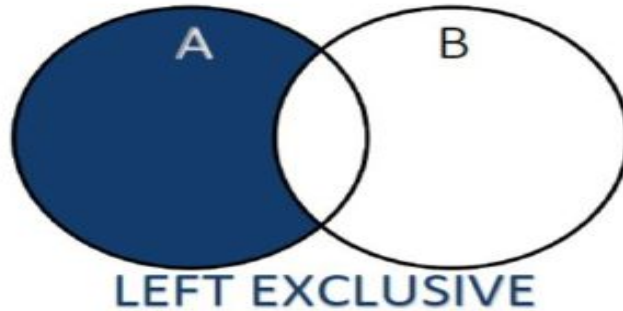
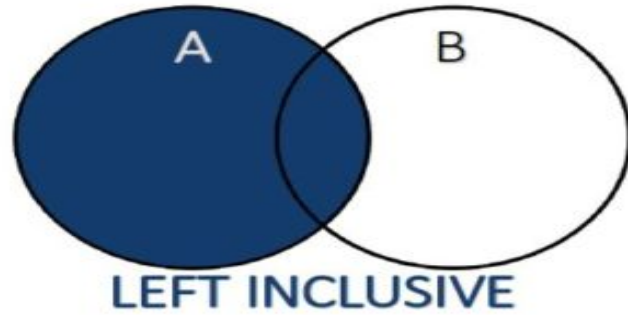
```
SELECT c_name, c_mktsegment, c_acctbal
FROM customer
WHERE (c_mktsegment = 'MACHINERY' AND c_acctbal = 20)
      OR (c_mktsegment = 'FURNITURE' AND c_acctbal = 334);
```

100 연습 2-2

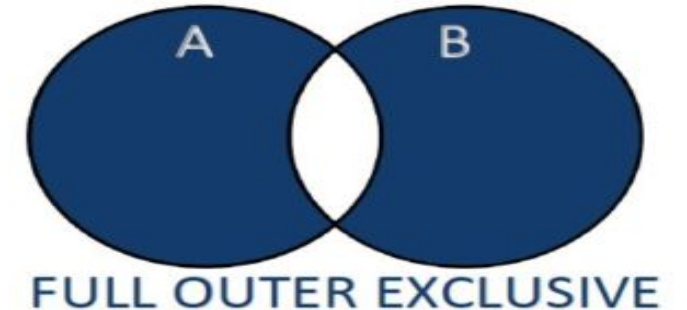
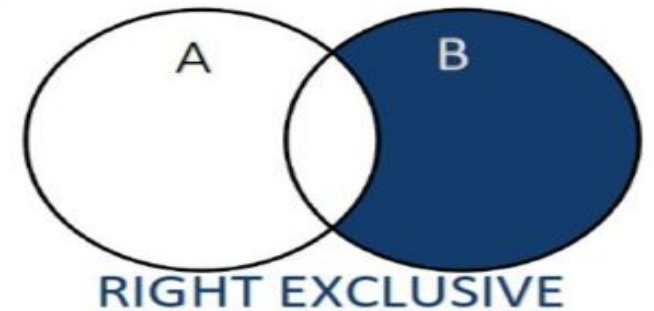
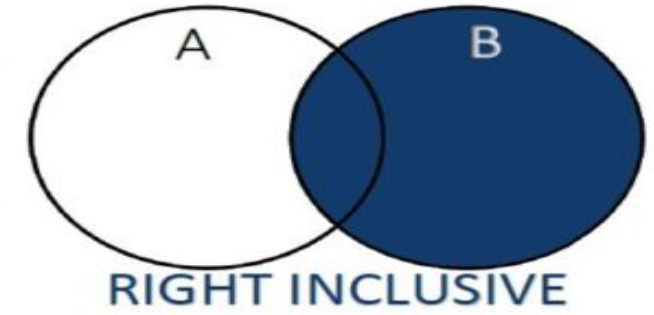
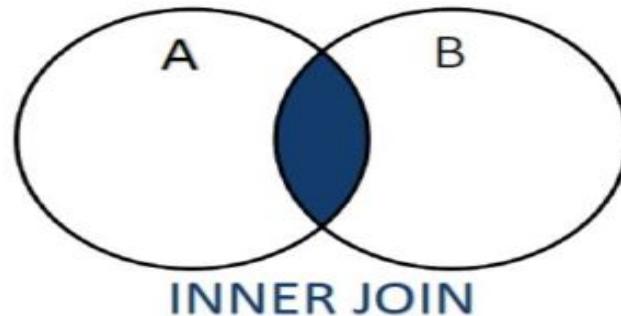
```
SELECT acct_num, acct_bal
FROM balances
WHERE acct_bal IS NULL OR acct_bal != 9;
```

3. 조인

3.1. 조인이란?



SQL JOINS	
LEFT INCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key	RIGHT INCLUSIVE SELECT [Select List] FROM TableA A RIGHT OUTER JOIN TableB B ON A.Key= B.Key
LEFT EXCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key WHERE B.Key IS NULL	RIGHT EXCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key WHERE A.Key IS NULL
FULL OUTER INCLUSIVE SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key	FULL OUTER EXCLUSIVE SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL OR B.Key IS NULL
INNER JOIN SELECT [Select List] FROM TableA A INNER JOIN TableB B ON A.Key = B.Key	



3.1. 조인이란?

보고서에는 각 주문의 주문 날짜, 주문 우선순위, 그리고 해당 주문을 한 고객의 이름 정보가 포함되어야 합니다. 따라서 이 보고서에 필요한 데이터를 검색하려면, orders 테이블과 customer 테이블 모두의 데이터를 가져올 수 있도록 이 두 테이블을 조인하는 쿼리를 작성해야 합니다.

DESCRIBE TABLE customer;

name	type	kind	null?	default	primary key	unique key	check	expression	comment	policy name	privacy domain
C_CUSTKEY	NUMBER(38,0)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
C_NAME	VARCHAR(25)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
C_ADDRESS	VARCHAR(40)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
C_NATIONKEY	NUMBER(38,0)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
C_PHONE	VARCHAR(15)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
C_ACCTBAL	NUMBER(12,2)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
C_MKTSEGMENT	VARCHAR(10)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
C_COMMENT	VARCHAR(117)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL

DESCRIBE TABLE orders;

name	type	kind	null?	default	primary key	unique key	check	expression	comment	policy name	privacy domain
O_ORDERKEY	NUMBER(38,0)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
O_CUSTKEY	NUMBER(38,0)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
O_ORDERSTATUS	VARCHAR(1)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
O_TOTALPRICE	NUMBER(12,2)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
O_ORDERDATE	DATE	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
O_ORDERPRIORITY	VARCHAR(15)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
O_CLERK	VARCHAR(15)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
O_SHIPPRIORITY	NUMBER(38,0)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
O_COMMENT	VARCHAR(79)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL

3.1.1. 조인 쿼리

customer 테이블의 모든 행은 c_custkey 열에 저장된 고유한 숫자 식별자를 가지고 있으며, orders 테이블의 모든 행에는 고객의 고유 식별자를 담고 있는 o_custkey 열이 있습니다. 따라서 customer.c_custkey와 orders.o_custkey가 두 테이블을 조인하는 데 사용될 열입니다.

```
SELECT o_orderkey, o_orderstatus, o_orderdate, c_name
FROM orders JOIN customer
      ON orders.o_custkey = customer.c_custkey
LIMIT 10;
```

O_ORDERKEY	O_ORDERSTATUS	O_ORDERDATE	C_NAME
600006	O	1996-09-16	Customer#000083098
600037	F	1994-06-26	Customer#000107722
600064	O	1997-11-04	Customer#000089008
600065	F	1993-09-15	Customer#000146441
600132	O	1998-01-08	Customer#000131644
600165	F	1992-12-08	Customer#000139328
600228	F	1992-03-01	Customer#000046379
600262	O	1997-03-03	Customer#000011323
600327	O	1997-02-25	Customer#000133094
600484	O	1997-08-11	Customer#000120598

10 Row(s) produced. Time Elapsed: 1.071s

3.2. 테이블 별칭

FROM 절에 있는 테이블에 별칭을 지정할 수도 있는데, 이는 쿼리의 가독성을 높이는 데 유용하며 특정 경우에는 필수적이기도 합니다. 이전 섹션의 쿼리를 가져와 orders 테이블에는 o 라는 별칭을, customer 테이블에는 c 라는 별칭을 추가해 보겠습니다.

```
SELECT o.o_orderkey, o.o_orderstatus, o.o_orderdate, c.c_name
FROM orders AS o
JOIN customer AS c
  ON o.o_custkey = c.c_custkey
LIMIT 10;
```

O_ORDERKEY	O_ORDERSTATUS	O_ORDERDATE	C_NAME
600006	O	1996-09-16	Customer#000083098
600037	F	1994-06-26	Customer#000107722
600064	O	1997-11-04	Customer#000089008
600065	F	1993-09-15	Customer#000146441
600132	O	1998-01-08	Customer#000131644
600165	F	1992-12-08	Customer#000139328
600228	F	1992-03-01	Customer#000046379
600262	O	1997-03-03	Customer#000011323
600327	O	1997-02-25	Customer#000133094
600484	O	1997-08-11	Customer#000120598

10 Row(s) produced. Time Elapsed: 1.160s

3.3. Inner 조인

```
CREATE TABLE customer_simple (custkey, custname)
AS SELECT *
FROM (VALUES (101, 'BOB'), (102, 'KIM'), (103, 'JIM'));
```

```
+-----+
| status |
+-----+
| Table CUSTOMER_SIMPLE successfully created. |
+-----+
1 Row(s) produced. Time Elapsed: 1.058s
```



```
SELECT * FROM customer_simple;
```

```
+-----+-----+
| CUSTKEY | CUSTNAME |
+-----+-----+
|      101 | BOB      |
|      102 | KIM      |
|      103 | JIM      |
+-----+-----+
3 Row(s) produced. Time Elapsed: 0.199s
```

```
CREATE TABLE orders_simple (ordernum, custkey)
AS SELECT *
FROM (VALUES (990, 101), (991, 102),
            (992, 101), (993, 104));
```

```
+-----+
| status |
+-----+
| Table ORDERS_SIMPLE successfully created. |
+-----+
1 Row(s) produced. Time Elapsed: 0.545s
```



```
SELECT * FROM orders_simple;
```

```
+-----+-----+
| ORDERNUM | CUSTKEY |
+-----+-----+
|      990 |      101 |
|      991 |      102 |
|      992 |      101 |
|      993 |      104 |
+-----+-----+
4 Row(s) produced. Time Elapsed: 0.592s
```


3.3.1. Inner 조인 쿼리

데이터를 살펴보면, `customer_simple` 테이블에는 `custkey` 값이 각각 101, 102, 103인 세 개의 행이 있습니다. `orders_simple` 테이블에는 네 개의 행이 있으며, 그중 세 개는 `custkey` 101과 102를 참조합니다. 하지만 나머지 한 행은 `custkey` 104를 참조하는데, 이 값은 `customer_simple` 테이블에는 존재하지 않습니다.

```
SELECT o.ordernum, o.custkey, c.custname
FROM orders_simple AS o
      INNER JOIN customer_simple AS c
      ON o.custkey = c.custkey;
```

ORDERNUM	CUSTKEY	CUSTNAME
990	101	BOB
991	102	KIM
992	101	BOB

3 Row(s) produced. Time Elapsed: 0.684s

```
EXPLAIN
SELECT o.ordernum, o.custkey, c.custname
FROM orders_simple AS o
      INNER JOIN customer_simple AS c
      ON o.custkey = c.custkey;
```


step	id	parentOperators	operation	objects	alias	expressions	partitionsTotal	partitionsAssigned	bytesAssigned
	NULL	NULL	GlobalStats	NULL	NULL	NULL	2	2	2048
1	0	NULL	Result	NULL	NULL	O.ORDERNUM, O.CUSTKEY, C.CUSTNAME	NULL	NULL	NULL
1	1	[0]	InnerJoin	NULL	NULL	joinKey: (C.CUSTKEY = O.CUSTKEY)	NULL	NULL	NULL
1	2	[1]	TableScan	LEARNING_SQL.PUBLIC.CUSTOMER_SIMPLE	C	CUSTKEY, CUSTNAME	1	1	1024
1	3	[1]	JoinFilter	NULL	NULL	joinKey: (C.CUSTKEY = O.CUSTKEY)	NULL	NULL	NULL
1	4	[3]	TableScan	LEARNING_SQL.PUBLIC.ORDERS_SIMPLE	O	ORDERNUM, CUSTKEY	1	1	1024

6 Row(s) produced. Time Elapsed: 0.161s

3.4. Outer 조인

다음으로 가장 흔한 조인 유형은 외부 조인(outer join)입니다. 이 조인은 B 테이블과의 조인 성공 여부에 관계없이 A 테이블의 모든 행을 반환합니다.

```
SELECT o.ordernum,  
       o.custkey, c.custname  
FROM orders_simple AS o  
     LEFT OUTER JOIN customer_simple AS c  
     ON o.custkey = c.custkey;
```




ORDERNUM	CUSTKEY	CUSTNAME
990	101	BOB
991	102	KIM
992	101	BOB
993	104	NULL

4 Row(s) produced. Time Elapsed: 0.593s

- 결과 집합의 첫 세 행은 내부 조인예시와 정확히 동일하게 보이지만, 네 번째 행에는 이제 customer_simple 테이블의 추가 행 (custkey 104, custname JIM)이 포함됩니다. 이때 두 개의 orders_simple 테이블 열 값은 NULL입니다.

- 결과 집합에는 orders_simple 테이블의 네 번째 행(ordernum 993)이 포함됩니다. 하지만 custname 열은 NULL인데, 이는 customer_simple 테이블에 custkey 104에 해당하는 행이 없기 때문입니다.
- 이 조인은 왼쪽 외부 조인으로 지정되었는데, 이는 조인 구문의 왼쪽에 있는 테이블(orders_simple)의 모든 행이 결과에 포함되어야 함을 의미합니다.

```
SELECT o.ordernum,  
       o.custkey, c.custname  
FROM orders_simple AS o  
     RIGHT OUTER JOIN customer_simple AS c  
     ON o.custkey = c.custkey;
```



ORDERNUM	CUSTKEY	CUSTNAME
990	101	BOB
991	102	KIM
992	101	BOB
NULL	NULL	JIM

4 Row(s) produced. Time Elapsed: 0.137s

3.5. Cross 조인

마지막 조인 유형은 교차 조인입니다. 이는 약간 부적절한 이름일 수 있는데, 실제로는 (조건에 따른) 조인이 발생하지 않기 때문입니다. 대신 두 테이블이 서로 병합되는데, 이는 두 테이블의 카티션 곱으로 알려져 있습니다. 예를 들어, 50개 행을 가진 테이블과 150개 행을 가진 테이블을 교차 조인하면, 결과 집합에는 7,500개의 행(50×150)이 포함될 것입니다.

```
SELECT years.yearnum, qtrs.qtrname, qtrs.startmonth, qtrs.endmonth
FROM (VALUES (2020), (2021), (2022)) AS years (yearnum)
     CROSS JOIN (
       VALUES ('Q1',1,3), ('Q2',4,6),
              ('Q3',7,9), ('Q4',10,12)
     ) AS qtrs (qtrname, startmonth, endmonth)
ORDER BY 1,2;
```

YEARNUM	QTRNAME	STARTMONTH	ENDMONTH
2020	Q1	1	3
2020	Q2	4	6
2020	Q3	7	9
2020	Q4	10	12
2021	Q1	1	3
2021	Q2	4	6
2021	Q3	7	9
2021	Q4	10	12
2022	Q1	1	3
2022	Q2	4	6
2022	Q3	7	9
2022	Q4	10	12

12 Row(s) produced. Time Elapsed: 0.666s

3.6. 3개 이상의 테이블 조인

DESCRIBE TABLE lineitem;

name	type	kind	null?	default	...
L_ORDERKEY	NUMBER(38,0)	COLUMN	Y	NULL	...
L_PARTKEY	NUMBER(38,0)	COLUMN	Y	NULL	...
L_SUPPKEY	NUMBER(38,0)	COLUMN	Y	NULL	...
L_LINENUMBER	NUMBER(38,0)	COLUMN	Y	NULL	...
L_QUANTITY	NUMBER(12,2)	COLUMN	Y	NULL	...
L_EXTENDEDPRICE	NUMBER(12,2)	COLUMN	Y	NULL	...
L_DISCOUNT	NUMBER(12,2)	COLUMN	Y	NULL	...
L_TAX	NUMBER(12,2)	COLUMN	Y	NULL	...
L_RETURNFLAG	VARCHAR(1)	COLUMN	Y	NULL	...
L_LINESTATUS	VARCHAR(1)	COLUMN	Y	NULL	...
L_SHIPDATE	DATE	COLUMN	Y	NULL	...
L_COMMITDATE	DATE	COLUMN	Y	NULL	...
L_RECEIPTDATE	DATE	COLUMN	Y	NULL	...
L_SHIPINSTRUCT	VARCHAR(25)	COLUMN	Y	NULL	...
L_SHIPMODE	VARCHAR(10)	COLUMN	Y	NULL	...
L_COMMENT	VARCHAR(44)	COLUMN	Y	NULL	...

DESCRIBE TABLE part;

name	type	kind	null?	default	...
P_PARTKEY	NUMBER(38,0)	COLUMN	Y	NULL	...
P_NAME	VARCHAR(55)	COLUMN	Y	NULL	...
P_MFGR	VARCHAR(25)	COLUMN	Y	NULL	...
P_BRAND	VARCHAR(10)	COLUMN	Y	NULL	...
P_TYPE	VARCHAR(25)	COLUMN	Y	NULL	...
P_SIZE	NUMBER(38,0)	COLUMN	Y	NULL	...
P_CONTAINER	VARCHAR(10)	COLUMN	Y	NULL	...
P_RETAILPRICE	NUMBER(12,2)	COLUMN	Y	NULL	...
P_COMMENT	VARCHAR(23)	COLUMN	Y	NULL	...

3.6. 3개 이상의 테이블 조인

이 쿼리는 상당히 복잡해 보이지만, 새로운 개념이 도입된 것은 아닙니다. 이는 단지 두 테이블 조인을 네 개의 테이블을 포함하도록 확장한 것뿐입니다. 각각의 조인은 적절한 조인 조건을 정의하기 위해 자체적인 ON 하위 절을 가집니다.

```
SELECT o.o_orderkey, o.o_orderdate,  
       c.c_name, p.p_name  
FROM orders AS o  
      INNER JOIN customer AS c ON o.o_custkey = c.c_custkey  
      INNER JOIN lineitem AS l ON o.o_orderkey = l.l_orderkey  
      INNER JOIN part AS p ON l.l_partkey = p.p_partkey  
LIMIT 10;
```

O_ORDERKEY	O_ORDERDATE	C_NAME	P_NAME
4200066	1995-06-02	Customer#000007693	indian blue chiffon slate lawn
4200068	1994-10-29	Customer#000055922	tan lavender chocolate orange burlywood
4200128	1992-02-03	Customer#000097453	brown burlywood indian peach forest
4200228	1992-06-04	Customer#000021313	azure gainsboro tomato green cornflower
4200295	1992-02-12	Customer#000119251	midnight hot antique sandy cornflower
4200354	1996-08-16	Customer#000081373	frosted grey rosy dark brown
4200358	1992-06-02	Customer#000024823	beige ivory pink cornsilk linen
4200385	1994-11-25	Customer#000112585	purple azure burlywood chocolate metallic
4200390	1993-06-16	Customer#000048991	deep blanchd yellow ivory misty
4200423	1998-05-10	Customer#000014165	gainsboro bisque orchid sandy cyan

10 Row(s) produced. Time Elapsed: 1.928s

3.6. 3개 이상의 테이블 조인

EXPLAIN

```
SELECT o.o_orderkey, o.o_orderdate,  
       c.c_name, p.p_name  
FROM orders AS o  
      INNER JOIN customer AS c ON o.o_custkey = c.c_custkey  
      INNER JOIN lineitem AS l ON o.o_orderkey = l.l_orderkey  
      INNER JOIN part AS p ON l.l_partkey = p.p_partkey  
LIMIT 10;
```

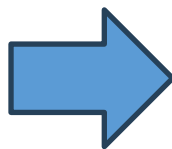
step	id	parentOperators	operation	objects	alias	expressions	partitionsTotal	partitionsAssigned	bytesAssigned
NULL	NULL	NULL	GlobalStats	NULL	NULL	NULL	4	4	11476992
1	0	NULL	Result	NULL	NULL	O.O_ORDERKEY, O.O_ORDERDATE, C.C_NAME, P.P_NAME	NULL	NULL	NULL
1	1	[0]	Limit	NULL	NULL	rowCount: 10	NULL	NULL	NULL
1	2	[1]	InnerJoin	NULL	NULL	joinKey: (P.P_PARTKEY = L.L_PARTKEY)	NULL	NULL	NULL
1	3	[2]	TableScan	LEARNING_SQL.PUBLIC.PART	P	P_PARTKEY, P_NAME	1	1	147456
1	4	[2]	InnerJoin	NULL	NULL	joinKey: (O.O_ORDERKEY = L.L_ORDERKEY)	NULL	NULL	NULL
1	5	[4]	InnerJoin	NULL	NULL	joinKey: (C.C_CUSTKEY = O.O_CUSTKEY)	NULL	NULL	NULL
1	6	[5]	TableScan	LEARNING_SQL.PUBLIC.CUSTOMER	C	C_CUSTKEY, C_NAME	1	1	4756992
1	7	[5]	JoinFilter	NULL	NULL	joinKey: (C.C_CUSTKEY = O.O_CUSTKEY)	NULL	NULL	NULL
1	8	[7]	TableScan	LEARNING_SQL.PUBLIC.ORDERS	O	O_ORDERKEY, O_CUSTKEY, O_ORDERDATE	1	1	3776000
1	9	[4]	JoinFilter	NULL	NULL	joinKey: (P.P_PARTKEY = L.L_PARTKEY)	NULL	NULL	NULL
1	10	[9]	TableScan	LEARNING_SQL.PUBLIC.LINEITEM	L	L_ORDERKEY, L_PARTKEY	1	1	2796544

12 Row(s) produced. Time Elapsed: 0.720s

3.6.1. Self 조인하기

```
CREATE TABLE employee_sample(  
    empid NUMBER,  
    emp_name VARCHAR(30),  
    mgr_empid NUMBER  
) AS  
SELECT *  
FROM (VALUES(1001, 'Bob Smith', null),  
    (1002, 'Susan Jackson', 1001),  
    (1003, 'Greg Carpenter', 1001),  
    (1004, 'Robert Butler', 1002),  
    (1005, 'Kim Josephs', 1003),  
    (1006, 'John Tyler', 1004)  
);
```

```
+-----+  
| status  
+-----+  
| Table EMPLOYEE_SAMPLE successfully created.  
+-----+  
1 Row(s) produced. Time Elapsed: 0.860s
```



```
SELECT *  
FROM employee_sample;
```

```
+-----+-----+-----+  
| EMPID | EMP_NAME      | MGR_EMPID |  
+-----+-----+-----+  
| 1001 | Bob Smith     | NULL      |  
| 1002 | Susan Jackson | 1001      |  
| 1003 | Greg Carpenter | 1001      |  
| 1004 | Robert Butler  | 1002      |  
| 1005 | Kim Josephs    | 1003      |  
| 1006 | John Tyler     | 1004      |  
+-----+-----+-----+  
6 Row(s) produced. Time Elapsed: 0.089s
```

3.6.1. Self 조인하기

```
SELECT e.empid, e.emp_name,  
       mgr.emp_name AS mgr_name  
FROM employee_sample AS e  
     INNER JOIN employee_sample AS mgr  
           ON e.mgr_empid = mgr.empid;
```

EMPID	EMP_NAME	MGR_NAME
1002	Susan Jackson	Bob Smith
1003	Greg Carpenter	Bob Smith
1004	Robert Butler	Susan Jackson
1005	Kim Josephs	Greg Carpenter
1006	John Tyler	Robert Butler

5 Row(s) produced. Time Elapsed: 2.051s

- Bob Smith는 mgr_empid 열에 NULL 값을 가지고 있습니다. 이는 Bob이 회사에서 최상위 관리자임을 의미합니다.
- Bob을 결과에 포함시키려면, 이전 쿼리를 외부 조인을 사용하도록 수정해야 합니다.

←

각 직원의 이름 및 ID와 함께 해당 직원의 관리자 이름을 반환하는 쿼리를 작성해 봅시다. 이를 위해서는 Employee 테이블을 자기 자신과 조인해야 합니다. 이때 mgr_empid 열(관리자 ID)을 사용하여 관리자의 empid 열(직원 ID)과 연결합니다.

```
SELECT e.empid, e.emp_name,  
       mgr.emp_name AS mgr_name  
FROM employee_sample e  
     LEFT OUTER JOIN employee_sample mgr  
           ON e.mgr_empid = mgr.empid
```

EMPID	EMP_NAME	MGR_NAME
1002	Susan Jackson	Bob Smith
1003	Greg Carpenter	Bob Smith
1004	Robert Butler	Susan Jackson
1005	Kim Josephs	Greg Carpenter
1006	John Tyler	Robert Butler
1001	Bob Smith	NULL

6 Row(s) produced. Time Elapsed: 0.720s

3.6.2. 같은 테이블 두 번 조인하기

```
ALTER TABLE employee_sample ADD COLUMN birth_nationkey INTEGER;
```

```
+-----+
| status |
+-----+
| Statement executed successfully. |
+-----+
1 Row(s) produced. Time Elapsed: 0.209s
```

```
ALTER TABLE employee_sample ADD COLUMN current_nationkey INTEGER;
```

```
+-----+
| status |
+-----+
| Statement executed successfully. |
+-----+
1 Row(s) produced. Time Elapsed: 0.200s
```

```
DESCRIBE employee_sample;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| name          | type          | kind  | null? | default | primary key | unique key | check | expression | comment | policy name | privacy domain |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| EMPID         | NUMBER(38,0)  | COLUMN | Y      | NULL    | N           | N           | NULL  | NULL       | NULL    | NULL       | NULL           |
| EMP_NAME      | VARCHAR(30)   | COLUMN | Y      | NULL    | N           | N           | NULL  | NULL       | NULL    | NULL       | NULL           |
| MGR_EMPID     | NUMBER(38,0)  | COLUMN | Y      | NULL    | N           | N           | NULL  | NULL       | NULL    | NULL       | NULL           |
| BIRTH_NATIONKEY | NUMBER(38,0)  | COLUMN | Y      | NULL    | N           | N           | NULL  | NULL       | NULL    | NULL       | NULL           |
| CURRENT_NATIONKEY | NUMBER(38,0)  | COLUMN | Y      | NULL    | N           | N           | NULL  | NULL       | NULL    | NULL       | NULL           |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
5 Row(s) produced. Time Elapsed: 0.103s
```

3.6.2. 같은 테이블 두 번 조인하기

```
UPDATE employee_sample
SET birth_nationkey = empid - 1000,
    current_nationkey = empid - 999;
```

number of rows updated	number of multi-joined rows updated
6	0

```
SELECT empid, emp_name, birth_nationkey, current_nationkey
FROM employee_sample;
```

EMPID	EMP_NAME	BIRTH_NATIONKEY	CURRENT_NATIONKEY
1001	Bob Smith	1	2
1002	Susan Jackson	2	3
1003	Greg Carpenter	3	4
1004	Robert Butler	4	5
1005	Kim Josephs	5	6
1006	John Tyler	6	7

6 Row(s) produced. Time Elapsed: 0.683s

birth_nation 값은 테이블 별칭이 n1인 Nation 테이블과의 첫 번째 조인에서 가져오며, current_nation 값은 별칭이 n2인 Nation 테이블과의 두 번째 조인에서 가져옵니다.

```
SELECT e.empid, e.emp_name,
       n1.n_name AS birth_nation,
       n2.n_name AS current_nation
FROM employee_sample e
     INNER JOIN nation AS n1
     ON e.birth_nationkey = n1.n_nationkey
     INNER JOIN nation AS n2
     ON e.current_nationkey = n2.n_nationkey;
```

EMPID	EMP_NAME	BIRTH_NATION	CURRENT_NATION
1001	Bob Smith	ARGENTINA	BRAZIL
1002	Susan Jackson	BRAZIL	CANADA
1003	Greg Carpenter	CANADA	EGYPT
1004	Robert Butler	EGYPT	ETHIOPIA
1005	Kim Josephs	ETHIOPIA	FRANCE
1006	John Tyler	FRANCE	GERMANY

6 Row(s) produced. Time Elapsed: 0.482s

3.7. 복습 하기

? 연습 3-1

소유자/반려동물 연습을 확장하여 반려동물이 소유자를 0명, 1명 또는 2명까지 가질 수 있다고 가정해 보겠습니다.

pet_owner

	# OWNER_ID	<u>A</u> OWNER_NAME
1	1	John
2	2	Cindy
3	3	Laura
4	4	Mark

pet

	# PET_ID	# OWNER_ID1	# OWNER_ID2	<u>A</u> PET_NAME
1	101	1	null	Fluffy
2	102	3	2	Spot
3	103	4	1	Rover
4	104	null	null	Rosco

각 반려동물의 **이름**과 **소유자 #1**과 **소유자 #2**의 이름을 반환합니다. 결과 집합에는 각 반려동물에 대해 하나의 행이 있어야 합니다. (총 4개) 일부 소유자 이름은 null 입니다.

3.7. 복습 하기(정답)

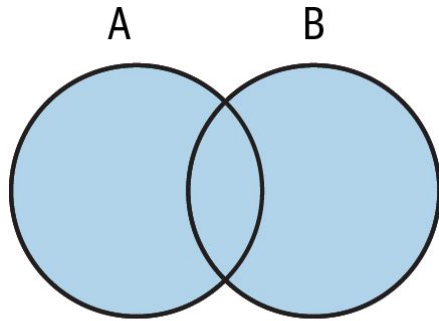
100 연습 3-1

```
SELECT p.pet_name,  
       po1.owner_name AS owner_1,  
       po2.owner_name AS owner_2  
FROM pet AS p  
     LEFT OUTER JOIN pet_owner AS po1  
         ON po1.owner_id = p.owner_id1  
     LEFT OUTER JOIN pet_owner AS po2  
         ON po2.owner_id = p.owner_id2;
```

	<u>A</u> PET_NAME	<u>A</u> OWNER_1	<u>A</u> OWNER_2
1	Fluffy	John	null
2	Spot	Laura	Cindy
3	Rover	Mark	John
4	Rosco	null	null

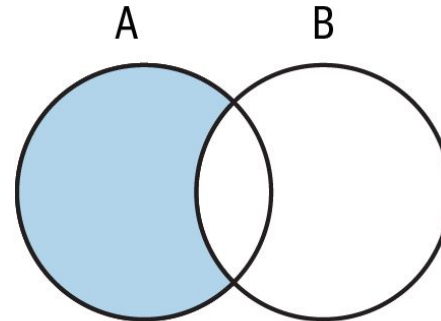
4. 집합으로 작업하기

4.1. 집합 이란



□ A union B

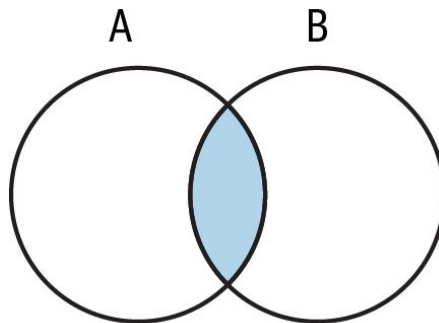
$A = \{1, 2, 4, 7, 9\}$
 $B = \{3, 5, 7, 9\}$
 $A \cup B = \{1, 2, 3, 4, 5, 7, 9\}$



□ A except B

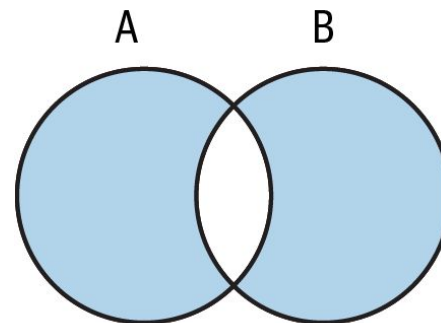
$A = \{1, 2, 4, 7, 9\}$
 $B = \{3, 5, 7, 9\}$
 $A \text{ except } B = \{1, 2, 4\}$

$A = \{1, 2, 4, 7, 9\}$
 $B = \{3, 5, 7, 9\}$
 $B \text{ except } A = \{3, 5\}$



□ A intersect B

$A = \{1, 2, 4, 7, 9\}$
 $B = \{3, 5, 7, 9\}$
 $A \cap B = \{7, 9\}$



□ ???

$(A \cup B) \text{ except } (A \cap B)$
-- or --
 $(A \text{ except } B) \cup (B \text{ except } A)$

4.2. UNION 연산자

가장 간단한 경우로, 중복되는 값 없이 두 개의 단일 행 결과 집합이 결합하여 두 행짜리 결과 집합을 만드는 예입니다. 집합 연산자로 구분된 두 개의 독립적인 쿼리를 포함하기 때문에, 이런 유형의 문장을 복합 쿼리 라고 부릅니다.

```
SELECT 1 AS numeric_col, 'ABC' AS string_col
UNION
SELECT 2 AS numeric_col, 'XYZ' AS string_col;
```

NUMERIC_COL	STRING_COL
1	ABC
2	XYZ

2 Row(s) produced. Time Elapsed: 0.498s

결합되는 두 집합(쿼리 결과)과 관련된 몇 가지 제한 사항을 알아보겠습니다.

1. 두 집합(쿼리 결과)은 반드시 동일한 개수의 열을 가져야 합니다.
2. 각 열의 데이터 타입이 순서대로 일치해야 합니다.

```
SELECT 1 AS numeric_col, 'ABC' AS string_col
UNION
SELECT 'XYZ' AS numeric_col, 2 AS string_col;
100038 (22018): Numeric value 'ABC' is not recognized
```

```
SELECT 1 AS numeric_col, 'ABC' AS string_col
UNION
SELECT 2 AS numeric_col, 'XYZ' AS string_col, 99 AS extra_col;
001789 (42601): SQL compilation error:
invalid number of result columns for set operator input branches, expected 2, got 3 in branch 2
```

4.2.1. UNION과 UNION ALL

두 집합 모두 7과 9를 포함하지만 이 값들은 결과 집합에는 한 번만 나타납니다. 이는 UNION 연산자가 값들을 정렬하고 중복을 제거하기 때문입니다. 이것이 기본 동작 방식이지만, 중복이 제거되기를 원하지 않는 경우도 있습니다. 이런 경우에는 UNION ALL을 사용할 수 있습니다.

```
SELECT integer_val
FROM (VALUES (1), (2), (4), (7), (9))
) AS set_a (integer_val)
```

UNION

```
SELECT integer_val
FROM (VALUES (3), (5), (7), (9))
) AS set_b (integer_val);
```

INTEGER_VAL
1
2
4
7
9
3
5

7 Row(s) produced. Time Elapsed: 0.225s

```
SELECT integer_val
FROM (VALUES (1), (2), (4), (7), (9))
) AS set_a (integer_val)
```

UNION ALL

```
SELECT integer_val
FROM (VALUES (3), (5), (7), (9))
) AS set_b (integer_val);
```

INTEGER_VAL
1
2
4
7
9
3
5
7
9

9 Row(s) produced. Time Elapsed: 0.678s

4.3. INTERSECT 연산자

집합 A와 B의 중복되는 부분(교집합)은 값 7과 9이며, 다른 모든 값들은 각 집합에만 고유하게 존재합니다.

```
SELECT integer_val
FROM (
    VALUES (1), (2), (4), (7), (9)
) AS set_a (integer_val)
INTERSECT
SELECT integer_val
FROM (
    VALUES (3), (5), (7), (9)
) AS set_b (integer_val);
```

INTEGER_VAL
7
9

2 Row(s) produced. Time Elapsed: 0.225s

4.4. EXCEPT 연산자

마지막으로 다룰 세 번째 집합 연산자는 EXCEPT이며, 이는 집합 B에 존재하는 행들을 제외하고 집합 A의 행들만을 반환하는 데 사용됩니다.

```
SELECT integer_val
FROM (
    VALUES (1), (2), (4), (7), (9)
) AS set_a (integer_val)
EXCEPT
SELECT integer_val
FROM (
    VALUES (3), (5), (7), (9)
) AS set_b (integer_val);
```

INTEGER_VAL
1
2
4

3 Row(s) produced. Time Elapsed: 0.225s

```
SELECT integer_val
FROM (
    VALUES (3), (5), (7), (9)
) AS set_b (integer_val)
EXCEPT
SELECT integer_val
FROM (
    VALUES (1), (2), (4), (7), (9)
) AS set_a (integer_val);
```

INTEGER_VAL
3
5

2 Row(s) produced. Time Elapsed: 0.225s

4.5. 집합 연산 규칙

집합 연산자를 사용할 때 결과 집합을 정렬하고 싶다면, 다음 규칙들을 따라야 합니다.

1. 단 하나의 ORDER BY 절만 허용되며, 반드시 전체 복합 쿼리 문장의 맨 마지막에 와야 합니다.
2. ORDER BY 절에서 참조하는 열 이름이나 별칭은 반드시 첫 번째 SELECT 문의 열 이름 또는 별칭을 사용해야 합니다.

```
SELECT DISTINCT o_orderdate FROM orders
INTERSECT
SELECT DISTINCT l_shipdate FROM lineitem
ORDER BY o_orderdate
LIMIT 10;
```

O_ORDERDATE
1992-01-03
1992-01-04
1992-01-05
1992-01-06
1992-01-07
1992-01-08
1992-01-09
1992-01-10
1992-01-11
1992-01-12

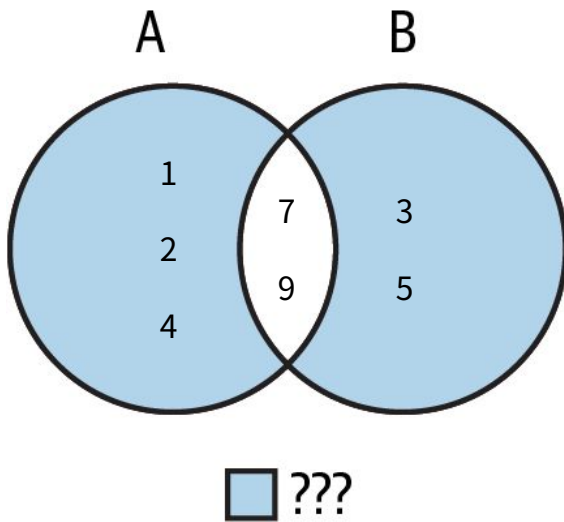
10 Row(s) produced. Time Elapsed: 0.544s

```
SELECT DISTINCT o_orderdate FROM orders
INTERSECT
SELECT DISTINCT l_shipdate FROM lineitem
ORDER BY l_shipdate
LIMIT 10;
```

000904 (42000): SQL compilation error: error line 4 at position 9
invalid identifier 'L_SHIPDATE'

4.5.1. 작업 우선순위 설정

세 개 이상의 집합 연산자를 사용하여 복합 쿼리를 만들 때는, 원하는 결과를 얻기 위해 여러 연산들을 어떻게 그룹화(묶어야) 할 필요가 있을지 반드시 고려해야 합니다.



```
(  
    SELECT integer_val  
    FROM (VALUES (1), (2), (4), (7), (9)) AS set_a (integer_val)  
    EXCEPT  
    SELECT integer_val  
    FROM (VALUES (3), (5), (7), (9)) AS set_b (integer_val)  
) UNION (  
    SELECT integer_val  
    FROM (VALUES (3), (5), (7), (9)) AS set_b (integer_val)  
    EXCEPT  
    SELECT integer_val  
    FROM (VALUES (1), (2), (4), (7), (9)) AS set_a (integer_val)  
)
```

INTEGER_VAL
1
2
4
5
3

5 Row(s) produced. Time Elapsed: 0.444s

4.6. 복습 하기

? 연습 4-1

다음 세 집합이 주어졌을 때:

$A = \{3, 5, 7, 9\}$

$B = \{4, 5, 6, 7, 8\}$

$C = \{8, 9, 10\}$

다음 연산을 수행하면 어떤 집합이 반환됩니까?

$(A \text{ except } B) \text{ intersect } C$

4.6. 복습 하기(정답)

100 연습 4-1

$A = \{3, 5, 7, 9\}$

$B = \{4, 5, 6, 7, 8\}$

$C = \{8, 9, 10\}$

$(A \text{ except } B) \text{ intersect } C \rightarrow \{9\}$

5. 데이터 생성과 수정

5.1. 데이터 종류

이전에서는 주로 select 문을 다루었습니다. 이제는 방향을 바꿔 테이블 생성과 함께 insert, update, delete 및 merge를 포함한 나머지 SQL 데이터 문을 살펴보겠습니다.

<https://docs.snowflake.com/ko/sql-reference-data-types>

5.1.1. 문자형

favorite_movie varchar(100) : Max Size 16MB

1. char, character, char varying
 2. nchar, nchar varying
 3. nvarchar, nvarchar2
 4. string, text
- 일반적으로 CHAR 데이터 유형은 고정 길이 문자열을 저장하는 데 사용되며, 문자열 길이가 최대 길이보다 짧은 경우 공백으로 채워집니다. 예를 들어, 다른 데이터베이스 시스템에서 CHAR(5)로 정의된 열에 'cat' 을 저장하면 'cat ' 처럼 공백이 추가됩니다. 하지만 스노우플레이크에서는 CHAR(5)로 정의된 열에 'cat' 을 저장하더라도 'cat' 으로 저장됩니다.
 - 스노우플레이크에서 CHAR를 사용할 때는 길이를 명시적으로 지정하는 것이 좋습니다. 길이를 지정하지 않으면 기본 길이가 1로 설정되어 예상치 못한 결과가 발생할 수 있습니다. 예를 들어, 길이를 지정하지 않고 CHAR로 열을 정의하면 'apple' 과 같은 문자열은 'a' 로 저장됩니다.
 - 데이터는 유니코드 UTF-8 문자 세트를 사용하여 저장되므로 단일 varchar 열에 저장할 수 있는 문자 수의 상한은 단일 바이트 문자를 저장하는지 아니면 멀티 바이트 문자를 저장하는지에 따라 다릅니다.

5.1.1. 문자형

- 문자 타입 열에 값을 제공할 때는, 작은따옴표(')를 구분자로 사용해야 합니다.

```
SELECT 'here is a string' AS output_string;
```

```
+-----+  
| OUTPUT_STRING |  
+-----+  
| here is a string |  
+-----+
```

1 Row(s) produced. Time Elapsed: 0.092s

- 문자열 안에 작은따옴표가 포함되어야 한다면, 작은따옴표 두 개를 연속('')으로 사용하여 스노우플레이크에 이것이 문자열의 끝이 아님을 알려줄 수 있습니다.

```
SELECT 'you haven''t reached the end yet' AS output_string;
```

```
+-----+  
| OUTPUT_STRING |  
+-----+  
| you haven't reached the end yet |  
+-----+
```

1 Row(s) produced. Time Elapsed: 0.231s

- 복잡한 문자열을 다룰 때, 두 개의 달러 기호(\$\$)를 구분자로 사용할 수 있습니다. 이 방식을 사용하면 스노우플레이크는 \$\$ 사이의 문자열을 작성된 내용 그대로 인식합니다.

```
SELECT $$string with 4 single quotes '$'$ AS output_string;
```

```
+-----+  
| OUTPUT_STRING |  
+-----+  
| string with 4 single quotes '$'$ |  
+-----+
```

1 Row(s) produced. Time Elapsed: 0.106s

5.1.2. 숫자형

전체 자릿수를 정밀도라고 하며, (소수 부분이 있는 경우) 소수점 오른쪽의 최대 자릿수를 스케일이라고 합니다. 전체 자릿수는 38을 초과할 수 없으며, 숫자 열을 정의할 때 정밀도와 스케일을 지정하지 않으면 기본값은 NUMBER(38,0)이 됩니다. 예를 들어, -999.99에서 999.99 사이의 숫자와 같이 특정 형식의 데이터를 저장하는 경우, 해당 열을 NUMBER(5,2)로 정의할 수 있습니다. 숫자 데이터에 필요한 저장 공간은 실제 자릿수에 따라 가변적이므로, 모든 숫자 열을 NUMBER(38,0)으로 정의한다고 해서 반드시 낭비적인 것은 아닙니다.

- 0
- -1
- 99,999,999,999,999,999,999,999,999,999,999,999
- 0.0000000000000000000000000000000000000001
- 1,234,567.123456789012345678901234567890

새 열을 정의할 때는, 정수를 위해서는 INTEGER 타입을, 부동 소수점 숫자를 위해서는 NUMBER 타입을 사용하는 것이 좋을 수 있습니다.

1. decimal, numeric, real
2. tinyint, smallint, int, integer, bigint
3. double, float, float4, float8

5.1.3. 시간형

Data type	Allowable range
date	1582-01-01 to 9999-12-31
time	00:00:00 to 23:59:59.999999999
timestamp	1582-01-01 00:00:00 to 9999-12-31 23:59:59.999999999

- timestamp_ntz : 특정 시간대 없음
- timestamp_ltz : 현재 세션의 표준 시간대 사용
- timestamp_tz : 시간대를 지정할 수 있음

```
SHOW PARAMETERS LIKE 'timez%';
```

key	value	default	level	description	type
TIMEZONE	Asia/Seoul	America/Los_Angeles	ACCOUNT	time zone	STRING

1 Row(s) produced. Time Elapsed: 0.236s

```
ALTER SESSION SET TIMEZONE='Asia/Seoul';
```

status
Statement executed successfully.

1 Row(s) produced. Time Elapsed: 0.102s



```
SHOW PARAMETERS LIKE 'timez%';
```

key	value	default	level	description	type
TIMEZONE	Asia/Seoul	America/Los_Angeles	SESSION	time zone	STRING

1 Row(s) produced. Time Elapsed: 0.113s

5.1.3. 시간형

```
SELECT CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP;
```

CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP
2025-03-17	15:33:23	2025-03-17 15:33:23.867 +0900

- 타임스탬프 값 끝의 +0900 표시는 저의 시간대가 그리니치 평균시(GMT)보다 9시간 빠르다는 것을 의미합니다. 해당 시간대가 현재 서버타임을 적용 중인지 여부에 따라 이 오프셋 값은 한 시간 달라질 수 있다는 점을 유의하세요.

```
SHOW PARAMETERS LIKE 'date_output%';
```

key	value	default	level	description	type
DATE_OUTPUT_FORMAT	YYYY-MM-DD	YYYY-MM-DD		display format for date	STRING

1 Row(s) produced. Time Elapsed: 1.605s

```
ALTER SESSION SET DATE_OUTPUT_FORMAT='DD/MM/YYYY';
```

status
Statement executed successfully.

1 Row(s) produced. Time Elapsed: 0.149s

```
SELECT CURRENT_DATE
```

CURRENT_DATE
03/17/2025

1 Row(s) produced. Time Elapsed: 0.068s

5.1.4. 기타 데이터형

불리언 데이터 타입은 논리 데이터 타입으로도 알려져 있으며, true와 false 값을 가질 수 있습니다. 스노우플레이크는 불리언 타입의 열이나 변수에 값을 할당할 때 상당히 유연하며, 다음 중 어떤 값이라도 허용합니다.

Boolean

- Strings 'true', '1', 'yes', 't', 'y', 'on' for true
- Strings 'false', '0', 'no', 'f', 'n', 'off' for false
- Number 0 for false
- Any non-zero number for true

```
SELECT true, false, true = true, true = false;
```

```
+-----+-----+-----+-----+
| TRUE | FALSE | TRUE = TRUE | TRUE = FALSE |
|-----+-----+-----+-----+
| True | False | True         | False         |
+-----+-----+-----+-----+
1 Row(s) produced. Time Elapsed: 0.082s
```

5.1.4. 기타 데이터형

VARIANT는 데이터 타입계의 '스위스 아미 나이프(만능 칼)' 와 같아서, 어떤 타입의 데이터든 담을 수 있습니다. 이는 반정형 데이터를 저장할 때 상당한 유연성을 제공합니다. VARIANT 타입의 열에 값을 삽입할 때는, 다음 예시처럼 :: 연산자를 사용하여 문자열, 숫자, 날짜 등을 VARIANT 타입으로 캐스팅 할 수 있습니다.

```
SELECT 1::VARIANT, 'abc'::VARIANT, current_date::VARIANT;
```

1::VARIANT	'ABC'::VARIANT	CURRENT_DATE::VARIANT
1	"abc"	"03/17/2025"

1 Row(s) produced. Time Elapsed: 1.097s

```
SELECT TYPEOF('this is a character string'::VARIANT);
```

TYPEOF('THIS IS A CHARACTER STRING'::VARIANT)
VARCHAR

1 Row(s) produced. Time Elapsed: 0.788s

- 스노우플레이크는 저장된 데이터가 어떤 타입인지 알려주는 내장 함수 typeof()를 제공합니다

```
SELECT TYPEOF(false::VARIANT);
```

TYPEOF(FALSE::VARIANT)
BOOLEAN

1 Row(s) produced. Time Elapsed: 0.102s

```
SELECT TYPEOF(CURRENT_TIMESTAMP::VARIANT);
```

TYPEOF(CURRENT_TIMESTAMP::VARIANT)
TIMESTAMP_LTZ

1 Row(s) produced. Time Elapsed: 1.393s

5.1.4. 기타 데이터형

배열 데이터 타입은 VARIANT 값들로 구성된 가변 길이 배열입니다. 배열은 길이가 0 또는 그 이상으로 생성될 수 있으며, 나중에 확장될 수도 있습니다. 배열의 요소 개수에는 상한선이 없지만, 전체 크기는 16MB로 제한됩니다. 대괄호([와])는 배열 리터럴의 구분자로 사용됩니다.

```
SELECT [123, 'ABC', CURRENT_TIME] AS my_array;
```

```
+-----+
| MY_ARRAY |
+-----+
| [        |
|   123,   |
|   "ABC", |
|   "09:13:54" |
| ]        |
+-----+
```

1 Row(s) produced. Time Elapsed: 0.647s

결과 집합은 배열 타입의 단일 열을 포함하는 단일 행입니다. 배열은 여러 행으로 펼쳐질 수 있습니다 (이 예시의 경우 3개의 행). 이는 table() 함수와 flatten() 함수의 조합을 사용하여 수행됩니다:

```
SELECT value
FROM TABLE(FLATTEN(INPUT => [123, 'ABC', CURRENT_TIME]));
```

```
+-----+
| VALUE |
+-----+
| 123   |
| "ABC" |
| "09:14:22" |
+-----+
```

3 Row(s) produced. Time Elapsed: 0.319s

5.1.4. 기타 데이터형

스노우플레이크의 OBJECT 타입은 키-값 쌍들을 저장하며, 이때 키는 VARCHAR 타입이고 값은 VARIANT 타입입니다. 객체 리터럴은 중괄호({ 와 })를 사용하여 생성되며, 키와 값은 콜론(:)으로 구분됩니다.

```
SELECT {'new_years' : '01/01',
       'independence_day' : '07/04',
       'christmas' : '12/25'}
AS my_object;
```

MY_OBJECT
{ "christmas": "12/25", "independence_day": "07/04", "new_years": "01/01" }

1 Row(s) produced. Time Elapsed: 0.295s

```
SELECT key, value
FROM TABLE(
  FLATTEN(
    {'new_years' : '01/01',
     'independence_day' : '07/04',
     'christmas' : '12/25'}
  )
);
```

KEY	VALUE
christmas	"12/25"
independence_day	"07/04"
new_years	"01/01"

3 Row(s) produced. Time Elapsed: 0.315s

```
SELECT value
FROM TABLE(
  FLATTEN(
    {'new_years' : '01/01',
     'independence_day' : '07/04',
     'christmas' : '12/25'}
  )
)
WHERE key = 'new_years';
```

VALUE
"01/01"

1 Row(s) produced. Time Elapsed: 0.326s

5.2. 테이블 생성

```
CREATE TABLE person (  
  first_name      VARCHAR(50),  
  last_name       VARCHAR(50),  
  birth_date      DATE,  
  eye_color       VARCHAR(10),  
  occupation      VARCHAR(50),  
  children        ARRAY,  
  years_of_education NUMBER(2,0)  
);
```

```
+-----+  
| status                               |  
+-----+  
| Table PERSON successfully created. |  
+-----+  
1 Row(s) produced. Time Elapsed: 0.219s
```

- First and last name
- Birth date
- Eye color
- Occupation
- Names of children
- Years of education

5.3. 테이블 채우기와 수정

```
INSERT INTO person (first_name, last_name, birth_date, eye_color, occupation, children, years_of_education)
VALUES ('Bob', 'Smith', '22-JAN-2000', 'blue', 'teacher', null, 18);
```

```
+-----+
| number of rows inserted |
+-----+
| 1 |
+-----+
```

1 Row(s) produced. Time Elapsed: 0.694s

```
INSERT INTO person (first_name, last_name, birth_date, eye_color, occupation, years_of_education)
VALUES ('Gina', 'Peters', '03-MAR-2001', 'brown', 'student', 12), ('Tim', 'Carpenter', '05-MAR-2001', 'blue', 'student', 15);
```

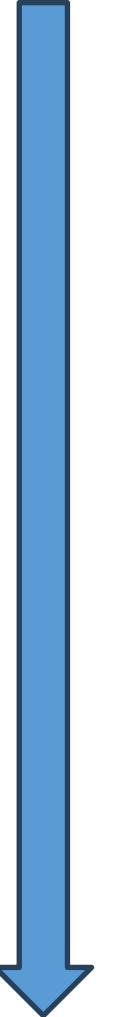
```
+-----+
| number of rows inserted |
+-----+
| 2 |
+-----+
```

1 Row(s) produced. Time Elapsed: 0.642s

```
INSERT INTO person (first_name, last_name, birth_date, eye_color,
occupation, children, years_of_education)
SELECT 'Sharon' AS first_name, last_name, birth_date, eye_color,
'doctor' AS occupation,
['Sue'::VARIANT, 'Shawn'::VARIANT] AS children,
20 AS years_of_education
FROM person
WHERE first_name = 'Tim' AND last_name = 'Carpenter';
```

```
+-----+
| number of rows inserted |
+-----+
| 1 |
+-----+
```

1 Row(s) produced. Time Elapsed: 0.649s



5.3.1. 데이터 입력

```
INSERT INTO person (first_name, last_name, birth_date, eye_color, occupation, years_of_education)
VALUES ('Tim','Carpenter','09-JUL-2002','green','salesman', 16),
      ('Kathy','Little','29-AUG-2001','blue','professor', 20),
      ('Sam','Jacobs','13-FEB-2003','brown','lawyer', 18);
```

```
+-----+
| number of rows inserted |
+-----+
|                3       |
+-----+
```

3 Row(s) produced. Time Elapsed: 0.664s

현재 Person 테이블에는 다섯 개의 행이 있지만, 모든 행의 눈 색깔 정보가 잘못 입력되어 처음부터 다시 시작하고 싶다고 가정해 봅시다. 스노우플레이크는 overwrite 옵션을 제공하는데, 이 옵션은 새 행을 삽입하기 전에 테이블의 모든 기존 행을 먼저 제거합니다. 다섯 명 모두를 삭제하고 다시 추가하는 구문은 다음과 같습니다.

```
INSERT OVERWRITE INTO person (first_name, last_name, birth_date, eye_color, occupation, years_of_education)
VALUES ('Bob','Smith','22-JAN-2000','brown','teacher', 18),
      ('Gina','Peters','03-MAR-2001','green','student', 12),
      ('Tim','Carpenter','09-JUL-2002','blue','salesman', 16),
      ('Kathy','Little','29-AUG-2001','brown','professor', 20),
      ('Sam','Jacobs','13-FEB-2003','blue','lawyer', 18);
```

```
+-----+
| number of rows inserted |
+-----+
|                5       |
+-----+
```

5.3.2. 데이터 삭제

```
SELECT first_name, last_name  
FROM person;
```

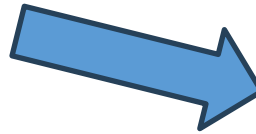
FIRST_NAME	LAST_NAME
Bob	Smith
Gina	Peters
Tim	Carpenter
Kathy	Little
Sam	Jacobs
Sharon	Carpenter

6 Row(s) produced. Time Elapsed: 0.731s

```
SELECT first_name, last_name  
FROM person;
```

FIRST_NAME	LAST_NAME
Bob	Smith
Gina	Peters
Tim	Carpenter
Kathy	Little
Sharon	Carpenter

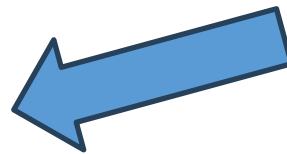
5 Row(s) produced. Time Elapsed: 0.056s



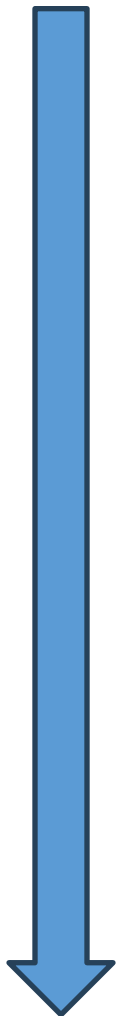
```
DELETE FROM person  
WHERE first_name = 'Sam' AND last_name = 'Jacobs';
```

number of rows deleted
1

1 Row(s) produced. Time Elapsed: 0.887s



5.3.2. 데이터 삭제



```
SELECT first_name, last_name
FROM person
WHERE last_name = 'Smith';
```

FIRST_NAME	LAST_NAME
Bob	Smith

1 Row(s) produced. Time Elapsed: 0.685s

```
SELECT emp_name
FROM employee_sample
WHERE emp_name like '%Smith';
```

EMP_NAME
Bob Smith

1 Row(s) produced. Time Elapsed: 0.575s

```
DELETE FROM person
USING employee_sample
WHERE employee_sample.emp_name = concat(person.first_name, ' ', person.last_name);
```

number of rows deleted
1

```
SELECT first_name, last_name FROM person;
```

FIRST_NAME	LAST_NAME
Gina	Peters
Tim	Carpenter
Kathy	Little
Sharon	Carpenter

4 Row(s) produced. Time Elapsed: 0.439s

```
select emp_name FROM employee_sample;
```

EMP_NAME
Bob Smith
Susan Jackson
Greg Carpenter
Robert Butler
Kim Josephs
John Tyler

6 Row(s) produced. Time Elapsed: 0.924s



5.3.2. 데이터 삭제(앗 이런)

다행히 스노우플레이크에는 타임 트래블이라는 기능이 있습니다. 이 기능을 사용하면 특정 시점의 데이터 상태를 볼 수 있습니다. 스노우플레이크 스탠다드 에디션을 사용하는 경우, 최대 24시간 전까지의 데이터 상태를 볼 수 있습니다. 스노우플레이크 엔터프라이즈 에디션의 경우, 데이터 보존 기간을 과거 최대 90일까지 설정할 수 있습니다.

```
DELETE FROM employee_sample WHERE emp_name = 'Greg Carpenter';
```

number of rows deleted
1

1 Row(s) produced. Time Elapsed: 1.087s

```
SELECT * FROM employee_sample AT(OFFSET => -600)
WHERE emp_name = 'Greg Carpenter';
```

EMPID	EMP_NAME	MGR_EMPID	BIRTH_NATIONKEY	CURRENT_NATIONKEY
1003	Greg Carpenter	1001	3	4

1 Row(s) produced. Time Elapsed: 0.400s

```
INSERT INTO employee_sample
```

```
SELECT * FROM employee_sample AT(OFFSET => -600)
WHERE emp_name = 'Greg Carpenter';
```

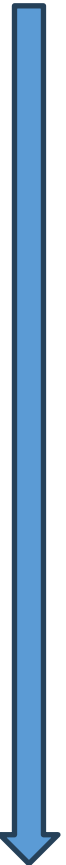
number of rows inserted
1

1 Row(s) produced. Time Elapsed: 0.558s

```
SELECT * FROM employee_sample;
```

EMPID	EMP_NAME	MGR_EMPID	BIRTH_NATIONKEY	CURRENT_NATIONKEY
1001	Bob Smith	NULL	1	2
1002	Susan Jackson	1001	2	3
1004	Robert Butler	1002	4	5
1005	Kim Josephs	1003	5	6
1006	John Tyler	1004	6	7
1003	Greg Carpenter	1001	3	4

6 Row(s) produced. Time Elapsed: 0.349s



5.3.3. 데이터 수정

```
UPDATE person
SET occupation = 'musician', eye_color = 'grey'
WHERE first_name = 'Kathy' and last_name = 'Little';
```

number of rows updated	number of multi-joined rows updated
1	0

1 Row(s) produced. Time Elapsed: 0.816s

```
UPDATE person AS p
SET occupation = 'boss'
FROM employee_sample AS e
WHERE e.emp_name = concat(p.first_name, ' ', p.last_name)
AND e.mgr_empid IS NULL;
```

number of rows updated	number of multi-joined rows updated
1	0

1 Row(s) produced. Time Elapsed: 0.580s

```
INSERT INTO person (first_name, last_name, birth_date, eye_color,
occupation, years_of_education)
VALUES ('Bob', 'Smith', '22-JAN-2000', 'blue', 'teacher', 18);
```

number of rows inserted
1

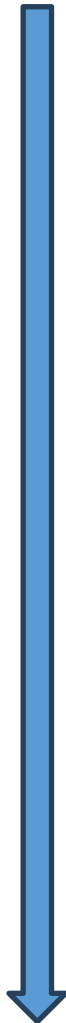
1 Row(s) produced. Time Elapsed: 0.646s

```
SELECT first_name, last_name, occupation
FROM person;
```

FIRST_NAME	LAST_NAME	OCCUPATION
Gina	Peters	student
Tim	Carpenter	salesman
Kathy	Little	musician
Sharon	Carpenter	doctor
Bob	Smith	boss

5 Row(s) produced. Time Elapsed: 0.695s

5.3.3. 데이터 수정



```
UPDATE person AS p
  SET p.years_of_education = e.empid - 1000
FROM employee_sample AS e
WHERE e.empid < 1003;
```

number of rows updated	number of multi-joined rows updated
5	5

10 Row(s) produced. Time Elapsed: 0.731s

```
SELECT first_name, last_name, years_of_education
FROM person;
```

FIRST_NAME	LAST_NAME	YEARS_OF_EDUCATION
Gina	Peters	1
Tim	Carpenter	1
Kathy	Little	1
Sharon	Carpenter	1
Bob	Smith	1

5 Row(s) produced. Time Elapsed: 0.356s

5.3.4. 데이터 병합

```
CREATE TABLE person_refresh AS
SELECT *
FROM (
  VALUES ('Bob','Smith','no','22-JAN-2000','blue','manager'), ('Gina','Peters','no','03-MAR-2001','brown','student'),
    ('Tim','Carpenter','no','09-JUL-2002','green','salesman'), ('Carl','Langford','no','16-JUN-2001','blue','tailor'),
    ('Sharon','Carpenter','yes',null,null,null), ('Kathy','Little','yes',null,null,null)
)
AS hr_list (fname, lname, remove, dob, eyes, profession);
```

```
+-----+
| status                                     |
+-----+
| Table PERSON_REFRESH successfully created. |
+-----+
1 Row(s) produced. Time Elapsed: 1.157s
```

```
SELECT * FROM person_refresh;
```

FNAME	LNAME	REMOVE	DOB	EYES	PROFESSION
Bob	Smith	no	22-JAN-2000	blue	manager
Gina	Peters	no	03-MAR-2001	brown	student
Tim	Carpenter	no	09-JUL-2002	green	salesman
Carl	Langford	no	16-JUN-2001	blue	tailor
Sharon	Carpenter	yes	NULL	NULL	NULL
Kathy	Little	yes	NULL	NULL	NULL

6 Row(s) produced. Time Elapsed: 0.354s

```
SELECT first_name, last_name, birth_date,
       eye_color, occupation
FROM person;
```

FIRST_NAME	LAST_NAME	BIRTH_DATE	EYE_COLOR	OCCUPATION
Gina	Peters	2001-03-03	green	student
Tim	Carpenter	2002-07-09	blue	salesman
Kathy	Little	2001-08-29	grey	musician
Sharon	Carpenter	2002-07-09	blue	doctor
Bob	Smith	2000-01-22	blue	boss

5 Row(s) produced. Time Elapsed: 0.466s

5.3.4. 데이터 병합

```
MERGE INTO person AS p
  USING person_refresh AS pr
    ON p.first_name = pr.fname AND p.last_name = pr.lname
  WHEN MATCHED AND pr.remove = 'yes' THEN DELETE
  WHEN MATCHED THEN
    UPDATE SET p.birth_date = pr.dob, p.eye_color = pr.eyes, p.occupation = pr.profession
  WHEN NOT MATCHED THEN
    INSERT (first_name, last_name, birth_date, eye_color, occupation)
      VALUES (pr.fname, pr.lname, pr.dob, pr.eyes, pr.profession);
```

number of rows inserted	number of rows updated	number of rows deleted
1	3	2

```
SELECT first_name, last_name, birth_date,
       eye_color, occupation
FROM person;
```

FIRST_NAME	LAST_NAME	BIRTH_DATE	EYE_COLOR	OCCUPATION
Carl	Langford	2001-06-16	blue	tailor
Gina	Peters	2001-03-03	brown	student
Tim	Carpenter	2002-07-09	green	salesman
Bob	Smith	2000-01-22	blue	manager

4 Row(s) produced. Time Elapsed: 0.836s

5.4. 복습 하기

? 연습 5-1

Pet_Owner 및 Pet 테이블에 대한 다음 데이터가 주어졌을 때 Pet.Owner 열을 Pet_Owner.Owner_Name 열의 연결된 소유자의 이름으로 설정하는 업데이트 문을 작성합니다.

Pet_Owner			Pet			
OWNER_ID	OWNER_NAME		PET_ID	OWNER_ID	PET_NAME	OWNER
1	John		101	1	Fluffy	NULL
2	Cindy		102	3	Spot	NULL
3	Laura		103	4	Rover	NULL
4	Mark		104	NULL	Rosco	NULL

5.4. 복습 하기(정답)

100 연습 5-1

```
UPDATE pet AS p
SET p.owner = po.owner_name
FROM pet_owner AS po
WHERE p.owner_id = po.owner_id;
```

6. 데이터 생성, 변환, 조작

6.1. 문자형 데이터와 작업하기

스노우플레이크에는 문자열을 연결하거나, 부분 문자열을 찾거나, 대소문자를 바꾸는 등, 문자 데이터를 다루는 데 필요한 거의 모든 작업을 위한 다양한 내장 함수들이 있습니다.

```
SELECT 'here is my string';
```

```
+-----+  
| 'HERE IS MY STRING' |  
|-----|  
| here is my string   |  
+-----+
```

```
1 Row(s) produced. Time Elapsed: 0.173s
```

<https://docs.snowflake.com/ko/sql-reference-functions>

6.1.1. 문자열 생성과 조작

```
SELECT 'string 1' || ' and ' || 'string2';
```

'STRING 1' ' AND ' 'STRING2'
string 1 and string2

1 Row(s) produced. Time Elapsed: 0.138s

```
SELECT CONCAT('string1',' and ','string2');
```

CONCAT('STRING1',' AND ','STRING2')
string1 and string2

1 Row(s) produced. Time Elapsed: 0.098s

```
SELECT CONCAT('I spent ', CHAR(8364), '357 in Paris');
```

CONCAT('I SPENT ',CHAR(8364),'357 IN PARIS')
I spent €357 in Paris

1 Row(s) produced. Time Elapsed: 0.541s

```
SELECT UPPER(str.val), LOWER(str.val), INITCAP(str.val)  
FROM (VALUES ('which case is best?')) AS str(val);
```

UPPER(STR.VAL)	LOWER(STR.VAL)	INITCAP(STR.VAL)
WHICH CASE IS BEST?	which case is best?	Which Case Is Best?

1 Row(s) produced. Time Elapsed: 0.103s

6.1.1. 문자열 생성과 조작

```
SELECT REVERSE('?siht daer uoy nac');
```

REVERSE('?SIHT DAER UOY NAC')
can you read this?

1 Row(s) produced. Time Elapsed: 0.382s

```
SELECT LTRIM(str.val), RTRIM(str.val), TRIM(str.val)
FROM (VALUES (' abc ')) AS str(val);
```

LTRIM(STR.VAL)	RTRIM(STR.VAL)	TRIM(STR.VAL)
abc	abc	abc

1 Row(s) produced. Time Elapsed: 0.113s

```
SELECT LENGTH(LTRIM(str.val)) AS str1_len,
       LENGTH(RTRIM(str.val)) AS str2_len,
       LENGTH(TRIM(str.val)) AS str3_len
FROM (VALUES (' abc ')) AS str(val);
```

STR1_LEN	STR2_LEN	STR3_LEN
4	4	3

1 Row(s) produced. Time Elapsed: 0.336s

```
SELECT TRANSLATE('(857)-234-5678', '()-','');
```

TRANSLATE('(857)-234-5678','()-','')
8572345678

1 Row(s) produced. Time Elapsed: 0.412s

```
SELECT TRANSLATE('AxBzCz', 'ABC', 'XYZ');
```

TRANSLATE('AXBYCZ','ABC','XYZ')
XxYyZz

1 Row(s) produced. Time Elapsed: 0.271s

6.1.2. 문자열 검색과 추출

```
SELECT POSITION('here', str.val) AS pos1,  
       POSITION('here', str.val, 10) AS pos2,  
       POSITION('nowhere', str.val) AS pos3  
FROM (VALUES ('here, there, and everywhere')) AS str(val);
```

POS1	POS2	POS3
1	24	0

1 Row(s) produced. Time Elapsed: 0.637s

```
SELECT SUBSTR(str.val, 1, 10) AS start_of_string,  
       SUBSTR(str.val, 11) AS rest_of_string  
FROM (VALUES ('beginning ending')) AS str(val);
```

START_OF_STRING	REST_OF_STRING
beginning	ending

1 Row(s) produced. Time Elapsed: 0.119s

```
SELECT SUBSTR(str.val, POSITION('every', str.val))  
FROM (VALUES ('here, there, and everywhere')) AS str(val);
```

SUBSTR(STR.VAL, POSITION('EVERY',STR.VAL))
everywhere

1 Row(s) produced. Time Elapsed: 0.425s

6.1.2. 문자열 검색과 추출

```
SELECT str.val
FROM (VALUES ('here, there, and everywhere')) AS str(val)
WHERE STARTSWITH(str.val, 'here');
```

VAL
here, there, and everywhere

```
SELECT str.val
FROM (VALUES ('here, there, and everywhere')) AS str(val)
WHERE ENDSWITH(str.val, 'where');
```

VAL
here, there, and everywhere

```
FROM (VALUES ('here, there, and everywhere')) AS str(val)
WHERE CONTAINS(str.val, 'there');
```

VAL
here, there, and everywhere

6.2. 숫자형 데이터와 작업하기

숫자 데이터를 생성하는 것은 비교적 간단합니다. 숫자를 직접 입력하거나, 테이블 열에서 값을 가져오거나, 산술 연산자(+, -, *, /)를 사용하여 계산할 수 있습니다

```
SELECT 10 AS radius, 2 * 3.14159 * 10 AS circumference;
```

RADIUS	CIRCUMFERENCE
10	62.83180

1 Row(s) produced. Time Elapsed: 0.531s

```
SELECT (3 * 6) - (10 / 2);
```

(3 * 6) - (10 / 2)
13.000000

1 Row(s) produced. Time Elapsed: 0.278s

6.2.1. 숫자형 함수

```
SELECT 10 AS radius,  
       2 * 3.14159 * 10 AS circumference,  
       3.14159 * POWER(10,2) AS area;
```

RADIUS	CIRCUMFERENCE	AREA
10	62.83180	314.159

1 Row(s) produced. Time Elapsed: 0.629s

```
SELECT 10 AS radius,  
       2 * PI() * 10 AS circumference,  
       3.14159 * POWER(10,2) AS area;
```

RADIUS	CIRCUMFERENCE	AREA
10	62.83180	314.159265359

1 Row(s) produced. Time Elapsed: 0.309s

```
SELECT MOD(70, 9);
```

MOD(70, 9)
7

1 Row(s) produced. Time Elapsed: 0.271s

```
SELECT SIGN(-7.5233), ABS(-7.5233)
```

SIGN(-7.5233)	ABS(-7.5233)
-1	7.5233

1 Row(s) produced. Time Elapsed: 0.263s

```
SELECT TRUNC(6.49), ROUND(6.49, 1), FLOOR(6.49), CEIL(6.49);
```

TRUNC(6.49)	ROUND(6.49, 1)	FLOOR(6.49)	CEIL(6.49)
6	6.5	6	7

1 Row(s) produced. Time Elapsed: 0.329s

6.2.2. 숫자형 변환

문자열을 숫자로 변환해야 하는 경우, 몇 가지 전략이 있습니다. 하나는 문자열이 숫자로 변환되도록 명시적으로 지정하는 **명시적 변환**이고, 다른 하나는 스노우플레이크가 변환의 필요성을 인지하고 스스로 변환을 시도하는 **암시적 변환**(강제 변환 이라고도 함)입니다.

```
SELECT 123.45 AS real_num  
UNION  
SELECT '678.90' AS real_num;
```

```
+-----+  
| REAL_NUM |  
+-----+  
| 123.45 |  
| 678.90 |  
+-----+
```

2 Row(s) produced. Time Elapsed: 0.509s

```
SELECT 123.45 AS real_num  
UNION  
SELECT 'AAA.BB' AS real_num;
```

100038 (22018): Numeric value 'AAA.BB' is not recognized

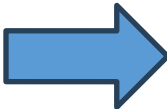
6.2.2. 숫자형 변환

```
SELECT CAST(str.val AS NUMBER(7, 2)) AS cast_val,  
       str.val::NUMBER(7, 2) AS cast_opr_val,  
       TO_DECIMAL(str.val, 7, 2) AS to_dec_val  
FROM (VALUES ('15873.26')) AS str(val);
```

CAST_VAL	CAST_OPR_VAL	TO_DEC_VAL
15873.26	15873.26	15873.26

1 Row(s) produced. Time Elapsed: 0.472s

```
SELECT CAST(str.val AS NUMBER(7, 2)) AS cast_val,  
       str.val::NUMBER(7, 2) AS cast_opr_val,  
       TO_DECIMAL(str.val, 7, 2) AS to_dec_val  
FROM (VALUES ('$15873.26')) AS str(val);  
100038 (22018): Numeric value '$15873.26' is not recognized
```



- The cast() function
- The cast operator ::
- A specific conversion function such as to_decimal()

```
SELECT to_decimal(str.val, '$99999.99', 7, 2) AS to_dec_val  
FROM (VALUES ('$15873.26')) AS str(val);
```

TO_DEC_VAL
15873.26

1 Row(s) produced. Time Elapsed: 0.509s

```
SELECT TRY_TO_DECIMAL(str.val, '$99999.99', 7, 2) AS good,  
       TRY_TO_DECIMAL(str.val, '999.9', 4, 2) AS bad  
FROM (VALUES ('$15873.26')) AS str(val);
```

GOOD	BAD
15873.26	NULL

1 Row(s) produced. Time Elapsed: 0.072s

6.2.3. 숫자형 데이터 생성

테스트 데이터를 만드는 것과 같이 숫자 집합을 생성해야 하는 상황을 위해, 스노우플레이크는 여러 편리한 내장 함수를 제공합니다. 첫 단계는 데이터 행을 생성하는 테이블 함수(table function)를 사용하는 것입니다. 테이블 함수는 쿼리의 FROM 절에서 호출되며 지금은 generator()를 설정 가능한 수의 행을 반환하는 함수 정도로 생각하시면 됩니다.

```
SELECT RANDOM()  
FROM TABLE(GENERATOR(ROWCOUNT => 5));
```

RANDOM()
5406908293433119964
-4117443017875668966
7561089648946528664
6103006136731227981
4670044485419758081

5 Row(s) produced. Time Elapsed: 0.592s

```
SELECT SEQ1()  
FROM TABLE(GENERATOR(ROWCOUNT => 5));
```

SEQ1()
0
1
2
3
4

5 Row(s) produced. Time Elapsed: 0.293s

```
SELECT  
    TO_DATE('01/' || TO_CHAR(SEQ1() + 1) || '/2023', 'DD/MM/YYYY') AS first_of_month  
FROM TABLE(GENERATOR(ROWCOUNT => 12));
```

FIRST_OF_MONTH
2023-01-01
2023-02-01
2023-03-01
2023-04-01
2023-05-01
2023-06-01
2023-07-01
2023-08-01
2023-09-01
2023-10-01
2023-11-01
2023-12-01

12 Row(s) produced. Time Elapsed: 0.292s

6.3. 시간 데이터와 작업하기

시간 데이터는 이 장에서 살펴본 세 가지 데이터 타입 중 가장 복잡합니다. 이는 주로 날짜를 기술하는 방법이 매우 다양하기 때문입니다. 날짜/시간을 형식화(format)할 수 있는 몇 가지 방법은 다음과 같습니다.

- 11/03/2022
- Thursday, November 3, 2022
- 2022-11-03 15:48:56.092 -0700
- 03-NOV-2022 06:48:56 PM EST
- Stardate 76307.5 (a little humor for you Trekkies out there)

```
SELECT TO_TIMESTAMP('04-NOV-2022 18:48:56', 'DD-MON-YYYY HH24:MI:SS') AS now;
```

```
+-----+
| NOW   |
+-----+
| 2022-11-04 18:48:56.000 |
+-----+
```

1 Row(s) produced. Time Elapsed: 0.220s

```
SHOW PARAMETERS LIKE 'timestamp_out%';
```

```
+-----+-----+-----+-----+-----+-----+
| key          | value                                | default                                | level | description                                | type |
+-----+-----+-----+-----+-----+-----+
| TIMESTAMP_OUTPUT_FORMAT | YYYY-MM-DD HH24:MI:SS.FF3 TZHTZM | YYYY-MM-DD HH24:MI:SS.FF3 TZHTZM |       | Default display format for all timestamp types. | STRING |
+-----+-----+-----+-----+-----+-----+
```

1 Row(s) produced. Time Elapsed: 0.105s

```
ALTER SESSION SET TIMESTAMP_OUTPUT_FORMAT = 'MM/DD/YYYY HH12:MI:SS AM TZh:Tz';
```

6.3.1. 날짜와 타임스탬프 생성

시간 값을 생성해야 할 때, 날짜 리터럴을 제공하고 스노우플레이크가 (암시적 변환을 통해) 알아서 처리해주기를 기대할 수도 있고, 형식화 문자열을 (변환 함수와 함께) 제공할 수도 있으며, 또는 날짜(date), 시간(time), 타임스탬프(timestamp) 값을 생성하기 위해 내장 함수 중 하나를 사용할 수도 있습니다.

```
SELECT
    DATE_FROM_PARTS(2025, SEQ1() + 2, 0) AS month_end
FROM TABLE(GENERATOR(ROWCOUNT => 12));
```

MONTH_END
2025-01-31
2025-02-28
2025-03-31
2025-04-30
2025-05-31
2025-06-30
2025-07-31
2025-08-31
2025-09-30
2025-10-31
2025-11-30
2025-12-31

12 Row(s) produced. Time Elapsed: 0.506s

```
SELECT DATE_FROM_PARTS(2025, 3, 15) AS my_date,
    TIME_FROM_PARTS(10, 22, 47) AS my_time;
```

MY_DATE	MY_TIME
2025-03-15	10:22:47

1 Row(s) produced. Time Elapsed: 0.514s

```
SELECT TIMESTAMP_FROM_PARTS(
    DATE_FROM_PARTS(2025, 3, 15),
    TIME_FROM_PARTS(10, 22, 47)
) AS my_timestamp;
```

MY_TIMESTAMP
2025-03-15 10:22:47.000

1 Row(s) produced. Time Elapsed: 1.163s

6.3.2. 날짜와 타임스탬프 조작하기

날짜를 반환하는 시간 함수

```
SELECT DATE_TRUNC('YEAR', dt.val) AS start_of_year,  
       DATE_TRUNC('MONTH', dt.val) AS start_of_month,  
       DATE_TRUNC('QUARTER', dt.val) AS start_of_quarter  
FROM (VALUES (TO_DATE('26-MAY-2025', 'DD-MON-YYYY'))) AS dt(val);
```

START_OF_YEAR	START_OF_MONTH	START_OF_QUARTER
2025-01-01	2025-05-01	2025-04-01

1 Row(s) produced. Time Elapsed: 0.085s

```
SELECT DATEADD(MONTH, 1, TO_DATE('01-JAN-2025', 'DD-MON-YYYY')) AS date1,  
       DATEADD(MONTH, 1, TO_DATE('15-JAN-2025', 'DD-MON-YYYY')) AS date2,  
       DATEADD(MONTH, 1, TO_DATE('31-JAN-2025', 'DD-MON-YYYY')) AS date3;
```

DATE1	DATE2	DATE3
2025-02-01	2025-02-15	2025-02-28

1 Row(s) produced. Time Elapsed: 0.051s

```
SELECT  
       DATEADD(YEAR, -1, TO_DATE('29-FEB-2024', 'DD-MON-YYYY')) AS new_date;
```

NEW_DATE
2023-02-28

1 Row(s) produced. Time Elapsed: 0.051s

6.3.2. 날짜와 타임스탬프 조작하기

문자열 반환하는 시간 함수

```
SELECT DAYNAME(CURRENT_DATE), MONTHNAME(CURRENT_DATE);
```

DAYNAME(CURRENT_DATE)	MONTHNAME(CURRENT_DATE)
Tue	Mar

1 Row(s) produced. Time Elapsed: 0.491s

숫자를 반환하는 시간 함수

```
SELECT DATE_PART(YEAR, dt.val) AS year_num,
       DATE_PART(QUARTER, dt.val) AS qtr_num,
       DATE_PART(MONTH, dt.val) AS month_num,
       DATE_PART(WEEK, dt.val) AS week_num
FROM (VALUES (TO_DATE('24-APR-2023', 'DD-MON-YYYY'))) AS dt(val);
```

YEAR_NUM	QTR_NUM	MONTH_NUM	WEEK_NUM
2023	2	4	17

1 Row(s) produced. Time Elapsed: 0.109s

```
SELECT DATE_PART(HOUR, dt.val) AS hour_num,
       DATE_PART(MINUTE, dt.val) AS min_num,
       DATE_PART(SECOND, dt.val) AS sec_num,
       DATE_PART(NANOSECOND, dt.val) AS nsec_num
FROM (VALUES (CURRENT_TIMESTAMP)) AS dt(val);
```

HOUR_NUM	MIN_NUM	SEC_NUM	NSEC_NUM
11	32	58	808000000

1 Row(s) produced. Time Elapsed: 0.109s

6.3.2. 날짜와 타임스탬프 조작하기

숫자를 반환하는 시간 함수

```
SELECT DATEDIFF(YEAR, dt.val1, dt.val2) AS num_years,  
       DATEDIFF(MONTH, dt.val1, dt.val2) AS num_months,  
       DATEDIFF(DAY, dt.val1, dt.val2) AS num_days,  
       DATEDIFF(HOUR, dt.val1, dt.val2) AS num_hours  
FROM (  
  VALUES(TO_DATE('12-FEB-2022', 'DD-MON-YYYY'),  
          TO_DATE('06-MAR-2023', 'DD-MON-YYYY'))  
) AS dt(val1, val2);
```

NUM_YEARS	NUM_MONTHS	NUM_DAYS	NUM_HOURS
1	13	387	9288

1 Row(s) produced. Time Elapsed: 0.736s

```
SELECT DATEDIFF(YEAR, dt.val1, dt.val2) AS num_years,  
       DATEDIFF(MONTH, dt.val1, dt.val2) AS num_months,  
       DATEDIFF(DAY, dt.val1, dt.val2) AS num_days,  
       DATEDIFF(HOUR, dt.val1, dt.val2) AS num_hours  
FROM (  
  VALUES(TO_DATE('06-MAR-2023', 'DD-MON-YYYY'),  
          TO_DATE('12-FEB-2022', 'DD-MON-YYYY'))  
) AS dt(val1, val2);
```

NUM_YEARS	NUM_MONTHS	NUM_DAYS	NUM_HOURS
-1	-13	-387	-9288

1 Row(s) produced. Time Elapsed: 0.535s



6.3.3. 날짜 변환

```
SELECT CAST('23-SEP-2025' AS DATE) AS format1,  
       CAST('09/23/2025' AS DATE) AS format2,  
       CAST('2025-09-23' AS DATE) AS format3;
```

FORMAT1	FORMAT2	FORMAT3
2025-09-23	2025-09-23	2025-09-23

1 Row(s) produced. Time Elapsed: 0.051s

```
SELECT CAST('09-23-2023' AS DATE);  
100040 (22007): Date '09-23-2023' is not recognized
```

```
SELECT TRY_CAST('09-23-2023' AS DATE);
```

TRY_CAST('09-23-2023' AS DATE)
NULL

1 Row(s) produced. Time Elapsed: 0.343s

```
SELECT '09/23/2023'::DATE AS date_val,  
       '23-SEP-2023'::TIMESTAMP AS tmstmp_val,  
       '123.456'::NUMBER(6,3) AS num_val;
```

DATE_VAL	TMSTMP_VAL	NUM_VAL
2023-09-23	2023-09-23 00:00:00.000	123.456

1 Row(s) produced. Time Elapsed: 0.301s

6.4. 복습 하기

? 연습 6-1

2024년 1월 1일과 2025년 8월 15일 사이의 날짜 수를 반환하는 쿼리를 작성하십시오.

? 연습 6-2

2025년 9월 27일의 숫자 연도, 월, 일을 합산하는 쿼리를 작성하십시오.

6.4. 복습 하기(정답)

100 연습 6-1

```
SELECT DATEDIFF(DAY, '01-JAN-2024', '15-AUG-2025')  
AS num_days;
```

100 연습 6-2

```
SELECT DATE_PART(YEAR, dt.val)  
      + DATE_PART(MONTH, dt.val)  
      + DATE_PART(DAY, dt.val)  
FROM (VALUES('27-SEP-2025'::DATE)) AS dt(val);
```


7. 그룹화와 집계

7.1. 그룹핑 개념

```
SELECT o_custkey,  
       SUM(o_totalprice) AS total_sales,  
       COUNT(*) AS number_of_orders  
FROM orders  
GROUP BY o_custkey  
LIMIT 10;
```

O_CUSTKEY	TOTAL_SALES	NUMBER_OF_ORDERS
83098	498229.89	2
107722	417425.91	3
89008	599101.53	3
131644	163655.31	1
139328	481518.67	2
46379	222257.92	1
11323	383366.17	2
133094	107213.82	1
120598	290308.76	2
105646	282919.39	1

10 Row(s) produced. Time Elapsed: 0.214s

마케팅 부서의 부사장으로부터 우수 고객에게 다음 주문 시 25% 할인을 제공하는 특별 프로모션 진행을 도와달라는 요청을 받았다고 가정해 봅시다. 이 프로모션은 \$1,800,000 이상 구매했거나 8회 이상 주문한 고객들을 대상으로 합니다. 당신의 임무는 어떤 고객들이 이 프로모션 대상 자격이 되는지 알아내는 것입니다.

```
SELECT o_custkey,  
       SUM(o_totalprice) AS total_sales,  
       COUNT(*) AS number_of_orders  
FROM orders  
GROUP BY o_custkey  
HAVING SUM(o_totalprice) >= 1800000  
       OR COUNT(*) >= 8;
```

O_CUSTKEY	TOTAL_SALES	NUMBER_OF_ORDERS
119674	1391170.52	8
7693	1300612.14	8
97156	1406682.16	8
37825	1696570.50	9
36316	1826901.18	8
45088	1744140.58	8
19942	1482217.71	8
7597	957698.37	8
7033	1749076.08	8
141352	1583313.81	8
50542	1261984.14	8
41581	1585564.71	8
120202	1870539.99	8
64804	1422490.86	8
66103	1846159.58	8
121909	1610405.31	8

16 Row(s) produced. Time Elapsed: 0.166s

7.2. 집계 함수

```
SELECT COUNT(*) AS num_orders,  
       MIN(o_totalprice) AS min_price,  
       MAX(o_totalprice) AS max_price,  
       AVG(o_totalprice) AS avg_price  
FROM orders;
```

NUM_ORDERS	MIN_PRICE	MAX_PRICE	AVG_PRICE
115269	885.75	555285.16	187845.84979500

1 Row(s) produced. Time Elapsed: 0.707s

```
SELECT DATE_PART(YEAR, o_orderdate) AS order_year,  
       COUNT(*) AS num_orders,  
       MIN(o_totalprice) AS min_price,  
       MAX(o_totalprice) AS max_price,  
       AVG(o_totalprice) AS avg_price  
FROM orders;  
000979 (42601): SQL compilation error:  
[ORDERS.O_ORDERDATE] is not a valid group by expression
```



집계 함수는 그룹 내의 모든 행에 걸쳐 특정 연산을 수행합니다. 예를 들어 행의 개수를 세거나(counting), 숫자 필드를 합산하거나(summing), 평균을 계산하는(calculating averages) 것과 같습니다. 쿼리에 GROUP BY 절이 없더라도, 여전히 하나의 그룹(결과 집합 전체)이 존재합니다.

```
SELECT DATE_PART(YEAR, o_orderdate) AS order_year,  
       COUNT(*) AS num_orders,  
       MIN(o_totalprice) AS min_price,  
       MAX(o_totalprice) AS max_price,  
       AVG(o_totalprice) AS avg_price  
FROM orders  
GROUP BY DATE_PART(YEAR, o_orderdate);
```

ORDER_YEAR	NUM_ORDERS	MIN_PRICE	MAX_PRICE	AVG_PRICE
1996	17657	1036.89	498599.91	186689.32167356
1994	17479	1021.55	489099.53	187566.44502946
1997	17408	992.28	471220.08	186987.97800322
1993	17392	946.60	460118.47	188041.41387649
1998	10190	1059.39	502742.76	188929.97532090
1995	17637	885.75	499753.01	188100.11965924
1992	17506	1087.90	555285.16	189062.87926368

7 Row(s) produced. Time Elapsed: 0.362s

7.2.1. COUNT() 함수

각 그룹에 속하는 행의 수를 세는 것은 매우 흔한 작업이며, 이 장에서 이미 여러 예시를 보셨습니다. 하지만 논의할 만한 몇 가지 변형이 있습니다. 첫 번째는 count() 함수와 DISTINCT 연산자를 조합하는 것입니다.

```
SELECT COUNT(*) AS total_orders,  
       COUNT(DISTINCT o_custkey) AS num_customers,  
       COUNT(DISTINCT DATE_PART(YEAR, o_orderdate)) AS num_years  
FROM orders;
```

TOTAL_ORDERS	NUM_CUSTOMERS	NUM_YEARS
115269	66076	7

1 Row(s) produced. Time Elapsed: 0.638s

count() 함수의 또 다른 유용한 변형으로는 count_if()가 있습니다. 이 함수는 주어진 조건이 참(true)으로 평가되는 행의 수를 셉니다.

```
SELECT  
       COUNT_IF(1992 = DATE_PART(YEAR, o_orderdate)) AS num_1992,  
       COUNT_IF(1995 = DATE_PART(YEAR, o_orderdate)) AS num_1995  
FROM orders;
```

NUM_1992	NUM_1995
17506	17637

1 Row(s) produced. Time Elapsed: 0.396s

7.2.2. MIN(), MAX(), AVG(), SUM() 함수

숫자 열을 포함하는 데이터를 그룹화할 때는, 그룹 내에서 최대값 또는 최소값을 찾거나, 그룹의 평균값을 계산하거나, 그룹 내 모든 행의 값을 합산해야 하는 경우가 많습니다. 이러한 목적을 위해 max(), min(), avg(), sum() 집계 함수가 사용되며, max()와 min()은 날짜 열에도 자주 사용됩니다.

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       MIN(o_orderdate) AS first_order,  
       MAX(o_orderdate) AS last_order,  
       AVG(o_totalprice) AS avg_price,  
       SUM(o_totalprice) AS tot_sales  
FROM orders  
GROUP BY DATE_PART(YEAR, o_orderdate);
```

YEAR	FIRST_ORDER	LAST_ORDER	AVG_PRICE	TOT_SALES
1996	1996-01-01	1996-12-31	186689.32167356	3296373352.79
1994	1994-01-01	1994-12-31	187566.44502946	3278473892.67
1997	1997-01-01	1997-12-31	186987.97800322	3255086721.08
1993	1993-01-01	1993-12-31	188041.41387649	3270416270.14
1998	1998-01-01	1998-08-02	188929.97532090	1925196448.52
1995	1995-01-01	1995-12-31	188100.11965924	3317521810.43
1992	1992-01-01	1992-12-31	189062.87926368	3309734764.39

7 Row(s) produced. Time Elapsed: 0.673s

7.2.3. LISTAGG() 함수

XML 또는 JSON 문서 생성 시 흔히 필요한 작업인 데이터 플래튼이 필요할 경우, listagg() 함수가 매우 유용하다는 것을 알게 될 것입니다. listagg() 함수는 여러 값들을 지정된 구분자로 연결하여 하나의 목록으로 만든 후, 이를 단일 열로 반환합니다.

```
SELECT r.r_name, LISTAGG(n.n_name, ',') AS nation_list
FROM region r INNER JOIN nation n
    ON r.r_regionkey = n.n_regionkey
GROUP BY r.r_name;
```

R_NAME	NATION_LIST
AFRICA	ALGERIA, ETHIOPIA, KENYA, MOROCCO, MOZAMBIQUE
MIDDLE EAST	EGYPT, IRAN, IRAQ, JORDAN, SAUDI ARABIA
AMERICA	ARGENTINA, BRAZIL, CANADA, PERU, UNITED STATES
EUROPE	FRANCE, GERMANY, ROMANIA, RUSSIA, UNITED KINGDOM
ASIA	CHINA, INDIA, INDONESIA, JAPAN, VIETNAM

5 Row(s) produced. Time Elapsed: 0.890s

이제 결과 집합은 지역 이름으로 정렬되어 있으며, 각 지역별 국가 목록은 국가 이름 순서로 정렬되어 있습니다. listagg()로 생성된 목록에 중복된 값이 있다면 DISTINCT를 지정할 수 있으며, 이 경우 목록에는 고유한 값들만 포함됩니다.



```
SELECT r.r_name,
    LISTAGG(n.n_name, ',')
        WITHIN GROUP (ORDER BY n.n_name) AS nation_list
FROM region r
    INNER JOIN nation n
        ON r.r_regionkey = n.n_regionkey
GROUP BY r.r_name
ORDER BY r.r_name;
```

R_NAME	NATION_LIST
AFRICA	ALGERIA, ETHIOPIA, KENYA, MOROCCO, MOZAMBIQUE
AMERICA	ARGENTINA, BRAZIL, CANADA, PERU, UNITED STATES
ASIA	CHINA, INDIA, INDONESIA, JAPAN, VIETNAM
EUROPE	FRANCE, GERMANY, ROMANIA, RUSSIA, UNITED KINGDOM
MIDDLE EAST	EGYPT, IRAN, IRAQ, JORDAN, SAUDI ARABIA

7.3. 그룹 생성

group by 절은 데이터 행을 그룹화하는 메커니즘입니다. 여기에서는 여러 열을 기준으로 데이터를 그룹화하는 방법, 표현식을 사용하여 데이터를 그룹화하는 방법, 그룹 내에서 롤업을 생성하는 방법을 살펴보겠습니다.

7.3.1. 다중칼럼 그룹화

지금까지의 모든 예시는 단일 열을 기준으로 데이터를 그룹화했습니다. 하지만 원하는 만큼 여러 개의 열을 기준으로 그룹화할 수도 있습니다. 다음 예시는 'America' 지역 고객들에 대해 각 국가 및 시장 세그먼트별 고객 수를 계산합니다.

```
SELECT n.n_name, c.c_mktsegment, COUNT(*)
FROM customer c
INNER JOIN nation n
    ON c.c_nationkey = n.n_nationkey
WHERE n.n_regionkey = 1
GROUP BY n.n_name, c.c_mktsegment
ORDER BY 1, 2
LIMIT 10;
```

N_NAME	C_MKTSEGMENT	COUNT(*)
ARGENTINA	AUTOMOBILE	521
ARGENTINA	BUILDING	580
ARGENTINA	FURNITURE	488
ARGENTINA	HOUSEHOLD	516
ARGENTINA	MACHINERY	533
BRAZIL	AUTOMOBILE	503
BRAZIL	BUILDING	551
BRAZIL	FURNITURE	492
BRAZIL	HOUSEHOLD	521
BRAZIL	MACHINERY	547

10 Row(s) produced. Time Elapsed: 0.532s

7.3.2. 표현식을 사용한 그룹화

그룹화는 열(column)에만 국한되지 않으며, 그룹화를 생성하기 위해 여러 표현식(expression)을 사용할 수도 있습니다.

```
SELECT DATE_PART(YEAR, o.o_orderdate) AS year,
       DATEDIFF(MONTH, o.o_orderdate, l.l_shipdate) AS months_to_ship,
       COUNT(*)
FROM orders o INNER JOIN lineitem l ON o.o_orderkey = l.l_orderkey
WHERE o.o_orderdate >= '01-JAN-1997'::DATE
GROUP BY DATE_PART(YEAR, o.o_orderdate),
         DATEDIFF(MONTH, o.o_orderdate, l.l_shipdate)
ORDER BY 1, 2;
```

YEAR	MONTHS_TO_SHIP	COUNT(*)
1997	0	2195
1997	1	4601
1997	2	4644
1997	3	4429
1997	4	2245
1997	5	2
1998	0	1295
1998	1	2602
1998	2	2628
1998	3	2724
1998	4	1356
1998	5	1

12 Row(s) produced. Time Elapsed: 0.883s

스노우플레이크는 SQL 구현을 포함한 많은 서버 함수에 새로운 기능들이 추가되면서 끊임없이 발전하고 있습니다. 스노우플레이크는 GROUP BY ALL 옵션을 추가했습니다. 이는 표현식을 사용하여 데이터를 그룹화할 때 유용한 단축 기능입니다.

```
SELECT DATE_PART(YEAR, o.o_orderdate) AS year,
       DATEDIFF(MONTH, o.o_orderdate, l.l_shipdate) AS months_to_ship,
       COUNT(*)
FROM orders o INNER JOIN lineitem l ON o.o_orderkey = l.l_orderkey
WHERE o.o_orderdate >= '01-JAN-1997'::DATE
GROUP BY ALL
ORDER BY 1, 2;
```

7.3.3. 롤업 생성

각 국가 및 시장 세그먼트별 집계(counts)와 더불어, 모든 시장 세그먼트를 포함하는 각 국가별 총 집계(total counts)도 알고 싶다고 가정해 봅시다. 이는 GROUP BY 절의 ROLLUP 옵션을 사용하여 수행할 수 있습니다.

```
SELECT n.n_name, c.c_mktsegment, COUNT(*)
FROM customer c
INNER JOIN nation n
    ON c.c_nationkey = n.n_nationkey
WHERE n.n_regionkey = 1
GROUP BY n.n_name, c.c_mktsegment
ORDER BY 1, 2
LIMIT 10;
```

N_NAME	C_MKTSEGMENT	COUNT(*)
ARGENTINA	AUTOMOBILE	521
ARGENTINA	BUILDING	580
ARGENTINA	FURNITURE	488
ARGENTINA	HOUSEHOLD	516
ARGENTINA	MACHINERY	533
BRAZIL	AUTOMOBILE	503
BRAZIL	BUILDING	551
BRAZIL	FURNITURE	492
BRAZIL	HOUSEHOLD	521
BRAZIL	MACHINERY	547

10 Row(s) produced. Time Elapsed: 0.150s

```
SELECT n.n_name, c.c_mktsegment, COUNT(*)
FROM customer c
INNER JOIN nation n
    ON c.c_nationkey = n.n_nationkey
WHERE n.n_regionkey = 1
GROUP BY ROLLUP(n.n_name, c.c_mktsegment)
ORDER BY 1, 2;
```

N_NAME	C_MKTSEGMENT	COUNT(*)
ARGENTINA	AUTOMOBILE	521
ARGENTINA	BUILDING	580
...		
ARGENTINA	NULL	2638
BRAZIL	AUTOMOBILE	503
...		
BRAZIL	MACHINERY	547
BRAZIL	NULL	2614
CANADA	AUTOMOBILE	499
...		
CANADA	MACHINERY	522
CANADA	NULL	2631
PERU	AUTOMOBILE	560
...		
PERU	MACHINERY	470
PERU	NULL	2625
UNITED STATES	AUTOMOBILE	514
...		
UNITED STATES	MACHINERY	519
UNITED STATES	NULL	2603
NULL	NULL	13111

31 Row(s) produced. Time Elapsed: 0.896s



7.3.3. 롤업 생성

```
SELECT n.n_name, c.c_mktsegment, COUNT(*)
FROM customer c
INNER JOIN nation n
    ON c.c_nationkey = n.n_nationkey
WHERE n.n_regionkey = 1
GROUP BY ROLLUP(c.c_mktsegment, n.n_name)
ORDER BY 1, 2;
```

N_NAME	C_MKTSEGMENT	COUNT(*)
ARGENTINA	AUTOMOBILE	521
...		
UNITED STATES	AUTOMOBILE	514
UNITED STATES	BUILDING	516
UNITED STATES	FURNITURE	522
UNITED STATES	HOUSEHOLD	532
UNITED STATES	MACHINERY	519
NULL	AUTOMOBILE	2597
NULL	BUILDING	2743
NULL	FURNITURE	2529
NULL	HOUSEHOLD	2651
NULL	MACHINERY	2591
NULL	NULL	13111

31 Row(s) produced. Time Elapsed: 0.907s

이제 각 시장 세그먼트별 소계(subtotal)와 전체 행에 대한 최종 합계(final total)는 있지만, 각 국가별 소계는 사라졌습니다. 만약 두 열 모두에 대해 소계를 생성해야 한다면, ROLLUP 대신 CUBE 옵션을 사용할 수 있습니다:

```
SELECT n.n_name, c.c_mktsegment, COUNT(*)
FROM customer c
INNER JOIN nation n
    ON c.c_nationkey = n.n_nationkey
WHERE n.n_regionkey = 1
GROUP BY CUBE(c.c_mktsegment, n.n_name)
ORDER BY 1, 2;
```

N_NAME	C_MKTSEGMENT	COUNT(*)
ARGENTINA	AUTOMOBILE	521
...		
ARGENTINA	NULL	2638
BRAZIL	AUTOMOBILE	503
...		
BRAZIL	NULL	2614
CANADA	AUTOMOBILE	499
...		
CANADA	NULL	2631
PERU	AUTOMOBILE	560
...		
PERU	NULL	2625
UNITED STATES	AUTOMOBILE	514
...		
UNITED STATES	NULL	2603
NULL	AUTOMOBILE	2597
NULL	BUILDING	2743
NULL	FURNITURE	2529
NULL	HOUSEHOLD	2651
NULL	MACHINERY	2591
NULL	NULL	13111

36 Row(s) produced. Time Elapsed: 0.676s



7.4. 그룹화된 데이터에서 필터링

```
SELECT o_custkey, SUM(o_totalprice)
FROM orders
WHERE 1998 = DATE_PART(YEAR, o_orderdate)
GROUP BY o_custkey
ORDER BY 1
LIMIT 10;
```

O_CUSTKEY	SUM(O_TOTALPRICE)
19	302071.17
56	275833.86
58	299919.14
61	205460.32
70	247555.65
71	103723.77
82	169060.30
124	268944.99
181	168293.04
194	210880.67

10 Row(s) produced. Time Elapsed: 0.299s

모든 그룹화 연산은 WHERE 절이 평가된 후에 수행되므로, 집계 함수를 포함하는 필터 조건을 WHERE 절에 넣는 것은 불가능합니다. 대신, 바로 이 목적(집계 결과 필터링)을 위해 특별히 존재하는 HAVING 절이 있습니다.



```
SELECT o_custkey, SUM(o_totalprice)
FROM orders
WHERE 1998 = DATE_PART(YEAR, o_orderdate)
      AND SUM(o_totalprice) >= 700000
GROUP BY o_custkey
ORDER BY 1;
```

002035 (42601): SQL compilation error:
Invalid aggregate function in where clause [SUM(ORDERS.O_TOTALPRICE)]



```
SELECT o_custkey, SUM(o_totalprice)
FROM orders
WHERE 1998 = DATE_PART(YEAR, o_orderdate)
GROUP BY o_custkey
HAVING SUM(o_totalprice) >= 700000
ORDER BY 1;
```

O_CUSTKEY	SUM(O_TOTALPRICE)
4309	719354.94
5059	734893.37
33487	727194.76
65434	1030712.34
90608	727770.25
138724	717744.86

6 Row(s) produced. Time Elapsed: 0.665s

7.5. 복습 하기

? 연습 7-1

Supplier 테이블의 행 수를 세고 s_acctbal 열의 최소값과 최대값을 결정하는 쿼리를 작성하십시오.

? 연습 7-2

연습 문제 7-1의 쿼리를 수정하여 전체 테이블이 아닌 s_nationkey의 각 값에 대해 동일한 계산을 수행하십시오.

7.5. 복습 하기(정답)

100 연습 7-1

```
SELECT
    COUNT(*),
    MIN(s_acctbal),
    MAX(s_acctbal)
FROM supplier;
```

100 연습 7-2

```
SELECT
    s_nationkey,
    COUNT(*),
    MIN(s_acctbal),
    MAX(s_acctbal)
FROM supplier
GROUP BY s_nationkey;
```

8. 서버 쿼리

8.1. 서브 쿼리 정의

서브쿼리는 다른 SQL 문 내에 포함된 쿼리입니다. 서브쿼리는 항상 괄호(())로 묶이며, 일반적으로 포함하는 문장보다 먼저 실행됩니다. 모든 쿼리와 마찬가지로, 서브쿼리도 결과 집합을 반환하며, 이는 단일 행 또는 여러 행, 그리고 단일 열 또는 여러 열로 구성될 수 있습니다. 서브쿼리가 반환하는 결과 집합의 유형에 따라, 포함하는 문장이 해당 결과 데이터와 상호 작용하기 위해 사용할 수 있는 연산자가 결정됩니다.

```
SELECT n_nationkey, n_name
FROM nation
WHERE n_regionkey = (
    SELECT r_regionkey
    FROM region
    WHERE r_name = 'ASIA'
);
```

N_NATIONKEY	N_NAME
8	INDIA
9	INDONESIA
12	JAPAN
18	CHINA
21	VIETNAM

5 Row(s) produced. Time Elapsed: 0.938s



```
SELECT n.n_nationkey, n.n_name
FROM nation n
    INNER JOIN region r
    ON n.n_regionkey = r.r_regionkey
WHERE r.r_name = 'ASIA';
```

N_NATIONKEY	N_NAME
8	INDIA
9	INDONESIA
12	JAPAN
18	CHINA
21	VIETNAM

5 Row(s) produced. Time Elapsed: 0.462s

서브쿼리 대신 조인(join)을 사용해도 동일한 결과 집합을 얻을 수 있다고 생각하셨다면, 맞습니다.

8.2. 서브 쿼리 종류

서브쿼리에는 두 가지 유형이 있으며, 차이점은 서브쿼리가 포함하는 쿼리와 별도로 실행될 수 있는지 여부에 있습니다. 다음 몇 섹션에서는 이러한 두 가지 서브쿼리 유형을 살펴보고 이들과 상호 작용하는 데 사용되는 다양한 연산자를 보여줍니다.

1. 상관 관계가 없는 서브 쿼리
2. 상관 관계가 있는 서브 쿼리

8.2.1. 상관 관계가 없는 서브 쿼리

이 예시의 서브쿼리는 비상관일 뿐만 아니라, 스칼라 서브쿼리라고도 합니다. 이는 서브쿼리가 단일 행과 단일 열만을 반환함을 의미합니다. 스칼라 서브쿼리는 =, <>, <, >, <=, >= 와 같은 비교 연산자를 사용하여 조건의 양쪽에 사용될 수 있습니다.

```
SELECT n_nationkey, n_name
FROM nation
WHERE n_regionkey <> (
    SELECT r_regionkey
    FROM region
    WHERE r_name = 'ASIA'
)
LIMIT 10;
```

N_NATIONKEY	N_NAME
0	ALGERIA
1	ARGENTINA
2	BRAZIL
3	CANADA
4	EGYPT
5	ETHIOPIA
6	FRANCE
7	GERMANY
10	IRAN
11	IRAQ

10 Row(s) produced. Time Elapsed: 0.847s

```
SELECT n_nationkey, n_name FROM nation
WHERE n_regionkey = (
    SELECT r_regionkey FROM region WHERE r_name <> 'ASIA'
);
```

090150 (22000): Single-row subquery returns more than one row.



```
SELECT r_regionkey FROM region WHERE r_name <> 'ASIA';
```

R_REGIONKEY
0
1
3
4

4 Row(s) produced. Time Elapsed: 0.513s

8.2.1. 상관 관계가 없는 서브 쿼리

다중 행, 단일 열 하위 쿼리

```
SELECT n_nationkey, n_name
FROM nation
WHERE n_regionkey IN (
    SELECT r_regionkey
    FROM region
    WHERE r_name <> 'ASIA'
)
LIMIT 10;
```

N_NATIONKEY	N_NAME
0	ALGERIA
1	ARGENTINA
2	BRAZIL
3	CANADA
4	EGYPT
5	ETHIOPIA
6	FRANCE
7	GERMANY
10	IRAN
11	IRAQ

10 Row(s) produced. Time Elapsed: 0.113s

```
SELECT n_nationkey, n_name
FROM nation
WHERE n_regionkey IN (
    SELECT r_regionkey
    FROM region
    WHERE r_name = 'AMERICA' OR r_name = 'EUROPE'
)
LIMIT 10;
```

N_NATIONKEY	N_NAME
0	ALGERIA
4	EGYPT
5	ETHIOPIA
8	INDIA
9	INDONESIA
10	IRAN
11	IRAQ
12	JAPAN
13	JORDAN
14	KENYA

10 Row(s) produced. Time Elapsed: 0.383s

8.2.1. 상관 관계가 없는 서브 쿼리

다중 행, 단일 열 하위 쿼리

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       MAX(o_totalprice) AS max_price  
FROM orders  
WHERE 1997 <> DATE_PART(YEAR, o_orderdate)  
GROUP BY DATE_PART(YEAR, o_orderdate)  
ORDER BY 1;
```

DATE_PART(YEAR, O_ORDERDATE)	MAX(O_TOTALPRICE)
1992	555285.16
1993	460118.47
1994	489099.53
1995	499753.01
1996	498599.91
1998	502742.76

```
SELECT o_custkey, COUNT(*) AS num_orders FROM orders  
WHERE 1996 = DATE_PART(YEAR, o_orderdate)  
GROUP BY o_custkey  
HAVING COUNT(*) > ALL (SELECT COUNT(*) FROM orders  
WHERE 1997 = DATE_PART(YEAR, o_orderdate) GROUP BY o_custkey);
```

O_CUSTKEY	NUM_ORDERS
43645	5

1 Row(s) produced. Time Elapsed: 0.765s

```
SELECT o_custkey, o_orderdate, o_totalprice FROM orders  
WHERE 1997 = DATE_PART(YEAR, o_orderdate)  
AND o_totalprice > ANY (SELECT MAX(o_totalprice)  
FROM orders WHERE 1997 <> DATE_PART(YEAR, o_orderdate)  
GROUP BY DATE_PART(YEAR, o_orderdate));
```

O_CUSTKEY	O_ORDERDATE	O_TOTALPRICE
148348	1997-01-31	465610.95
140506	1997-12-21	461118.75
54602	1997-02-09	471220.08

8.2.1. 상관 관계가 없는 서브 쿼리

다중 열 하위 쿼리

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       MAX(o_totalprice) AS max_price  
FROM orders  
GROUP BY DATE_PART(YEAR, o_orderdate)  
ORDER BY 1;
```

DATE_PART(YEAR, O_ORDERDATE)	MAX(O_TOTALPRICE)
1992	555285.16
1993	460118.47
1994	489099.53
1995	499753.01
1996	498599.91
1997	471220.08
1998	502742.76

7 Row(s) produced. Time Elapsed: 0.694s

이 주문들 각각에 대한 추가 세부 정보(예: custkey, orderdate 값)를 알고 싶다고 가정해 봅시다. 이 행들을 찾으려면, 포함하는 쿼리를 만들어야 하는데, 이 쿼리는 각 주문의 연도 및 총 가격을 서브쿼리가 반환한 두 개의 열과 비교해야 합니다.

```
SELECT o_custkey, o_orderdate, o_totalprice  
FROM orders  
WHERE (DATE_PART(YEAR, o_orderdate), o_totalprice) IN (  
    SELECT DATE_PART(YEAR, o_orderdate), MAX(o_totalprice)  
    FROM orders  
    GROUP BY DATE_PART(YEAR, o_orderdate)  
)  
ORDER BY 2;
```

O_CUSTKEY	O_ORDERDATE	O_TOTALPRICE
21433	1992-11-30	555285.16
95069	1993-02-28	460118.47
121546	1994-10-20	489099.53
52516	1995-08-15	499753.01
56620	1996-05-22	498599.91
54602	1997-02-09	471220.08
100159	1998-07-28	502742.76

7 Row(s) produced. Time Elapsed: 0.228s

8.2.2. 상관 관계가 있는 서브 쿼리

지금까지 다룬 모든 서브쿼리는 포함하는 문장과 독립적이었습니다. 즉, 포함하는 문장이 실행되기 전에 서브쿼리가 먼저 한 번 실행된다는 의미입니다. 반면에, 상관 서브쿼리는 포함하는 문장의 열을 하나 이상 참조하며, 이는 서브쿼리와 포함하는 쿼리가 함께 실행되어야 함을 의미합니다 (개념적으로는 외부 쿼리의 각 행에 대해 서브쿼리가 실행될 수 있습니다).

```
SELECT c.c_name
FROM customer c
WHERE 150000 <= (
    SELECT SUM(o.o_totalprice)
    FROM orders o
    WHERE o.o_custkey = c.c_custkey
)
LIMIT 10;
```

C_NAME
Customer#000007033
Customer#000036316
Customer#000037825
Customer#000066103
Customer#000067051
Customer#000045088
Customer#000039604
Customer#000041581
Customer#000078727
Customer#000120202

10 Row(s) produced. Time Elapsed: 0.345s

상관 서브쿼리는 종종 EXISTS 연산자와 함께 사용되는데, 이는 양에 관계없이 특정 관계의 존재 여부만을 확인하고자 할 때 유용합니다.

```
SELECT c.c_name
FROM customer c
WHERE EXISTS (
    SELECT 1
    FROM orders o
    WHERE o.o_custkey = c.c_custkey
    AND o.o_totalprice > 500000
);
```

C_NAME
Customer#000008936
Customer#000021433
Customer#000100159

3 Row(s) produced. Time Elapsed: 0.878s

8.2.2. 상관 관계가 있는 서브 쿼리

```
DELETE FROM customer c
WHERE NOT EXISTS (
    SELECT 1
    FROM orders o
    WHERE o.o_custkey = c.c_custkey
    AND o.o_orderdate > DATEADD(CURRENT_DATE, -5, 'YEAR')
);
```

```
ALTER TABLE employee_sample ADD inactive VARCHAR(1);
```

```
+-----+
| status |
+-----+
| Statement executed successfully. |
+-----+
1 Row(s) produced. Time Elapsed: 0.131s
```

```
UPDATE customer c
SET inactive = 'Y'
WHERE NOT EXISTS (
    SELECT 1
    FROM orders o
    WHERE o.o_custkey = c.c_custkey
    AND o.o_orderdate > DATEADD(CURRENT_DATE, -5, 'YEAR')
);
```

```
UPDATE employee_sample e
```

```
SET e.inactive = 'Y'
```

```
WHERE NOT EXISTS (
```

```
    SELECT 1
```

```
    FROM person p
```

```
    WHERE p.first_name || ' ' || p.last_name = e.emp_name
```

```
);
```

```
+-----+-----+
| number of rows updated | number of multi-joined rows updated |
+-----+-----+
| 5 | 0 |
+-----+-----+
```

```
5 Row(s) produced. Time Elapsed: 1.400s
```

8.3. 데이터 소스로 서브 쿼리

테이블은 여러 행과 여러 열로 구성되며, 서브쿼리가 반환하는 결과 집합 또한 여러 행과 열로 이루어져 있습니다. 따라서 테이블과 서브쿼리 모두 쿼리의 FROM 절에 사용될 수 있으며, 서로 조인될 수도 있습니다.

```
SELECT c.c_name, o.total_dollars
FROM (
    SELECT o_custkey, SUM(o_totalprice) AS total_dollars
    FROM orders
    WHERE 1998 = DATE_PART(YEAR, o_orderdate)
    GROUP BY o_custkey
    HAVING SUM(o_totalprice) >= 650000
) o INNER JOIN customer c ON c.c_custkey = o.o_custkey
ORDER BY 1
LIMIT 10;
```

C_NAME	TOTAL_DOLLARS
Customer#000002948	663115.18
Customer#000004309	719354.94
Customer#000005059	734893.37
Customer#000022924	686947.21
Customer#000026729	654376.71
Customer#000033487	727194.76
Customer#000044116	699699.98
Customer#000061120	656770.28
Customer#000065434	1030712.34
Customer#000074695	665357.12

10 Row(s) produced. Time Elapsed: 0.344s

8.3.1. CTE 테이블

서브쿼리를 FROM 절에 넣는 방법 외에도, 서브쿼리를 WITH 절로 옮겨 정의하는 방법이 있습니다. WITH 절은 항상 쿼리의 가장 앞, SELECT 절보다 먼저 위치해야 합니다.

```
WITH big_orders AS (  
    SELECT o_custkey, SUM(o_totalprice) AS total_dollars FROM orders  
    WHERE 1998 = DATE_PART(YEAR, o_orderdate)  
    GROUP BY o_custkey  
    HAVING SUM(o_totalprice) >= 650000  
)  
SELECT c.c_name, big_orders.total_dollars  
FROM big_orders INNER JOIN customer c ON c.c_custkey = big_orders.o_custkey  
ORDER BY 1 LIMIT 10;
```

C_NAME	TOTAL_DOLLARS
Customer#000002948	663115.18
Customer#000004309	719354.94
Customer#000005059	734893.37
Customer#000022924	686947.21
Customer#000026729	654376.71
Customer#000033487	727194.76
Customer#000044116	699699.98
Customer#000061120	656770.28
Customer#000065434	1030712.34
Customer#000074695	665357.12

10 Row(s) produced. Time Elapsed: 1.126s

8.3.1. CTE 테이블

WITH 절 안의 서브쿼리는 공통 테이블 표현식(Common Table Expression), 즉 CTE라고 합니다. 단일 CTE를 사용하면 쿼리의 가독성을 높일 수 있으며, WITH 절 안에 여러 CTE가 있을 경우, 동일한 WITH 절 내에서 앞서 정의된 다른 CTE를 참조할 수도 있습니다.

```
WITH big_orders AS (  
    SELECT o_custkey, SUM(o_totalprice) AS total_dollars FROM orders  
    WHERE 1998 = DATE_PART(YEAR, o_orderdate)  
    GROUP BY o_custkey HAVING SUM(o_totalprice) >= 650000  
)  
big_orders_with_names AS (  
    SELECT c.c_name, big_orders.total_dollars FROM big_orders  
    INNER JOIN customer c ON c.c_custkey = big_orders.o_custkey  
)  
SELECT * FROM big_orders_with_names ORDER BY 1 LIMIT 10;
```

C_NAME	TOTAL_DOLLARS
Customer#000002948	663115.18
Customer#000004309	719354.94
Customer#000005059	734893.37
Customer#000022924	686947.21
Customer#000026729	654376.71
Customer#000033487	727194.76
Customer#000044116	699699.98
Customer#000061120	656770.28
Customer#000065434	1030712.34
Customer#000074695	665357.12

8.3.1. CTE 테이블

```
WITH dollar_ranges AS (  
    SELECT * FROM (  
        VALUES (3, 'Bottom Tier', 650000, 700000),  
        (2, 'Middle Tier', 700001, 730000),  
        (1, 'Top Tier', 730001, 9999999)  
    ) AS dr (range_num, range_name, low_val, high_val)  
),  
big_orders AS (  
    SELECT o_custkey, SUM(o_totalprice) AS total_dollars FROM orders  
    WHERE 1998 = DATE_PART(YEAR, o_orderdate)  
    GROUP BY o_custkey HAVING SUM(o_totalprice) >= 650000  
),  
big_orders_with_names AS (  
    SELECT c.c_name, big_orders.total_dollars FROM big_orders  
    INNER JOIN customer c ON c.c_custkey = big_orders.o_custkey  
)  
SELECT dr.range_name,  
    SUM(ROUND(bon.total_dollars, 0)) AS rng_sum,  
    LISTAGG(bon.c_name, ',') WITHIN GROUP (ORDER BY bon.c_name) AS name_list  
FROM big_orders_with_names AS bon INNER JOIN dollar_ranges AS dr  
    ON bon.total_dollars BETWEEN dr.low_val AND dr.high_val  
GROUP BY dr.range_name;
```

- CTE(공통 테이블 표현식)를 사용하는 것은 자바나 파이썬에서 함수를 만드는 것과 약간 비슷합니다. 하나의 코드(쿼리) 내에서 (CTE로 정의된) 서브쿼리를 원하는 만큼 여러 번 사용할 수 있다는 점에서는 그렇습니다. 차이점은 SQL에서는 문장 실행이 완료되면 CTE의 결과가 폐기된다는 점입니다.
- 또한 CTE는 데이터베이스에 존재하지 않는 데이터 집합을 임시로 만드는 데에도 사용될 수 있습니다.

RANGE_NAME	RNG_SUM	NAME_LIST
Top Tier	1765605	Customer#000005059, Customer#000065434
Middle Tier	2892065	Customer#000004309, Customer#000033487, Customer#000090608, Customer#000138724
Bottom Tier	6722566	Customer#000002948, Customer#000022924, Customer#000026729, Customer#000044116, Customer#000061120, Customer#000074695, Customer#000074807, Customer#000097519, Customer#000098410, Customer#000102904

8.4. 복습 하기

? 연습 8-1

1997년에 정확히 4개의 주문을 한 모든 고객에 대해 `c_custkey` 및 `c_name` 열을 반환하는 `Customer` 테이블에 대한 쿼리를 작성하십시오. `Orders` 테이블에 대해 상관 없는 서브쿼리를 사용하십시오.

? 연습 8-2

연습 문제 8-1의 쿼리를 수정하여 상관된 서브쿼리를 사용하여 동일한 결과를 반환하십시오.

8.4. 복습 하기(정답)

100 연습 8-1

```
SELECT c_custkey, c_name
FROM customer
WHERE c_custkey IN (
    SELECT o_custkey
    FROM orders
    WHERE DATE_PART(YEAR, o_orderdate) = 1997
    GROUP BY o_custkey
    HAVING COUNT(*) = 4
);
```

100 연습 8-2

```
SELECT c_custkey, c_name
FROM customer c
WHERE 4 = (
    SELECT COUNT(*)
    FROM orders o
    WHERE DATE_PART(YEAR, o_orderdate) = 1997
    AND o.o_custkey = c.c_custkey
);
```

9. 계층적 From 절

9.1. 계층적 쿼리

```
SELECT empid, emp_name, mgr_empid FROM employee_sample
ORDER BY 1;
```

EMPID	EMP_NAME	MGR_EMPID
1001	Bob Smith	NULL
1002	Susan Jackson	1001
1003	Greg Carpenter	1001
1004	Robert Butler	1002
1005	Kim Josephs	1003
1006	John Tyler	1004

6 Row(s) produced. Time Elapsed: 0.105s

```
SELECT e_1.emp_name, e_2.emp_name, e_3.emp_name, e_4.emp_name
FROM employee_sample e_1
INNER JOIN employee_sample e_2
    ON e_1.mgr_empid = e_2.empid
INNER JOIN employee_sample e_3
    ON e_2.mgr_empid = e_3.empid
INNER JOIN employee_sample e_4
    ON e_3.mgr_empid = e_4.empid
WHERE e_1.emp_name = 'John Tyler';
```

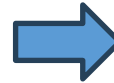
EMP_NAME	EMP_NAME	EMP_NAME	EMP_NAME
John Tyler	Robert Butler	Susan Jackson	Bob Smith

1 Row(s) produced. Time Elapsed: 0.836s

```
SELECT emp_name FROM employee_sample
START WITH emp_name = 'John Tyler'
CONNECT BY PRIOR mgr_empid = empid;
```

EMP_NAME
John Tyler
Robert Butler
Susan Jackson
Bob Smith

4 Row(s) produced. Time Elapsed: 0.665s



```
SELECT emp_name FROM employee_sample
START WITH emp_name = 'Bob Smith'
CONNECT BY PRIOR empid = mgr_empid;
```

EMP_NAME
Bob Smith
Susan Jackson
Greg Carpenter
Robert Butler
Kim Josephs
John Tyler

6 Row(s) produced. Time Elapsed: 0.545s

9.1.1. SYS_CONNECT_BY_PATH

이번에는 Bob Smith의 행이 시작점이며, 위로 탐색하는 대신 트리 아래로 내려가기 때문에 CONNECT BY 절은 반대로 작성되었습니다. (자신을 포함하여) 모든 직원이 궁극적으로 Bob Smith에게 보고하므로, 여섯 명의 직원이 모두 결과에 포함됩니다.

하지만 이 결과만으로는 Robert Butler가 Susan Jackson에게 보고하는 것과 같은 중간 관계는 알 수 없습니다. 이러한 관계를 확인하려면, 해당 지점까지의 전체 계층 구조 설명을 보여주는 내장 함수 sys_connect_by_path()를 사용할 수 있습니다.

```
SELECT emp_name,  
       SYS_CONNECT_BY_PATH(emp_name, ' : ') AS management_path  
FROM employee_sample  
START WITH emp_name = 'Bob Smith'  
CONNECT BY PRIOR empid = mgr_empid;
```

EMP_NAME	MANAGEMENT_PATH
Bob Smith	: Bob Smith
Susan Jackson	: Bob Smith : Susan Jackson
Greg Carpenter	: Bob Smith : Greg Carpenter
Robert Butler	: Bob Smith : Susan Jackson : Robert Butler
Kim Josephs	: Bob Smith : Greg Carpenter : Kim Josephs
John Tyler	: Bob Smith : Susan Jackson : Robert Butler : John Tyler

6 Row(s) produced. Time Elapsed: 0.579s

9.2. 시간 추적

```
INSERT INTO employee_sample (empid, emp_name, mgr_empid)
VALUES (9999, 'Tim Traveler', 1006);
```

number of rows inserted
1

1 Row(s) produced. Time Elapsed: 1.073s

```
SELECT empid, emp_name, mgr_empid FROM employee_sample;
```

EMPID	EMP_NAME	MGR_EMPID
1001	Bob Smith	NULL
1002	Susan Jackson	1001
1004	Robert Butler	1002
1005	Kim Josephs	1003
1006	John Tyler	1004
1003	Greg Carpenter	1001
9999	Tim Traveler	1006

7 Row(s) produced. Time Elapsed: 0.145s

스노우플레이크의 타임 트래블 기능은 과거 특정 시점의 데이터 상태 그대로 쿼리를 실행할 수 있게 해줍니다. 이를 위해 특정 시간을 지정하거나 현재 시간으로부터의 오프셋을 지정하는 AT 키워드를 사용할 수 있으며, 그러면 스노우플레이크는 해당 시점의 상태 그대로 데이터를 검색합니다.

```
SELECT empid, emp_name, mgr_empid
FROM employee_sample AT(OFFSET => -3600);
```

EMPID	EMP_NAME	MGR_EMPID
1001	Bob Smith	NULL
1002	Susan Jackson	1001
1004	Robert Butler	1002
1005	Kim Josephs	1003
1006	John Tyler	1004
1003	Greg Carpenter	1001

6 Row(s) produced. Time Elapsed: 0.293s

```
SELECT empid, emp_name, mgr_empid
FROM employee_sample
MINUS
SELECT empid, emp_name, mgr_empid
FROM employee_sample AT(OFFSET => -3600);
```

EMPID	EMP_NAME	MGR_EMPID
9999	Tim Traveler	1006

1 Row(s) produced. Time Elapsed: 0.592s

9.3. 피벗 쿼리

피벗(Pivoting)은 데이터 분석에서 흔한 작업으로, 데이터의 행(row)을 열(column)로 변환하는 것을 말합니다.

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       ROUND(SUM(o_totalprice)) AS total_sales  
FROM orders  
WHERE 1995 <= DATE_PART(YEAR, o_orderdate)  
GROUP BY DATE_PART(YEAR, o_orderdate)  
ORDER BY 1;
```

YEAR	TOTAL_SALES
1995	3317521810
1996	3296373353
1997	3255086721
1998	1925196449



```
SELECT ROUND(yr_1995) AS "1995_sales", ROUND(yr_1996) AS "1996_sales",  
       ROUND(yr_1997) AS "1997_sales", ROUND(yr_1998) AS "1998_sales"  
FROM (  
    SELECT DATE_PART(YEAR, o_orderdate) AS year, o_totalprice  
    FROM orders  
    WHERE 1995 <= DATE_PART(YEAR, o_orderdate)  
) PIVOT (  
    SUM(o_totalprice) FOR year IN (1995, 1996, 1997, 1998)  
) AS pvt(yr_1995, yr_1996, yr_1997, yr_1998);
```

1995_sales	1996_sales	1997_sales	1998_sales
3317521810	3296373353	3255086721	1925196449

9.3.1. UNPIVOT

```
SELECT ROUND(yr_1995) AS "1995_sales",
       ROUND(yr_1996) AS "1996_sales",
       ROUND(yr_1997) AS "1997_sales",
       ROUND(yr_1998) AS "1998_sales"
FROM (
  SELECT DATE_PART(YEAR, o_orderdate) AS year, o_totalprice
  FROM orders
  WHERE 1995 <= DATE_PART(YEAR, o_orderdate)
) PIVOT (
  SUM(o_totalprice) FOR year IN (1995, 1996, 1997, 1998)
) AS pvt(yr_1995, yr_1996, yr_1997, yr_1998);
```

1995_sales	1996_sales	1997_sales	1998_sales
3317521810	3296373353	3255086721	1925196449

스노우플레이크는 반대 변환(열의 데이터를 행으로 되돌리는 것)을 수행하는 UNPIVOT 절도 제공합니다.

```
WITH year_pvt AS (
  SELECT ROUND(yr_1995) AS "1995", ROUND(yr_1996) AS "1996",
         ROUND(yr_1997) AS "1997", ROUND(yr_1998) AS "1998"
  FROM (
    SELECT DATE_PART(YEAR, o_orderdate) AS year, o_totalprice
    FROM orders
    WHERE 1995 <= DATE_PART(YEAR, o_orderdate)
  ) PIVOT (
    SUM(o_totalprice) FOR year IN (1995, 1996, 1997, 1998)
  ) AS pvt(yr_1995, yr_1996, yr_1997, yr_1998)
)
SELECT *
FROM year_pvt UNPIVOT (
  total_sales FOR year IN ("1995", "1996", "1997", "1998")
);
```

YEAR	TOTAL_SALES
1995	3317521810
1996	3296373353
1997	3255086721
1998	1925196449

4 Row(s) produced. Time Elapsed: 1.306s

9.4. 랜덤 샘플링

테스트와 같은 작업을 위해 테이블의 일부만을 검색하는 것이 유용할 때가 있습니다. 또한 매번 다른 부분 집합을 얻고 싶을 수도 있습니다. 이러한 목적을 위해, 스노우플레이크는 반환받고자 하는 행의 비율을 지정할 수 있는 SAMPLE 절을 제공합니다.

```
SELECT s_suppkey, s_name, s_nationkey
FROM supplier SAMPLE (0.1);
```

S_SUPPKEY	S_NAME	S_NATIONKEY
5351	Supplier#000005351	10
6818	Supplier#000006818	4
4751	Supplier#000004751	15
7789	Supplier#000007789	3
2922	Supplier#000002922	23
1978	Supplier#000001978	5

6 Row(s) produced. Time Elapsed: 0.676s

```
SELECT s_suppkey, s_name, s_nationkey
FROM supplier SAMPLE (10 ROWS);
```

S_SUPPKEY	S_NAME	S_NATIONKEY
6542	Supplier#000006542	9
8487	Supplier#000008487	13
4504	Supplier#000004504	23
8581	Supplier#000008581	17
8379	Supplier#000008379	0
5570	Supplier#000005570	16
60	Supplier#000000060	8
230	Supplier#000000230	1
9201	Supplier#000009201	17
2069	Supplier#000002069	21

10 Row(s) produced. Time Elapsed: 0.549s

9.5. Full Outer 조인

```
SELECT orders.ordernum, orders.custkey, customer.custname
FROM (
    VALUES (990, 101), (991, 102), (992, 101), (993, 104)
) AS orders (ordernum, custkey)
FULL OUTER JOIN (
    VALUES (101, 'BOB'), (102, 'KIM'), (103, 'JIM')
) AS customer (custkey, custname)
ON orders.custkey = customer.custkey;
```

ORDERNUM	CUSTKEY	CUSTNAME
990	101	BOB
991	102	KIM
992	101	BOB
993	104	NULL
NULL	NULL	JIM

5 Row(s) produced. Time Elapsed: 0.447s

```
SELECT orders.ordernum,
    NVL(orders.custkey, customer.custkey) AS custkey,
    customer.custname
FROM (
    VALUES (990, 101), (991, 102), (992, 101), (993, 104)
) AS orders (ordernum, custkey)
FULL OUTER JOIN (
    VALUES (101, 'BOB'), (102, 'KIM'), (103, 'JIM')
) AS customer (custkey, custname)
ON orders.custkey = customer.custkey;
```

ORDERNUM	CUSTKEY	CUSTNAME
990	101	BOB
991	102	KIM
992	101	BOB
993	104	NULL
NULL	103	JIM

5 Row(s) produced. Time Elapsed: 0.288s

9.6. Lateral 조인

```
SELECT ord.o_orderdate, ord.o_totalprice,
       cst.c_name, cst.c_address
FROM orders ord INNER JOIN LATERAL (
  SELECT c.c_name, c.c_address
  FROM customer c
  WHERE c.c_custkey = ord.o_custkey
) cst
WHERE 1995 <= DATE_PART(YEAR, ord.o_orderdate)
      AND ord.o_totalprice > 475000;
```

O_ORDERDATE	O_TOTALPRICE	C_NAME	C_ADDRESS
1995-08-15	499753.01	Customer#000052516	BUePeY10PR3 35zwkJF4NA7FKE8gKtI0cR
1996-05-22	498599.91	Customer#000056620	QAnxRzFcVPARTjvvG3SvYnfCOMVqR5 yX
1996-09-16	491096.90	Customer#000111926	yDC67043iroadcywMl
1998-07-28	502742.76	Customer#000100159	fcjfnCnKTf4wvvY0Nq9p,aYnTLmf1rpbu

4 Row(s) produced. Time Elapsed: 0.071s

스노우플레이크는 FROM 절의 서브쿼리가 같은 FROM 절 내의 다른 테이블을 참조하는 것을 허용하며, 이는 해당 서브쿼리가 마치 상관 서브쿼리처럼 동작함을 의미합니다. 이는 LATERAL 키워드를 지정하여 수행하면 됩니다.

```
SELECT ord.o_orderdate, ord.o_totalprice,
       li.num_line_items, li.last_shipdate
FROM orders ord INNER JOIN LATERAL (
  SELECT COUNT(*) AS num_line_items,
         MAX(l_shipdate) AS last_shipdate
  FROM lineitem AS l
  WHERE l.l_orderkey = ord.o_orderkey
) li
WHERE 1995 <= DATE_PART(YEAR, ord.o_orderdate)
      AND ord.o_totalprice > 475000;
```

O_ORDERDATE	O_TOTALPRICE	NUM_LINE_ITEMS	LAST_SHIPDATE
1996-09-16	491096.90	1	1996-10-23
1996-05-22	498599.91	1	1996-08-28
1995-08-15	499753.01	1	1995-10-01
1998-07-28	502742.76	1	1998-11-25

4 Row(s) produced. Time Elapsed: 0.810s

9.7. 테이블 리터럴

스노우플레이크에서는 table() 함수를 사용하여 테이블 이름을 문자열 형태로 쿼리에 전달할 수 있습니다.

```
SELECT * FROM TABLE('region');
```

R_REGIONKEY	R_NAME	R_COMMENT
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are final requests. regular accounts are according to
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. carefully final asymptotes haggle furiousl

5 Row(s) produced. Time Elapsed: 0.804s

```
SELECT * FROM TABLE('learning_sql.public.region');
```

R_REGIONKEY	R_NAME	R_COMMENT
0	AFRICA	lar deposits. blithely final packages cajole. regular waters are final requests. regular accounts are according to
1	AMERICA	hs use ironic, even requests. s
2	ASIA	ges. thinly even pinto beans ca
3	EUROPE	ly final courts cajole furiously final excuse
4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. carefully final asymptotes haggle furiousl

5 Row(s) produced. Time Elapsed: 0.706s

9.8. 복습 하기

? 연습 9-1

다음 쿼리는 각 시장 세그먼트의 고객 수를 반환합니다.

```
SELECT c_mktsegment AS mktseg, count(*) tot_custs
FROM customer
GROUP BY c_mktsegment;
```

MKTSEG	TOT_CUSTS
MACHINERY	13185
AUTOMOBILE	13192
FURNITURE	13125
BUILDING	13360
HOUSEHOLD	13214

이 쿼리를 피벗 쿼리의 기초로 사용하여 5개의 열을 가진 단일 행이 생성되도록 하고, 각 열은 시장 세그먼트의 이름을 갖도록 하십시오.

9.8. 복습 하기(정답)

100 연습 9-1

```
SELECT automobile, machinery, building, furniture, household
FROM (
    SELECT c_mktsegment AS mktseg, count(*) AS tot_custs
    FROM customer
    GROUP BY c_mktsegment
) PIVOT (
    MAX(tot_custs)
    FOR mktseg IN ('AUTOMOBILE', 'MACHINERY', 'BUILDING', 'FURNITURE', 'HOUSEHOLD')
) AS pvt(automobile, machinery, building, furniture, household);
```

10. 조건 논리

10.1. 조건 논리가 뭘 까요?

```
SELECT c_custkey, c_name, c_acctbal,  
CASE  
    WHEN c_acctbal < 0 THEN 'generate refund'  
    WHEN c_acctbal = 0 THEN 'no action'  
    ELSE 'send bill'  
END AS month_end_action  
FROM customer  
LIMIT 15;
```

C_CUSTKEY	C_NAME	C_ACCTBAL	MONTH_END_ACTION
30001	Customer#000030001	8848.47	send bill
30004	Customer#000030004	3308.55	send bill
30005	Customer#000030005	-278.54	generate refund
30007	Customer#000030007	3912.67	send bill
30010	Customer#000030010	8599.71	send bill
30011	Customer#000030011	4442.02	send bill
30016	Customer#000030016	6670.55	send bill
30017	Customer#000030017	8992.52	send bill
30019	Customer#000030019	1848.59	send bill
30020	Customer#000030020	3144.52	send bill
30023	Customer#000030023	5299.36	send bill
30025	Customer#000030025	6615.97	send bill
30028	Customer#000030028	2347.74	send bill
30031	Customer#000030031	-541.78	generate refund
30034	Customer#000030034	5590.43	send bill

15 Row(s) produced. Time Elapsed: 0.668s

특정 상황에서는 특정 열이나 표현식의 값에 따라 SQL 문이 다르게 동작하기를 원할 수 있는데, 이를 조건부 논리 라고 합니다. SQL 문에서 조건부 논리를 위해 사용되는 메커니즘은 CASE 표현식이며, 이는 INSERT, UPDATE, DELETE 문뿐만 아니라 SELECT 문의 모든 절에서도 활용될 수 있습니다.

```
SELECT num.val,  
CASE  
    WHEN num.val > 90 THEN 'huge number'  
    WHEN num.val > 50 THEN 'really big number'  
    WHEN num.val > 20 THEN 'big number'  
    WHEN num.val > 10 THEN 'medium number'  
    WHEN num.val <= 10 THEN 'small number'  
END AS num_size  
FROM (  
    VALUES (11), (12), (25), (99), (3)  
) AS num (val);
```

VAL	NUM_SIZE
11	medium number
12	medium number
25	big number
99	huge number
3	small number

5 Row(s) produced. Time Elapsed: 0.245s

10.2. CASE 구문의 종류

Searched Case Expressions

- 검색된 CASE 표현식은 여러 개의 WHEN 절을 가질 수 있으며, 어떤 WHEN 절도 참으로 평가되지 않을 경우 반환될 선택적 ELSE 절을 가질 수 있습니다.

Simple Case Expressions

- 이러한 유형의 명령문에서는 표현식이 평가되어 값의 집합과 비교됩니다. 일치하는 항목이 발견되면 해당 표현식이 반환되고, 일치하는 항목이 없으면 선택적 ELSE 절의 표현식이 반환됩니다.

10.2.1. 검색된 CASE 구문

```
SELECT p_partkey, p_retailprice,  
       CASE  
         WHEN p_retailprice > 2000 THEN  
           (SELECT COUNT(*)  
            FROM lineitem li  
            WHERE li.l_partkey = p.p_partkey)  
         ELSE 0  
       END AS num_bigticket_orders  
FROM part p  
WHERE p_retailprice BETWEEN 1990 AND 2010  
LIMIT 10;
```

P_PARTKEY	P_RETAILPRICE	NUM_BIGTICKET_ORDERS
140958	1998.95	0
136958	1994.95	0
184908	1992.90	0
194908	2002.90	26
198908	2006.90	31
135958	1993.95	0
139958	1997.95	0
190908	1998.90	0
151958	2009.95	36
134958	1992.95	0

검색된 CASE 표현식은 여러 개의 WHEN 절을 가질 수 있으며, 어떤 WHEN 절도 참으로 평가되지 않을 경우 반환될 선택적인 ELSE 절을 포함할 수 있습니다. CASE 표현식은 숫자, 문자열, 날짜는 물론 심지어 서브쿼리까지 어떤 타입의 표현식이든 반환할 수 있습니다.

10.2.2. 간단한 CASE 구문

```
SELECT o_orderkey,  
       CASE o_orderstatus  
         WHEN 'P' THEN 'Partial'  
         WHEN 'F' THEN 'Filled'  
         WHEN 'O' THEN 'Open'  
       END AS status  
FROM orders  
LIMIT 10;
```

O_ORDERKEY	STATUS
600006	Open
600037	Filled
600064	Open
600065	Filled
600132	Open
600165	Filled
600228	Filled
600262	Open
600327	Open
600484	Open

10 Row(s) produced. Time Elapsed: 0.780s

단순 CASE 표현식에서는, 하나의 표현식이 평가된 후 여러 값들과 비교됩니다. 일치하는 값을 찾으면 해당하는 THEN 다음의 표현식이 반환되고, 일치하는 값을 찾지 못하면 선택적인 ELSE 절의 표현식이 반환됩니다.

10.3. CASE 구문의 사용

- Pivot Operations
- Checking for Existence
- Conditional Updates

10.3.1. 피봇 연산

```
SELECT DATE_PART(YEAR, o_orderdate), ROUND(SUM(o_totalprice))
FROM orders
WHERE DATE_PART(YEAR, o_orderdate) >= 1995
GROUP BY DATE_PART(YEAR, o_orderdate)
ORDER BY 1;
```

DATE_PART(YEAR, O_ORDERDATE)	ROUND(SUM(O_TOTALPRICE))
1995	3317521810
1996	3296373353
1997	3255086721
1998	1925196449

네 개의 열 각각은 CASE 표현식이 반환하는 값들을 합산(sum)하며, 각 CASE 표현식은 주문이 지정된 연도(1995, 1996, 1997 또는 1998)에 이루어진 경우에만 0이 아닌 값을 반환합니다. 각 열 값이 집계 함수(이 예시에서는 sum())를 사용하여 생성되므로, GROUP BY 절은 필요하지 않습니다.

```
SELECT
    ROUND(SUM(CASE WHEN 1995 = DATE_PART(YEAR, o_orderdate) THEN o_totalprice ELSE 0 END)) AS "1995",
    ROUND(SUM(CASE WHEN 1996 = DATE_PART(YEAR, o_orderdate) THEN o_totalprice ELSE 0 END)) AS "1996",
    ROUND(SUM(CASE WHEN 1997 = DATE_PART(YEAR, o_orderdate) THEN o_totalprice ELSE 0 END)) AS "1997",
    ROUND(SUM(CASE WHEN 1998 = DATE_PART(YEAR, o_orderdate) THEN o_totalprice ELSE 0 END)) AS "1998"
FROM orders
WHERE DATE_PART(YEAR, o_orderdate) >= 1995;
```

1995	1996	1997	1998
3317521810	3296373353	3255086721	1925196449

10.3.2. 존재 여부 확인

발생 횟수와 관계없이 특정 관계의 존재 여부만 알아야 하는 상황에 처할 수 있습니다. 예를 들어, 고객이 \$400,000를 초과하는 주문을 한 번이라도 했는지 여부를 알고 싶다고 가정해 봅시다. 이때 그러한 주문을 몇 번 했는지는 중요하지 않습니다.

```
SELECT c_custkey, c_name,  
       CASE WHEN EXISTS (SELECT 1 FROM orders o WHERE o.o_custkey = c.c_custkey AND o.o_totalprice > 400000)  
             THEN 'Big Spender' ELSE 'Regular'  
       END AS cust_type  
FROM customer c  
WHERE c_custkey BETWEEN 74000 AND 74020;
```

C_CUSTKEY	C_NAME	CUST_TYPE
74011	Customer#000074011	Big Spender
74003	Customer#000074003	Big Spender
74017	Customer#000074017	Regular
74015	Customer#000074015	Regular
74012	Customer#000074012	Regular
74020	Customer#000074020	Regular
74009	Customer#000074009	Regular
74000	Customer#000074000	Regular
74014	Customer#000074014	Regular
74008	Customer#000074008	Regular
74005	Customer#000074005	Regular

11 Row(s) produced. Time Elapsed: 0.738s

10.3.3. 조건부 업데이트

```
ALTER TABLE customer ADD c_cust_type VARCHAR(50);
```

```
+-----+
| status |
+-----+
| Statement executed successfully. |
+-----+
1 Row(s) produced. Time Elapsed: 0.203s
```

```
UPDATE customer AS c
SET c_cust_type = CASE
    WHEN EXISTS (SELECT 1 FROM orders o
        WHERE o.o_custkey = c.c_custkey
        AND o.o_totalprice > 400000) THEN 'Big Spender'
    ELSE 'Regular'
END;
```

```
+-----+-----+
| number of rows updated | number of multi-joined rows updated |
+-----+-----+
| 66076 | 0 |
+-----+-----+
66076 Row(s) produced. Time Elapsed: 1.889s
```



```
SELECT c_custkey, c_name, c_cust_type
FROM customer
WHERE c_custkey BETWEEN 74000 AND 74020
LIMIT 10;
```

```
+-----+-----+-----+
| C_CUSTKEY | C_NAME | C_CUST_TYPE |
+-----+-----+-----+
| 74000 | Customer#000074000 | Regular |
| 74003 | Customer#000074003 | Big Spender |
| 74005 | Customer#000074005 | Regular |
| 74008 | Customer#000074008 | Regular |
| 74009 | Customer#000074009 | Regular |
| 74011 | Customer#000074011 | Big Spender |
| 74012 | Customer#000074012 | Regular |
| 74014 | Customer#000074014 | Regular |
| 74015 | Customer#000074015 | Regular |
| 74017 | Customer#000074017 | Regular |
+-----+-----+-----+
10 Row(s) produced. Time Elapsed: 0.706s
```

10.3.3. 조건부 업데이트

이 구문은 \$1,000를 초과하는 주문을 한 번도 한 적이 없거나 1996년 또는 그 이후에 주문한 기록이 없는 고객들을 제거할 것입니다. 만약 데이터를 제거(remove)했다가 다시 되돌리고 싶다면, 언제든지 스노우플레이크의 타임 트래블기능을 사용하여 데이터를 복구할 수 있습니다.

```
DELETE FROM customer c
WHERE 1 = CASE WHEN NOT EXISTS (
    SELECT 1
    FROM orders o
    WHERE o.o_custkey = c.c_custkey
    AND o.o_totalprice > 1000)
    THEN 1
    WHEN '31-DEC-1995' > (
        SELECT MAX(o_orderdate)
        FROM orders o
        WHERE o.o_custkey = c.c_custkey)
    THEN 1
    ELSE 0
END;
```

```
+-----+
| number of rows deleted |
+-----+
|           30426       |
+-----+
```

30426 Row(s) produced. Time Elapsed: 1.696s

10.4. 조건 논리를 위한 함수

- iff() Function
- ifnull() and nvl() Functions
- decode() Function

10.4.1. IFF() 함수

```
SELECT c_custkey, c_name,  
       CASE WHEN EXISTS (SELECT 1  
                          FROM orders o  
                          WHERE o.o_custkey = c.c_custkey  
                                AND o.o_totalprice > 400000  
                          ) THEN 'Big Spender' ELSE 'Regular'  
       END AS cust_type_case,  
       IFF(EXISTS (SELECT 1  
                  FROM orders o  
                  WHERE o.o_custkey = c.c_custkey  
                        AND o.o_totalprice > 400000  
                  ), 'Big Spender', 'Regular') AS cust_type_iff  
FROM customer c  
WHERE c_custkey BETWEEN 74000 AND 74020;
```

C_CUSTKEY	C_NAME	CUST_TYPE_CASE	CUST_TYPE_IFF
74011	Customer#000074011	Big Spender	Big Spender
74003	Customer#000074003	Big Spender	Big Spender
74020	Customer#000074020	Regular	Regular
74009	Customer#000074009	Regular	Regular
74017	Customer#000074017	Regular	Regular
74008	Customer#000074008	Regular	Regular

6 Row(s) produced. Time Elapsed: 0.933s

- 단일 조건을 가진 간단한 if-then-else 표현식만 필요한 경우 iff() 함수를 사용할 수 있습니다.
- CASE 표현식과 iff() 함수 모두 단일 조건(orders 테이블에 행이 존재하는지 여부)을 평가하여 'Big Spender' 또는 'Regular' 문자열을 결과로 반환합니다. 여러 조건을 평가해야 하는 경우에는 iff() 함수를 사용할 수 없다는 점에 유의하세요.

10.4.2. IFNULL(), NVL() 함수

```
SELECT name,  
       NVL(favorite_color, 'Unknown') AS favorite_color_nvl,  
       IFNULL(favorite_color, 'Unknown') AS favorite_color_isnull  
FROM (VALUES ('Thomas', 'yellow'), ('Catherine', 'red'),  
            ('Richard', 'blue'), ('Rebecca', null)  
      ) AS person (name, favorite_color);
```

NAME	FAVORITE_COLOR_NVL	FAVORITE_COLOR_ISNULL
Thomas	yellow	yellow
Catherine	red	red
Richard	blue	blue
Rebecca	Unknown	Unknown

4 Row(s) produced. Time Elapsed: 0.512s

보고서를 작성할 때, 열 값이 NULL인 경우 'unknown'이나 'N/A' 같은 특정 값으로 대체하고 싶은 상황에 자주 마주치게 될 것입니다. 이러한 상황에서는 ifnull() 또는 nvl() 함수를 사용할 수 있습니다.

```
SELECT orders.ordernum,  
       CASE  
         WHEN orders.custkey IS NOT NULL THEN orders.custkey  
         WHEN customer.custkey IS NOT NULL THEN customer.custkey  
       END AS custkey_case,  
       NVL(orders.custkey, customer.custkey) AS custkey_nvl,  
       IFNULL(orders.custkey, customer.custkey) AS custkey_ifnull,  
       customer.custname AS name  
FROM (  
  VALUES (990, 101), (991, 102), (992, 101), (993, 104)  
) AS orders (ordernum, custkey) FULL OUTER JOIN (  
  VALUES (101, 'BOB'), (102, 'KIM'), (103, 'JIM')  
) AS customer (custkey, custname)  
ON orders.custkey = customer.custkey;
```

ORDERNUM	CUSTKEY_CASE	CUSTKEY_NVL	CUSTKEY_IFNULL	NAME
990	101	101	101	BOB
991	102	102	102	KIM
992	101	101	101	BOB
993	104	104	104	NULL
NULL	103	103	103	JIM

5 Row(s) produced. Time Elapsed: 0.272s

10.4.3. DECODE() 함수

```
SELECT o_orderkey,  
       CASE o_orderstatus  
         WHEN 'P' THEN 'Partial'  
         WHEN 'F' THEN 'Filled'  
         WHEN 'O' THEN 'Open'  
       END AS status_case,  
       DECODE(o_orderstatus, 'P', 'Partial',  
              'F', 'Filled',  
              'O', 'Open') AS status_decode  
FROM orders  
LIMIT 10;
```

O_ORDERKEY	STATUS_CASE	STATUS_DECODE
600006	Open	Open
600037	Filled	Filled
600064	Open	Open
600065	Filled	Filled
600132	Open	Open
600165	Filled	Filled
600228	Filled	Filled
600262	Open	Open
600327	Open	Open
600484	Open	Open

10 Row(s) produced. Time Elapsed: 0.255s

일부 사람들은 decode()가 더 간결하기 때문에 선호하지만, 저는 case를 사용하는 것을 선호합니다. 왜냐하면 이해하기 쉽고 다른 데이터베이스 서버 간에 이식성이 좋기 때문입니다.

10.5. 복습 하기

? 연습 10-1

다음 쿼리에 order_status라는 열을 추가하여, ps_availqty 값이 100 미만이면 'order now', 101과 1000 사이이면 'order soon', 그렇지 않으면 'plenty in stock' 값을 반환하는 CASE 표현식을 사용하도록 하세요:

```
SELECT ps_partkey, ps_supkey, ps_availqty  
FROM partsupp  
WHERE ps_partkey BETWEEN 148300 AND 148450;
```

PS_PARTKEY	PS_SUPPKEY	PS_AVAILQTY
148308	8309	9570
148308	823	7201
148308	3337	7917
148308	5851	8257
148358	8359	9839
148358	873	6917
148358	3387	1203
148358	5901	1
148408	8409	74
148408	923	341
148408	3437	4847
148408	5951	1985

10.5. 복습 하기(정답)

100 연습 10-1

```
SELECT ps_partkey, ps_suppkey, ps_availqty,  
       CASE  
         WHEN ps_availqty <= 100 THEN 'order now'  
         WHEN ps_availqty <= 1000 THEN 'order soon'  
         ELSE 'plenty in stock'  
       END AS order_status  
FROM partsupp  
WHERE ps_partkey BETWEEN 148300 AND 148450;
```

11. 뷰

11.1. 뷰가 뭘 까요?

- 뷰(View)는 테이블과 유사한 데이터베이스 객체이지만, 뷰는 쿼리만 가능합니다.
- 뷰는 (구체화된 뷰는 제외하고, 이는 나중에 논의됩니다) 데이터 저장과 관련이 없습니다.
- 뷰를 생각하는 한 가지 방법은 쉽게 사용할 수 있도록 데이터베이스에 저장된 명명된 쿼리라고 생각하는 것입니다.
- 매월 마지막 영업일에 보고서를 실행하는 경우, 보고서 생성에 사용된 쿼리를 포함하는 뷰를 생성하고 매달 해당 뷰를 쿼리할 수 있습니다.

11.1.1. 뷰 만들기

```
CREATE VIEW employee_vw
AS
SELECT empid, emp_name, mgr_empid, inactive
FROM employee_sample;
```

status
View EMPLOYEE_VW successfully created.
1 Row(s) produced. Time Elapsed: 0.212s

```
SELECT * FROM employee_vw
LIMIT 10;
```

EMPID	EMP_NAME	MGR_EMPID	INACTIVE
1001	Bob Smith	NULL	NULL
1002	Susan Jackson	1001	Y
1004	Robert Butler	1002	Y
1005	Kim Josephs	1003	Y
1006	John Tyler	1004	Y
1003	Greg Carpenter	1001	Y
9999	Tim Traveler	1006	NULL

7 Row(s) produced. Time Elapsed: 0.711s

```
DESCRIBE employee_vw;
```

name	type	kind	null?	default	primary key	unique key	check	expression	comment	policy name	privacy domain
EMPID	NUMBER(38,0)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
EMP_NAME	VARCHAR(30)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
MGR_EMPID	NUMBER(38,0)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
INACTIVE	VARCHAR(1)	COLUMN	Y	NULL	N	N	NULL	NULL	NULL	NULL	NULL
4 Row(s) produced. Time Elapsed: 0.079s											

11.1.1. 뷰 만들기

```
CREATE VIEW person_vw (fname, lname, dob, eyes)
```

```
AS
```

```
SELECT first_name, last_name, birth_date, eye_color
```

```
FROM person;
```

```
+-----+
| status |
+-----+
| View PERSON_VW successfully created. |
+-----+
1 Row(s) produced. Time Elapsed: 0.196s
```

```
SELECT * FROM person_vw LIMIT 10;
```

```
+-----+-----+-----+-----+
| FNAME | LNAME | DOB | EYES |
+-----+-----+-----+-----+
| Gina | Peters | 2001-03-03 | brown |
| Tim | Carpenter | 2002-07-09 | green |
| Bob | Smith | 2000-01-22 | blue |
| Carl | Langford | 2001-06-16 | blue |
+-----+-----+-----+-----+
4 Row(s) produced. Time Elapsed: 0.652s
```

```
DESCRIBE person_vw;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| name | type | kind | null? | default | primary key | unique key | check | expression | comment | policy name | privacy domain |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| FNAME | VARCHAR(50) | COLUMN | Y | NULL | N | N | NULL | NULL | NULL | NULL | NULL |
| LNAME | VARCHAR(50) | COLUMN | Y | NULL | N | N | NULL | NULL | NULL | NULL | NULL |
| DOB | DATE | COLUMN | Y | NULL | N | N | NULL | NULL | NULL | NULL | NULL |
| EYES | VARCHAR(10) | COLUMN | Y | NULL | N | N | NULL | NULL | NULL | NULL | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
4 Row(s) produced. Time Elapsed: 0.055s
```

11.1.1. 뷰 만들기

```
CREATE VIEW big_spenders_1998_vw (custkey, cust_name, total_order_dollars)
AS
SELECT o_custkey, c.c_name, SUM(o_totalprice)
FROM orders AS o INNER JOIN customer AS c
    ON o.o_custkey = c.c_custkey
WHERE 1998 = DATE_PART(YEAR, o.o_orderdate)
GROUP BY o.o_custkey, c.c_name
HAVING SUM(o.o_totalprice) >= 500000;
```

```
+-----+
| status                                     |
+-----+
| View BIG_SPENDERS_1998_VW successfully created. |
+-----+
1 Row(s) produced. Time Elapsed: 0.145s
```

뷰는 마케팅 및 영업 부서 구성원들에게 유용할 수 있으며,
판매 프로모션 대상 고객을 식별하는 데 도움이 됩니다.

```
SELECT * FROM big_spenders_1998_vw LIMIT 10;
```

```
+-----+-----+-----+
| CUSTKEY | CUST_NAME          | TOTAL_ORDER_DOLLARS |
+-----+-----+-----+
| 103918 | Customer#000103918 | 538378.10           |
| 65434  | Customer#000065434 | 1030712.34          |
| 52015  | Customer#000052015 | 575325.51           |
| 131371 | Customer#000131371 | 585674.44           |
| 44101  | Customer#000044101 | 555095.74           |
| 119080 | Customer#000119080 | 576321.31           |
| 119650 | Customer#000119650 | 512338.32           |
| 110620 | Customer#000110620 | 528555.55           |
| 140092 | Customer#000140092 | 602625.67           |
| 132016 | Customer#000132016 | 527385.88           |
+-----+-----+-----+
10 Row(s) produced. Time Elapsed: 0.588s
```

11.1.2. 뷰 사용하기

```
SELECT p.fname, p.lname, e.empid
FROM person_vw AS p INNER JOIN employee_vw AS e
    ON e.emp_name = CONCAT(p.fname, ' ', p.lname);
```

FNAME	LNAME	EMPID
Bob	Smith	1001

1 Row(s) produced. Time Elapsed: 0.957s

스노우플레이크에서는 뷰를 통해 데이터를 수정할 수는 없지만, SELECT 문 안에서는 테이블과 거의 동일하게 뷰를 사용할 수 있습니다.

```
WITH p AS (
    SELECT CONCAT(fname, ' ', lname) AS full_name, dob
    FROM person_vw
)
SELECT p.full_name, p.dob, e.empid
FROM p INNER JOIN employee_sample AS e
    ON p.full_name = e.emp_name;
```

FULL_NAME	DOB	EMPID
Bob Smith	2000-01-22	1001

1 Row(s) produced. Time Elapsed: 0.440s

11.2. 복습 하기

? 연습 11-1

다음 뷰에 대한 쿼리 및 결과 집합을 고려하십시오.

```
SELECT * FROM region_totalsales_vw;
```

REGION_NAME	SUM_TOTALPRICE
ASIA	4378591175.90
AMERICA	4321075685.27
EUROPE	4391712838.03
AFRICA	4239225325.42
MIDDLE EAST	4322198235.40

region_totalsales_vw 뷰에 대한 뷰 정의를 작성하세요.

Region, Nation, Customer, 그리고 Orders 테이블을 조인해야 합니다. 컬럼 이름은 결과 집합에 표시된 것과 일치해야 합니다.

11.2. 복습 하기(정답)

100 연습 11-1

```
CREATE VIEW region_totalsales_vw
AS
SELECT r.r_name AS region_name, SUM(o.o_totalprice) AS sum_totalprice
FROM region AS r
      INNER JOIN nation AS n ON r.r_regionkey = n.n_regionkey
      INNER JOIN customer AS c ON c.c_nationkey = n.n_nationkey
      INNER JOIN orders AS o ON o.o_custkey = c.c_custkey
GROUP BY r.r_name;
```

```
+-----+
| status                                     |
+-----+
| View REGION_TOTALSALES_VW successfully created. |
+-----+
```

```
SELECT * FROM region_totalsales_vw;
```

```
+-----+-----+
| REGION_NAME | SUM_TOTALPRICE |
+-----+-----+
| AFRICA      | 2655009469.52  |
| ASIA        | 2760849584.93  |
| EUROPE      | 2754496546.31  |
| MIDDLE EAST | 2721329023.25  |
| AMERICA     | 2721044464.48  |
+-----+-----+
```

12. 윈도우 함수

12.1. 윈도우 개념

- GROUP BY 절은 데이터 값을 기준으로 행들을 부분 집합으로 그룹화하므로, 이 절을 사용해 보셨다면 이 개념에 이미 익숙하실 것입니다.
- GROUP BY를 사용하면 각 그룹의 행들에 대해 max(), min(), count(), sum()과 같은 집계 함수를 적용할 수 있습니다.

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS quarter,  
       SUM(o_totalprice) AS tot_sales  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QUARTER	TOT_SALES
1995	1	828280426.28
1995	2	818992304.21
1995	3	845652776.68
1995	4	824596303.26
1996	1	805551195.59
1996	2	809903462.32
1996	3	841091513.43
1996	4	839827181.45
1997	1	793402839.95
1997	2	824211569.74
1997	3	824176170.61
1997	4	813296140.78

12 Row(s) produced. Time Elapsed: 1.048s

12.1.1. 데이터 윈도우

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS qrter,  
       SUM(o_totalprice) AS tot_sales,  
       SUM(SUM(o_totalprice)) OVER (PARTITION BY DATE_PART(YEAR, o_orderdate)) AS tot_yrly_sales  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QRTER	TOT_SALES	TOT_YRLY_SALES
1995	1	828280426.28	3317521810.43
1995	2	818992304.21	3317521810.43
1995	3	845652776.68	3317521810.43
1995	4	824596303.26	3317521810.43
1996	1	805551195.59	3296373352.79
1996	2	809903462.32	3296373352.79
1996	3	841091513.43	3296373352.79
1996	4	839827181.45	3296373352.79
1997	1	793402839.95	3255086721.08
1997	2	824211569.74	3255086721.08
1997	3	824176170.61	3255086721.08
1997	4	813296140.78	3255086721.08

12 Row(s) produced. Time Elapsed: 0.150s

데이터 윈도우는 그룹과 유사하지만, 윈도우는 SELECT 절이 평가될 때 생성된다는 차이점이 있습니다. 데이터 윈도우가 정의되면, 각 데이터 윈도우 내의 데이터에 max()나 rank()와 같은 윈도우 함수를 적용할 수 있습니다.

12.1.1. 데이터 윈도우

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS qrter,  
       SUM(o_totalprice) AS tot_sales,  
       ROUND(SUM(o_totalprice) / SUM(SUM(o_totalprice)) OVER (PARTITION BY DATE_PART(YEAR, o_orderdate)) * 100, 1) AS pct_of_yrly_sales  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QRTER	TOT_SALES	PCT_OF_YRLY_SALES
1995	1	828280426.28	25.0
1995	2	818992304.21	24.7
1995	3	845652776.68	25.5
1995	4	824596303.26	24.9
1996	1	805551195.59	24.4
1996	2	809903462.32	24.6
1996	3	841091513.43	25.5
1996	4	839827181.45	25.5
1997	1	793402839.95	24.4
1997	2	824211569.74	25.3
1997	3	824176170.61	25.3
1997	4	813296140.78	25.0

12 Row(s) produced. Time Elapsed: 0.860s

첫 번째 합산(summation)은 각 분기별 총 매출을 생성하고, 바깥쪽 합산(윈도우 함수)은 동일한 데이터 윈도우(해당 연도) 내 모든 분기의 총 매출(즉, 연간 총 매출)을 생성합니다. 이제 연간 총계(yearly totals)를 얻었으므로, 이 윈도우 함수 결과를 분모로 사용하여 백분율을 계산함으로써 쿼리를 완성할 수 있습니다.

12.1.2. 파티션과 정렬

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS qtrter,  
       SUM(o_totalprice) AS tot_sales,  
       RANK() OVER (PARTITION BY DATE_PART(YEAR, o_orderdate) ORDER BY SUM(o_totalprice) DESC) AS qtr_rank_per_year  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QTRTER	TOT_SALES	QTR_RANK_PER_YEAR
1995	1	828280426.28	2
1995	2	818992304.21	4
1995	3	845652776.68	1
1995	4	824596303.26	3
1996	1	805551195.59	4
1996	2	809903462.32	3
1996	3	841091513.43	1
1996	4	839827181.45	2
1997	1	793402839.95	4
1997	2	824211569.74	1
1997	3	824176170.61	2
1997	4	813296140.78	3

12 Row(s) produced. Time Elapsed: 0.610s

PARTITION BY 절을 사용하여 데이터 윈도우를 정의하는 예시를 보았습니다. sum()이나 avg()와 같은 일부 윈도우 함수에서는 데이터 윈도우를 정의하는 것만으로 충분합니다. 하지만 다른 유형의 윈도우 함수에서는 각 윈도우 내의 행들을 정렬할 필요가 있으며, 이를 위해 ORDER BY 절을 사용해야 합니다.

12.1.2. 파티션과 정렬

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS qtrter,  
       SUM(o_totalprice) AS tot_sales,  
       RANK() OVER (ORDER BY SUM(o_totalprice) DESC) AS qtr_ranking  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QTRTER	TOT_SALES	QTR_RANKING
1995	1	828280426.28	4
1995	2	818992304.21	8
1995	3	845652776.68	1
1995	4	824596303.26	5
1996	1	805551195.59	11
1996	2	809903462.32	10
1996	3	841091513.43	2
1996	4	839827181.45	3
1997	1	793402839.95	12
1997	2	824211569.74	6
1997	3	824176170.61	7
1997	4	813296140.78	9

12 Row(s) produced. Time Elapsed: 0.944s

데이터 윈도우 내부의 행 정렬과 최종 결과 집합의 정렬에 사용되는 절의 이름이 모두 'order by'로 동일하여 혼동을 야기할 수 있습니다. 윈도우 함수 내의 'order by'(일반적으로 'over(...) ' 절 안에서 사용)는 최종 결과 집합의 순서에는 영향을 주지 않으므로, 쿼리 결과를 원하는 순서로 정렬하려면 쿼리 마지막 부분에 별도의 'order by' 절을 추가해야 합니다.

12.2. 랭킹

- 사람들은 순위 매기기를 좋아합니다. 여행 관련 기사에서는 "최고의 휴가지 10선(Top 10 Vacation Destinations)"을 선정할 수 있고, "역대 최고의 노래 100곡(Top 100 Songs of All Time!)"을 외치는 수많은 노래 목록도 찾아볼 수 있습니다! 기업들 역시 순위 매기기를 좋아하지만, 이는 좀 더 실용적인 목적을 위한 것입니다. 예를 들어, 최고의 제품이나 시장 지역을 파악하는 것은 조직이 미래 투자에 대한 전략적 결정을 내리는 데 도움이 됩니다.
- 스노우플레이크는 여러 다른 종류(flavor)의 순위 함수(ranking function)를 제공합니다.

12.2.1. 랭킹 함수

- row_number(): 각 행에 고유한 순위를 할당하며, 동점은 임의로 처리하고, 순위에는 간격이 없습니다.
- rank() : 동점의 경우 동일한 순위를 할당하고, 순위에 간격이 생깁니다.
- dense_rank(): 동점의 경우 동일한 순위를 할당하고, 순위에 간격이 없습니다.

- 세 가지 순위 함수 모두 order by count(*) desc를 순위 기준으로 지정하는데, 이는 가장 많은 주문 수를 가진 고객에게 순위 1을 부여하게 됩니다. 세 함수 모두 첫 두 행에 대해서는 동일한 순위를 반환하지만, 세 번째 행(네 개의 주문을 한 두 번째 고객에 해당)부터 차이가 나타나기 시작합니다.

- 결과 집합의 아래쪽을 보면, dense_rank() 함수는 1부터 3까지만 순위를 할당하는 반면, rank() 함수는 1, 2, 7 순위를 할당함을 알 수 있습니다. 따라서 순위에 공백을 돌지 여부를 결정하는 것은, 특히 순위 기준값이 동일한 행이 많을 경우, 큰 차이를 만들 수 있습니다.

```
SELECT o_custkey, COUNT(*) AS num_orders,  
       ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) AS row_num_rnk,  
       RANK() OVER (ORDER BY COUNT(*) DESC) AS rank_rnk,  
       DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) AS dns_rank_rnk  
FROM orders o  
WHERE DATE_PART(YEAR, o.o_orderdate) = 1996  
GROUP BY o_custkey  
HAVING o_custkey IN (43645, 55120, 71731, 60250, 55849, 104692, 20743, 118636, 4618, 63620)  
ORDER BY 2 DESC;
```

O_CUSTKEY	NUM_ORDERS	ROW_NUM_RNK	RANK_RNK	DNS_RANK_RNK
43645	5	1	1	1
60250	4	2	2	2
55120	4	3	2	2
71731	4	4	2	2
55849	4	5	2	2
104692	4	6	2	2
118636	3	7	7	3
20743	3	8	7	3
63620	3	9	7	3
4618	3	10	7	3

10 Row(s) produced. Time Elapsed: 0.831s

12.2.2. Top/Bottom/Nth 랭킹

- 1995년부터 1997년까지 각 분기별 총 매출을 합산

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS qrter,  
       SUM(o_totalprice) AS tot_sales  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QRTER	TOT_SALES
1995	1	828280426.28
1995	2	818992304.21
1995	3	845652776.68
1995	4	824596303.26
1996	1	805551195.59
1996	2	809903462.32
1996	3	841091513.43
1996	4	839827181.45
1997	1	793402839.95
1997	2	824211569.74
1997	3	824176170.61
1997	4	813296140.78

12 Row(s) produced. Time Elapsed: 0.572s

12.2.2. Top/Bottom/Nth 랭킹

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS qrter,  
       SUM(o_totalprice) AS tot_sales,  
       FIRST_VALUE(SUM(o_totalprice)) OVER (ORDER BY SUM(o_totalprice) DESC) AS top_sales,  
       LAST_VALUE(SUM(o_totalprice)) OVER (ORDER BY SUM(o_totalprice) DESC) AS btm_sales  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QRTER	TOT_SALES	TOP_SALES	BTM_SALES
1995	1	828280426.28	845652776.68	793402839.95
1995	2	818992304.21	845652776.68	793402839.95
1995	3	845652776.68	845652776.68	793402839.95
1995	4	824596303.26	845652776.68	793402839.95
1996	1	805551195.59	845652776.68	793402839.95
1996	2	809903462.32	845652776.68	793402839.95
1996	3	841091513.43	845652776.68	793402839.95
1996	4	839827181.45	845652776.68	793402839.95
1997	1	793402839.95	845652776.68	793402839.95
1997	2	824211569.74	845652776.68	793402839.95
1997	3	824176170.61	845652776.68	793402839.95
1997	4	813296140.78	845652776.68	793402839.95

12 Row(s) produced. Time Elapsed: 0.302s

- 전체 결과 집합에 걸쳐 순위를 매기고 싶은 수많은 상황이 있지만, 때로는 가장 순위가 높거나 가장 낮은 행에만 관심이 있을 수도 있습니다. 게다가, 예를 들어 결과 집합 내 특정 열의 값과 같이, 가장 순위가 높거나 낮은 행 자체의 어떤 값을 알고 싶을 수도 있습니다. 이런 종류의 기능을 위해 first_value()와 last_value() 함수를 사용할 수 있습니다.

12.2.2. Top/Bottom/Nth 랭킹

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS qrter,  
       SUM(o_totalprice) AS tot_sales,  
       ROUND(SUM(o_totalprice) / FIRST_VALUE(SUM(o_totalprice)) OVER (ORDER BY SUM(o_totalprice) DESC) * 100, 1) AS pct_top_sales,  
       ROUND(SUM(o_totalprice) / LAST_VALUE(SUM(o_totalprice)) OVER (ORDER BY SUM(o_totalprice) DESC) * 100, 1) AS pct_btm_sales  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QRTER	TOT_SALES	PCT_TOP_SALES	PCT_BTM_SALES
1995	1	828280426.28	97.9	104.4
1995	2	818992304.21	96.8	103.2
1995	3	845652776.68	100.0	106.6
1995	4	824596303.26	97.5	103.9
1996	1	805551195.59	95.3	101.5
1996	2	809903462.32	95.8	102.1
1996	3	841091513.43	99.5	106.0
1996	4	839827181.45	99.3	105.9
1997	1	793402839.95	93.8	100.0
1997	2	824211569.74	97.5	103.9
1997	3	824176170.61	97.5	103.9
1997	4	813296140.78	96.2	102.5

12 Row(s) produced. Time Elapsed: 0.044s

- 이제 각 행에 전체 12개 분기의 최고 매출(best)과 최저 매출(worst) 정보가 포함되었으므로, 이 값들을 분모로 사용하여 간단하게 백분율을 계산할 수 있습니다.

12.2.2. Top/Bottom/Nth 랭킹

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,
       DATE_PART(QUARTER, o_orderdate) AS qrter,
       SUM(o_totalprice) AS tot_sales,
       ROUND(SUM(o_totalprice) /
             FIRST_VALUE(SUM(o_totalprice)) OVER (PARTITION BY DATE_PART(YEAR, o_orderdate) ORDER BY SUM(o_totalprice) DESC) * 100, 1) AS pct_top_sales,
       ROUND(SUM(o_totalprice) /
             LAST_VALUE(SUM(o_totalprice)) OVER (PARTITION BY DATE_PART(YEAR, o_orderdate) ORDER BY SUM(o_totalprice) DESC) * 100, 1) AS pct_btm_sales
FROM orders
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997
GROUP BY DATE_PART(YEAR, o_orderdate),
         DATE_PART(QUARTER, o_orderdate)
ORDER BY 1, 2;
```

YEAR	QRTER	TOT_SALES	PCT_TOP_SALES	PCT_BTM_SALES
1995	1	828280426.28	97.9	101.1
1995	2	818992304.21	96.8	100.0
1995	3	845652776.68	100.0	103.3
1995	4	824596303.26	97.5	100.7
1996	1	805551195.59	95.8	100.0
1996	2	809903462.32	96.3	100.5
1996	3	841091513.43	100.0	104.4
1996	4	839827181.45	99.8	104.3
1997	1	793402839.95	96.3	100.0
1997	2	824211569.74	100.0	103.9
1997	3	824176170.61	100.0	103.9
1997	4	813296140.78	98.7	102.5

12 Row(s) produced. Time Elapsed: 0.053s

- 비교가 세 개 연도 전체가 아닌 각 연도 내에서 이루어져야 한다고 가정해 봅시다. 다시 말해, 동일한 연도 내의 최고 및 최저 실적 분기 대비 각 분기 매출의 백분율 비교를 보여줘야 합니다. 이를 위해서는 first_value()와 last_value() 함수(의 OVER 절 내부)에 PARTITION BY 절을 추가하기만 하면 됩니다.

12.2.2. Top/Bottom/Nth 랭킹

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS qtr,  
       SUM(o_totalprice),  
       FIRST_VALUE(DATE_PART(QUARTER, o_orderdate)) OVER (PARTITION BY DATE_PART(YEAR, o_orderdate) ORDER BY SUM(o_totalprice) DESC) AS best_qtr,  
       NTH_VALUE(DATE_PART(QUARTER, o_orderdate), 2) OVER (PARTITION BY DATE_PART(YEAR, o_orderdate) ORDER BY SUM(o_totalprice) DESC) AS next_best_qtr  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QTR	SUM(O_TOTALPRICE)	BEST_QTR	NEXT_BEST_QTR
1995	1	828280426.28	3	1
1995	2	818992304.21	3	1
1995	3	845652776.68	3	1
1995	4	824596303.26	3	1
1996	1	805551195.59	3	4
1996	2	809903462.32	3	4
1996	3	841091513.43	3	4
1996	4	839827181.45	3	4
1997	1	793402839.95	2	3
1997	2	824211569.74	2	3
1997	3	824176170.61	2	3
1997	4	813296140.78	2	3

12 Row(s) produced. Time Elapsed: 0.598s

- first_value() 및 last_value() 함수 외에도, 맨 위(top) 또는 맨 아래(bottom) 행 대신 N번째 순위(2위, 3위 등)에 해당하는 행의 값을 가져오고 싶을 때는 nth_value() 함수를 사용할 수 있습니다.
- 다음 쿼리는 first_value()와 nth_value()를 사용하여 연도별 최고 및 두 번째 최고 실적 분기 번호를 알아냅니다.

12.2.2. Top/Bottom/Nth 랭킹

```
SELECT DATE_PART(YEAR, o_orderdate) AS year, DATE_PART(QUARTER, o_orderdate) AS qtr,
       SUM(o_totalprice),
       FIRST_VALUE(DATE_PART(QUARTER, o_orderdate))
         OVER (PARTITION BY DATE_PART(YEAR, o_orderdate) ORDER BY SUM(o_totalprice) DESC) AS best_qtr,
       NTH_VALUE(DATE_PART(QUARTER, o_orderdate), 2)
         FROM LAST OVER (PARTITION BY DATE_PART(YEAR, o_orderdate) ORDER BY SUM(o_totalprice) DESC) AS next_worst_qtr
FROM orders
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997
GROUP BY DATE_PART(YEAR, o_orderdate), DATE_PART(QUARTER, o_orderdate)
ORDER BY 1, 2;
```

YEAR	QTR	SUM(O_TOTALPRICE)	BEST_QTR	NEXT_WORST_QTR
1995	1	828280426.28	3	4
1995	2	818992304.21	3	4
1995	3	845652776.68	3	4
1995	4	824596303.26	3	4
1996	1	805551195.59	3	2
1996	2	809903462.32	3	2
1996	3	841091513.43	3	2
1996	4	839827181.45	3	2
1997	1	793402839.95	2	4
1997	2	824211569.74	2	4
1997	3	824176170.61	2	4
1997	4	813296140.78	2	4

12 Row(s) produced. Time Elapsed: 0.876s

- nth_value() 함수는 "N"값, 즉 몇 번째 순위의 값을 가져올지 지정하는 추가 매개변수를 가집니다 (이 예시에서는 2). 또한 순위의 맨 위에서부터 N번째 값을 원하는지, 아니면 맨 아래에서부터 N번째 값을 원하는지 지정하는 옵션도 있습니다. 따라서 다음은 연도별 최고 실적 분기와 두 번째로 실적이 낮은(second-worst) 분기를 보여주는 또 다른 변형 쿼리입니다.

12.2.3. Qualify 구문

```
SELECT name, num_suppliers
FROM (
    SELECT n.n_name AS name, COUNT(*) AS num_suppliers,
           RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
    FROM supplier AS s
    INNER JOIN nation AS n
        ON n.n_nationkey = s.s_nationkey
    GROUP BY n.n_name
) AS top_suppliers
WHERE rnk <= 5;
```

NAME	NUM_SUPPLIERS
PERU	325
ALGERIA	318
ARGENTINA	312
CHINA	310
IRAQ	309

5 Row(s) produced. Time Elapsed: 0.918s

- 순위 함수(ranking function)의 결과로 필터링할 때마다 서브쿼리를 사용해야 하는 것은 다소 번거롭습니다. 이러한 목적을 위해, 스노우플레이크는 QUALIFY라는 새로운 절을 추가했습니다. 이는 윈도우 함수의 결과에 기반하여 필터링하기 위해 특별히 설계되었습니다.

```
SELECT n.n_name AS name, COUNT(*) AS num_suppliers,
       RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
FROM supplier AS s
INNER JOIN nation AS n
    ON n.n_nationkey = s.s_nationkey
GROUP BY n.n_name
QUALIFY rnk <= 5;
```

NAME	NUM_SUPPLIERS	RNK
PERU	325	1
ALGERIA	318	2
ARGENTINA	312	3
CHINA	310	4
IRAQ	309	5

5 Row(s) produced. Time Elapsed: 0.817s

12.2.3. Qualify 구문

윈도우 함수를 QUALIFY 절 내부에 직접 넣을 수도 있는데, 이는 실제 순위 값 자체는 필요 없고 상위 N개의 행만 결과로 받고 싶을 때 유용합니다.

```
SELECT n.n_name AS name, COUNT(*) AS num_suppliers
FROM supplier AS s
INNER JOIN nation AS n
    ON n.n_nationkey = s.s_nationkey
GROUP BY n.n_name
QUALIFY RANK() OVER (ORDER BY COUNT(*) DESC) <= 5;
```

NAME	NUM_SUPPLIERS
PERU	325
ALGERIA	318
ARGENTINA	312
CHINA	310
IRAQ	309

5 Row(s) produced. Time Elapsed: 0.473s

- 순위를 기반으로 상위 N개의 행을 찾는 다른 방법은 SELECT 절의 TOP 하위 절을 사용하는 것입니다.

```
SELECT TOP 5
    n.n_name AS name, COUNT(*) AS num_suppliers,
    RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
FROM supplier AS s
INNER JOIN nation AS n
    ON n.n_nationkey = s.s_nationkey
GROUP BY n.n_name
ORDER BY 3;
```

NAME	NUM_SUPPLIERS	RNK
PERU	325	1
ALGERIA	318	2
ARGENTINA	312	3
CHINA	310	4
IRAQ	309	5

5 Row(s) produced. Time Elapsed: 0.603s

12.3. 리포팅 함수

```
SELECT n.n_name, DATE_PART(YEAR, o.o_orderdate) AS year,  
       SUM(o.o_totalprice) AS total_sales  
FROM region AS r  
      INNER JOIN nation AS n ON r.r_regionkey = n.n_regionkey  
      INNER JOIN customer AS c ON n.n_nationkey = c.c_nationkey  
      INNER JOIN orders AS o ON o.o_custkey = c.c_custkey  
WHERE r.r_name = 'ASIA'  
GROUP BY n.n_name, DATE_PART(YEAR, o.o_orderdate)  
ORDER BY 1, 2;
```

- 순위를 생성하는 것 외에도, 윈도우 함수의 또 다른 흔한 용도는 전체 데이터 집합 (또는 파티션)에 걸쳐 이상치(outlier)(예: 최소값 또는 최대값)를 찾거나 합계 또는 평균을 생성하는 것입니다. 이런 유형의 사용 사례에서는 `min()`, `max()`, `sum()` (및 `avg()`)과 같은 집계 함수를 사용하게 됩니다. 하지만 `GROUP BY` 절과 함께 사용하는 대신, 이 함수들을 (`OVER` 절 내의) `PARTITION BY` 및/또는 `ORDER BY` 절과 함께 사용합니다.

N_NAME	YEAR	TOTAL_SALES
CHINA	1992	55786937.46
CHINA	1993	51795296.40
CHINA	1994	62098819.23
CHINA	1995	56719858.43
CHINA	1996	140645511.48
CHINA	1997	129644871.64
CHINA	1998	80074704.96
INDIA	1992	49113818.67
INDIA	1993	50568781.99
INDIA	1994	50191174.00
INDIA	1995	52456057.49
INDIA	1996	132290605.57
INDIA	1997	128123889.04
INDIA	1998	83999645.32
INDONESIA	1992	51329638.93
INDONESIA	1993	47308643.93
INDONESIA	1994	49905364.83
INDONESIA	1995	47577315.99
INDONESIA	1996	123524947.21
INDONESIA	1997	133559192.95
INDONESIA	1998	79817061.71
JAPAN	1992	55479691.91
JAPAN	1993	53885426.14
JAPAN	1994	43457206.90
JAPAN	1995	54232289.25
JAPAN	1996	128591246.23
JAPAN	1997	133507984.20
JAPAN	1998	78711228.07
VIETNAM	1992	53528987.88
VIETNAM	1993	54807694.60
VIETNAM	1994	49265762.05
VIETNAM	1995	53507042.43
VIETNAM	1996	139046819.16
VIETNAM	1997	128811392.94
VIETNAM	1998	77484675.94

12.3.1. 리포팅 함수

```
SELECT n.n_name, DATE_PART(YEAR, o.o_orderdate) AS year,
       SUM(o.o_totalprice) AS total_sales,
       SUM(SUM(o.o_totalprice))
         OVER (PARTITION BY n.n_name) AS tot_cntry_sls,
       SUM(SUM(o.o_totalprice))
         OVER (PARTITION BY DATE_PART(YEAR, o.o_orderdate)) AS tot_yrly_sls
FROM region AS r
INNER JOIN nation AS n
  ON r.r_regionkey = n.n_regionkey
INNER JOIN customer AS c
  ON n.n_nationkey = c.c_nationkey
INNER JOIN orders AS o
  ON o.o_custkey = c.c_custkey
WHERE r.r_name = 'ASIA'
GROUP BY n.n_name, DATE_PART(YEAR, o.o_orderdate)
ORDER BY 1, 2;
```

- 모든 연도에 걸친 국가별 총 매출을 보여주는 열 하나와, 각 연도별 모든 국가의 총 매출을 보여주는 다른 열, 이렇게 두 개의 추가 열을 더해 봅시다. 이를 위해, 두 개의 **sum()** 함수를 각각 적절한 데이터 윈도우를 생성하는 **PARTITION BY** 절과 함께 사용할 것입니다. 쿼리는 다음과 같습니다.

N_NAME	YEAR	TOTAL_SALES	TOT_CNTRY_SLS	TOT_YRLY_SLS
CHINA	1992	55786937.46	576765999.60	265239074.85
CHINA	1993	51795296.40	576765999.60	258365843.06
CHINA	1994	62098819.23	576765999.60	254918327.01
CHINA	1995	56719858.43	576765999.60	264492563.59
CHINA	1996	140645511.48	576765999.60	664099129.65
CHINA	1997	129644871.64	576765999.60	653647330.77
CHINA	1998	80074704.96	576765999.60	400087316.00
INDIA	1992	49113818.67	546743972.08	265239074.85
INDIA	1993	50568781.99	546743972.08	258365843.06
INDIA	1994	50191174.00	546743972.08	254918327.01
INDIA	1995	52456057.49	546743972.08	264492563.59
INDIA	1996	132290605.57	546743972.08	664099129.65
INDIA	1997	128123889.04	546743972.08	653647330.77
INDIA	1998	83999645.32	546743972.08	400087316.00
INDONESIA	1992	51329638.93	533022165.55	265239074.85
INDONESIA	1993	47308643.93	533022165.55	258365843.06
INDONESIA	1994	49905364.83	533022165.55	254918327.01
INDONESIA	1995	47577315.99	533022165.55	264492563.59
INDONESIA	1996	123524947.21	533022165.55	664099129.65
INDONESIA	1997	133559192.95	533022165.55	653647330.77
INDONESIA	1998	79817061.71	533022165.55	400087316.00
JAPAN	1992	55479691.91	547865072.70	265239074.85
JAPAN	1993	53885426.14	547865072.70	258365843.06
JAPAN	1994	43457206.90	547865072.70	254918327.01
JAPAN	1995	54232289.25	547865072.70	264492563.59
JAPAN	1996	128591246.23	547865072.70	664099129.65
JAPAN	1997	133507984.20	547865072.70	653647330.77
JAPAN	1998	78711228.07	547865072.70	400087316.00
VIETNAM	1992	53528987.88	556452375.00	265239074.85
VIETNAM	1993	54807694.60	556452375.00	258365843.06
VIETNAM	1994	49265762.05	556452375.00	254918327.01
VIETNAM	1995	53507042.43	556452375.00	264492563.59
VIETNAM	1996	139046819.16	556452375.00	664099129.65
VIETNAM	1997	128811392.94	556452375.00	653647330.77
VIETNAM	1998	77484675.94	556452375.00	400087316.00

12.3.1. 리포팅 함수

```
SELECT n.n_name, date_part(year, o_orderdate) as year,  
       SUM(o.o_totalprice) AS total_sales,  
       MAX(SUM(o.o_totalprice))  
         OVER (PARTITION BY n.n_name) AS max_cntry_sls,  
       AVG(ROUND(SUM(o.o_totalprice)))  
         OVER (PARTITION BY DATE_PART(YEAR, o.o_orderdate)) AS avg_yrly_sls  
FROM region AS r  
     INNER JOIN nation AS n ON r.r_regionkey = n.n_regionkey  
     INNER JOIN customer AS c ON n.n_nationkey = c.c_nationkey  
     INNER JOIN orders AS o ON o.o_custkey = c.c_custkey  
WHERE r.r_name = 'ASIA'  
GROUP BY n.n_name, DATE_PART(YEAR, o.o_orderdate)  
ORDER BY 1, 2;
```

- 이 추가 열들을 사용하여 국가별 또는 연도별 백분율을 계산할 수 있습니다. 또한 윈도우 내의 평균값이나 최대값과 비교하는 것에도 관심이 있을 수 있습니다.

N_NAME	YEAR	TOTAL_SALES	MAX_CNTRY_SLS	AVG_YRLY_SLS
CHINA	1992	55786937.46	140645511.48	53047815.000
CHINA	1993	51795296.40	140645511.48	51673168.600
CHINA	1994	62098819.23	140645511.48	50983665.400
CHINA	1995	56719858.43	140645511.48	52898512.400
CHINA	1996	140645511.48	140645511.48	132819825.800
CHINA	1997	129644871.64	140645511.48	130729466.200
CHINA	1998	80074704.96	140645511.48	80017463.200
INDIA	1992	49113818.67	132290605.57	53047815.000
INDIA	1993	50568781.99	132290605.57	51673168.600
INDIA	1994	50191174.00	132290605.57	50983665.400
INDIA	1995	52456057.49	132290605.57	52898512.400
INDIA	1996	132290605.57	132290605.57	132819825.800
INDIA	1997	128123889.04	132290605.57	130729466.200
INDIA	1998	83999645.32	132290605.57	80017463.200
INDONESIA	1992	51329638.93	133559192.95	53047815.000
INDONESIA	1993	47308643.93	133559192.95	51673168.600
INDONESIA	1994	49905364.83	133559192.95	50983665.400
INDONESIA	1995	47577315.99	133559192.95	52898512.400
INDONESIA	1996	123524947.21	133559192.95	132819825.800
INDONESIA	1997	133559192.95	133559192.95	130729466.200
INDONESIA	1998	79817061.71	133559192.95	80017463.200
JAPAN	1992	55479691.91	133507984.20	53047815.000
JAPAN	1993	53885426.14	133507984.20	51673168.600
JAPAN	1994	43457206.90	133507984.20	50983665.400
JAPAN	1995	54232289.25	133507984.20	52898512.400
JAPAN	1996	128591246.23	133507984.20	132819825.800
JAPAN	1997	133507984.20	133507984.20	130729466.200
JAPAN	1998	78711228.07	133507984.20	80017463.200
VIETNAM	1992	53528987.88	139046819.16	53047815.000
VIETNAM	1993	54807694.60	139046819.16	51673168.600
VIETNAM	1994	49265762.05	139046819.16	50983665.400
VIETNAM	1995	53507042.43	139046819.16	52898512.400
VIETNAM	1996	139046819.16	139046819.16	132819825.800
VIETNAM	1997	128811392.94	139046819.16	130729466.200
VIETNAM	1998	77484675.94	139046819.16	80017463.200

12.4. 포지션 윈도우

```
SELECT date_part(year, o_orderdate) year,  
       DATE_PART(QUARTER, o_orderdate) AS qrter,  
       SUM(o_totalprice) AS total_sales  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate),  
         DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QRTER	TOTAL_SALES
1995	1	828280426.28
1995	2	818992304.21
1995	3	845652776.68
1995	4	824596303.26
1996	1	805551195.59
1996	2	809903462.32
1996	3	841091513.43
1996	4	839827181.45
1997	1	793402839.95
1997	2	824211569.74
1997	3	824176170.61
1997	4	813296140.78

12 Row(s) produced. Time Elapsed: 0.662s

- 값 기준 대신 근접성(proximity), 즉 행의 상대적인 위치를 기반으로 데이터 윈도우를 정의해야 하는 경우도 있습니다. 예를 들어, 이전 행, 현재 행, 다음 행을 포함하는 데이터 윈도우를 각 행마다 정의하여 이동 평균(rolling average)을 계산해야 할 수 있습니다. 첫 번째 행부터 현재 행까지를 포함하는 데이터 윈도우를 구성하는 누계(running total) 계산도 또 다른 예입니다.
- 이런 유형의 계산을 위해서는 행의 순서를 정의하는 ORDER BY 절과, 데이터 윈도우에 포함될 행들을 지정하는 ROWS(또는 RANGE) 절을 사용해야 합니다.

12.4.1. preceding/following

```
SELECT DATE_PART(YEAR, o_orderdate) AS year,  
       DATE_PART(QUARTER, o_orderdate) AS qrter,  
       SUM(o_totalprice) AS total_sales,  
       AVG(SUM(o_totalprice))  
         OVER (ORDER BY DATE_PART(YEAR, o_orderdate), DATE_PART(QUARTER, o_orderdate)  
              ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS rolling_avg  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate), DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QRTER	TOTAL_SALES	ROLLING_AVG
1995	1	828280426.28	823636365.24500
1995	2	818992304.21	830975169.05666
1995	3	845652776.68	829747128.05000
1995	4	824596303.26	825266758.51000
1996	1	805551195.59	813350320.39000
1996	2	809903462.32	818848723.78000
1996	3	841091513.43	830274052.40000
1996	4	839827181.45	824773844.94333
1997	1	793402839.95	819147197.04666
1997	2	824211569.74	813930193.43333
1997	3	824176170.61	820561293.71000
1997	4	813296140.78	818736155.69500

12 Row(s) produced. Time Elapsed: 1.187s

- 각 분기별 3개월 이동 평균(three-month running average)을 계산하라는 요청을 받았다고 가정해 봅시다. 이를 위해서는 각 행마다 이전 분기, 현재 분기, 다음 분기를 포함하는 데이터 윈도우가 필요합니다. 예를 들어, 1996년 1분기의 3개월 이동 평균은 1995년 4분기, 1996년 1분기, 그리고 1996년 2분기의 값들을 포함하게 됩니다.
- ROWS 절을 사용하여 데이터 윈도우를 정의하는 방법은 다음과 같습니다.

12.4.2. unbound

```
SELECT DATE_PART(YEAR, o_orderdate) AS year, DATE_PART(QUARTER, o_orderdate) AS qrter,
       SUM(o_totalprice) AS total_sales,
       SUM(SUM(o_totalprice))
         OVER (ORDER BY DATE_PART(YEAR, o_orderdate), DATE_PART(QUARTER, o_orderdate)
              ROWS UNBOUNDED PRECEDING) AS running_total
FROM orders
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997
GROUP BY DATE_PART(YEAR, o_orderdate),
         DATE_PART(QUARTER, o_orderdate)
ORDER BY 1, 2;
```

YEAR	QRTER	TOTAL_SALES	RUNNING_TOTAL
1995	1	828280426.28	828280426.28
1995	2	818992304.21	1647272730.49
1995	3	845652776.68	2492925507.17
1995	4	824596303.26	3317521810.43
1996	1	805551195.59	4123073006.02
1996	2	809903462.32	4932976468.34
1996	3	841091513.43	5774067981.77
1996	4	839827181.45	6613895163.22
1997	1	793402839.95	7407298003.17
1997	2	824211569.74	8231509572.91
1997	3	824176170.61	9055685743.52
1997	4	813296140.78	9868981884.30

12 Row(s) produced. Time Elapsed: 0.988s

- 누계(running total)를 계산하려면, 첫 번째 행에서 시작하여 현재 행에서 끝나는 데이터 윈도우를 정의해야 합니다. ROWS UNBOUNDED PRECEDING 구문을 사용하여 분기별 매출 누계를 생성하는 방법은 다음과 같습니다.

12.4.3. lag/lead

```
SELECT DATE_PART(YEAR, o_orderdate) AS year, DATE_PART(QUARTER, o_orderdate) AS qrter,  
       SUM(o_totalprice) AS total_sales,  
       LAG(SUM(o_totalprice), 1)  
         OVER (ORDER BY DATE_PART(YEAR, o_orderdate), DATE_PART(QUARTER, o_orderdate)) AS prior_qtr  
FROM orders  
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997  
GROUP BY DATE_PART(YEAR, o_orderdate), DATE_PART(QUARTER, o_orderdate)  
ORDER BY 1, 2;
```

YEAR	QRTER	TOTAL_SALES	PRIOR_QTR
1995	1	828280426.28	NULL
1995	2	818992304.21	828280426.28
1995	3	845652776.68	818992304.21
1995	4	824596303.26	845652776.68
1996	1	805551195.59	824596303.26
1996	2	809903462.32	805551195.59
1996	3	841091513.43	809903462.32
1996	4	839827181.45	841091513.43
1997	1	793402839.95	839827181.45
1997	2	824211569.74	793402839.95
1997	3	824176170.61	824211569.74
1997	4	813296140.78	824176170.61

12 Row(s) produced. Time Elapsed: 0.691s

- 만약 결과 집합 내의 위치를 기반으로 단일 행의 값만 가져와야 한다면, lag() 함수와 lead() 함수를 사용할 수 있습니다. 이 함수들을 사용하면 (정렬된 순서상) 이전 행이나 다음 행의 열 값을 가져올 수 있습니다. 예를 들어, 이전 분기 대비 백분율 변화를 계산하고자 할 때 유용할 것입니다.
- lag() 함수를 사용하여 이전 분기의 총 매출을 찾는 방법은 다음과 같습니다.

12.4.3. lag/lead

```
SELECT DATE_PART(YEAR, o_orderdate) AS year, DATE_PART(QUARTER, o_orderdate) AS qrter,
       SUM(o_totalprice) AS total_sales,
       LEAD(SUM(o_totalprice), 1)
       OVER (ORDER BY DATE_PART(YEAR, o_orderdate), DATE_PART(QUARTER, o_orderdate)) AS next_qtr
FROM orders
WHERE DATE_PART(YEAR, o_orderdate) BETWEEN 1995 AND 1997
GROUP BY DATE_PART(YEAR, o_orderdate), DATE_PART(QUARTER, o_orderdate)
ORDER BY 1, 2;
```

YEAR	QRTER	TOTAL_SALES	NEXT_QTR
1995	1	828280426.28	818992304.21
1995	2	818992304.21	845652776.68
1995	3	845652776.68	824596303.26
1995	4	824596303.26	805551195.59
1996	1	805551195.59	809903462.32
1996	2	809903462.32	841091513.43
1996	3	841091513.43	839827181.45
1996	4	839827181.45	793402839.95
1997	1	793402839.95	824211569.74
1997	2	824211569.74	824176170.61
1997	3	824176170.61	813296140.78
1997	4	813296140.78	NULL

12 Row(s) produced. Time Elapsed: 0.390s

- 만약 이전 행 대신 정렬 순서상 다음 행의 값을 가져오고 싶다면, lead() 함수를 사용할 수 있습니다.
- next_qtr 열은 다음 분기의 total_sales 값을 가져옵니다. 예상하시겠지만, 마지막 행의 next_qtr 값은 NULL인데, 이는 결과 집합에 더 이상 다음 행이 없기 때문입니다.

12.5. 다른 윈도우 함수

```
SELECT r.r_name as region, LISTAGG(n.n_name, ',') WITHIN GROUP (ORDER BY n.n_name) AS nation_list
FROM region AS r
INNER JOIN nation AS n ON r.r_regionkey = n.n_regionkey
GROUP BY r.r_name
ORDER BY 1;
```

REGION	NATION_LIST
AFRICA	ALGERIA, ETHIOPIA, KENYA, MOROCCO, MOZAMBIQUE
AMERICA	ARGENTINA, BRAZIL, CANADA, PERU, UNITED STATES
ASIA	CHINA, INDIA, INDONESIA, JAPAN, VIETNAM
EUROPE	FRANCE, GERMANY, ROMANIA, RUSSIA, UNITED KINGDOM
MIDDLE EAST	EGYPT, IRAN, IRAQ, JORDAN, SAUDI ARABIA

- listagg() 함수는 데이터를 구분자로 연결된 단일 문자열로 만듭니다. 다음 쿼리는 각 지역과 해당 지역에 연관된 국가들의 심표로 구분된 목록을 함께 반환하는 예시입니다.

```
SELECT DISTINCT r.r_name AS region,
LISTAGG(n.n_name, ',') WITHIN GROUP (ORDER BY n.n_name)
OVER (PARTITION BY r.r_name) AS nation_list
FROM region AS r
INNER JOIN nation AS n ON r.r_regionkey = n.n_regionkey
ORDER BY 1;
```

REGION	NATION_LIST
AFRICA	ALGERIA, ETHIOPIA, KENYA, MOROCCO, MOZAMBIQUE
AMERICA	ARGENTINA, BRAZIL, CANADA, PERU, UNITED STATES
ASIA	CHINA, INDIA, INDONESIA, JAPAN, VIETNAM
EUROPE	FRANCE, GERMANY, ROMANIA, RUSSIA, UNITED KINGDOM
MIDDLE EAST	EGYPT, IRAN, IRAQ, JORDAN, SAUDI ARABIA

5 Row(s) produced. Time Elapsed: 0.606s

12.6. 복습 하기

? 연습 12-1

원본 쿼리에서 다시 시작하여 모든 연도의 총 매출(총합계)을 계산하는 열을 추가하십시오. 각 행의 값은 동일해야 합니다.

```
SELECT DATE_PART(YEAR, o_orderdate) AS order_year,  
       COUNT(*) AS num_orders,  
       SUM(o_totalprice) AS tot_sales  
FROM orders  
GROUP BY DATE_PART(YEAR, o_orderdate);
```

ORDER_YEAR	NUM_ORDERS	TOT_SALES
1996	17657	3296373352.79
1994	17479	3278473892.67
1997	17408	3255086721.08
1993	17392	3270416270.14
1998	10190	1925196448.52
1995	17637	3317521810.43
1992	17506	3309734764.39

12.6. 복습 하기(정답)

100 연습 12-1

```
SELECT DATE_PART(YEAR, o_orderdate) AS order_year,  
       COUNT(*) AS num_orders,  
       SUM(o_totalprice) AS tot_sales,  
       SUM(SUM(o_totalprice)) OVER () AS grnd_tot_sales  
FROM orders  
GROUP BY DATE_PART(YEAR, o_orderdate);
```

13. 반구조화 데이터

13.1. 관계형 데이터에서 JSON 생성

1. 관계형 데이터베이스는 복잡한 개념과 관계를 처리하는 데 능숙하지만, 모든 데이터가 행과 열로 쉽게 표현되지는 않습니다. XML, Parquet, Avro, JSON과 같은 반정형 데이터 형식은 미리 정의된 스키마를 따를 필요 없이 유연한 데이터 저장을 가능하게 합니다.
 2. 반정형 데이터 지원이 나중에 추가된 일부 데이터베이스 서버들과는 달리, 스노우플레이크의 아키텍처는 처음부터 정형 및 반정형 데이터를 모두 지원하도록 설계되었습니다. 이 장에서는 스노우플레이크에서 JSON 문서의 생성, 저장 및 검색 방법을 살펴볼 것입니다.
- `object_construct()`: 키-값 쌍의 집합을 객체 타입으로 반환합니다.
 - `array_agg()`: 값의 집합을 피벗하고 배열 타입을 반환합니다.
 - `parse_json()`: JSON 문서를 파싱하고 `variant` 타입을 반환합니다.
 - `try_parse_json()`: JSON 문서를 파싱하고, 유효한 JSON 문서가 아니면 `null`을 반환하고, 그렇지 않으면 `variant` 타입을 반환합니다.

13.1.1. array_agg(), object_construct()

```
SELECT
  OBJECT_CONSTRUCT('Regions',
    ARRAY_AGG(OBJECT_CONSTRUCT('Region_Key', r_regionkey, 'Region_Name', r_name))) AS my_doc
FROM region;
```

```
+-----+
| MY_DOC |
+-----+
| {      |
|   "Regions": [ |
|     {        |
|       "Region_Key": 0, |
|       "Region_Name": "AFRICA" |
|     },      |
|     {        |
|       "Region_Key": 1, |
|       "Region_Name": "AMERICA" |
|     },      |
|     {        |
|       "Region_Key": 2, |
|       "Region_Name": "ASIA" |
|     },      |
|     {        |
|       "Region_Key": 3, |
|       "Region_Name": "EUROPE" |
|     },      |
|     {        |
|       "Region_Key": 4, |
|       "Region_Name": "MIDDLE EAST" |
|     }        |
|   ]         |
| }          |
+-----+
```

- 두 개의 키-값 쌍(키는 Region_Key와 Region_Name)으로 구성된 객체(object)를 생성한 다음, 그것들을 배열(array)로 변환합니다. 최종 결과는 JSON 문서를 포함하는 단일 행 출력입니다.

13.2. JSON 문서 저장하기

```
CREATE TABLE my_docs (doc VARIANT);
```

```
+-----+
| status |
+-----+
| Table MY_DOCS successfully created. |
+-----+
1 Row(s) produced. Time Elapsed: 0.196s
```



```
INSERT INTO my_docs
WITH ntn_tot AS (
    SELECT n.n_regionkey, n.n_nationkey,
           SUM(o.o_totalprice) AS tot_ntn_sales,
           SUM(SUM(o.o_totalprice))
              OVER (PARTITION BY n.n_regionkey) AS tot_rgn_sales
    FROM orders o
         INNER JOIN customer c ON c.c_custkey = o.o_custkey
         INNER JOIN nation n ON c.c_nationkey = n.n_nationkey
    GROUP BY n.n_regionkey, n.n_nationkey
), ntn AS (
    SELECT n.n_regionkey,
           ARRAY_AGG(OBJECT_CONSTRUCT(
               'Nation_Name', n.n_name,
               'Tot_Nation_Sales', ntn_tot.tot_ntn_sales
           )) AS nation_list
    FROM nation n
         INNER JOIN ntn_tot ON n.n_nationkey = ntn_tot.n_nationkey
    GROUP BY n.n_regionkey
)
SELECT
    OBJECT_CONSTRUCT('Regions', ARRAY_AGG(OBJECT_CONSTRUCT(
        'Region_Name', r.r_name,
        'Tot_Region_Sales', rgn_tot.tot_rgn_sales,
        'Nations', ntn.nation_list
    ))) AS region_summary_doc
FROM region r
     INNER JOIN ntn ON r.r_regionkey = ntn.n_regionkey
     INNER JOIN (SELECT DISTINCT n_regionkey, tot_rgn_sales
                  FROM ntn_tot
                 ) rgn_tot ON rgn_tot.n_regionkey = r.r_regionkey;
```

Variant 열(Column)에 JSON 문서 저장 시 제한 사항

Variant 열을 사용하여 문서를 저장하는 것이 편리하기는 하지만, 다음과 같은 몇 가지 제한 사항을 고려해야 합니다:

- **크기 제한:** Variant 열은 **16MB**로 크기가 제한되어 있어, 매우 큰 문서는 단일 열에 저장하기 어렵습니다.
- **시간 데이터 처리 성능:** 날짜(date), 시간(time), 타임스탬프(timestamp)와 같은 시간 관련 데이터 필드는 **문자열(string)**로 저장됩니다. 이 때문에 `add_month()`나 날짜 차이(date diff) 계산과 같은 연산을 수행할 때 성능 저하가 발생할 수 있습니다.

13.3. JSON 문서 조회하기

```
SELECT DISTINCT d.key, typeof(d.value) AS data_type
FROM my_docs
      INNER JOIN LATERAL FLATTEN(doc, recursive => true) d
WHERE typeof(d.value) <> 'OBJECT'
ORDER BY 1;
```

KEY	DATA_TYPE
Nation_Name	VARCHAR
Nations	ARRAY
Region_Name	VARCHAR
Regions	ARRAY
Tot_Nation_Sales	DECIMAL
Tot_Region_Sales	DECIMAL

- `flatten()` 함수는 여러 열을 반환하는데, 이 예시에서는 그중 `key`와 `value` 열을 사용하며, 이 쿼리에서는 각 필드의 데이터 타입을 알아내기 위해 `typeof()` 함수를 사용합니다.

```
SELECT ARRAY_SIZE(doc:Regions)
FROM my_docs;
```

ARRAY_SIZE(DOC:REGIONS)
5

1 Row(s) produced. Time Elapsed: 0.395s

- `Regions` 필드가 배열 타입임을 확인했으므로, `array_size()` 함수를 사용하여 배열에 요소(element)가 몇 개 있는지 알아낼 수 있습니다.

```
SELECT
      doc:Regions[0].Region_Name AS R0_Name,
      doc:Regions[1].Region_Name AS R1_Name,
      doc:Regions[2].Region_Name AS R2_Name,
      doc:Regions[3].Region_Name AS R3_Name,
      doc:Regions[4].Region_Name AS R4_Name
```

```
FROM my_docs;
```

R0_NAME	R1_NAME	R2_NAME	R3_NAME	R4_NAME
"AFRICA"	"ASIA"	"EUROPE"	"MIDDLE EAST"	"AMERICA"

1 Row(s) produced. Time Elapsed: 0.639s

```
SELECT
      r.value:Region_Name::STRING AS region_name
FROM my_docs
      INNER JOIN LATERAL FLATTEN(INPUT => doc:regions) r;
```

REGION_NAME
AFRICA
ASIA
EUROPE
MIDDLE EAST
AMERICA

5 Row(s) produced. Time Elapsed: 0.426s

- `Region_Name` 값을 문자열로 캐스팅 하여 값 양쪽의 큰따옴표를 제거했습니다.

13.3.1. JSON Join

```
SELECT
  rgn.r_regionkey,
  r.value:Region_Name::STRING AS region_name,
  ntn.n_nationkey,
  n.value:Nation_Name::STRING AS nation_name
FROM my_docs
  INNER JOIN LATERAL FLATTEN(INPUT => doc:Regions) r
  INNER JOIN region rgn ON r.value:Region_Name = rgn.r_name
  INNER JOIN LATERAL FLATTEN(INPUT => r.value:Nations) n
  INNER JOIN nation ntn ON n.value:Nation_Name = ntn.n_name
ORDER BY 1, 3
LIMIT 10;
```

R_REGIONKEY	REGION_NAME	N_NATIONKEY	NATION_NAME
0	AFRICA	0	ALGERIA
0	AFRICA	5	ETHIOPIA
0	AFRICA	14	KENYA
0	AFRICA	15	MOROCCO
0	AFRICA	16	MOZAMBIQUE
1	AMERICA	1	ARGENTINA
1	AMERICA	2	BRAZIL
1	AMERICA	3	CANADA
1	AMERICA	17	PERU
1	AMERICA	24	UNITED STATES

10 Row(s) produced. Time Elapsed: 0.632s

- 첫 번째 `flatten()` 호출은 **Regions** 배열의 각 요소를 행으로 변환하고, 두 번째 `flatten()` 호출은 첫 번째 변환으로 생성된 각 행 내의 **Nations** 배열을 다시 행으로 변환합니다.
- 이제 문서가 개별 필드를 가진 행들로 분해되었으므로, 다음 쿼리에서 보여주는 것처럼, 이 플래튼된 (**flattened**) 데이터 집합을 다른 테이블들과 조인할 수 있습니다.

13.4. 복습 하기

? 연습 13-1

Part 테이블에 대해 다음 쿼리가 주어집니다.

```
SELECT p_partkey, p_name, p_brand
FROM part
WHERE p_mfgr = 'Manufacturer#1'
      AND p_type = 'ECONOMY POLISHED STEEL';
```

P_PARTKEY	P_NAME	P_BRAND
95608	royal thistle floral frosted midnight	Brand#12
100308	azure honeydew grey aquamarine black	Brand#11
103808	steel lemon tomato brown blush	Brand#13
68458	spring white lime dim peru	Brand#14
70808	gainsboro chiffon papaya green khaki	Brand#12
112758	turquoise saddle moccasin magenta pink	Brand#14

6 Row(s) produced. Time Elapsed: 0.812s

동일한 결과를 포함하는 JSON 문서를 생성하는 쿼리를 작성합니다. 이 문서는 Partkey, Name 및 Brand 태그가 있는 6개의 항목 배열이 포함된 Parts라는 이름의 개체로 구성되어야 합니다.

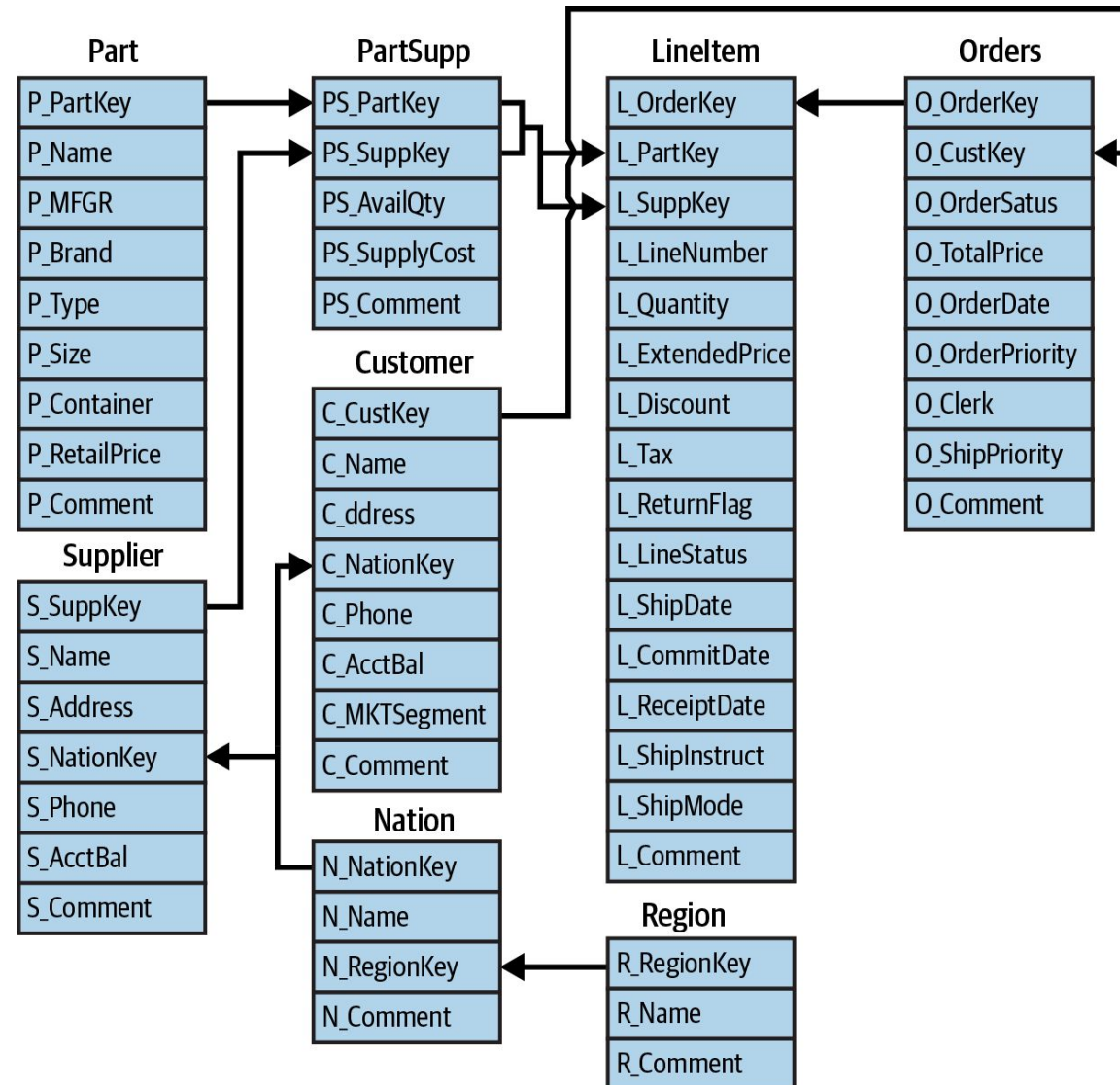
13.4. 복습 하기(정답)

100 연습 13-1

```
SELECT OBJECT_CONSTRUCT('Parts',  
    ARRAY_AGG(  
        OBJECT_CONSTRUCT(  
            'Partkey', p_partkey,  
            'Name', p_name,  
            'Brand', p_brand  
        )  
    )  
    ) AS part_doc  
FROM part  
WHERE p_mfgr = 'Manufacturer#1'  
    AND p_type = 'ECONOMY POLISHED STEEL';
```

참고. 실습환경

1. 테스트 데이터베이스



1.1. 구성

-- 데이터베이스 생성

create database learning_sql

-- 테이블 생성

create table region
as select * from snowflake_sample_data.tpch_sf1.region;

create table nation
as select * from snowflake_sample_data.tpch_sf1.nation;

create table part as
select * from snowflake_sample_data.tpch_sf1.part
where mod(p_partkey,50) = 8;

create table partsupp as
select * from snowflake_sample_data.tpch_sf1.partsupp
where mod(ps_partkey,50) = 8;

create table supplier as
with sp as (select distinct ps_suppkey from partsupp)
select s.* from snowflake_sample_data.tpch_sf1.supplier s
inner join sp
on s.s_suppkey = sp.ps_suppkey;

create table lineitem as
select l.* from snowflake_sample_data.tpch_sf1.lineitem l
inner join part p
on p.p_partkey = l.l_partkey;

create table orders as
with li as (select distinct l_orderkey from lineitem)
select o.* from snowflake_sample_data.tpch_sf1.orders o
inner join li on o.o_orderkey = li.l_orderkey;

create table customer as
with o as (select distinct o_custkey from orders)
select c.* from snowflake_sample_data.tpch_sf1.customer c
inner join o on c.c_custkey = o.o_custkey;

감사합니다.