#데이터베이스 #SQL #AdvancedSQL

Agenda

- Join
- Views
- Window functions
- Keys

Views #Views

- 모든 사용자가 데이터의 전체 논리적 모델(Logical Model)을 보는 것이 항상 바람직한 것은 아닙니다.
  - 예: 강사의 이름과 학과는 알아야 하지만 급여는 몰라도 되는 사용자를 고려해 봅시다.
    - 이 사용자는 SQL에서 다음 관계만 보면 됩니다:

```
SELECT ID, name, dept_name
FROM instructor;
```

- View: 특정 사용자의 관점에서 특정 데이터를 숨기는 메커니즘을 제공합니다.
  - View는 저장된 테이블(base tables) 및 다른 뷰를 기반으로 정의된 relation입니다.
  - 개념적 모델(Conceptual Model)에는 속하지 않지만 사용자에게 "가상 관계(virtual relation)"로 표시되는 모든 관계를 view라고 합니다.

Important

뷰는 특정 사용자로부터 특정 데이터를 숨기는 메커니즘을 제공합니다.

View Syntax

- Syntax:

```
CREATE VIEW v AS < query expression >
```

  - 여기서 `<query expression>`은 유효한 SQL expression이며, `v`는 view name을 나타냅니다.
- 뷰가 정의되면, 뷰 이름은 뷰가 생성하는 virtual relation을 참조하는 데 사용될 수 있습니다.
- View definition은 새로운 관계를 생성하는 것과는 다릅니다.
- View definition은 표현식을 저장하게 하며, 이 표현식은 뷰를 사용하는 쿼리에 대체됩니다.

View Examples

- 급여 정보가 없는 강사 뷰:

```
CREATE VIEW faculty AS
SELECT ID, name, dept_name
FROM instructor;
```

- 뷰에 대한 쿼리도 가능합니다:

```
SELECT name
FROM faculty
WHERE dept_name = 'Biology';
```

- 참고(C.f.), Biology 학과의 모든 강사 찾기 (원본 테이블 쿼리):

```sql
SELECT name
FROM instructor
WHERE dept_name = 'Biology';
```

- 뷰의 속성 이름은 명시적으로 지정할 수 있습니다:

```sql
CREATE VIEW departments_total_salary(dept_name,
total_salary) AS
SELECT dept_name, SUM(salary)
FROM instructor
GROUP BY dept_name;
```

> **Note**
> `SUM(salary)` 표현식 자체에는 이름이 없으므로, 뷰 정의에서 속성 이름을 명시적으로 지정했습니다. SUM, GROUP BY

- Class VM 이미지의 sakila 데이터베이스에는 7개의 샘플 뷰가 포함되어 있습니다 (예: `actor_info`, `customer_list`, `film_list` 등).

View Expansion

- View expansion: 다른 뷰를 기반으로 정의된 뷰의 의미를 정의하는 방법입니다.
- 뷰 v_1이 다른 뷰 관계를 포함할 수 있는 표현식 e_1에 의해 정의된다고 가정합니다.
- 표현식의 View expansion은 다음 대체 단계를 반복합니다:
  1. `repeat`
  2. e_1에서 뷰 관계 v_i 찾기
  3. 뷰 관계 v_i를 v_i를 정의하는 표현식으로 교체
  4. `until` e_1에 더 이상 뷰 관계가 없을 때까지
- 뷰 정의가 재귀적이지 않은 한, 이 루프는 종료됩니다.

예시:

- VIEW V3 AS SELECT * FROM BaseTable WHERE category = 'A'; (V3은 BaseTable 참조)
- VIEW V2 AS SELECT item, price FROM V3 WHERE price > 100; (V2는 V3 참조)
- VIEW V1 AS SELECT * FROM V2 WHERE item LIKE 'Book%'; (V1은 V2 참조)

Views Defined Using Other Views

- 하나의 뷰가 다른 뷰를 정의하는 표현식에서 사용될 수 있습니다.
- 뷰 관계 v_1은 v_1을 정의하는 표현식에서 v_2가 사용되면 뷰 관계 v_2에 직접 의존한다고 합니다.
- 뷰 관계 v_1은 v_1이 v_2에 직접 의존하거나 v_1에서 v_2로의 의존성 경로가 있는 경우 뷰 관계 v_2에 의존한다고 합니다.
- 뷰 관계 v는 자신에게 의존하는 경우 재귀적이라고 합니다. view relation

Examples

- ```
  CREATE VIEW physics_fall_2017 AS
  SELECT course.course_id, sec_id, building, room_number
  FROM course, section
  WHERE course.course_id = section.course_id
    AND course.dept_name = 'Physics'
    AND section.semester = 'Fall'
    AND section.year = '2017';
    -- Depends on: [[course]], [[section]]
  ```

- ```
  CREATE VIEW physics_fall_2017_watson AS
  SELECT course_id, room_number
  FROM physics_fall_2017 -- Depends on the view above
  ```

```
WHERE building= 'Watson';
-- Depends on: [[physics_fall_2017]]
```

- **View Expansion Example:** 다음 두 쿼리는 view expansion을 통해 동일합니다.
  1. **Original Query using View:**

```
-- Creating the view that depends on another view
CREATE VIEW physics_fall_2017_watson AS
SELECT course_id, room_number
FROM physics_fall_2017
WHERE building= 'Watson';
```

  2. **Expanded Query:**

```
-- Creating the view with the base view's definition
substituted
CREATE VIEW physics_fall_2017_watson AS
SELECT course_id, room_number
FROM ( -- This is the definition of physics_fall_2017
     SELECT course.course_id, sec_id, building,
room_number
     FROM course, section
     WHERE course.course_id = section.course_id
       AND course.dept_name = 'Physics'
       AND section.semester = 'Fall'
       AND section.year = '2017'
    )
WHERE building= 'Watson';
```

Materialized Views

- **두 종류의 뷰:**

    - Virtual 뷰: 데이터베이스에 저장되지 않음; 관계를 구성하기 위한 쿼리일 뿐입니다.
    - Materialized 뷰: 물리적으로 구성되고 저장됩니다.
- Materialized view: 쿼리의 사전 계산된(구체화된) 결과입니다.
    - 단순 VIEW와 달리 Materialized View의 결과는 일반적으로 테이블과 같은 곳에 저장됩니다.
    - 사용 시기:
        - 즉각적인 응답이 필요할 때
        - Materialized View의 기반이 되는 쿼리가 결과를 생성하는 데 너무 오래 걸릴 때
    - Materialized Views는 가끔씩 새로 고침되어야 합니다.

> **Note**
>
> MySQL은 materialized views를 지원하지 않습니다.

Update via a View

- 앞서 정의한 `[[faculty]]` 뷰에 새 튜플 추가:

```
INSERT INTO faculty VALUES ('30765', 'Green', 'Music');
```

    - 이 삽입은 `[[instructor]]` 관계에 대한 삽입으로 표현되어야 합니다.
    - 문제점: `salary`에 대한 값이 있어야 합니다.
- `salary` 값 문제 해결 방법:
    1. 삽입 거부 (Reject the insert)
    2. `[[instructor]]` 관계에 `('30765', 'Green', 'Music', null)` 튜플 삽입. (NULL)
- 일부 업데이트는 고유하게 변환될 수 없습니다 (`[[INSERT INTO]]`).
    - 예: `[[instructor]]`와 `[[department]]`를 조인하는 뷰 생성

```sql
CREATE VIEW instructor_info AS
SELECT ID, name, building
FROM instructor, department
WHERE instructor.dept_name = department.dept_name;
```

- 그런 다음 뷰에 삽입:

```sql
INSERT INTO instructor_info VALUES ('69987', 'White',
'Taylor');
```

- 문제점:
  - Taylor 건물에 여러 학과가 있다면 어느 학과인가?
  - Taylor 건물에 학과가 없다면 어떻게 되는가?

> ⚠ **Caution**
> MySQL에서는 조인 뷰에 대한 삽입 시 "SQL error (1394): Can not insert into join view without fields list" 오류가 발생할 수 있습니다.

- 다른 업데이트 예제:

```sql
CREATE VIEW history_instructors AS
SELECT *
FROM instructor
WHERE dept_name='History';
```

- `history_instructors` 뷰에 다음을 삽입하면 어떻게 될까요?
  `sql INSERT INTO history_instructors VALUES ('25566', 'Brown', 'Biology', 100000);`

> 🖉 **Note**

이 삽입은 기본 테이블인 `[[instructor]]`에는 성공하지만, `WHERE dept_name='History'` 조건을 만족하지 않으므로 삽입된 행은 `[[history_instructors]]` 뷰 자체에서는 보이지 않습니다.

- 업데이트 가능 뷰 (일반 규칙):
  - 대부분의 SQL 구현은 simple views에 대해서만 업데이트를 허용합니다.

> **Important**
> 단순 뷰 (Simple View) 조건:
> - `[[FROM]]` 절에는 단 하나의 데이터베이스 관계만 포함됩니다.
> - `[[SELECT]]` 절에는 관계의 속성 이름만 포함되며, 표현식, 집계 함수, 또는 DISTINCT 지정이 없습니다.
> - `[[SELECT]]` 절에 나열되지 않은 모든 속성은 NULL로 설정될 수 있습니다 (즉, NOT NULL 제약 조건이 없거나 기본값이 있어야 함).
> - 쿼리에 `[[GROUP BY]]` 또는 `[[HAVING]]` 절이 없습니다.

Window Functions in SQL #WindowFunctions

- SQL:2003 표준 SQL에 처음 도입되었습니다.
- 내장 함수로서 레코드 간의 관계를 정의합니다.
  - "윈도우 함수는 현재 행과 어떤 방식으로든 관련된 테이블 행 집합에 대해 계산을 수행합니다... 내부적으로 윈도우 함수는 쿼리 결과의 현재 행뿐만 아니라 그 이상에 접근할 수 있습니다." (PostgreSQL)
  - 순위(ranks), 백분위수(percentiles), 합계/평균(sums/averages), 행 번호(row numbers) 등을 찾을 수 있습니다.
- 집계 함수의 경우, 이동 합계, 이동 평균 등을 구현할 수 있습니다.
  - WINDOW_FUNCTION clause를 사용하여 윈도우 크기를 변경할 수 있습니다. (구문은 아래 참조)
- `[[GROUP BY]]` 절과 함께 사용할 수 없습니다.

> ⚠ Warning
> `[[PARTITION]]` (윈도우 함수)와 `[[GROUP BY]]`는 모두 데이터를 분할하고 일부 통계를 계산하지만, 윈도우 함수는 결과의 레코드 수를 줄이지 않습니다. `GROUP BY`는 그룹당 하나의 행으로 결과를 축소합니다.

Window function types

- 집계 윈도우 함수:
  - `[[SUM()]]`, `[[MAX()]]`, `[[MIN()]]`, `[[AVG()]]`, `[[COUNT()]]`, ...
- 순위 윈도우 함수:
  - `[[RANK()]]`, `[[DENSE_RANK()]]`, `[[PERCENT_RANK()]]`, `[[ROW_NUMBER()]]`, `[[NTILE()]]`
- 값 윈도우 함수:
  - `[[LAG()]]`, `[[LEAD()]]`, `[[FIRST_VALUE()]]`, `[[LAST_VALUE()]]`, `[[CUME_DIST()]]`, `[[NTH_VALUE()]]`

Syntax

- ```
  SELECT
      [어떤 계산을 할지] ( [무엇을 가지고 계산할지] )  -- 윈도우 함수 부분
      OVER (
          [어떤 기준으로 그룹을 나눌지]                -- PARTITION BY
          [나눠진 그룹 안에서 어떻게 정렬할지]      -- ORDER BY
          [정렬된 그룹 안에서 어디까지 계산에 포함할지] -- frame_clause
      )
  FROM
      [데이터가 있는 테이블];
  ```

- `[[WINDOW_FUNCTION]]`: 윈도우 함수의 이름을 지정합니다.
- `[[ALL]]` (선택 사항): ALL을 포함하면 중복 값을 포함하여 모든 값을 계산합니다.

> 🖉 Note
  ```

`[[DISTINCT]]` 는 윈도우 함수에서 지원되지 않습니다.

- `[[OVER]]` : 집계 함수에 대한 윈도우 절을 지정합니다.
  - `[[PARTITION BY partition_list]]` : 윈도우 함수가 작동하는 윈도우(행 집합)를 정의합니다.
    - `PARTITION BY` 가 지정되지 않으면 전체 테이블에 대해 그룹화되고 값이 집계됩니다.
  - `[[ORDER BY order_list]]` : 각 파티션 내의 행을 정렬합니다.
    - `ORDER BY` 가 지정되지 않으면 전체 테이블(또는 프레임 정의에 따라 다름)이 사용됩니다.
  - `[ frame_clause ]` : 파티션 내에서 함수가 작동할 행의 하위 집합(프레임)을 정의합니다. (아래 참조)

Running Examples

DEPT Table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DE
7839	KING	PRESIDENT	NULL	1981-11-17	5000.00	NULL	10

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DE
7698	BLAKE	MANAGER	7839	1981-05-01	2850.00	NULL	30
7782	CLARK	MANAGER	7839	1981-05-09	2450.00	NULL	10
7566	JONES	MANAGER	7839	1981-04-01	2975.00	NULL	20
7654	MARTIN	SALESMAN	7698	1981-09-10	1250.00	1400.00	30
7499	ALLEN	SALESMAN	7698	1981-02-11	1600.00	300.00	30
7844	TURNER	SALESMAN	7698	1981-08-21	1500.00	0.00	30
7900	JAMES	CLERK	7698	1981-12-11	950.00	NULL	30
7521	WARD	SALESMAN	7698	1981-02-23	1250.00	500.00	30
7902	FORD	ANALYST	7566	1981-12-11	3000.00	NULL	20
7369	SMITH	CLERK	7902	1980-12-09	800.00	NULL	20
7788	SCOTT	ANALYST	7566	1982-12-22	3000.00	NULL	20
7876	ADAMS	CLERK	7788	1983-01-15	1100.00	NULL	20
7934	MILLER	CLERK	7782	1982-01-11	1300.00	NULL	10

DIY Table Creation

- DEPT Table:

```
CREATE TABLE DEPT (
    DEPTNO INT,
    DNAME VARCHAR(14),
    LOC VARCHAR(13)
);
INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');
INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');
```

- EMP Table:

```
CREATE TABLE EMP (
    EMPNO INT NOT NULL,
    ENAME VARCHAR(10),
    JOB VARCHAR(9),
    MGR INT,
    HIREDATE DATE,
    SAL DECIMAL(7,2),
    COMM DECIMAL(7,2),
    DEPTNO INT
);
INSERT INTO EMP VALUES (7839,'KING','PRESIDENT',NULL,'1981-
11-17',5000,NULL,10);
INSERT INTO EMP VALUES (7698,'BLAKE','MANAGER',7839,'1981-
05-01',2850,NULL,30);
INSERT INTO EMP VALUES (7782,'CLARK','MANAGER',7839,'1981-
05-09',2450,NULL,10);
INSERT INTO EMP VALUES (7566,'JONES','MANAGER',7839,'1981-
04-01',2975,NULL,20);
INSERT INTO EMP VALUES (7654,'MARTIN','SALESMAN',7698,'1981-
09-10',1250,1400,30);
INSERT INTO EMP VALUES (7499,'ALLEN','SALESMAN',7698,'1981-
02-11',1600,300,30);
INSERT INTO EMP VALUES (7844,'TURNER','SALESMAN',7698,'1981-
08-21',1500,0,30);
INSERT INTO EMP VALUES (7900,'JAMES','CLERK',7698,'1981-12-
11',950,NULL,30);
```

```
INSERT INTO EMP VALUES (7521,'WARD','SALESMAN',7698,'1981-
02-23',1250,500,30);
INSERT INTO EMP VALUES (7902,'FORD','ANALYST',7566,'1981-12-
11',3000,NULL,20);
INSERT INTO EMP VALUES (7369,'SMITH','CLERK',7902,'1980-12-
09',800,NULL,20);
INSERT INTO EMP VALUES (7788,'SCOTT','ANALYST',7566,'1982-
12-22',3000,NULL,20);
INSERT INTO EMP VALUES (7876,'ADAMS','CLERK',7788,'1983-01-
15',1100,NULL,20);
INSERT INTO EMP VALUES (7934,'MILLER','CLERK',7782,'1982-01-
11',1300,NULL,10);
```

Aggregation Examples #AggregationWindowFunctions

- 각 관리자(MGR)별 급여 합계:

```
SELECT ENAME, SAL, MGR,
       SUM(SAL) OVER (PARTITION BY MGR) AS SUM_MGR
FROM EMP;
```

Result:

ENAME	SAL	MGR	SUM_MGR
KING	5000.00	NULL	5000.00
FORD	3000.00	7566	6000.00
SCOTT	3000.00	7566	6000.00
MARTIN	1250.00	7698	6550.00
ALLEN	1600.00	7698	6550.00
TURNER	1500.00	7698	6550.00
JAMES	950.00	7698	6550.00
WARD	1250.00	7698	6550.00

ENAME	SAL	MGR	SUM_MGR
MILLER	1300.00	7782	1300.00
ADAMS	1100.00	7788	1100.00
BLAKE	2850.00	7839	8275.00
CLARK	2450.00	7839	8275.00
JONES	2975.00	7839	8275.00
SMITH	800.00	7902	800.00

- 각 직업(JOB)별 평균 급여 (Window Function 사용):

```
SELECT ENAME, SAL, JOB,
       AVG(SAL) OVER (PARTITION BY JOB) AS AVG_SAL_JOB
FROM EMP;
```

Result:

ENAME	SAL	JOB	AVG_SAL_JOB
FORD	3000.00	ANALYST	3000.000000
SCOTT	3000.00	ANALYST	3000.000000
JAMES	950.00	CLERK	1037.500000
SMITH	800.00	CLERK	1037.500000
ADAMS	1100.00	CLERK	1037.500000
MILLER	1300.00	CLERK	1037.500000
BLAKE	2850.00	MANAGER	2758.333333
CLARK	2450.00	MANAGER	2758.333333
JONES	2975.00	MANAGER	2758.333333
KING	5000.00	PRESIDENT	5000.000000
MARTIN	1250.00	SALESMAN	1400.000000
ALLEN	1600.00	SALESMAN	1400.000000
TURNER	1500.00	SALESMAN	1400.000000

ENAME	SAL	JOB	AVG_SAL_JOB
WARD	1250.00	SALESMAN	1400.000000

- 참고(C.f.), 그룹별 집계 (`[[GROUP BY]]` 사용):

```
SELECT JOB, AVG(SAL)
FROM EMP
GROUP BY JOB;
```

Result:

JOB	AVG(SAL)
PRESIDENT	5000.000000
MANAGER	2758.333333
SALESMAN	1400.000000
CLERK	1037.500000
ANALYST	3000.000000

> **Note**
> `[[GROUP BY]]` 는 결과를 그룹당 하나의 행으로 축소하지만, 윈도우 함수는 원본 행 수를 유지하면서 각 행에 집계 값을 추가합니다.

Ranking Examples #RankingFunctions

- 기본 쿼리 (전체 합계 - 프레임 명시): 전체 급여 합계를 각 행에 표시 (전체 파티션 프레임 사용).

```
SELECT EMPNO, ENAME, SAL,
       SUM(SAL) OVER(ORDER BY SAL
                     ROWS BETWEEN UNBOUNDED PRECEDING AND
```

```
UNBOUNDED FOLLOWING) AS TOTSAL
FROM EMP;
```

Result: (모든 행의 TOTSAL이 전체 합계인 29025.00)

- **누적 합계 (Cumulative Sum):** 급여 순서대로 누적 합계 계산.

```
SELECT EMPNO, ENAME, SAL,
       SUM(SAL) OVER(ORDER BY SAL
                     ROWS BETWEEN UNBOUNDED PRECEDING AND
CURRENT ROW) AS TOTSAL
FROM EMP;
```

Result:

EMPNO	ENAME	SAL	TOTSAL
7369	SMITH	800.00	800.00
7900	JAMES	950.00	1750.00
7876	ADAMS	1100.00	2850.00
7654	MARTIN	1250.00	4100.00
7521	WARD	1250.00	5350.00
...	...	...	...
7839	KING	5000.00	29025.00

- **전체 순위 및 파티션된 순위 (`[[RANK()]]`):** 전체 급여 순위 및 직업(JOB) 내 급여 순위.

```
SELECT ENAME, SAL,
       RANK() OVER (ORDER BY SAL DESC) AS ALL_RANK,
       RANK() OVER (PARTITION BY JOB ORDER BY SAL DESC) AS
JOB_RANK
FROM EMP;
```

Result: (RANK는 동점자 다음 순위에 갭 발생)

ENAME	SAL	ALL_RANK	JOB_RANK
FORD	3000.00	2	1
SCOTT	3000.00	2	1
JONES	2975.00	4	1
…	…	…	…
WARD	1250.00	10	3

- 전체 순위 및 파티션된 순위 (`[[DENSE_RANK()]]`): `DENSE_RANK` 는 동점자 다음 순위에 갭 없음.

```
SELECT ENAME, SAL,
       RANK() OVER (ORDER BY SAL DESC) AS ALL_RANK, -- For
comparison
       DENSE_RANK() OVER (PARTITION BY JOB ORDER BY SAL
DESC) AS JOB_RANK
FROM EMP;
```

Result: (JOB_RANK에서 DENSE_RANK 확인)

ENAME	SAL	ALL_RANK	JOB_RANK
FORD	3000.00	2	1
SCOTT	3000.00	2	1
MILLER	1300.00	9	1
ADAMS	1100.00	12	2
JAMES	950.00	13	3
SMITH	800.00	14	4
…	…	…	…

- 행 번호 및 순위 (`[[ROW_NUMBER()]]` , `[[RANK()]]`):

```
SELECT ROW_NUMBER() OVER (ORDER BY SAL DESC) AS ROW_NUM,
       ENAME, SAL,
```

```
       RANK() OVER (ORDER BY SAL DESC) AS ALL_RANK
FROM EMP;
```

Result: (ROW_NUM은 항상 고유, ALL_RANK는 동점 허용)

ROW_NUM	ENAME	SAL	ALL_RANK
1	KING	5000.00	1
2	FORD	3000.00	2
3	SCOTT	3000.00	2
4	JONES	2975.00	4
...	...	...	...

Nonaggregation Examples (Ranking Focus) #RankingFunctions

- 급여(SAL) 기준 순위 비교:

```
SELECT ENAME, SAL, JOB, HIREDATE,
       ROW_NUMBER() OVER (ORDER BY SAL) AS ROW_NUMBER_SAL,
       RANK() OVER (ORDER BY SAL) AS RANK_SAL,
       DENSE_RANK() OVER (ORDER BY SAL) AS DENSE_RANK_SAL
FROM EMP;
```

Result: (동점자 처리 방식 비교)

- 고용일자(HIREDATE) 기준 순위 비교:

```
SELECT ENAME, SAL, JOB, HIREDATE,
       ROW_NUMBER() OVER (ORDER BY HIREDATE) AS
ROW_NUMBER_HIREDATE,
       RANK() OVER (ORDER BY HIREDATE) AS RANK_HIREDATE,
       DENSE_RANK() OVER (ORDER BY HIREDATE) AS
DENSE_RANK_HIREDATE
FROM EMP;
```

Result: (동일 날짜 입사자 처리 방식 비교)

- 각 직업(JOB) 내 고용일자(HIREDATE) 기준 순위 (내림차순):

```
SELECT ENAME, SAL, JOB, HIREDATE,
       RANK() OVER (PARTITION BY JOB ORDER BY HIREDATE DESC)
AS RANK_HIREDATE
FROM EMP;
```

Result: (각 직업별 가장 최근 입사자 순위)

- 위와 동일 + `[[WINDOW]]` 절 사용:

```
SELECT ENAME, SAL, JOB, HIREDATE,
       RANK() OVER w AS RANK_HIREDATE
FROM EMP
WINDOW w AS (PARTITION BY JOB ORDER BY HIREDATE DESC);
```

> Tip
> `WINDOW` 절은 복잡하거나 재사용되는 윈도우 정의의 가독성을 높일 수 있습니다.
> Result: (이전 쿼리와 동일)

- 급여 기준 백분위 순위 (`[[RANK()]]`, `[[CUME_DIST()]]`, `[[PERCENT_RANK()]]`):

```
SELECT ENAME, SAL, JOB, HIREDATE,
       RANK() OVER (ORDER BY SAL) AS RANK_SAL,
       CUME_DIST() OVER (ORDER BY SAL) AS CUME_DIST_SAL,
       PERCENT_RANK() OVER (ORDER BY SAL) AS
PERCENT_RANK_SAL
FROM EMP;
```

Result: (상대적 순위 지표 확인)

- 위와 동일 + `[[WINDOW]]` 절 사용:

```
SELECT ENAME, SAL, JOB, HIREDATE,
       RANK() OVER w AS RANK_SAL,
       CUME_DIST() OVER w AS CUME_DIST_SAL,
       PERCENT_RANK() OVER w AS PERCENT_RANK_SAL
FROM EMP
WINDOW w AS (ORDER BY SAL);
```

Result: (이전 쿼리와 동일)

Value Window Examples #ValueWindowFunctions

Orders Table Example

ID	ORD_DATE	CUSTOMER_NAME	CITY	ORD_AMT
1001	2017-04-01	David Smith	GuildFord	10000.00
1002	2017-04-02	David Jones	Arlington	20000.00
1003	2017-04-03	John Smith	Shalford	5000.00
1004	2017-04-04	Michael Smith	GuildFord	15000.00
1005	2017-04-05	David Williams	Shalford	7000.00
1006	2017-04-06	Paum Smith	GuildFord	25000.00
1007	2017-04-10	Andrew Smith	Arlington	15000.00
1008	2017-04-11	David Brown	Arlington	2000.00
1009	2017-04-20	Robert Smith	Shalford	1000.00
1010	2017-04-25	Peter Smith	GuildFord	500.00

DIY Table Creation (Orders)

```
CREATE TABLE ORDERS (
    ID INT,
    ORD_DATE DATE,
```

```
    CUSTOMER_NAME VARCHAR(250),
    CITY VARCHAR(100),
    ORD_AMT DECIMAL(9,2)
);

INSERT INTO ORDERS(ID, ORD_DATE, CUSTOMER_NAME, CITY, ORD_AMT)
SELECT '1001','2017-04-01','David Smith','GuildFord',10000 UNION
ALL
SELECT '1002','2017-04-02','David Jones','Arlington',20000 UNION
ALL
SELECT '1003','2017-04-03','John Smith', 'Shalford', 5000 UNION
ALL
SELECT '1004','2017-04-04','Michael Smith','GuildFord', 15000
UNION ALL
SELECT '1005','2017-04-05','David Williams', 'Shalford', 7000
UNION ALL
SELECT '1006','2017-04-06','Paum Smith','GuildFord', 25000 UNION
ALL
SELECT '1007','2017-04-10','Andrew Smith','Arlington',15000
UNION ALL
SELECT '1008','2017-04-11','David Brown','Arlington',2000 UNION
ALL
SELECT '1009','2017-04-20', 'Robert Smith','Shalford', 1000
UNION ALL
SELECT '1010','2017-04-25','Peter Smith','GuildFord',500;
```

- 각 파티션(CITY)의 첫 번째 및 마지막 레코드:

```
SELECT ID, CITY, ORD_DATE,
       FIRST_VALUE(ORD_DATE) OVER(PARTITION BY CITY ORDER BY
ORD_DATE ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS FIRST_VAL,
       LAST_VALUE(ORD_DATE) OVER(PARTITION BY CITY ORDER BY
ORD_DATE ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) AS LAST_VAL
FROM ORDERS;
```

> **Note**
> `[[FIRST_VALUE()]]` 및 `[[LAST_VALUE()]]`가 파티션 전체에서 의미있는 첫/마지막 값을 반환하도록 하려면 `ORDER BY`와 함께 전체 파티션을 포함하는 프레임(`ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`)을 명시적으로 지정하는 것이 가장 좋습니다. 슬라이드의 원본 쿼리에는 `ORDER BY`와 프레임이 없었으나, 표시된 결과는 이를 암시하는 것처럼 보입니다. 위 코드는 명시적 정의를 사용합니다.

Result: (각 도시별 첫 주문일과 마지막 주문일)

- **이전 및 다음 레코드 (Offset 1):** 전체 주문을 날짜순으로 정렬했을 때, 이전 주문일과 다음 주문일.

```
SELECT ID, CUSTOMER_NAME, CITY, ORD_AMT, ORD_DATE,
       LAG(ORD_DATE, 1) OVER(ORDER BY ORD_DATE) AS
PREV_ORD_DAT,
       LEAD(ORD_DATE, 1) OVER(ORDER BY ORD_DATE) AS
NEXT_ORD_DAT
FROM ORDERS;
```

Result: (첫 행의 PREV_ORD_DAT는 NULL, 마지막 행의 NEXT_ORD_DAT는 NULL)

- **이전 및 다음 레코드 (Offset 2):** 전체 주문을 날짜순으로 정렬했을 때, 2개 이전 주문일과 2개 다음 주문일.

```
SELECT ID, CUSTOMER_NAME, CITY, ORD_AMT, ORD_DATE,
       LAG(ORD_DATE, 2) OVER(ORDER BY ORD_DATE) AS
PREV_ORD_DAT,
       LEAD(ORD_DATE, 2) OVER(ORDER BY ORD_DATE) AS
NEXT_ORD_DAT
FROM ORDERS;
```

Result: (처음 두 행의 PREV_ORD_DAT는 NULL, 마지막 두 행의 NEXT_ORD_DAT는 NULL)

Frame Specification #WindowFrame

- 프레임: 현재 파티션의 하위 집합이며, 프레임 절은 하위 집합을 정의하는 방법을 지정합니다.
- 프레임은 현재 행을 기준으로 결정됩니다.
  - 프레임을 파티션 시작부터 현재 행까지 모든 행으로 정의하여 각 행에 대한 누적 합계를 계산할 수 있습니다.
  - 프레임을 현재 행 양쪽으로 N개의 행을 확장하도록 정의하여 이동 평균을 계산할 수 있습니다.
- 프레임 유형:
  - `[[ROWS]]` : 프레임은 시작 및 종료 행 위치(물리적 창)로 정의됩니다.
  - `[[RANGE]]` : 프레임은 값 범위 내의 행(논리적 창)으로 정의됩니다.
- 프레임 끝점 지정 (`[[BETWEEN ... AND ...]]`):
  - `[[UNBOUNDED PRECEDING]]` : 파티션의 첫 번째 행이 경계입니다.
  - `[[UNBOUNDED FOLLOWING]]` : 파티션의 마지막 행이 경계입니다.
  - `[[CURRENT ROW]]` :
    - `ROWS` 의 경우: 현재 행이 경계입니다.
    - `RANGE` 의 경우: 현재 행과 동일한 값을 갖는 모든 행(피어)이 경계입니다.

Frame Specification Examples

- 각 파티션 전체 합계 (명시적 프레임): 각 도시(CITY)별 주문 금액(ORD_AMT)의 평균을 계산하되, 프레임을 파티션 전체로 명시적으로 지정합니다.

```
SELECT ID, CITY, ORD_AMT, ORD_DATE,
       AVG(ORD_AMT) OVER(PARTITION BY CITY ORDER BY ORD_DATE
                         ROWS BETWEEN UNBOUNDED PRECEDING
AND UNBOUNDED FOLLOWING) AS AVG_AMT
FROM ORDERS;
```

Result: (각 도시 내 모든 행에 해당 도시의 전체 평균 ORD_AMT가 표시됨)

- 2개 레코드 이동 평균 (`ROWS BETWEEN 1 PRECEDING AND 0 FOLLOWING`): 현재 행과 바로 이전 행의 평균 (2개 레코드).

```
SELECT ID, CITY, ORD_AMT, ORD_DATE,
       AVG(ORD_AMT) OVER(PARTITION BY CITY ORDER BY ORD_DATE
                         ROWS BETWEEN 1 PRECEDING AND 0
FOLLOWING) AS AVG_AMT
FROM ORDERS;
```

> **Note**
> `0 FOLLOWING` 구문은 "현재 행까지"를 의미할 수 있으며, SQL 방언에 따라 다를 수 있습니다. 이 예제에서는 현재 행과 이전 행을 포함하는 2개 레코드 프레임을 의미하는 것으로 보입니다. 더 표준적인 방법은 `CURRENT ROW`를 사용하는 것입니다 (다음 예제 참조).

Result: (각 행은 자신과 이전 행의 ORD_AMT 평균을 가짐, 파티션 첫 행은 자신의 값만 가짐)

- 2개 레코드 이동 평균 (`ROWS BETWEEN 1 PRECEDING AND CURRENT ROW`): 현재 행과 바로 이전 행의 평균 (표준 구문).

```
SELECT ID, CITY, ORD_AMT, ORD_DATE,
       AVG(ORD_AMT) OVER(PARTITION BY CITY ORDER BY ORD_DATE
                         ROWS BETWEEN 1 PRECEDING AND
CURRENT ROW) AS AVG_AMT
FROM ORDERS;
```

Result: (이전 예제와 동일한 결과)

- 3일 이동 평균 (`RANGE` 사용): 현재 행의 날짜를 포함하여 이전 2일간의 모든 주문에 대한 평균 ORD_AMT.

```
SELECT ID, ORD_DATE, ORD_AMT,
       AVG(ORD_AMT) OVER(ORDER BY ORD_DATE
                         RANGE BETWEEN INTERVAL '2' DAY
```

```
PRECEDING AND CURRENT ROW) AS AVG_AMT
FROM ORDERS;
```

Result: (예: 4월 3일 행은 4월 1일, 2일, 3일 주문의 평균을 표시)

- `[[INTERVAL]]`에 유효한 단위:

unit Value	Expected expr Format
`MICROSECOND`	MICROSECONDS
`SECOND`	SECONDS
`MINUTE`	MINUTES
`HOUR`	HOURS
`DAY`	DAYS
`WEEK`	WEEKS
`MONTH`	MONTHS
`QUARTER`	QUARTERS
`YEAR`	YEARS
`SECOND_MICROSECOND`	'SECONDS.MICROSECONDS'
`MINUTE_MICROSECOND`	'MINUTES:SECONDS.MICROSECONDS'
`MINUTE_SECOND`	'MINUTES:SECONDS'
`HOUR_MICROSECOND`	'HOURS:MINUTES:SECONDS.MICROSECONDS'
`HOUR_SECOND`	'HOURS:MINUTES:SECONDS'
`HOUR_MINUTE`	'HOURS:MINUTES'
`DAY_MICROSECOND`	'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'
`DAY_SECOND`	'DAYS HOURS:MINUTES:SECONDS'
`DAY_MINUTE`	'DAYS HOURS:MINUTES'
`DAY_HOUR`	'DAYS HOURS'
`YEAR_MONTH`	'YEARS-MONTHS'

프레임 절 예시:

- `ROWS BETWEEN 10 PRECEDING AND CURRENT ROW`
- `RANGE BETWEEN INTERVAL 5 DAY PRECEDING AND INTERVAL 1 DAY PRECEDING`
- `ROWS BETWEEN CURRENT ROW AND 5 FOLLOWING`
- `RANGE BETWEEN CURRENT ROW AND INTERVAL '2:30' MINUTE_SECOND FOLLOWING`

Reference: MySQL Temporal Intervals Documentation

Keys #Keys #DatabaseKeys

- Key: 관계 내에서 데이터 튜플을 고유하게 식별하는 데 도움이 되는 속성 또는 속성 집합입니다.

  Example Table:

EmployeeID	Name	Branch	Email
10201	Cooper	DBMI	cooper@institute.edu
10203	Abraham	DBMI	laboriel@institute.edu
10204	Abraham	CS	abe@institute.edu
10207	Elly	EE	elly@institute.edu

  - **Q:** 이 속성들 중 어떤 것이 키가 될 수 있을까요? (EmployeeID, Email)
- **키가 필요한 이유:**
  - 데이터의 식별성을 강제합니다.
  - 데이터 무결성 유지를 보장합니다.
  - 관계 간의 관계를 설정합니다.
- **키의 종류 (Types of Keys):**
  - Super key
  - Candidate key
  - Primary key (PK)

- Alternate key
- Foreign key (FK)
- Composite key
- Compound key
- Surrogate key

Super Keys #SuperKey

- 가능한 모든 고유 식별자.
- 관계에서 데이터 튜플을 식별하는 데 사용할 수 있는 모든 속성 또는 속성 집합입니다. 즉, 다음 중 하나입니다:
  - 고유한 값을 가진 속성 또는
  - 속성들의 조합
- **예시 (아래 테이블 기준):

EmployeeID	FileCD	Name	Branch	Email
10201	D-201-C	Cooper	DBMI	cooper@institute.edu
10203	D-203-A	Abraham	DBMI	laboriel@institute.edu
10204	C-204-A	Abraham	CS	abe@institute.edu
10207	E-207-E	Elly	EE	elly@institute.edu

  - `{[[EmployeeID]]}`
  - `{[[FileCD]]}`
  - `{[[Email]]}`
  - `{EmployeeID, FileCD}`
  - `{EmployeeID, Name}`
  - `{FileCD, Branch}` (이 예제에서는 고유하지 않을 수 있지만, 고유하다면 수퍼키)
  - `{EmployeeID, FileCD, Name, Branch, Email}` (전체 속성 집합은 항상 수퍼키)

- ... 등등, 튜플을 고유하게 식별하는 모든 조합.

Candidate Keys #CandidateKey

- 수퍼 키의 최소 하위 집합.
- 수퍼 키의 진부분집합(proper subset) 또한 수퍼 키라면, 그 원래의 (더 큰) 수퍼 키는 후보 키가 될 수 없습니다.
- 예시 (위 테이블 기준):
  - 수퍼 키: `{EmployeeID}`, `{FileCD}`, `{Email}`, `{EmployeeID, FileCD}`, `{EmployeeID, Email}`, ...
  - `{EmployeeID}`는 최소입니다 (더 작은 부분집합 없음). -> 후보 키
  - `{FileCD}`는 최소입니다. -> 후보 키
  - `{Email}`는 최소입니다. -> 후보 키
  - `{EmployeeID, FileCD}`는 수퍼 키이지만, 부분집합인 `{EmployeeID}`와 `{FileCD}`가 모두 수퍼 키이므로, `{EmployeeID, FileCD}`는 *후보 키가 아닙니다*.
  - 다른 조합들도 마찬가지입니다.
  - 후보 키 목록: `[[EmployeeID]]`, `[[FileCD]]`, `[[Email]]`

Primary Keys (PKs) #PrimaryKey

- 관계에서 데이터의 각 행(row)을 고유하게 식별하기 위해 선택된 후보 키.
- 규칙:
  - 어떤 두 행도 동일한 PK 값을 가질 수 없습니다.
  - PK 값은 NULL일 수 없습니다 (모든 행은 기본 키 값을 가져야 합니다).

> **Important**
> 기본 키(PK) 값은 NULL이 될 수 없습니다.

- 예시: 위 후보 키 (`EmployeeID` , `FileCD` , `Email`) 중에서 하나를 PK로 선택합니다. 예를 들어, `[[EmployeeID]]` 를 PK로 선택할 수 있습니다. (Pick any one as PK)

Alternate Keys #AlternateKey

- 관계에서 기본 키(PK)로 선택되지 않은 후보 키들.
- 예시: 만약 `[[EmployeeID]]` 를 PK로 선택했다면, `[[FileCD]]` 와 `[[Email]]` 은 대체 키가 됩니다.

Foreign Keys (FKs) #ForeignKey

- 다른 관계와의 관계를 정의하는 데 사용되는 한 관계 내의 속성(들). (보통 다른 테이블의 PK를 참조)
- 외래 키 사용은 관계형 테이블 간의 데이터 무결성 유지에 도움이 됩니다.
- 예시:

  Employee Table:

EmployeeID	FileCD	Name	Branch	Email
10201	D-201-C	Cooper	DBMI	cooper@institute.edu
10203	D-203-A	Abraham	DBMI	laboriel@institute.edu
10204	C-204-A	Abraham	CS	abe@institute.edu
10207	E-207-E	Elly	EE	elly@institute.edu

  - `Branch` (FK referencing Branch.Branch)

  Branch Table:

Branch	Address
DBMI	5607 Baum Blvd
CS	260 S Bouquet St

Branch	Address
EE	3700 O'Hara St
BIO	4249 Fifth Ave

- `Branch` (PK)
- Employee 테이블의 `[[Branch]]` 속성은 Branch 테이블의 `Branch` (PK)를 참조하는 Foreign key입니다.

Composite & Compound Keys #CompositeKey #CompoundKey

- 복합 키: 둘 이상의 속성으로 구성된 모든 키 (수퍼 키, 후보 키, 기본 키 등).
  - 예시: `{EmployeeID, FileCD}`, `{EmployeeID, Email}`, `{FileCD, Email}` 등은 (이 예제에서는 후보 키는 아니지만) 복합 수퍼 키입니다. 만약 `{FileCD, Branch}`가 고유성을 보장하여 키(예: 후보 키)로 정의되었다면, 이것은 복합 키가 됩니다.
- 화합 키: 하나 이상의 속성이 외래 키인 복합 키.
  - 예시: 만약 `{FileCD, Branch}`가 복합 키로 정의되었고, `Branch` 속성이 Branch 테이블을 참조하는 외래 키라면, `{FileCD, Branch}`는 화합 키입니다.

Surrogate Keys #SurrogateKey

- 관계에 키로 사용할 수 있는 (자연스러운) 속성이 없는 경우, 이 목적을 위해 생성하는 인공 속성.
- 데이터 자체에는 의미를 추가하지 않지만, 테이블 내에서 튜플을 고유하게 식별하는 유일한 목적을 수행합니다.
- 예시: 종종 auto increment 기능이 있는 `_ID` 열 (예: `CustomerID`, `OrderID`).

EOF

- Coming next:
  - Transactions

---

4. 핵심 주요 키워드

- Join
- Views
- Window functions
- Keys
- 논리적 모델(Logical Model)
- View
- base tables
- relation
- 개념적 모델(Conceptual Model)
- 가상 관계(virtual relation)
- SQL expression
- view name
- View definition
- substituted into queries
- CREATE VIEW
- faculty
- instructor
- departments_total_salary
- SUM
- GROUP BY
- sakila
- View expansion
- recursive

- depend directly
- depend on
- view relation
- physics_fall_2017
- course
- section
- physics_fall_2017_watson
- Virtual
- Materialized
- Materialized view
- VIEW
- Materialized View
- Materialized Views
- refresh
- materialized views
- Update via a View
- INSERT INTO
- NULL
- department
- SQL error (1394)
- history_instructors
- simple views
- FROM
- SELECT
- aggregates
- DISTINCT
- HAVING
- Window Functions in SQL
- SQL:2003
- Built-in functions
- Window function

- ranks
- percentiles
- sums
- averages
- row numbers
- aggregation functions
- moving sums
- moving averages
- WINDOW_FUNCTION clause
- PARTITION
- Aggregate window functions
- SUM()
- MAX()
- MIN()
- AVG()
- COUNT()
- Ranking window functions
- RANK()
- DENSE_RANK()
- PERCENT_RANK()
- ROW_NUMBER()
- NTILE()
- Value window functions
- LAG()
- LEAD()
- FIRST_VALUE()
- LAST_VALUE()
- CUME_DIST()
- NTH_VALUE()
- WINDOW_FUNCTION
- ALL

- OVER
- PARTITION BY partition_list
- ORDER BY order_list
- frame_clause
- DEPT
- EMP
- CREATE TABLE
- Aggregation Examples
- PARTITION BY
- ORDER BY
- ROWS BETWEEN
- UNBOUNDED PRECEDING
- UNBOUNDED FOLLOWING
- CURRENT ROW
- WINDOW
- HIREDATE
- Nonaggregation Examples
- Orders
- Value Window Examples
- Frame Specification
- frame
- partition
- frame clause
- current row
- running totals
- rolling averages
- ROWS
- RANGE
- BETWEEN ... AND ...
- Frame Specification Examples
- RANGE BETWEEN

- INTERVAL
- identity
- integrity
- relationship
- Super key
- Candidate key
- Primary key
- Alternate key
- Foreign key
- Composite key
- Compound key
- Surrogate key
- unique identifier
- EmployeeID
- FileCD
- Email
- Minimal subset
- Primary Keys (PKs)
- Foreign Keys (FKs)
- data integrity
- Branch
- Composite & Compound Keys
- key
- Compound key
- Surrogate Keys
- artificial attribute
- auto increment
- Transactions