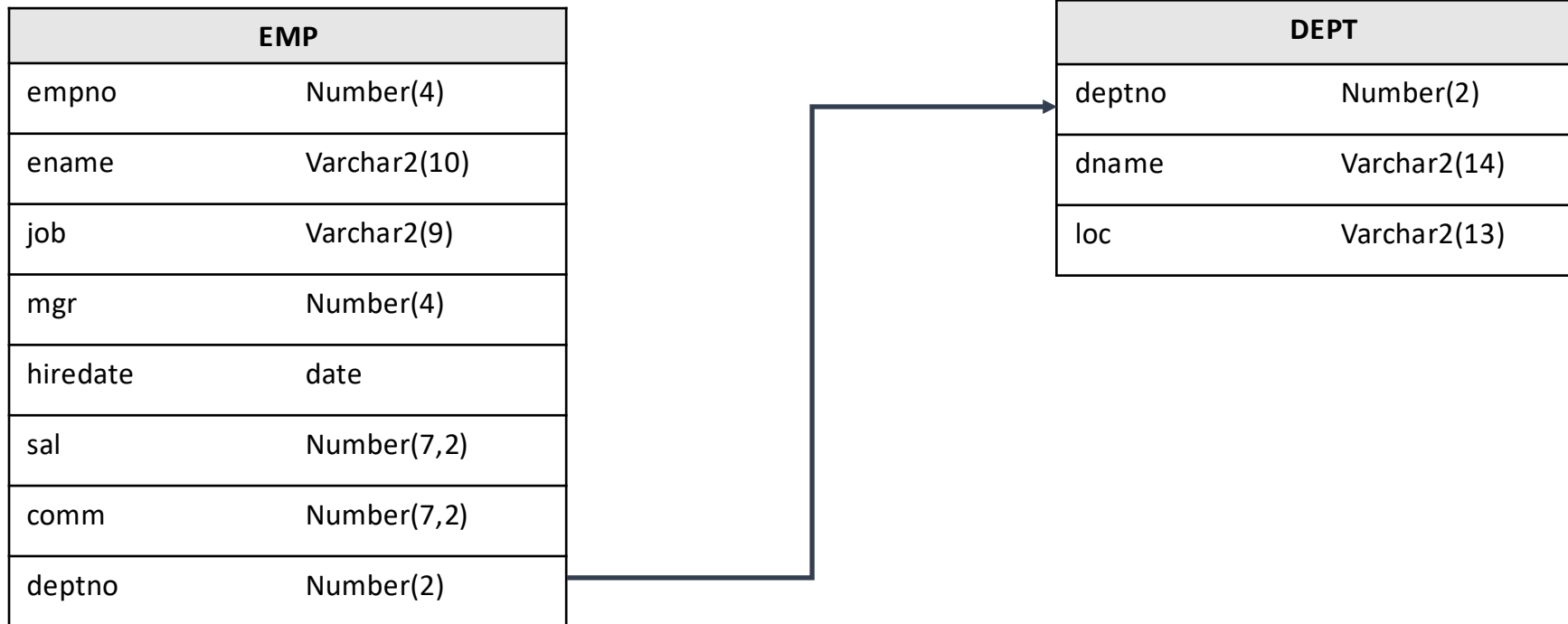


Basic SQL Practice

VLDB Lab.

Professor Sangwon Lee

Scott Schema



Recap – Basic SQL

- **Basic SQL**
 - CREATE TABLE / DROP TABLE
 - INSERT / UPDATE / DELETE
 - SELECT .. FROM .. WHERE
 - JOIN
 - ORDER BY / GROUP BY / HAVING
 - COUNT(), SUM(), AVG()

Airbnb Interview - Problem

Airbnb의 연도별 성장을 추정하세요. 여기서 성장을 등록된 호스트 수를 기반으로 측정됩니다. 성장률은 아래 공식으로 계산됩니다.

$$\text{성장률 (\%)} = \left(\frac{\text{현재 연도의 호스트 수} - \text{이전 연도의 호스트 수}}{\text{이전 연도의 호스트 수}} \right) \times 100$$

결과로 다음을 출력하세요: 연도, 현재 연도의 호스트 수, 이전 연도의 호스트 수, 성장률(정수로 반올림)

결과는 연도 순서대로 정렬하며, 데이터셋에는 중복된 호스트가 없다고 가정합니다.

airbnb_search_details

Preview

id:	int
price:	float
property_type:	varchar
room_type:	varchar
amenities:	varchar
accommodates:	int
bathrooms:	int
bed_type:	varchar
cancellation_policy:	varchar
cleaning_fee:	bool
city:	varchar
host_identity_verified:	varchar
host_response_rate:	varchar
host_since:	datetime
neighbourhood:	varchar
number_of_reviews:	int
review_scores_rating:	float
zipcode:	int
bedrooms:	int
beds:	int

Airbnb Interview - Solution

```
WITH each_year AS  
(  
  SELECT datepart(year,host_since) AS Year, count(*) AS hosts  
  FROM airbnb_search_details  
  GROUP BY datepart(year,host_since)  
)
```

```
SELECT year, hosts AS Current_year,
```

```
LAG(hosts) over(ORDER BY year ASC) AS Last_year,
```

```
Round((((hosts - LAG(hosts) over(ORDER BY year ASC)*1.0)/hosts)*100,0) AS [Growth Rate]
```

```
FROM each_year
```

```
ORDER BY year
```

➡ CTE 사용(WITH each_year AS)

➡ LAG 윈도우 함수를 사용해서 현재 행 이전의 행 (이전 연도)의 호스트 수를 가져옴.

➡ 성장률 공식에 따라서 성장률 계산

Google Interview - Problem

각 사용자별 이메일 활동 순위를 찾아야 한다. 이메일 활동 순위는 총 발송된 이메일 수로 정의된다. 가장 많은 이메일을 보낸 사용자가 순위 1을 가지며, 그 뒤로 순위가 매겨진다. 출력을 사용자, 총 이메일 수, 활동 순위를 포함해야 한다.

조건:

1. 이메일 총 수를 기준으로 내림차순으로 정렬한다.
2. 동일한 이메일 수를 가진 사용자는 사용자 이름 알파벳 순으로 정렬한다.
3. 순위는 고유한 값으로 반환된다. (즉, 동일한 이메일 수를 가진 사용자도 각자의 고유 순위를 가진다.)

google_gmail_emails

Preview

day:	bigint
from_user:	varchar
id:	bigint
to_user:	varchar

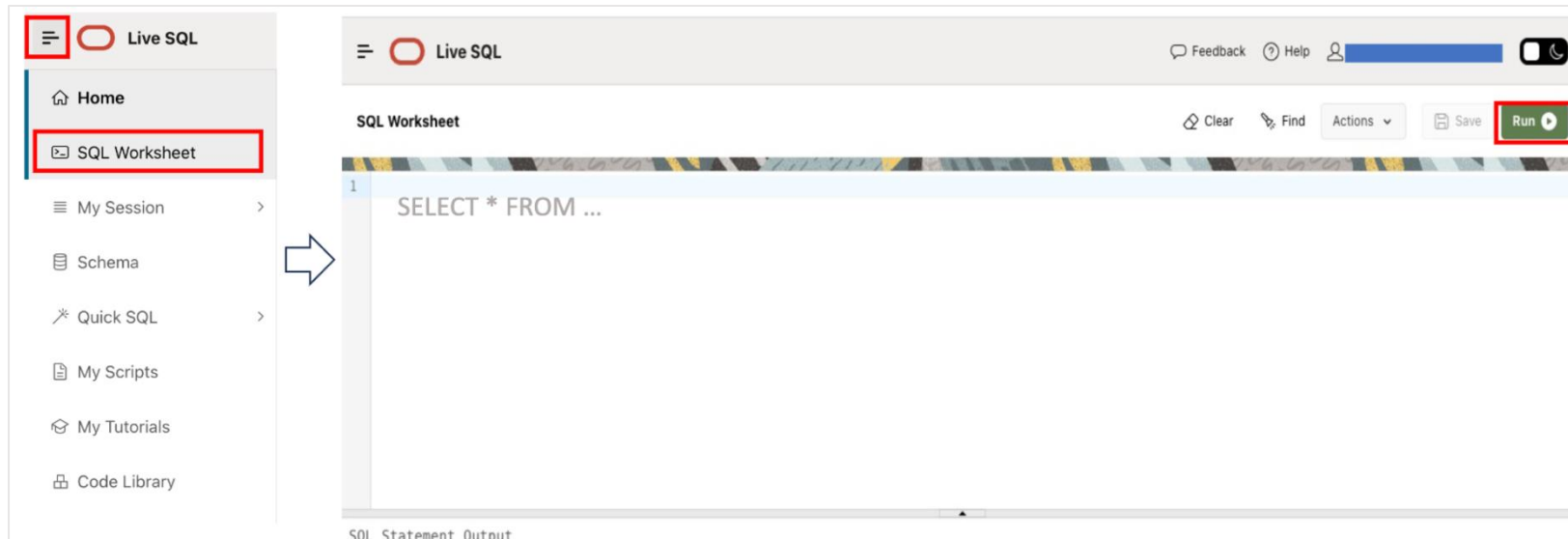
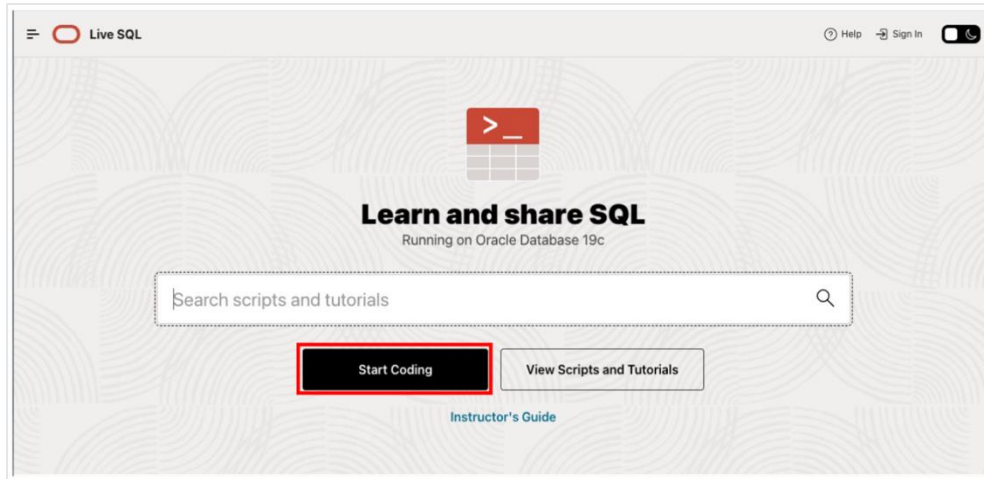
Google Interview - Solution

```
SELECT
  from_user,
  COUNT(id) AS [Total Emails],
  DENSE_RANK() OVER (ORDER BY COUNT(id) DESC, from_user) AS [Rank]
FROM
  google_gmail_emails
GROUP BY
  from_user
ORDER BY
  [Total Emails] DESC,
  from_user ASC;
```

- ➡ 사용자가 보낸 메일 수를 세어 Total Emails라는 이름으로 출력
- ➡ 이메일 수를 기준으로 순위를 매기며, 내림차순 정렬
- ➡ 사용자별 그룹화
- ➡ Total Emails 기준으로 내림차순. 동일한 이메일 수를 가진 경우 from_user 알파벳 순으로 오름차순

Advanced SQL using LiveSQL

1. Sign In to LiveSQL. (<https://livesql.oracle.com/>)

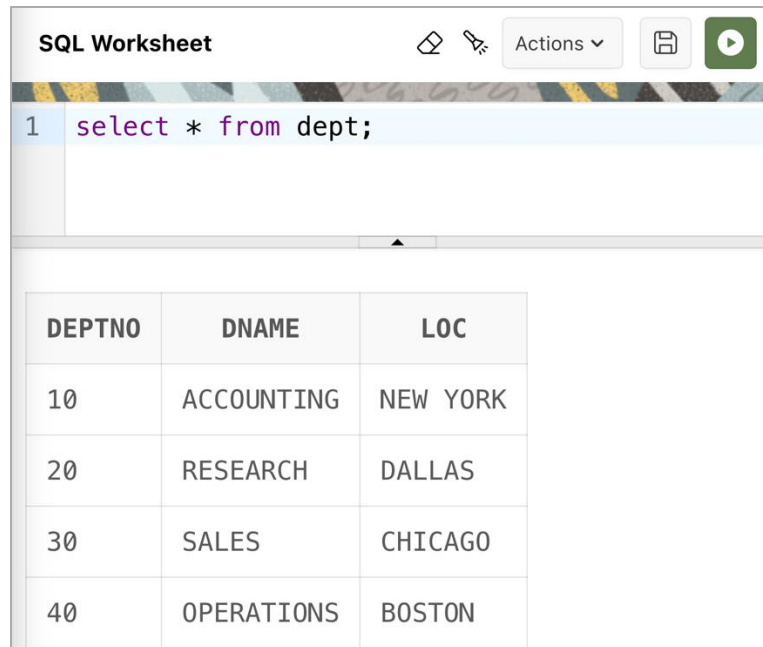


Advanced SQL using LiveSQL

2. LiveSQL worksheet에 scott schema 복사 후 실행

(<https://github.com/snu-vldb-ta/SNU-BigData-Fintech-2025-1H/blob/main/1/scott.sql>)

3. Scott schema 실행 후 데이터가 잘 들어갔는지 아래 select 문을 통해 확인



The screenshot shows a web interface for a SQL worksheet. At the top, there's a header bar with the text "SQL Worksheet" and several icons. Below the header, a text area contains the SQL query: `1 select * from dept;`. Below the text area, a table displays the results of the query. The table has three columns: DEPTNO, DNAME, and LOC. It contains four rows of data representing different departments.

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

Advanced SQL using LiveSQL – Cube, Rollup

```
SELECT deptno, job, sum(sal)
FROM emp
GROUP BY CUBE(deptno,job);
```

DEPTNO	JOB	SUM(SAL)
–	–	29025
–	CLERK	4150
–	ANALYST	6000
–	MANAGER	8275
–	SALESMAN	5600
–	PRESIDENT	5000
10	–	8750
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	–	10875
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30	–	9400
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

```
SELECT job, deptno, sum(sal)
FROM emp
GROUP BY ROLLUP(deptno,job);
```

JOB	DEPTNO	SUM(SAL)
CLERK	10	1300
MANAGER	10	2450
PRESIDENT	10	5000
–	10	8750
CLERK	20	1900
ANALYST	20	6000
MANAGER	20	2975
–	20	10875
CLERK	30	950
MANAGER	30	2850
SALESMAN	30	5600
–	30	9400
–	–	29025

Advanced SQL using LiveSQL – Cube, Rollup

다른 예제)

```
CREATE TABLE sales (  
    sale_date DATE,  
    category VARCHAR2(50),  
    region VARCHAR2(50),  
    amount NUMBER  
);
```

```
INSERT INTO sales (sale_date, category, region, amount)  
VALUES (TO_DATE('2024-01-01', 'YYYY-MM-DD'), 'Electronics', 'North', 1500);
```

```
INSERT INTO sales (sale_date, category, region, amount)  
VALUES (TO_DATE('2024-01-01', 'YYYY-MM-DD'), 'Electronics', 'South', 1200);
```

```
INSERT INTO sales (sale_date, category, region, amount)  
VALUES (TO_DATE('2024-01-01', 'YYYY-MM-DD'), 'Clothing', 'North', 800);
```

```
INSERT INTO sales (sale_date, category, region, amount)  
VALUES (TO_DATE('2024-01-01', 'YYYY-MM-DD'), 'Clothing', 'South', 600);
```

```
INSERT INTO sales (sale_date, category, region, amount)  
VALUES (TO_DATE('2024-02-01', 'YYYY-MM-DD'), 'Electronics', 'North', 1600);
```

```
INSERT INTO sales (sale_date, category, region, amount)  
VALUES (TO_DATE('2024-02-01', 'YYYY-MM-DD'), 'Clothing', 'South', 700);
```

```
INSERT INTO sales (sale_date, category, region, amount)  
VALUES (TO_DATE('2024-02-01', 'YYYY-MM-DD'), 'Electronics', 'South', 1300);
```

Advanced SQL using LiveSQL – Cube, Rollup

1. sale_date, category, region의 모든 조합에 대한 합계 (SUM(amount))는 무엇인가?
2. sale_date와 category의 조합별 매출을 집계하고, 각각의 조합 및 전체 합계를 포함한 결과를 SQL로 구해라.
(region의 차원은 무시하고, cube를 사용해서 구현해라.)
3. 매출 데이터를 sale_date, category, region의 계층적 조합에 따라 집계하고, 각 차원별 총합 및 전체 합계를 포함한 결과를 구해라.
4. sale_date와 category의 계층적 조합에 따라 매출을 집계하고, 각 차원의 합계 및 전체 매출 합계를 포함한 결과를 SQL로 구해라. region은 포함하지 말고, SQL ROLLUP을 사용해라.

Advanced SQL using LiveSQL – Cube, Rollup

1. sale_date, category, region의 모든 조합에 대한 합계 (SUM(amount))는 무엇인가?

```
SELECT sale_date, category, region, SUM(amount) AS total_amount FROM sales GROUP BY CUBE (sale_date, category, region);
```

2. sale_date와 category의 조합별 매출을 집계하고, 각각의 조합 및 전체 합계를 포함한 결과를 SQL로 구해라.

(region의 차원은 무시하고, cube를 사용해서 구현해라.)

```
SELECT sale_date, category, SUM(amount) AS total_amount FROM sales GROUP BY CUBE (sale_date, category);
```

3. 매출 데이터를 sale_date, category, region의 계층적 조합에 따라 집계하고, 각 차원별 총합 및 전체 합계를 포함한 결과를 구해라.

```
SELECT sale_date, category, region, SUM(amount) AS total_amount FROM sales GROUP BY ROLLUP (sale_date, category, region);
```

4. sale_date와 category의 계층적 조합에 따라 매출을 집계하고, 각 차원의 합계 및 전체 매출 합계를 포함한 결과를 SQL로 구해라. region은 포함하지 말고, SQL ROLLUP을 사용해라.

```
SELECT sale_date, category, SUM(amount) AS total_amount FROM sales GROUP BY ROLLUP (sale_date, category);
```

Advanced SQL using LiveSQL – Analytic Function

1. EMP 테이블에서 직원의 직업별 및 전체 급여 순위는 어떻게 되는가?

Advanced SQL using LiveSQL – Analytic Function

1. EMP 테이블에서 직원의 직업별 및 전체 급여 순위는 어떻게 되는가?

```
1 SELECT JOB, ENAME, SAL,  
2 RANK() OVER (ORDER BY SAL DESC) ALL_RANK,  
3 RANK() OVER (PARTITION BY Job ORDER BY SAL DESC) JOB_RANK  
4 FROM EMP;
```

JOB	ENAME	SAL	ALL_RANK	JOB_RANK
PRESIDENT	KING	5000	1	1
ANALYST	FORD	3000	2	1
ANALYST	SCOTT	3000	2	1
MANAGER	JONES	2975	4	1
MANAGER	BLAKE	2850	5	2
MANAGER	CLARK	2450	6	3
SALESMAN	ALLEN	1600	7	1
SALESMAN	TURNER	1500	8	2
CLERK	MILLER	1300	9	1
SALESMAN	WARD	1250	10	3
SALESMAN	MARTIN	1250	10	3
CLERK	ADAMS	1100	12	2
CLERK	JAMES	950	13	3
CLERK	SMITH	800	14	4

```
1 SELECT JOB, ENAME, SAL,  
2 RANK() OVER (ORDER BY SAL DESC) ALL_RANK,  
3 DENSE_RANK() OVER (PARTITION BY Job ORDER BY SAL DESC) JOB_RANK  
4 FROM EMP;
```

JOB	ENAME	SAL	ALL_RANK	JOB_RANK
PRESIDENT	KING	5000	1	1
ANALYST	FORD	3000	2	1
ANALYST	SCOTT	3000	2	1
MANAGER	JONES	2975	4	1
MANAGER	BLAKE	2850	5	2
MANAGER	CLARK	2450	6	3
SALESMAN	ALLEN	1600	7	1
SALESMAN	TURNER	1500	8	2
CLERK	MILLER	1300	9	1
SALESMAN	WARD	1250	10	3
SALESMAN	MARTIN	1250	10	3
CLERK	ADAMS	1100	12	2
CLERK	JAMES	950	13	3
CLERK	SMITH	800	14	4

Advanced SQL using LiveSQL – Analytic Function

2. 관리자별로 각 직원의 급여와 관리자 그룹의 총 급여 합계를 출력해라. (PARTITION BY 사용)

Advanced SQL using LiveSQL – Analytic Function

2. 관리자별로 각 직원의 급여와 관리자 그룹의 총 급여 합계를 출력해라. (PARTITION BY 사용)

```
1 SELECT MGR, ENAME, SAL,  
2 sum(sal) OVER (PARTITION BY MGR) MGR_SUM  
3 FROM EMP;
```

MGR	ENAME	SAL	MGR_SUM
7566	FORD	3000	6000
7566	SCOTT	3000	6000
7698	JAMES	950	6550
7698	ALLEN	1600	6550
7698	WARD	1250	6550
7698	TURNER	1500	6550
7698	MARTIN	1250	6550
7782	MILLER	1300	1300
7788	ADAMS	1100	1100
7839	BLAKE	2850	8275
7839	JONES	2975	8275
7839	CLARK	2450	8275
7902	SMITH	800	800
-	KING	5000	5000

Advanced SQL using LiveSQL – Analytic Function

3. 관리자별로 각 직원의 급여와 관리자 그룹의 총 급여 합계를 출력해라. (PARTITION BY 사용)

Advanced SQL using LiveSQL – Analytic Function

3. 관리자별로 각 직원의 급여와 관리자 그룹의 총 급여 합계를 출력해라. (PARTITION BY 사용)

```
1 select ename, sal, count(*) over
2   (order by sal range between 50 preceding and 150 following)
3   as sim_cnt from emp;
4
```

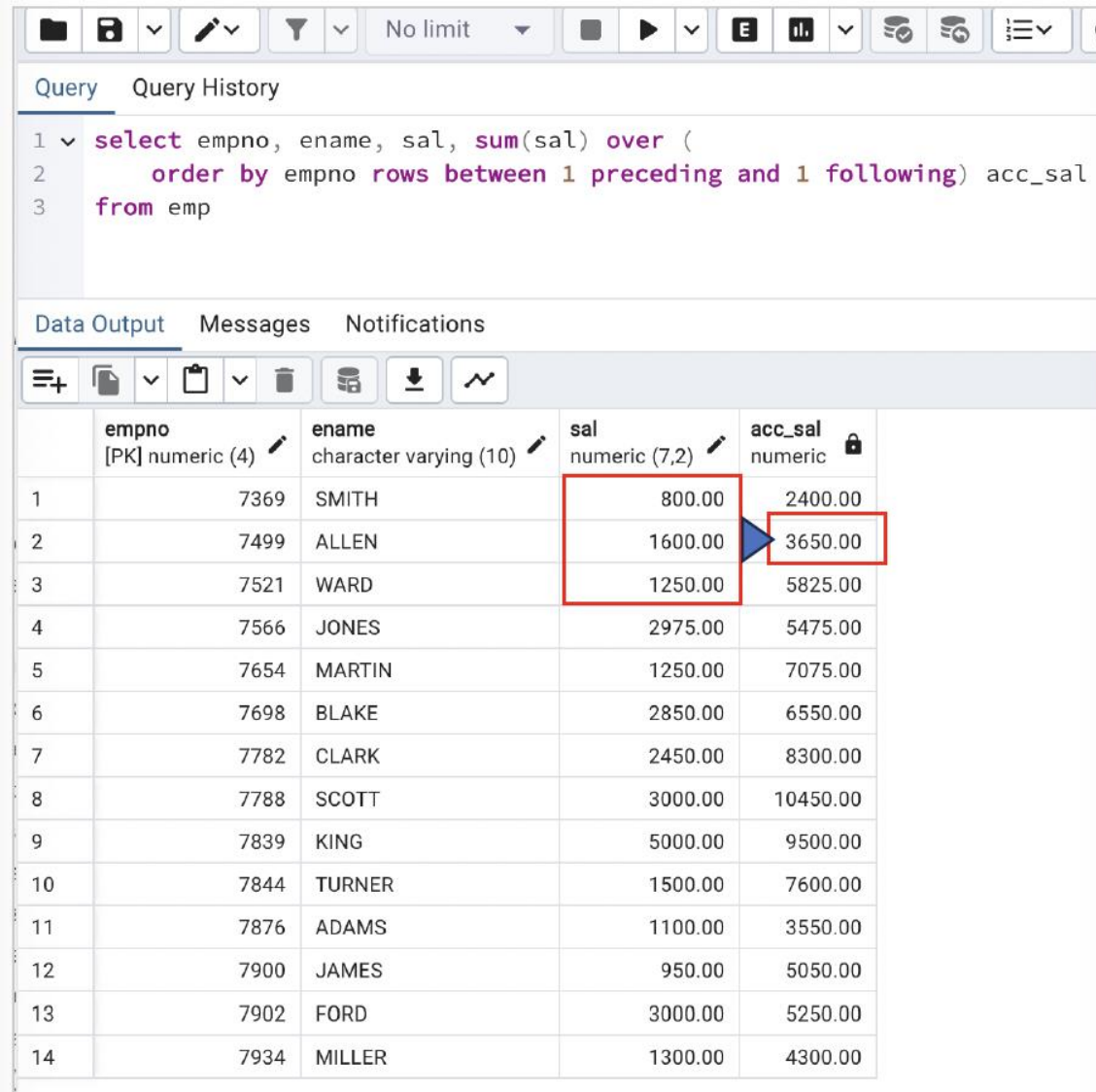
ENAME	SAL	SIM_CNT
SMITH	800	2
JAMES	950	2
ADAMS	1100	3
WARD	1250	3
MARTIN	1250	3
MILLER	1300	3
TURNER	1500	2
ALLEN	1600	1
CLARK	2450	1
BLAKE	2850	4
JONES	2975	3
SCOTT	3000	3
FORD	3000	3
KING	5000	1

Advanced SQL using LiveSQL – Analytic Function

다른 쿼리들도 따라해보면서 어떤 의미인지 파악해보세요.

- **select deptno, ename, sal, cume_dist() over (partition by deptno order by sal desc) as cume_dist from emp;**
- **select deptno, ename, sal, first_value(ename) over (partition by deptno order by sal desc rows unbounded preceding) as dept_rich from emp;**
- **select deptno, ename, sal, percent_rank() over (partition by deptno order by sal desc) as P_R from emp;**

Advanced SQL using LiveSQL – Window Function



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, query execution, and settings. Below the toolbar, the 'Query' tab is active, displaying the following SQL query:

```
1 select empno, ename, sal, sum(sal) over (  
2     order by empno rows between 1 preceding and 1 following) acc_sal  
3 from emp
```

The 'Data Output' tab is also active, showing the results of the query. The table has five columns: empno, ename, sal, and acc_sal. The acc_sal column represents the cumulative salary for each employee, calculated using a window function. The results are as follows:

	empno [PK] numeric (4)	ename character varying (10)	sal numeric (7,2)	acc_sal numeric
1	7369	SMITH	800.00	2400.00
2	7499	ALLEN	1600.00	3650.00
3	7521	WARD	1250.00	5825.00
4	7566	JONES	2975.00	5475.00
5	7654	MARTIN	1250.00	7075.00
6	7698	BLAKE	2850.00	6550.00
7	7782	CLARK	2450.00	8300.00
8	7788	SCOTT	3000.00	10450.00
9	7839	KING	5000.00	9500.00
10	7844	TURNER	1500.00	7600.00
11	7876	ADAMS	1100.00	3550.00
12	7900	JAMES	950.00	5050.00
13	7902	FORD	3000.00	5250.00
14	7934	MILLER	1300.00	4300.00

In the table, the rows for employees SMITH, ALLEN, and WARD are highlighted with red boxes. A blue arrow points from the 'sal' value of 1600.00 for ALLEN to the 'acc_sal' value of 3650.00, illustrating the cumulative sum calculation.

Advanced SQL using LiveSQL

나머지 예제)

- Analytic Functions (교수님 교안)
- Recursive CTE (교수님 교안, [github](#))