به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس آزمایشگاه پایگاه‌داده

پیش‌گزارش هفتم

نام و نام خانوادگی	میثاق محقق
شماره دانشجویی	**810199484**
تاریخ ارسال گزارش	**1402.09.26**

فهرست

پاسخ 1. توابع پنجره‌ای

1-0. جدول student_score

ابتدا یک جدول نمونه در دیتابیس ای جدید (به نام ex7) می‌سازیم:

```
DROP TABLE IF EXISTS student_score;

CREATE TABLE student_score (
    student_id SERIAL PRIMARY KEY,
    student_name VARCHAR(30),
    dep_name VARCHAR(40),
    score INT
);

INSERT INTO student_score VALUES (11, 'Ibrahim', 'Computer Science', 80);
INSERT INTO student_score VALUES (7, 'Taiwo', 'Microbiology', 76);
INSERT INTO student_score VALUES (9, 'Nurain', 'Biochemistry', 80);
INSERT INTO student_score VALUES (8, 'Joel', 'Computer Science', 90);
INSERT INTO student_score VALUES (10, 'Mustapha', 'Industrial Chemistry', 78);
INSERT INTO student_score VALUES (5, 'Muritadoh', 'Biochemistry', 85);
INSERT INTO student_score VALUES (2, 'Yusuf', 'Biochemistry', 70);
INSERT INTO student_score VALUES (3, 'Habeebah', 'Microbiology', 80);
INSERT INTO student_score VALUES (1, 'Tomiwa', 'Microbiology', 65);
INSERT INTO student_score VALUES (4, 'Gbadebo', 'Computer Science', 80);
INSERT INTO student_score VALUES (12, 'Tolu', 'Computer Science', 67);
```

```
ex7=# \dt
            List of relations
 Schema |     Name      | Type  |  Owner
--------+---------------+-------+----------
 public | student_score | table | postgres
(1 row)


ex7=# SELECT * FROM student_score;
 student_id | student_name |       dep_name       | score
------------+--------------+----------------------+-------
         11 | Ibrahim      | Computer Science     |    80
          7 | Taiwo        | Microbiology         |    76
          9 | Nurain       | Biochemistry         |    80
          8 | Joel         | Computer Science     |    90
         10 | Mustapha     | Industrial Chemistry |    78
          5 | Muritadoh    | Biochemistry         |    85
          2 | Yusuf        | Biochemistry         |    70
          3 | Habeebah     | Microbiology         |    80
          1 | Tomiwa       | Microbiology         |    65
          4 | Gbadebo      | Computer Science     |    80
         12 | Tolu         | Computer Science     |    67
(11 rows)
```

همانطور که می‌بینیم، داده‌ها به طور صحیح وارد جدول student_score شده اند.

1-1. محاسبه برای کل جدول

می‌خواهیم بیشترین و کمترین score در بین تمامی سطرهای جدول را حساب کنیم:

```
SELECT *, MAX(score) OVER() AS maximum_score, MIN(score) OVER() AS minimum_score
FROM student_score;
```

```
ex7=# SELECT *, MAX(score) OVER() AS maximum_score, MIN(score) OVER() AS minimum_score
ex7-# FROM student_score;
 student_id | student_name |       dep_name       | score | maximum_score | minimum_score
------------+--------------+----------------------+-------+---------------+---------------
         11 | Ibrahim      | Computer Science     |    80 |            90 |            65
          7 | Taiwo        | Microbiology         |    76 |            90 |            65
          9 | Nurain       | Biochemistry         |    80 |            90 |            65
          8 | Joel         | Computer Science     |    90 |            90 |            65
         10 | Mustapha     | Industrial Chemistry |    78 |            90 |            65
          5 | Muritadoh    | Biochemistry         |    85 |            90 |            65
          2 | Yusuf        | Biochemistry         |    70 |            90 |            65
          3 | Habeebah     | Microbiology         |    80 |            90 |            65
          1 | Tomiwa       | Microbiology         |    65 |            90 |            65
          4 | Gbadebo      | Computer Science     |    80 |            90 |            65
         12 | Tolu         | Computer Science     |    67 |            90 |            65
(11 rows)
```

این کار را می‌توانستیم بدون توابع پنجره‌ای و با استفاده از subquery-ها هم انجام دهیم:

```
SELECT *, (SELECT MAX(score) FROM student_score) AS maximum_score,
          (SELECT MIN(score) FROM student_score) AS minimum_score
FROM student_score;
```

```
ex7=# SELECT *, (SELECT MAX(score) FROM student_score) AS maximum_score,
ex7-#           (SELECT MIN(score) FROM student_score) AS minimum_score
ex7-# FROM student_score;
 student_id | student_name |       dep_name       | score | maximum_score | minimum_score
------------+--------------+----------------------+-------+---------------+---------------
         11 | Ibrahim      | Computer Science     |    80 |            90 |            65
          7 | Taiwo        | Microbiology         |    76 |            90 |            65
          9 | Nurain       | Biochemistry         |    80 |            90 |            65
          8 | Joel         | Computer Science     |    90 |            90 |            65
         10 | Mustapha     | Industrial Chemistry |    78 |            90 |            65
          5 | Muritadoh    | Biochemistry         |    85 |            90 |            65
          2 | Yusuf        | Biochemistry         |    70 |            90 |            65
          3 | Habeebah     | Microbiology         |    80 |            90 |            65
          1 | Tomiwa       | Microbiology         |    65 |            90 |            65
          4 | Gbadebo      | Computer Science     |    80 |            90 |            65
         12 | Tolu         | Computer Science     |    67 |            90 |            65
(11 rows)
```

همانطور که می‌بینیم، کمترین score عدد 65 و بیشترین آن 90 می‌باشد.

سینتکس تابع پنجره‌ای در اینجا ساده‌تر می‌باشد و با استفاده از کلیدواژه OVER() اتفاق می‌افتد.

از آنجا که داخل OVER() چیزی نوشته نشده، پنجره تمام سطرها در نظر گرفته می‌شود.

1-2. استفاده از Partition

با استفاده از PARTITION BY در OVER()، می‌توانیم سطرها را قسمت‌بندی کرده و در هر کدام aggregate را اجرا کنیم.

در کوئری زیر بیشینه score و میانگین آن را در میان همه سطرهایی که dep_name یکسان دارند به دست می‌آوریم.

```
SELECT *, MAX(score) OVER(PARTITION BY dep_name) AS dep_maximum_score,
          ROUND(AVG(score) OVER(PARTITION BY dep_name), 2) AS dep_average_score
FROM student_score;
```

```
ex7=# SELECT *, MAX(score) OVER(PARTITION BY dep_name) AS dep_maximum_score,
ex7-#         ROUND(AVG(score) OVER(PARTITION BY dep_name), 2) AS dep_average_score
ex7-# FROM student_score;
 student_id | student_name |       dep_name       | score | dep_maximum_score | dep_average_score
------------+--------------+----------------------+-------+-------------------+-------------------
          2 | Yusuf        | Biochemistry         |    70 |                85 |             78.33
          5 | Muritadoh    | Biochemistry         |    85 |                85 |             78.33
          9 | Nurain       | Biochemistry         |    80 |                85 |             78.33
         12 | Tolu         | Computer Science     |    67 |                90 |             79.25
          8 | Joel         | Computer Science     |    90 |                90 |             79.25
          4 | Gbadebo      | Computer Science     |    80 |                90 |             79.25
         11 | Ibrahim      | Computer Science     |    80 |                90 |             79.25
         10 | Mustapha     | Industrial Chemistry |    78 |                78 |             78.00
          3 | Habeebah     | Microbiology         |    80 |                80 |             73.67
          1 | Tomiwa       | Microbiology         |    65 |                80 |             73.67
          7 | Taiwo        | Microbiology         |    76 |                80 |             73.67
(11 rows)
```

از آنجا که 4 دپارتمان داریم، خروجی به 4 قسمت تقسیم شده و در هر کدام بیشینه و میانگین score حساب شده است. همچنین در اینجا با استفاده از تابع ROUND، تعداد رقم اعشار میانگین را به 2 تا محدود کرده ایم.

1-3. تابع ROW_NUMBER

با استفاده از این تابع، می‌توانیم به ردیف‌های پنجره عدد اساین کنیم.

در کوئری زیر کل ردیف‌ها برای پنجره در نظر گرفته شده، بر حسب student name سورت شده و سپس ستون name_serial_number را اضافه می‌کنیم که به ترتیب از 1 به ردیف‌ها عدد نسبت می‌دهد.

```
SELECT *, ROW_NUMBER() OVER(ORDER BY student_name) AS name_serial_number
FROM student_score;
```

```
ex7=# SELECT *, ROW_NUMBER() OVER(ORDER BY student_name) AS name_serial_number
ex7-# FROM student_score;
 student_id | student_name |       dep_name       | score | name_serial_number
------------+--------------+----------------------+-------+--------------------
          4 | Gbadebo      | Computer Science     |    80 |                  1
          3 | Habeebah     | Microbiology         |    80 |                  2
         11 | Ibrahim      | Computer Science     |    80 |                  3
          8 | Joel         | Computer Science     |    90 |                  4
          5 | Muritadoh    | Biochemistry         |    85 |                  5
         10 | Mustapha     | Industrial Chemistry |    78 |                  6
          9 | Nurain       | Biochemistry         |    80 |                  7
          7 | Taiwo        | Microbiology         |    76 |                  8
         12 | Tolu         | Computer Science     |    67 |                  9
          1 | Tomiwa       | Microbiology         |    65 |                 10
          2 | Yusuf        | Biochemistry         |    70 |                 11
(11 rows)
```

همانطور که می‌بینیم، به ترتیب نام، اعداد سطرها از 1 تا 11 داده شده است.

4-1. تابع RANK

با استفاده از این تابع، می‌توانیم رکوردهای پنجره را رتبه‌بندی کنیم.

در کوئری زیر، پنجره‌ها بر حسب dep_name تقسیم شده و در هر کدام، بر حسب score سورت می‌کند و با همان ترتیب آنها را رتبه‌بندی می‌کند.

```sql
SELECT *, RANK() OVER(PARTITION BY dep_name ORDER BY score DESC)
FROM student_score;
```

```
ex7=# SELECT *, RANK() OVER(PARTITION BY dep_name ORDER BY score DESC)
ex7-# FROM student_score;
 student_id | student_name |       dep_name       | score | rank
------------+--------------+----------------------+-------+------
          5 | Muritadoh    | Biochemistry         |    85 |    1
          9 | Nurain       | Biochemistry         |    80 |    2
          2 | Yusuf        | Biochemistry         |    70 |    3
          8 | Joel         | Computer Science     |    90 |    1
         11 | Ibrahim      | Computer Science     |    80 |    2
          4 | Gbadebo      | Computer Science     |    80 |    2
         12 | Tolu         | Computer Science     |    67 |    4
         10 | Mustapha     | Industrial Chemistry |    78 |    1
          3 | Habeebah     | Microbiology         |    80 |    1
          7 | Taiwo        | Microbiology         |    76 |    2
          1 | Tomiwa       | Microbiology         |    65 |    3
(11 rows)
```

همانطور که می‌بینیم، در هر پنجره (که dep_name یکسانی دارند) ستون rank بر حسب score رتبه می‌دهد. در پنجره Computer Science می‌بینیم که دو دانشجو score یکسانی دارند و به این خاطر، رتبه آنها نیز یکسان در نظر گرفته شده است و رتبه نفر بعدی، فاصله و gap عددی دارد.

5-1. تابع DENSE_RANK

این تابع مشابه تابع RANK است با این فرق که در صورت برابر بودن رتبه دو رکورد در پنجره، رکورد بعدی عددی پیوسته می‌گیرد و فاصله عددی نداریم.

کوئری زیر همان کوئری بخش قبل است و فقط به جای RANK از DENSE_RANK استفاده شده است:

```
SELECT *, DENSE_RANK() OVER(PARTITION BY dep_name ORDER BY score DESC)
FROM student_score;
```

```
ex7=# SELECT *, DENSE_RANK() OVER(PARTITION BY dep_name ORDER BY score DESC)
ex7-# FROM student_score;
 student_id | student_name |       dep_name       | score | dense_rank
------------+--------------+----------------------+-------+------------
          5 | Muritadoh    | Biochemistry         |    85 |          1
          9 | Nurain       | Biochemistry         |    80 |          2
          2 | Yusuf        | Biochemistry         |    70 |          3
          8 | Joel         | Computer Science     |    90 |          1
         11 | Ibrahim      | Computer Science     |    80 |          2
          4 | Gbadebo      | Computer Science     |    80 |          2
         12 | Tolu         | Computer Science     |    67 |          3
         10 | Mustapha     | Industrial Chemistry |    78 |          1
          3 | Habeebah     | Microbiology         |    80 |          1
          7 | Taiwo        | Microbiology         |    76 |          2
          1 | Tomiwa       | Microbiology         |    65 |          3
(11 rows)
```

همانطور که می‌بینیم، در پنجره Computer Science رتبه‌ها به ترتیب اند و بعد از دو تا رتبه 2، رتبه 3 آمده است. در بخش قبل که از RANK استفاده شد، بعد از دو تا رتبه 2، رتبه 4 آمده بود.

6-1. تابع LAG

با استفاده از این تابع، می‌توانیم مقداری از رکورد قبلی در لیست را در رکورد کنونی داشته باشیم.

در کوئری زیر، پنجره‌ها با dep_name جدا شده و بر حسب score مرتب می‌شوند. حال با گرفتن LAG روی score، مقدار score ردیف قبل خود را در هر پنجره می‌گیریم.

```
SELECT *, LAG(score) OVER(PARTITION BY dep_name ORDER BY score)
FROM student_score;
```

```
ex7=# SELECT *, LAG(score) OVER(PARTITION BY dep_name ORDER BY score)
ex7-# FROM student_score;
 student_id | student_name |       dep_name       | score | lag
------------+--------------+----------------------+-------+-----
          2 | Yusuf        | Biochemistry         |    70 |
          9 | Nurain       | Biochemistry         |    80 |  70
          5 | Muritadoh    | Biochemistry         |    85 |  80
         12 | Tolu         | Computer Science     |    67 |
         11 | Ibrahim      | Computer Science     |    80 |  67
          4 | Gbadebo      | Computer Science     |    80 |  80
          8 | Joel         | Computer Science     |    90 |  80
         10 | Mustapha     | Industrial Chemistry |    78 |
          1 | Tomiwa       | Microbiology         |    65 |
          7 | Taiwo        | Microbiology         |    76 |  65
          3 | Habeebah     | Microbiology         |    80 |  76
(11 rows)
```

همانطور که می‌بینیم، در هر پنجره، اولین ردیف مقداری در lag ندارد. این به این خاطر است که رکوردی قبل از آن وجود ندارد. برای بقیه رکوردها، مقدار lag همان مقدار score در رکورد قبلی می‌باشد.

7-1. استفاده از Frame

با استفاده از frame-ها، می‌توانیم به ازای هر سطر، ناحیه‌ای اطراف آن را برای محاسبه aggregate انتخاب کنیم. این کار با توابع RANK کار نمی‌کند.

برای تعیین یک frame، از کلیدواژه ROWS استفاده می‌کنیم. سپس بازه قبل و بعد از سطر کنونی برای frame را با کلیدواژه‌های N PRECEDING و N FOLLOWING که N یک عدد یا UNBOUNDED است تعیین می‌کنیم. همچنین CURRENT ROW به سطر کنونی اشاره می‌کند.

مثلا `ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING`، به ازای هر سطر، frame-ای که شامل خود سطر، سطر قبل و سطر بعد آن می‌شود را در نظر می‌گیرد.

در کوئری زیر، پنجره تمام سطرها را شامل می‌شود و بر حسب student_id مرتب شده اند. حال به ازای هر سطر، یک frame برای بازهٔ تمام سطرهای قبل تا خود سطر کنونی را در نظر می‌گیریم و جمع score در آنها را حساب می‌کنیم.

با این کار به جمع تجمعی score می‌رسیم.

```sql
SELECT *, SUM(score) OVER(ORDER BY student_id ROWS BETWEEN UNBOUNDED PRECEDING AND
                                                    CURRENT ROW) AS cummulative_sum
FROM student_score;
```

```
ex7=# SELECT *, SUM(score) OVER(ORDER BY student_id ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cummulative_sum
ex7-# FROM student_score;
 student_id | student_name |       dep_name       | score | cummulative_sum
------------+--------------+----------------------+-------+-----------------
          1 | Tomiwa       | Microbiology         |    65 |              65
          2 | Yusuf        | Biochemistry         |    70 |             135
          3 | Habeebah     | Microbiology         |    80 |             215
          4 | Gbadebo      | Computer Science     |    80 |             295
          5 | Muritadoh    | Biochemistry         |    85 |             380
          7 | Taiwo        | Microbiology         |    76 |             456
          8 | Joel         | Computer Science     |    90 |             546
          9 | Nurain       | Biochemistry         |    80 |             626
         10 | Mustapha     | Industrial Chemistry |    78 |             704
         11 | Ibrahim      | Computer Science     |    80 |             784
         12 | Tolu         | Computer Science     |    67 |             851
(11 rows)
```

همانطور که می‌بینیم، هر سطر از cummulative_sum، برابر جمع score خود سطر و همه score-های قبل از خودش است.

پاسخ 2. تریگرها

2-0. جدول users

ابتدا یک جدول نمونه در دیتابیس ex7 می‌سازیم:

```
CREATE TABLE users (
    fullname VARCHAR(120),
    email VARCHAR(120),
    username VARCHAR(30),
    password VARCHAR(60)
);
```

در کل trigger-ها به جدول‌ها وصل می‌شوند و در صورت رخداد INSERT، UPDATE یا DELETE بر روی یک جدول، کد trigger قبل یا بعد از انجام رخداد اجرا می‌شود.

با استفاده از CREATE TRIGGER name یک trigger جدید تعریف می‌کنیم و سپس مشخص می‌کنیم که BEFORE یا AFTER سه عمل مطرح شده روی هر سطر، باید اجرا بشود.

2-1. تریگر password_hasher

بر روی جدول ساخته شده، یک trigger می‌سازیم که قبل از insert شدن یک سطر به جدول، فیلد password آن را تغییر داده تا hash بشود.

```
CREATE TRIGGER password_hasher
BEFORE INSERT ON users
FOR EACH ROW
SET NEW.password = MD5(NEW.password);
```

کلیدواژه NEW وقتی از BEFORE INSERT/UPDATE استفاده می‌کنیم وجود دارد و به سطر جدید اشاره می‌کند. به طور مشابه کلیدواژه OLD وقتی از AFTER DELETE/UPDATE استفاده می‌کنیم وجود دارد و به سطر مد نظر قبل از تغییر یا حذف شدن اشاره می‌کند.

سینتکس داده شده در لینک سینتکس MySQL بوده و بر روی PostgreSQL کار نمی‌کند.

سینتکس ساخت trigger معادل بالا به شکل زیر است:

```
CREATE OR REPLACE FUNCTION password_md5()
RETURNS TRIGGER AS $$
BEGIN
  NEW.password = MD5(NEW.password);
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER password_hasher
BEFORE INSERT ON users
FOR EACH ROW
EXECUTE FUNCTION password_md5();
```

```
ex7=# CREATE OR REPLACE FUNCTION password_md5()
ex7-# RETURNS TRIGGER AS $$
ex7$# BEGIN
ex7$#   NEW.password = MD5(NEW.password);
ex7$#   RETURN NEW;
ex7$# END;
ex7$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
ex7=# CREATE TRIGGER password_hasher
ex7-# BEFORE INSERT ON users
ex7-# FOR EACH ROW
ex7-# EXECUTE FUNCTION password_md5();
CREATE TRIGGER
```

همانطور که می‌بینیم، در PostgreSQL باید ابتدا یک تابع تعریف کرد و کد مد نظر trigger را داخل آن بگذاریم. سپس در trigger با استفاده از EXECUTE FUNCTION آن را صدا می‌کنیم.

جهت آزمایش trigger ساخته شده، یک سطر به users اضافه می‌کنیم:

```
INSERT INTO users
VALUES ('idris babu', 'zubs@test.com', 'zubby1', 'password');
```

```
ex7=# INSERT INTO users
ex7-# VALUES ('idris babu', 'zubs@test.com', 'zubby1', 'password');
INSERT 0 1
ex7=# SELECT * FROM users;
  fullname  |     email     | username |             password
------------+---------------+----------+----------------------------------
 idris babu | zubs@test.com | zubby1   | 5f4dcc3b5aa765d61d8327deb882cf99
(1 row)
```

همانطور که می‌بینیم، پس از insert کردن، مقدار فیلد password سطر در جدول به هش MD5 ورودی تغییر یافته است.

2-2. حذف Trigger

جهت حذف یک trigger، از دستور زیر استفاده می‌کنیم:

```
DROP TRIGGER password_hasher ON users;
```

حال برای تست حذف شدن، همان سطر قبلی را دوباره اضافه می‌کنیم:

```
ex7=# DROP TRIGGER password_hasher ON users;
DROP TRIGGER
ex7=# INSERT INTO users
ex7-# VALUES ('idris babu', 'zubs@test.com', 'zubby1', 'password');
INSERT 0 1
ex7=# SELECT * FROM users;
  fullname  |     email     | username |             password
------------+---------------+----------+----------------------------------
 idris babu | zubs@test.com | zubby1   | 5f4dcc3b5aa765d61d8327deb882cf99
 idris babu | zubs@test.com | zubby1   | password
(2 rows)
```

همانطور که می‌بینیم، از آنجا که تریگر password_hasher دیگر وجود ندارد، مقدار password سطر insert شده هش نشده است.