



به نام خدا  
دانشگاه تهران  
دانشکده مهندسی برق و کامپیوتر



## درس آزمایشگاه پایگاه داده پیش گزارش هفتم

نام و نام خانوادگی	پاشا براهیمی
شماره دانشجویی	۸۱۰۱۹۹۳۸۵
تاریخ ارسال گزارش	۱۴۰۲/۰۹/۲۹

## فهرست

پاسخ ۱. معرفی توابع پنجره‌ای.....	2
۱-۱. نمونه ساده استفاده از OVER().....	2
۲-۱. دستور Partition By.....	3
۳-۱. تابع ROW_NUMBER() و استفاده از Order By.....	4
۴-۱. تابع RANK().....	5
۵-۱. تابع DENSE_RANK().....	6
۶-۱. تابع LAG.....	7
۷-۱. Frame Clause.....	8
پاسخ ۲. معرفی تریگرها.....	9
۱-۲. ایجاد تریگر.....	9
۲-۲. حذف تریگر.....	11

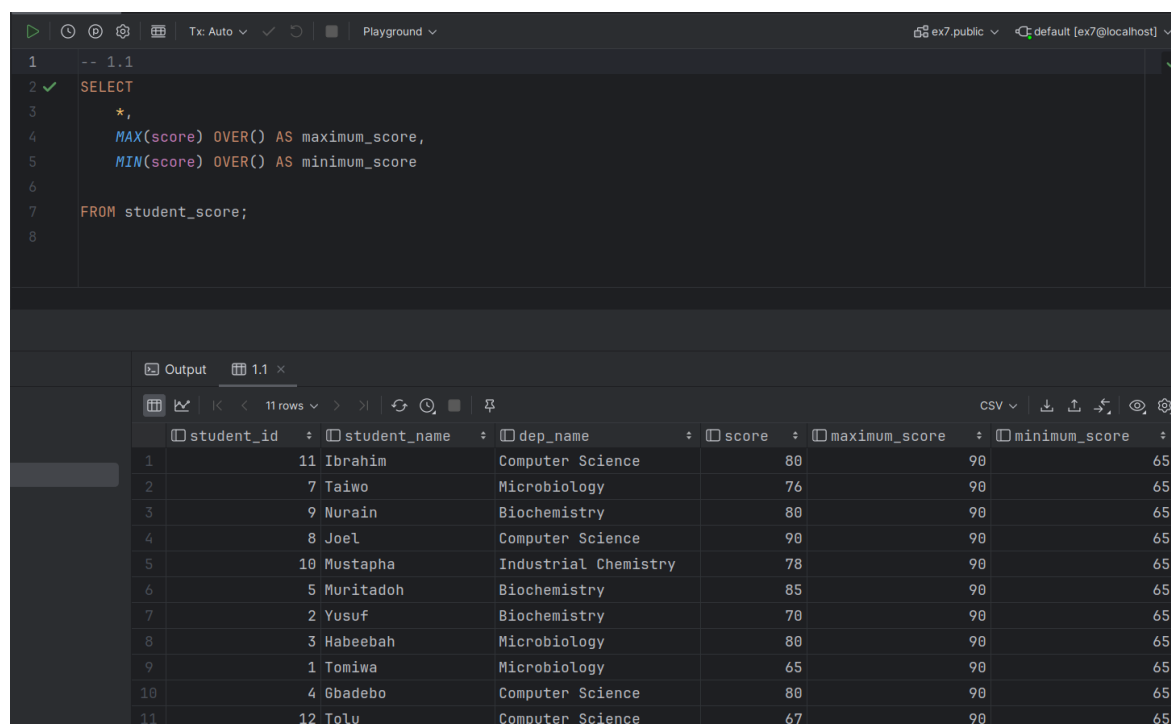
## پاسخ ۱. معرفی توابع پنجره‌ای

### ۱-۱. نمونه ساده استفاده از OVER()

در این مثال، می‌خواهیم در کنار تمام سطرها، بالاترین و پایین‌ترین نمره را نیز با نام‌های maximum\_score و minimum\_score نمایش دهیم.

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

خروجی به صورت زیر است:



The screenshot shows a SQL playground interface. The query editor contains the following SQL code:

```
-- 1.1
SELECT
*,
MAX(score) OVER() AS maximum_score,
MIN(score) OVER() AS minimum_score
FROM student_score;
```

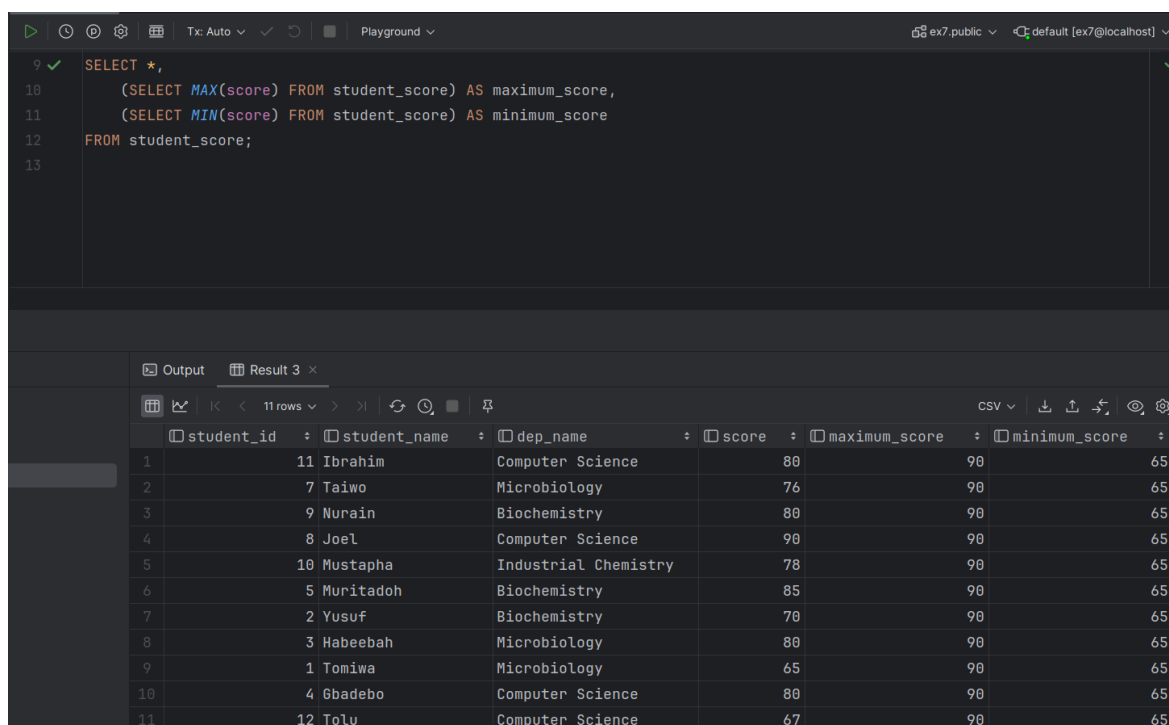
The output window displays a table with 11 rows. The columns are student\_id, student\_name, dep\_name, score, maximum\_score, and minimum\_score. The maximum\_score is consistently 90 and the minimum\_score is consistently 65 for all rows.

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

همانطور که مشاهده می‌شود، در هر سطر دو ستون maximum\_score با مقدار 90 و minimum\_score با مقدار 65 اضافه شده است. این دستور معادل با دستور زیر (بدون استفاده از توابع پنجره‌ای) است:

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

خروجی این دستور نیز مشابه با دستور قبلی و به صورت زیر است:



The screenshot shows a SQL playground interface. At the top, there's a toolbar with icons for running, saving, and other functions. Below the toolbar, a SQL query is entered in a text area. The query is: `SELECT *, (SELECT MAX(score) FROM student_score) AS maximum_score, (SELECT MIN(score) FROM student_score) AS minimum_score FROM student_score;`. Below the query, the results are displayed in a table. The table has 7 columns: `student_id`, `student_name`, `dep_name`, `score`, `maximum_score`, and `minimum_score`. There are 11 rows of data. The interface also shows a 'Result 3' tab and a 'csv' download button.

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

## ۲-۱. دستور Partition By

در دستور زیر، به ازای هر دپارتمان، ماکسیمم و میانگین score را بدست آورده و در دو ستون جدید در هر سطر نمایش می دهیم. همانطور که در نتیجه می شود دید، این مقادیر در یک دپارتمان ثابت اند.

```
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;
```

نتیجه به صورت زیر است:

14 -- 1.2 Partition By

15 ✓ SELECT

16 \*

17 MAX(score)OVER(PARTITION BY dep\_name) AS dep\_maximum\_score,

18 ROUND(AVG(score)OVER(PARTITION BY dep\_name), 2) AS dep\_average\_score

19 FROM student\_score;

20

Output 1.2 Partition By

11 rows

	student_id	student_name	dep_name	score	dep_maximum_score	dep_average_score
1	2	Yusuf	Biochemistry	70	85	78.33
2	5	Muritadoh	Biochemistry	85	85	78.33
3	9	Nurain	Biochemistry	80	85	78.33
4	12	Tolu	Computer Science	67	90	79.25
5	8	Joel	Computer Science	90	90	79.25
6	4	Gbadebo	Computer Science	80	90	79.25
7	11	Ibrahim	Computer Science	80	90	79.25
8	10	Mustapha	Industrial Chemistry	78	78	78
9	3	Habeebah	Microbiology	80	80	73.67
10	1	Tomiwa	Microbiology	65	80	73.67
11	7	Taiwo	Microbiology	76	80	73.67

### ۳-۱. تابع ROW\_NUMBER() و استفاده از Order By

در دستور زیر، ابتدا بر حسب نام دانشجویها، جدول را مرتب کرده و سپس به هر یک از سطرها یک شماره یکتا اختصاص داده می‌شود که شماره آن سطر را مشخص می‌کند. برای مثال سطر اول شماره 1 و سطر دوم شماره 2 خواهد گرفت.

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

خروجی به صورت زیر است:

21 -- 1.3 Row Number  
 22 ✓ SELECT  
 23 \*,  
 24 ROW\_NUMBER() OVER(ORDER BY student\_name) AS name\_serial\_number  
 25 FROM student\_score;  
 26

Output 1.3 Row Number

	student_id	student_name	dep_name	score	name_serial_number
1	4	Gbadebo	Computer Science	80	1
2	3	Habeebah	Microbiology	80	2
3	11	Ibrahim	Computer Science	80	3
4	8	Joel	Computer Science	90	4
5	5	Muritadoh	Biochemistry	85	5
6	10	Mustapha	Industrial Chemistry	78	6
7	9	Nurain	Biochemistry	80	7
8	7	Taiwo	Microbiology	76	8
9	12	Tolu	Computer Science	67	9
10	1	Tomiwa	Microbiology	65	10
11	2	Yusuf	Biochemistry	78	11

همانطور که دیده می‌شود، شماره هر سطر در ستون name\_serial\_number داده شده است.

#### ۴-۱. تابع RANK()

در کوئری زیر، برای هر دپارتمان، ابتدا بر اساس score و به صورت نزولی مرتب کرده و بر اساس همان score، یک rank به هر سطر اختصاص می‌دهد. بدیهی‌ست که دو سطر که دپارتمان و score مشابه دارند، rank یکسان هم خواهد داشت. همچنین لازم به ذکر است که این تابع، زمانی که rank یکسان برای دو سطر در نظر بگیرد، یک gap پس از آن قرار می‌دهد. برای مثال، اگر در یک دپارتمان، سه نفر score برابر با 80 داشته باشند، هر سه rank یکسان (برای مثال در اینجا برابر با 2) دارند و rank بعدی به جای 3 برابر با 5 خواهد بود.

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

خروجی به صورت زیر است:

27 -- 1.4 Rank  
 28 ✓ SELECT  
 29 \*,  
 30 RANK()OVER(PARTITION BY dep\_name ORDER BY score DESC)  
 31 FROM student\_score;  
 32

Output 1.4 Rank ×

11 rows

	student_id	student_name	dep_name	score	rank
1	5	Muritadoh	Biochemistry	85	1
2	9	Nurain	Biochemistry	80	2
3	2	Yusuf	Biochemistry	70	3
4	8	Joel	Computer Science	90	1
5	11	Ibrahim	Computer Science	80	2
6	4	Gbadebo	Computer Science	80	2
7	12	Tolu	Computer Science	67	4
8	10	Mustapha	Industrial Chemistry	78	1
9	3	Habeebah	Microbiology	80	1
10	7	Taiwo	Microbiology	76	2
11	1	Tomiwa	Microbiology	65	3

همانطور که دیده می‌شود، به ازای هر دپارتمان، دانشجویان بر حسب نمره‌شان یک rank گرفته‌اند و دانشجویان با score برابر، rank یکسان هم دارند.

## ۱-۵. تابع DENSE\_RANK()

این تابع همانند تابع RANK است با این تفاوت که در صورت برابری rank چند سطر، gap بعد از آن را نخواهیم داشت و rank بعدی دقیقاً یک واحد بیشتر از rank قبلی خواهد بود.

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

خروجی به صورت زیر است:

33 -- 1.5 Dense Rank  
 34 ✓ SELECT  
 35 \*,  
 36 DENSE\_RANK()OVER(PARTITION BY dep\_name ORDER BY score DESC)  
 37 FROM student\_score;  
 38

Output 1.5 Dense Rank

11 rows

	student_id	student_name	dep_name	score	dense_rank
1	5	Muritadoh	Biochemistry	85	1
2	9	Nurain	Biochemistry	80	2
3	2	Yusuf	Biochemistry	70	3
4	8	Joel	Computer Science	90	1
5	11	Ibrahim	Computer Science	80	2
6	4	Gbadebo	Computer Science	80	2
7	12	Tolu	Computer Science	67	3
8	10	Mustapha	Industrial Chemistry	78	1
9	3	Habeebah	Microbiology	80	1
10	7	Taiwo	Microbiology	76	2
11	1	Tomiwa	Microbiology	65	3

## ۶-۱. تابع LAG

در این تابع، می‌خواهیم مقداری از رکوردهای قبلی را در رکورد فعلی نمایش دهیم. این تابع به صورت دیفالت مقدار سطر قبلی را بازمی‌گرداند. برای مثال در تابع زیر، در کنار score هر دانشجو، score سطر قبلی را نیز قرار می‌دهیم. با توجه به اینکه از Partition By استفاده کردیم، این مورد در خصوص هر دپارتمان به صورت جداگانه عمل می‌کند.

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

نتیجه به صورت زیر است:



39 -- 1.6 Lag  
 40 ✓ SELECT  
 41 \*,  
 42 LAG(score) OVER(PARTITION BY dep\_name ORDER BY score)  
 43 FROM student\_score;  
 44

Output 1.6 Lag ×

11 rows

	student_id	student_name	dep_name	score	lag
1	2	Yusuf	Biochemistry	70	<null>
2	9	Nurain	Biochemistry	80	70
3	5	Muritadoh	Biochemistry	85	80
4	12	Tolu	Computer Science	67	<null>
5	11	Ibrahim	Computer Science	80	67
6	4	Gbadebo	Computer Science	80	80
7	8	Joel	Computer Science	90	80
8	10	Mustapha	Industrial Chemistry	78	<null>
9	1	Tomwa	Microbiology	65	<null>
10	7	Taiwo	Microbiology	76	65
11	3	Habeebah	Microbiology	80	76

همانطور که دیده می‌شود، اولین سطر مربوط به هر دپارتمان (چون سطر قبلی ندارد)، مقدار lag برابر با null است.

## ۷-۱. Frame Clause

در این تابع، می‌توانیم داده‌هایی از سطرهای قبل یا بعد از سطر فعلی را بدست آوریم. برای مثال در این کوئری می‌خواهیم به ازای هر دپارتمان، cumulative sum را در نمرات بدست آوریم. در واقع یک ستون در هر سطر داشته باشیم که مجموع نمره تمام سطرهای قبلی را به همراه سطر فعلی داشته باشد. در این بخش UNBOUND PRECEDING به معنای سطرهای قبلی و CURRENT ROW به معنای سطر فعلی است.

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

نتیجه به صورت زیر است:

45 -- 1.7 Frame Clause  
 46 ✓ SELECT  
 47 \*,  
 48 SUM(score)OVER(ORDER BY student\_id ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cummulative\_sum  
 49 FROM student\_score  
 50

Output 1.7 Frame Clause x

11 rows

	student_id	student_name	dep_name	score	cummulative_sum
1	1	Tomiwa	Microbiology	65	65
2	2	Yusuf	Biochemistry	70	135
3	3	Habeebah	Microbiology	80	215
4	4	Gbadebo	Computer Science	80	295
5	5	Muritadoh	Biochemistry	85	380
6	7	Taiwo	Microbiology	76	456
7	8	Joel	Computer Science	90	546
8	9	Nurain	Biochemistry	80	626
9	10	Mustapha	Industrial Chemistry	78	704
10	11	Ibrahim	Computer Science	80	784
11	12	Tolu	Computer Science	67	851

## پاسخ ۲. معرفی تریگرها

### ۲-۱. ایجاد تریگر

در این بخش چون سیتکس قرار داده شده در سایت برای MySQL است، آن را طوری تغییر دادم که در postgres اجرا شود. در این حالت، ابتدا یک تابع hash\_password داریم که در تریگر، به ازای هر یوزر اضافه شده، این تابع کال می‌شود. ابتدا جدول جدید را می‌سازیم:

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

سپس تریگر را به صورت ذکر شده ایجاد می‌کنیم:

```
CREATE OR REPLACE FUNCTION hash_password()
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 hash_password();

```

و در نهایت یوزر را اضافه می‌کنیم:

```

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

```

دستورهای اجرا شده به صورت کلی در تصویر زیر واضح است:

```

60 -- 2.1 Create Trigger
61 CREATE OR REPLACE FUNCTION hash_password()
62 RETURNS TRIGGER AS $$
63 BEGIN
64     NEW.password = MD5(NEW.password);
65     RETURN NEW;
66 END;
67 $$ LANGUAGE plpgsql;
68
69 CREATE TRIGGER password_hasher
70 BEFORE INSERT ON users
71 FOR EACH ROW
72 EXECUTE FUNCTION hash_password();
73
74 INSERT INTO
75     users
76 VALUES
77     (
78         'idris babu',
79         'zubs@test.com',
80         'zubby1',
81         'password'
82     );
83
84

```

پس از اضافه کردن یوزر، جدول را مشاهده کرده و فیلد password را بررسی می‌کنیم:

	fullname	email	username	password
1	idris babu	zubs@test.com	zubby1	5f4dcc3b5aa765d61d8327deb882cf99

می بینیم که به جای عبارت password، هش آن در جدول قرار گرفته است.

## ۲-۲. حذف تریگر

در این بخش، تریگر ایجاد شده در بخش قبل را حذف می کنیم:

```
DROP TRIGGER
  IF EXISTS password_hasher
  ON users;
```

و سپس یک یوزر جدید اضافه می کنیم:

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

خروجی به صورت زیر است:

```
84 -- 2.2 Drop Trigger
85 DROP TRIGGER
86     IF EXISTS password_hasher
87     ON users;
88
89 INSERT INTO
90     users
91     VALUES
92     (
93         'idris babu',
94         'zubs@test.com',
95         'zubby1',
96         'password'
97     );
98
```

Output 1.7 Frame Clause

```
ex7.public> DROP TRIGGER
          IF EXISTS password_hasher
          ON users
[2023-12-17 00:52:27] completed in 6 ms
ex7.public> INSERT INTO
          users
```

حال اگر جدول users را بررسی کنیم، می بینیم که مقدار password در جدول قرار گرفته و دیگر هش آن به جای خود پسورد در جدول نیست:

	fullname	email	username	password
1	idris babu	zubs@test.com	zubby1	5f4dcc3b5aa765d61d8327deb882cf99
2	idris babu	zubs@test.com	zubby1	password