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

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس آزمایشگاه پایگاه‌داده**  **پیش‌گزارش هفتم** | | |

**فهرست**

[**پاسخ 1**. **توابع پنجره‌ای** 2](#_Toc153669211)

[0-1. جدول student\_score 2](#_Toc153669212)

[1-1. محاسبه برای کل جدول 3](#_Toc153669213)

[2-1. استفاده از Partition 4](#_Toc153669214)

[3-1. تابع ROW\_NUMBER 4](#_Toc153669215)

[4-1. تابع RANK 5](#_Toc153669216)

[5-1. تابع DENSE\_RANK 6](#_Toc153669217)

[6-1. تابع LAG 6](#_Toc153669218)

[7-1. استفاده از Frame 7](#_Toc153669219)

[**پاسخ 2**. **تریگرها** 9](#_Toc153669220)

[0-2. جدول users 9](#_Toc153669221)

[1-2. تریگر password\_hasher 9](#_Toc153669222)

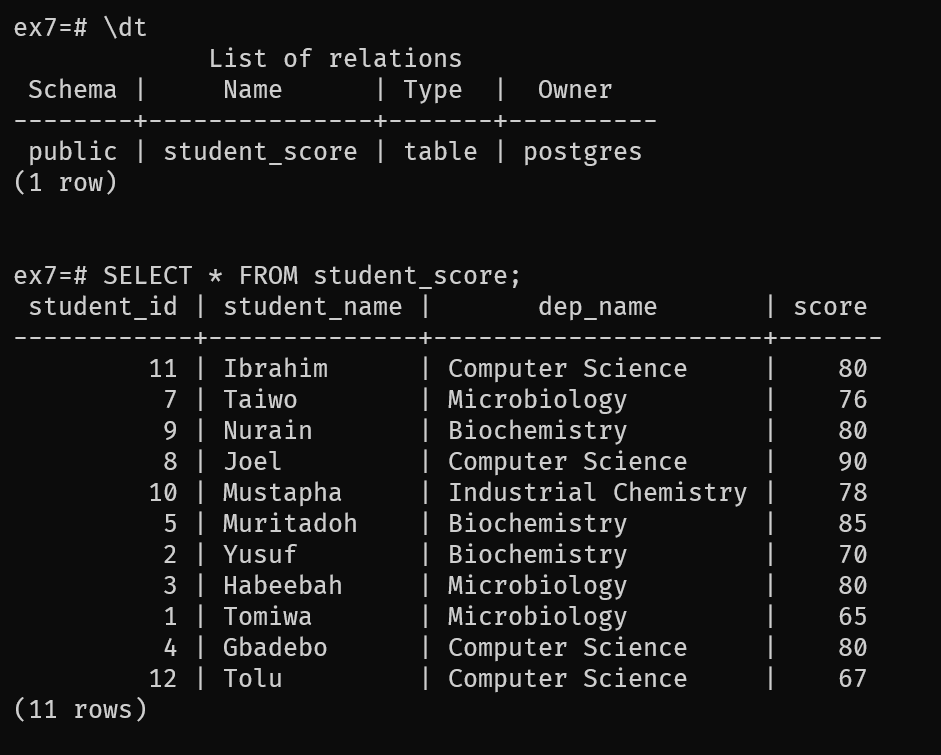
[2-2. حذف Trigger 12](#_Toc153669223)

# **پاسخ 1**. **توابع پنجره‌ای**

## 0-1. 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);

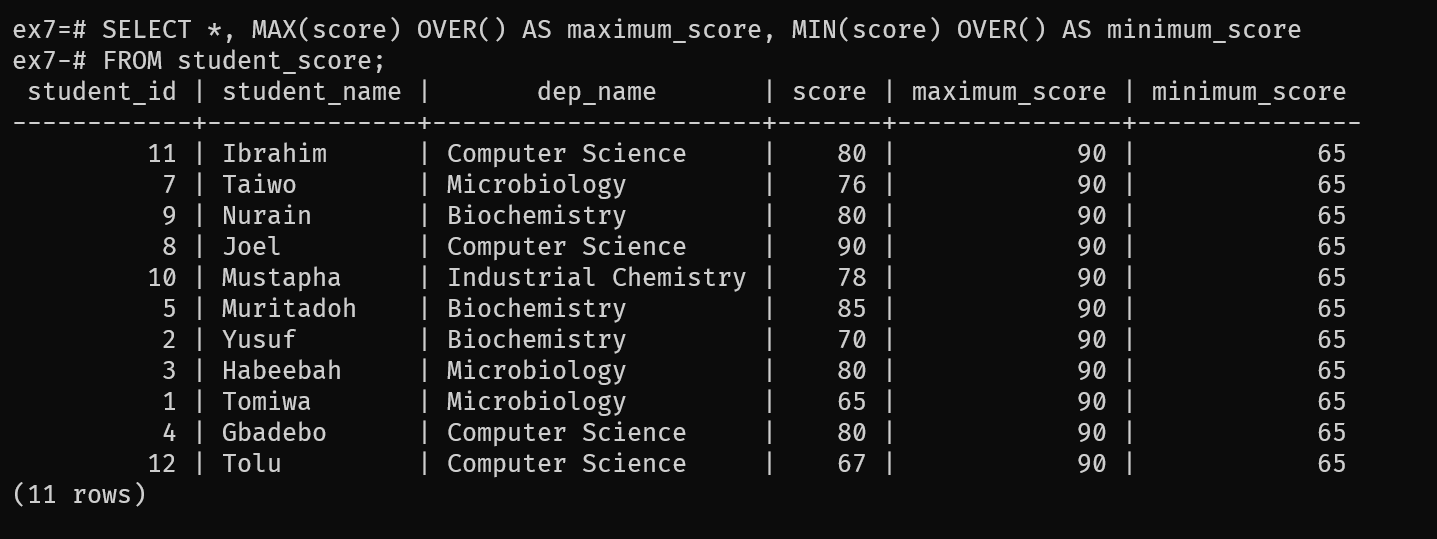


همانطور که می‌بینیم، داده‌ها به طور صحیح وارد جدول student\_score شده اند.

## 1-1. محاسبه برای کل جدول

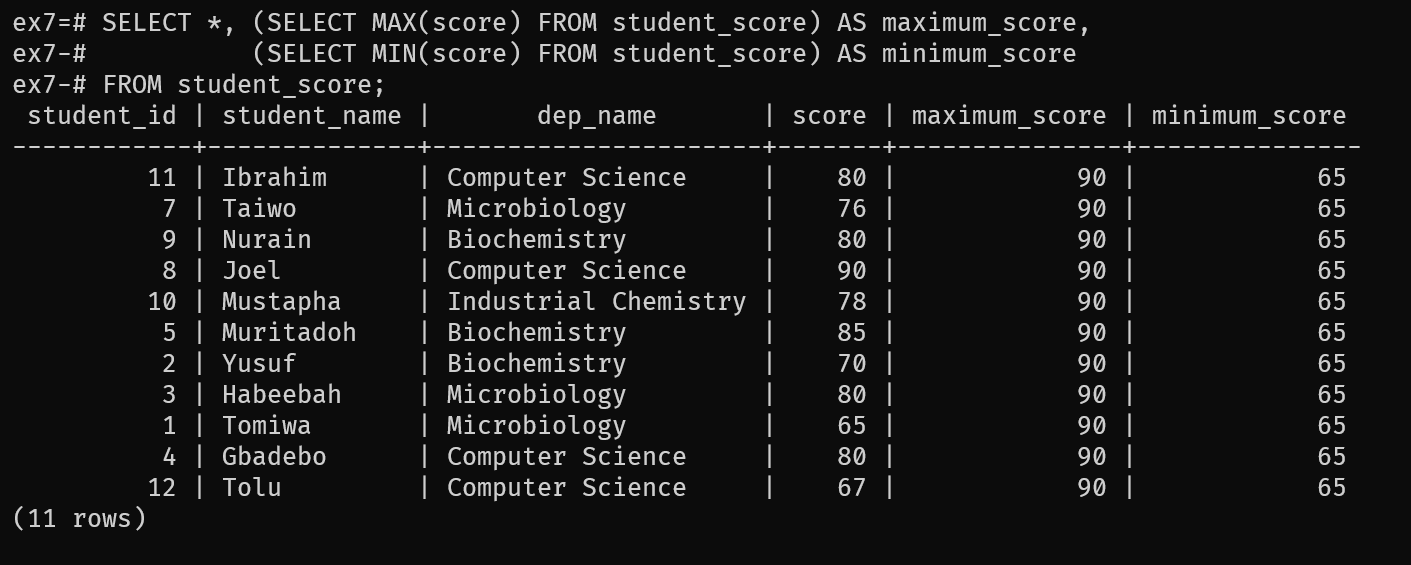
می‌خواهیم بیشترین و کمترین score در بین تمامی سطرهای جدول را حساب کنیم:

SELECT \*, MAX(score) OVER() AS maximum\_score, MIN(score) OVER() AS minimum\_score  
FROM student\_score;



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;



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.

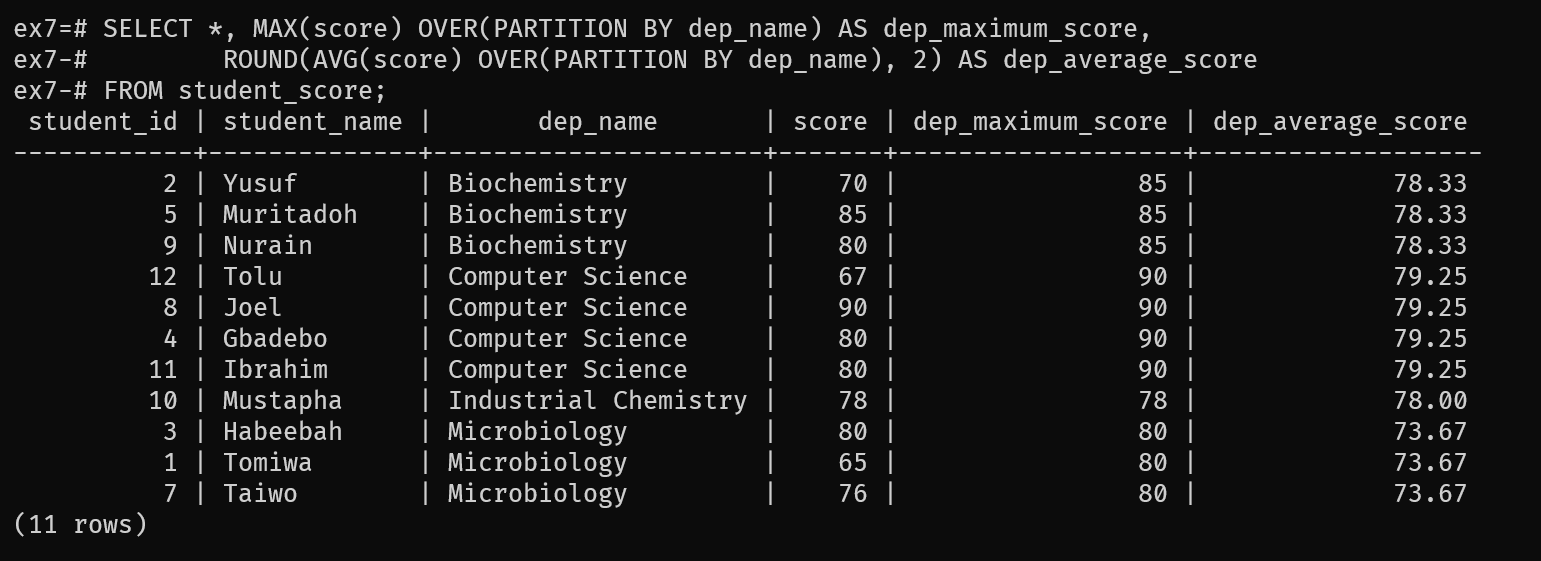
از آنجا که داخل OVER() چیزی نوشته نشده، پنجره تمام سطرها در نظر گرفته می‌شود.

## 2-1. 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;



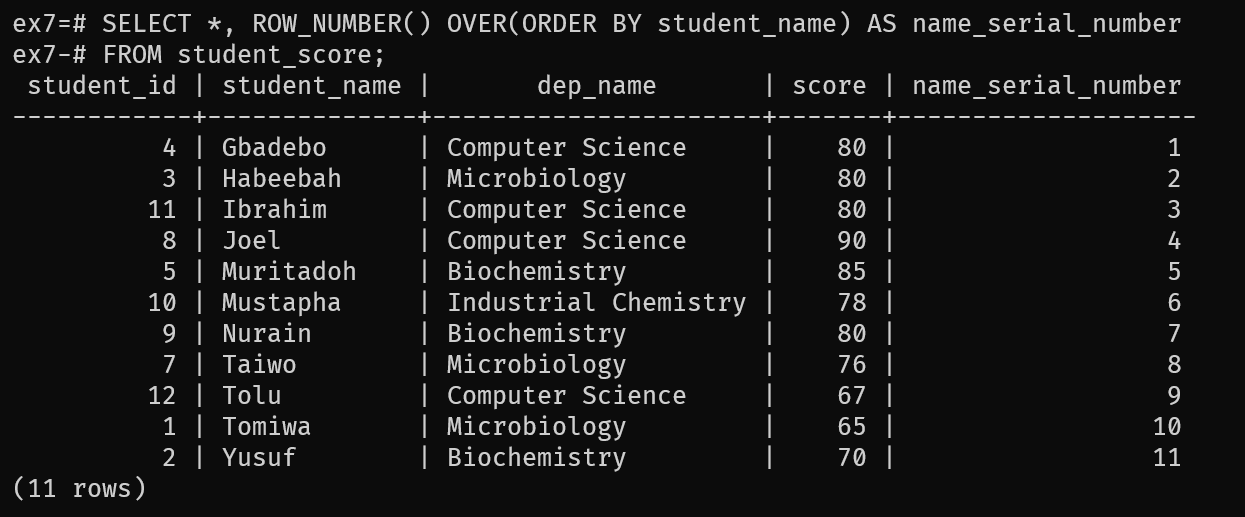
از آنجا که 4 دپارتمان داریم، خروجی به 4 قسمت تقسیم شده و در هر کدام بیشینه و میانگین score حساب شده است. همچنین در اینجا با استفاده از تابع ROUND، تعداد رقم اعشار میانگین را به 2 تا محدود کرده ایم.

## 3-1. 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;



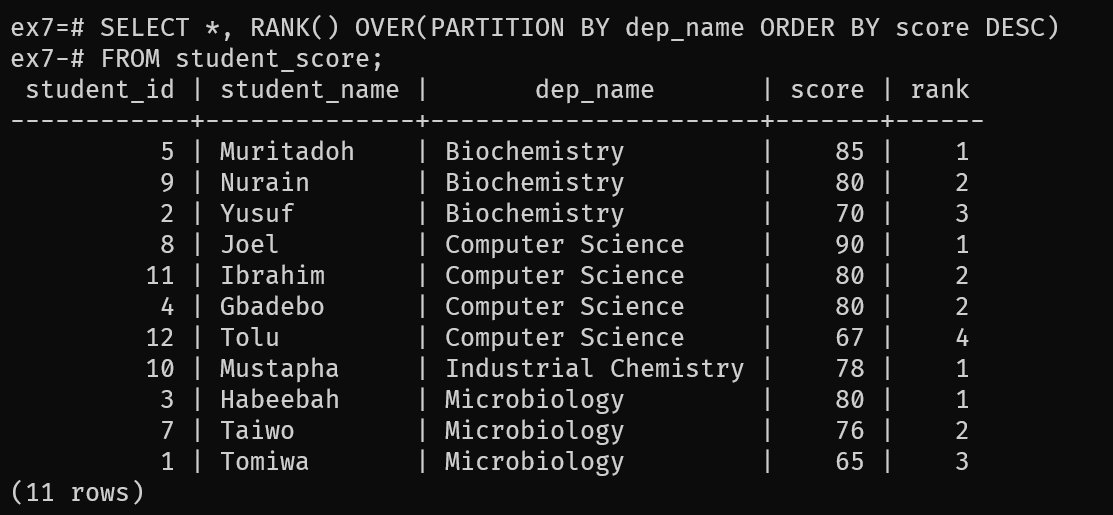
همانطور که می‌بینیم، به ترتیب نام، اعداد سطرها از 1 تا 11 داده شده است.

## 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.

SELECT \*, RANK() OVER(PARTITION BY dep\_name ORDER BY score DESC)  
FROM student\_score;



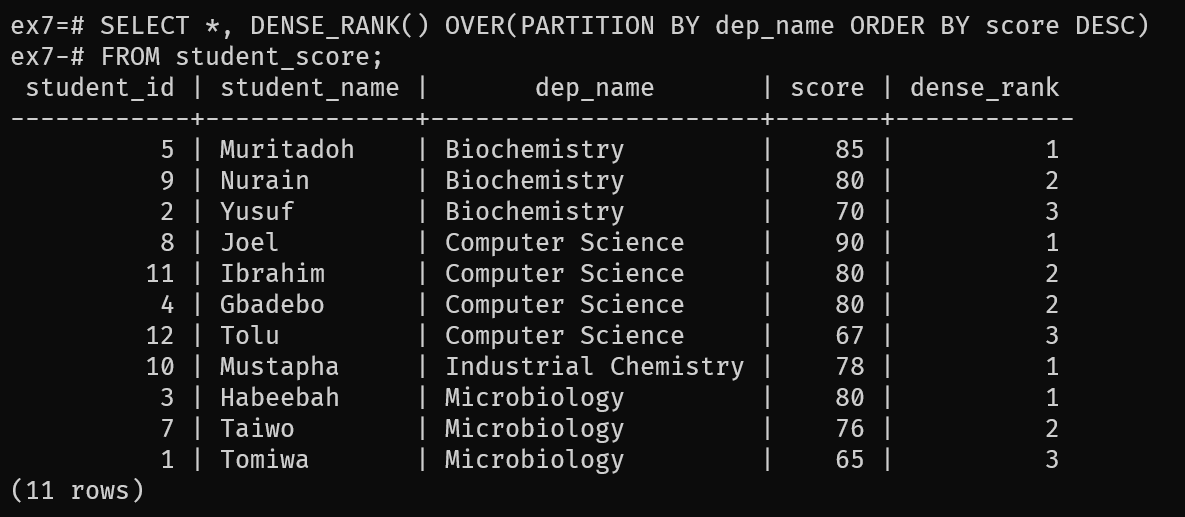
همانطور که می‌بینیم، در هر پنجره (که dep\_name یکسانی دارند) ستون rank بر حسب score رتبه می‌دهد. در پنجره Computer Science می‌بینیم که دو دانشجو score یکسانی دارند و به این خاطر، رتبه آنها نیز یکسان در نظر گرفته شده است و رتبه نفر بعدی، فاصله و gap عددی دارد.

## 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;



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;



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.

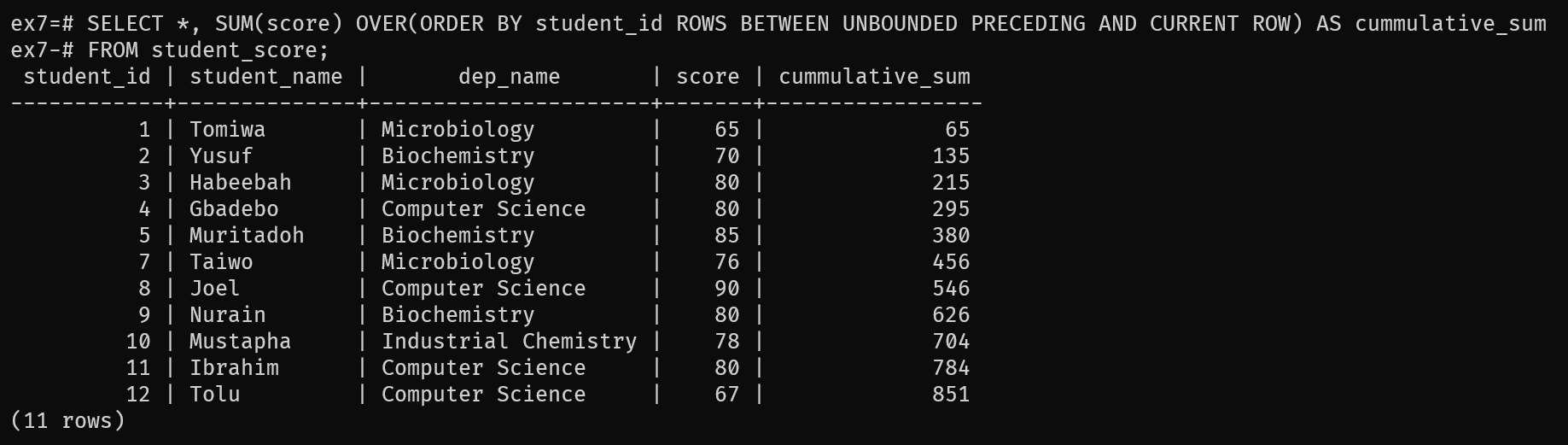
برای تعیین یک frame، از کلیدواژه ROWS استفاده می‌کنیم. سپس بازه قبل و بعد از سطر کنونی برای frame را با کلیدواژه‌های N PRECEDING و N FOLLOWING که N یک عدد یا UNBOUNDED است تعیین می‌کنیم. همچنین CURRENT ROW به سطر کنونی اشاره می‌کند.

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 بازۀ 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.

SELECT \*, SUM(score) OVER(ORDER BY student\_id ROWS BETWEEN UNBOUNDED PRECEDING AND  
 CURRENT ROW) AS cummulative\_sum  
FROM student\_score;



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

# **پاسخ 2**. **تریگرها**

## 0-2. 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.

## 1-2. 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.

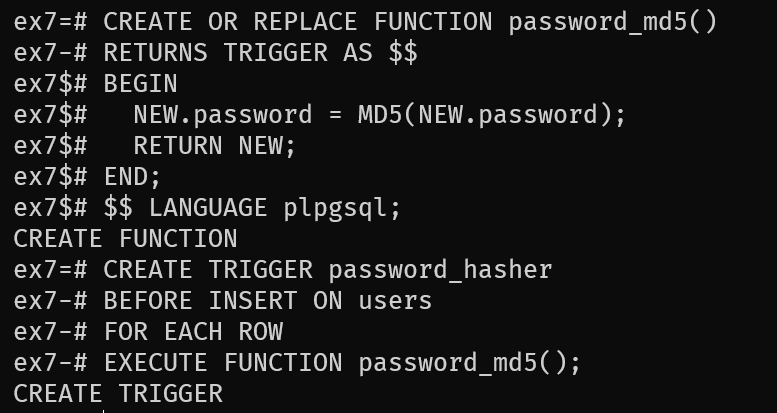
CREATE TRIGGER password\_hasher  
BEFORE INSERT ON users  
FOR EACH ROW  
SET NEW.password = MD5(NEW.password);

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:

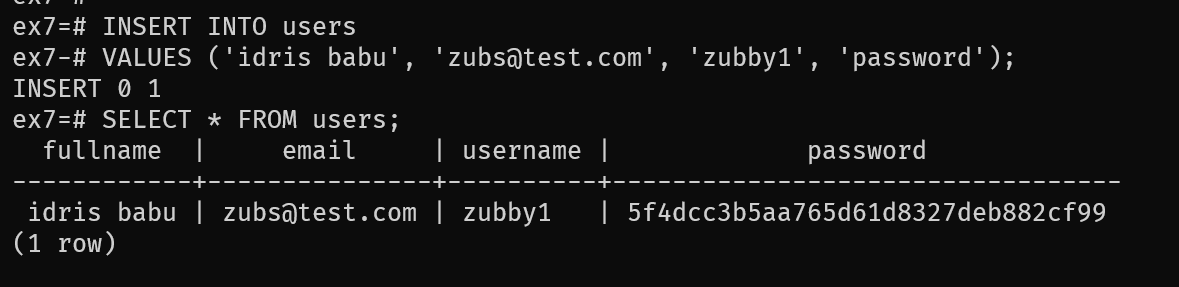
CREATE OR REPLACE FUNCTION password\_md5()  
RETURNS TRIGGER AS $$  
BEGIN  
 NEW.password = MD5(NEW.password);  
 RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER password\_hasher  
BEFORE INSERT ON users  
FOR EACH ROW  
EXECUTE FUNCTION password\_md5();



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');



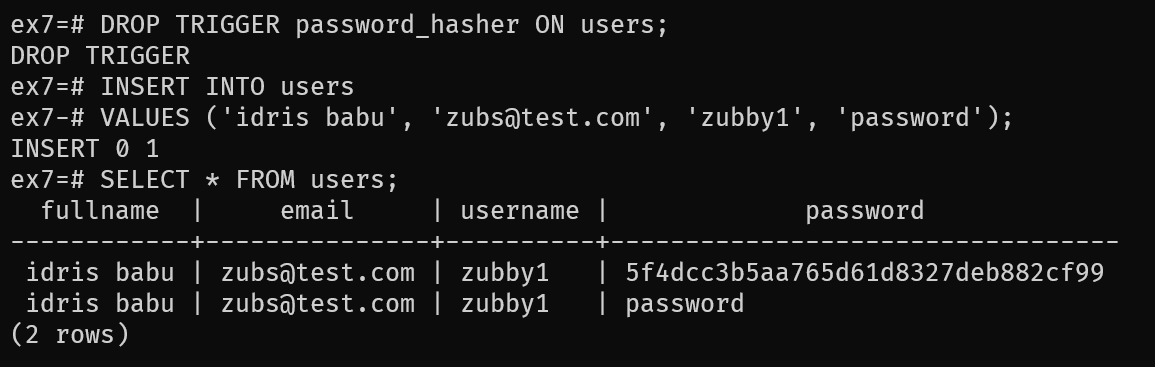
همانطور که می‌بینیم، پس از insert کردن، مقدار فیلد password سطر در جدول به هش MD5 ورودی تغییر یافته است.

## 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:



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