پایگاه داده‌ها

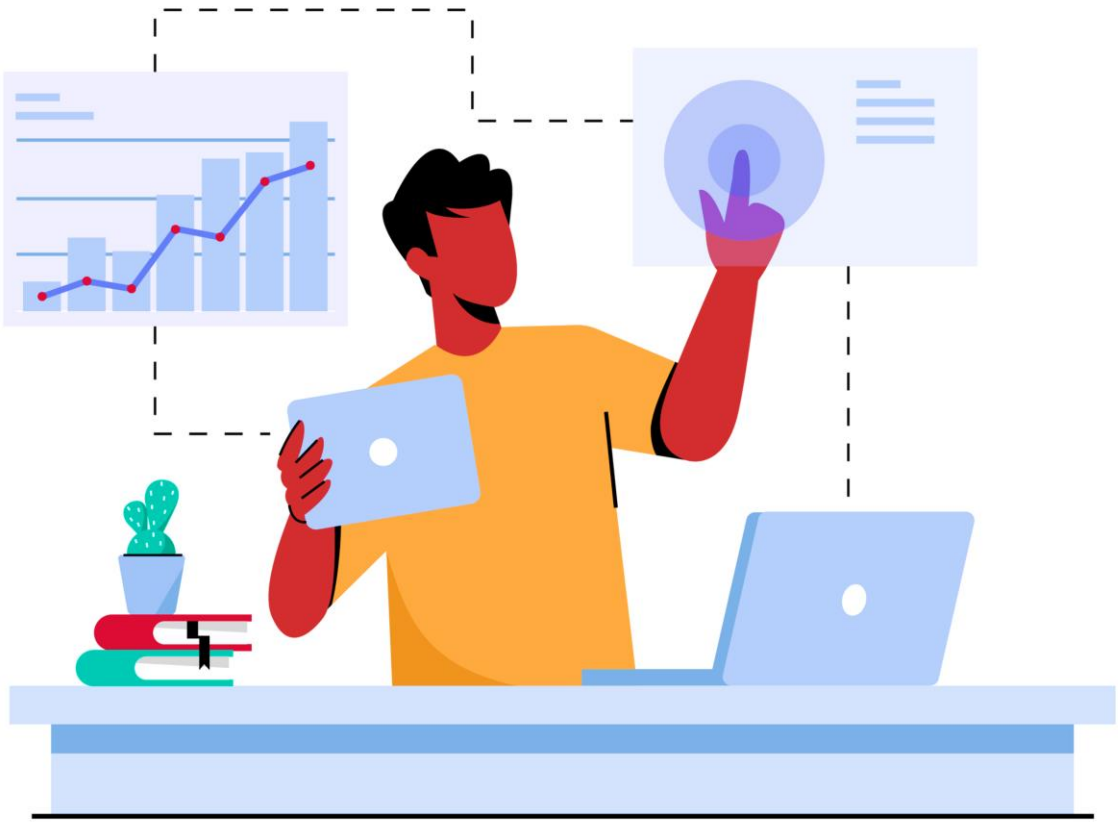
مدرس: میلاد وزان

دانشگاه شهید بهشتی - دانشکده ریاضی - گروه آموزشی آمار

نیم‌سال دوم ۱۴۰۴-۱۴۰۳

https://dbsbu.github.io

فصل ۵: مفاهیم پیشرفته SQL



اتصال (پیوند-ترکیب) جداول در پایگاه داده‌های رابطه‌ای با استفاده از JOIN

در پایگاه داده‌های رابطه‌ای، داده‌ها اغلب در جداول جداگانه ذخیره می‌شوند تا هم از نظر منطقی مرتب باشند و هم از نظر فنی قابل نگهداری، توسعه و مدیریت آسان‌تر داشته باشند. این طراحی که با هدف **کاهش افزونگی** (Redundancy) و **افزایش نرمال‌سازی** (Normalization) انجام می‌شود، مزایای بی‌شماری دارد، اما در عین حال چالشی مهم نیز ایجاد می‌کند: چگونه داده‌های ذخیره‌شده در چندین جدول را به گونه‌ای مؤثر و دقیق با یکدیگر ترکیب کنیم؟ **پاسخ این چالش در مفهومی بنیادین در زبان SQL نهفته است: دستور JOIN**. این دستور امکان اتصال ردیف‌های مرتبط از دو یا چند جدول را بر اساس شرایط منطقی مشخص (مانند اشتراک کلیدها) فراهم می‌سازد.

با استفاده از JOIN پایگاه داده‌ها به جای ذخیره داده‌های تکراری، قادرند داده‌هایی منسجم، منعطف و منظم تولید کنند که هم در حافظه کارآمدتر باشند و هم در بازیابی اطلاعات دقیق‌تر.

JOIN در SQL یک عملیات رابطه‌ای است که سطرهایی از دو یا چند جدول را بر اساس شرطی مشخص (معمولاً تطابق بین کلیدها) به یکدیگر مرتبط می‌کند تا یک مجموعه‌ی داده‌ی جدید تولید کند.

ما داده‌ها را در جداول مجزا ذخیره می‌کنیم، اما برای استخراج معنا، باید این جداول را به هم وصل کنیم.

مثال ۱: فروشگاه اینترنتی

- جدول اول: مشخصات مشتریان (نام، ایمیل، شماره تماس)
- جدول دوم: سفارش‌ها (کد سفارش، شناسه مشتری، تاریخ، مبلغ)

اگر فقط جدول مشتریان را داشته باشی، نمی‌دانی **چه کسی چه چیزی خرید.**

اگر فقط جدول سفارش‌ها را داشته باشی، نمی‌دانی **این سفارش مربوط به چه کسی است.**

پس نیاز داری این دو جدول را **با هم ترکیب کنی**، تا بفهمی:

"علیرضا جهانبخش در تاریخ ۲۵ فروردین یک گوشی به مبلغ ۴۵ میلیون تومان سفارش داده است".

مثال ۲: دانشگاه

- یک جدول داریم به نام "دانشجویان": فقط نام و شماره دانشجویی آن‌ها
- یک جدول دیگر داریم به نام "درس‌ها": شماره دانشجویی و نام درسی که انتخاب کرده‌اند

برای اینکه بفهمیم **هر دانشجو چه دروسی انتخاب کرده است**، باید داده‌های این دو جدول را با استفاده از ستون "شماره دانشجویی" به‌هم وصل کنیم.

مثال ۳: بانک

- جدول اول: حساب‌های بانکی (شماره حساب، نام صاحب حساب)
- جدول دوم: تراکنش‌ها (شماره حساب، مبلغ، نوع تراکنش، تاریخ)

برای تولید یک گزارش ساده مثل:

"لیست تراکنش‌های آرتین رمضانی در اردیبهشت ماه" نیاز داری اطلاعات تراکنش‌ها را با اطلاعات صاحبان حساب ترکیب کنی.

همان‌طور که انسان‌ها برای ساخت جامعه، باید ارتباط برقرار کنند، جداول هم برای خلق معنا، باید از طریق JOIN به هم متصل شوند.

فلسفه ذخیره‌سازی داده‌های جداگانه: چرا داده‌ها را اصلاً جداگانه ذخیره می‌کنیم؟

در نگاه اول، شاید این پرسش برای بسیاری از دانشجویان و حتی برنامه‌نویسان مطرح شود:

اگر در نهایت قرار است داده‌ها از جداول مختلف با یکدیگر ترکیب شوند، چرا از ابتدا تمام داده‌ها را در یک جدول بزرگ و واحد ذخیره نمی‌کنیم؟ آیا این کار باعث ساده‌تر شدن بازیابی اطلاعات نمی‌شود؟

این پرسش به ظاهر ساده، در دل خود یکی از عمیق‌ترین اصول طراحی سیستم‌های اطلاعاتی را پنهان کرده است. برای پاسخ به آن، باید ابتدا با مفهومی کلیدی در طراحی پایگاه داده‌های رابطه‌ای آشنا شویم: **نرمال‌سازی**.

نرمال‌سازی

نرمال‌سازی یک فرآیند علمی و ساختاریافته برای طراحی جداول پایگاه داده است که **هدف آن** **کاهش افزونگی**، **جلوگیری از ناسازگاری‌های داده‌ای**، **و بهبود قابلیت نگهداری و گسترش پایگاه داده است**.
در قالب فنی، نرمال‌سازی مجموعه‌ای از قوانین و فرم‌های استاندارد (مانند 1NF، 2NF، 3NF و ...) را ارائه می‌دهد که هرکدام گام به گام، ساختار منطقی جداول را بهبود می‌بخشند.

چرا نرمال‌سازی مهم است؟

۱. **جلوگیری از تکرار غیرضروری داده‌ها:**
   - مثال: اگر در جدول کارمندان، نام دپارتمان به ازای هر کارمند تکرار شود، برای ۱۰۰۰ کارمند دپارتمان فناوری اطلاعات، ۱۰۰۰ بار عبارت "فناوری اطلاعات" ذخیره خواهد شد.

۲. **کاهش خطا و ناسازگاری در داده‌ها:**
   - اگر در یک ردیف نام دپارتمان "فناوری اطلاعات" و در ردیف دیگر اشتباها "فناوری اطلاعت" نوشته شده باشد، آمارها دچار خطا خواهند شد.

۳. **افزایش انسجام و به‌روزرسانی‌پذیری داده‌ها:**
   - اگر روزی بخواهیم نام دپارتمان "فناوری اطلاعات" را به "توسعه فناوری" تغییر دهیم، در حالت نرمال‌نشده باید در صدها ردیف این کار انجام شود، اما در صورت استفاده از جدول جداگانه دپارتمان‌ها، فقط کافی است یک ردیف را ویرایش کنیم.

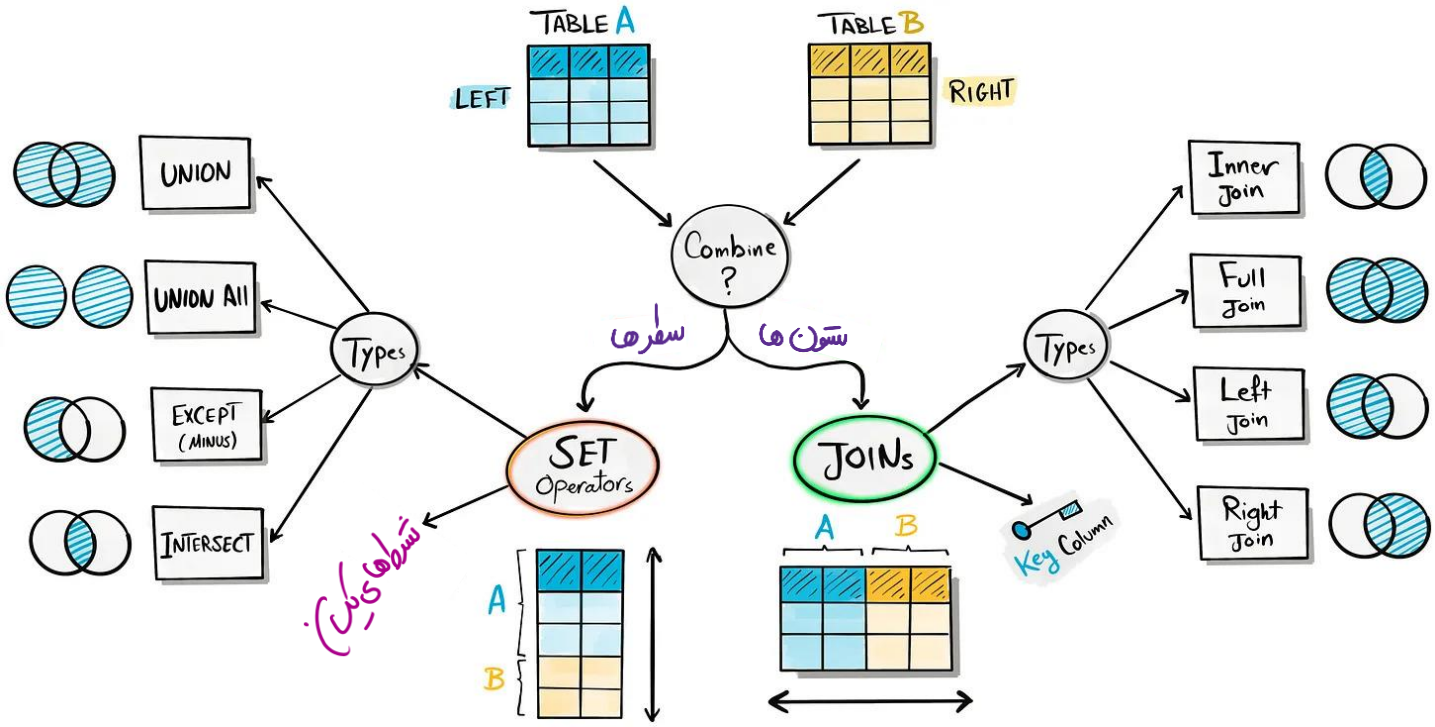
نتیجه: **تفکیک داده‌ها برای پیوستگی معنایی بهتر**
بنابراین، در طراحی پایگاه داده‌های حرفه‌ای، داده‌ها عمدا در جداول مجزا نگهداری می‌شوند. این تفکیک نه تنها از نظر فنی به صرفه است، بلکه از نظر مفهومی نیز باعث می‌شود هر جدول معنای مشخص و مستقلی داشته باشد. این استقلال اما ما را به ابزار **JOIN** نیازمند می‌کند، تا در مواقع نیاز، این اطلاعات جداگانه را به شکلی معنادار ترکیب کنیم.

تعریف JOIN

در پایگاه داده‌های رابطه‌ای، **JOIN** یک عملیات رابطه‌ای است که امکان ترکیب دو یا چند جدول را بر اساس یک شرط منطقی فراهم می‌کند. این شرط معمولا بر پایه ستون‌هایی است که رابطه‌ای معنایی با یکدیگر دارند (مانند کلیدهای اصلی و خارجی). **هدف JOIN این است که یک دید ترکیبی و معنادار از داده‌های پراکنده در جداول مختلف ایجاد کند.**

به طور کلی، ساختار **JOIN** به صورت زیر است:

```
SELECT columns
FROM table1
JOIN table2
ON table1.column = table2.column;
```



انواع JOIN

INNER JOIN

فرض کنید می‌خواهید ردیف‌های جدول X و Y را با استفاده از **INNER JOIN** ادغام کنید:

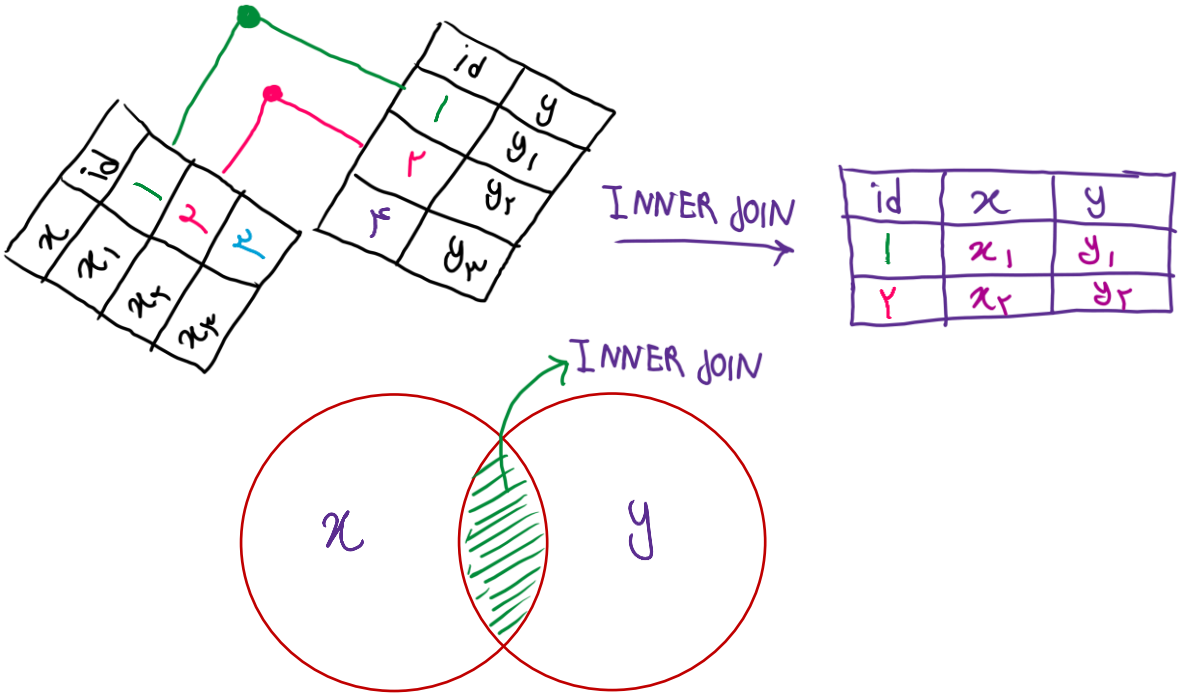
جدول X دو ستون دارد: `id` (کلید) و x

جدول Y نیز دو ستون دارد: `id` (کلید) و y

id	x
۱	x_1
۲	x_2
۳	x_3

id	y
۱	y_1
۲	y_2
۴	y_4

INNER JOIN شامل ردیف‌هایی با مقادیر منطبق در ستون‌های id است.



INNER JOIN فقط رکوردهایی که در هر دو جدول مطابق‌اند را نشان می‌دهد.

نحو پایه **INNER JOIN** به صورت زیر است:

```
SELECT
  table1.column1,
  table2.column2,
  ...
FROM
  table1
INNER JOIN table2 ON table1.column1 = table2.column1
```

در این نحو:

- ابتدا، نام جدول اول (`table1`) را در بخش `FROM` مشخص کنید.
- سپس، نام جدول دوم (`table2`) را که می‌خواهید ردیف‌های آن را با جدول اول ادغام کنید، در بخش `INNER JOIN` بنویسید.
- سوم، از یک شرط در بخش `ON` استفاده کنید تا ردیف‌های جدول اول (`table1`) را با ردیف‌های جدول دوم (`table2`) تطبیق دهید. این شرط ردیف‌ها را با مقایسه مقادیر `column1` در `table1` با مقادیر `column1` در `table2` تطبیق می‌دهد. توجه داشته باشید که می‌توان از عملگرهای مقایسه‌ای دیگر نیز استفاده کرد. اگر می‌خواهید از چندین شرط استفاده کنید، می‌توانید آن‌ها را با استفاده از عملگر منطقی `AND` در بخش `ON` ترکیب کنید.
- در نهایت، ستون‌هایی که می‌خواهید در نتیجه نهایی نشان داده شوند را از هر دو جدول در بخش `SELECT` فهرست کنید.

PostgreSQL ابتدا بخش FROM را ارزیابی می‌کند، سپس INNER JOIN را و در نهایت SELECT را.

نحوه عملکرد inner join در PostgreSQL به این صورت است:

- ابتدا، INNER JOIN مقادیر column1 در هر دو جدول را مقایسه می‌کند.
- سپس، اگر مقادیر برابر باشند، ردیف‌های مطابق را در یک جدول میانی ادغام می‌کند.
- در نهایت، ستون‌هایی را که در بخش SELECT مشخص شده‌اند در مجموعه نتایج نهایی انتخاب می‌کند.

مثال ۱: دو جدول به نام‌های employees (کارمندان) و departments (دپارتمان‌ها) ایجاد کرده‌ایم:

```
CREATE TABLE departments (
    department_id SERIAL PRIMARY KEY,
    department_name VARCHAR
);

CREATE TABLE employees (
    employee_id SERIAL PRIMARY KEY,
    name VARCHAR,
    department_id INT REFERENCES departments(department_id)
);
```

و داده‌های زیر را در آن درج شده‌اند:

```
-- درج دپارتمان‌ها
INSERT INTO departments (department_name) VALUES
('IT'),
('HR'),
('Finance'),
('Marketing'),
('Sales');

-- درج کارمندان
INSERT INTO employees (name, department_id) VALUES
('Ali', 1),
('Sara', 2),
('Reza', 1),
('Neda', 3),
('Mina', NULL),
('Omid', 4),
('Hamed', 2),
('Zahra', 5),
('Maryam', NULL),
('Javad', 3);
```

می‌خواهیم فقط کارمندانی که دپارتمان مشخصی دارند (یعنی department_id آن‌ها NULL نیست) را نشان بدهیم:

```
SELECT e.name, d.department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id;
```

خروجی:

```
name   department_name
Ali    IT
Sara   HR
Reza   IT
Neda   Finance
Omid   Marketing
Hamed  HR
Zahra  Sales
Javad  Finance
```

`employees e` یعنی جدول `employees` را با نام کوتاه `e` صدا می‌زنیم.

`departments d` یعنی جدول departments را با نام کوتاه `d` صدا می‌زنیم.

- `SELECT e.name, d.department_name`

این بخش مشخص می‌کند که می‌خواهیم چه ستون‌هایی را در خروجی ببینیم.

- `e.name` نام کارمند از جدول `employees`
- `d.department_name` نام دپارتمان از جدول `departments`

- `FROM employees e`

جدول `employees` را به‌عنوان جدول اصلی برای انتخاب داده‌ها مشخص می‌کنیم و به آن یک نام مستعار به نام `e` می‌دهیم.

- `INNER JOIN departments d`

جدول `departments` را با جدول `employees` به‌صورت `INNER JOIN` ترکیب می‌کنیم و به آن هم نام مستعار `d` می‌دهیم.

- `ON e.department_id = d.department_id`

شرط پیوند را مشخص می‌کنیم: ردیف‌هایی از هر دو جدول که مقدار `department_id` یکسان دارند به هم مرتبط می‌شوند.

مثال ۲: نام تمام کارمندانی که در دپارتمان "IT" هستند.

```
SELECT e.name
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
WHERE d.department_name = 'IT';
```

یا

```
SELECT e.name
FROM employees e
INNER JOIN departments d
 ON e.department_id = d.department_id
 AND d.department_name = 'IT';
```

خروجی:

```
name
Ali
Reza
```

از نظر عملکرد نهایی در `INNER JOIN`، این دو کوئری خروجی یکسانی دارند.

اما از نظر ترتیب اجرا:

در اولی، ابتدا `JOIN` کامل انجام می‌شود، سپس فیلتر با `WHERE`

در دومی، فقط ردیف‌هایی `JOIN` می‌شوند که از همان ابتدا `d.department_name = 'IT'` باشند.

مثال ۳: فقط کارمندانی که در دپارتمان "HR" یا "Sales" هستند:

```
SELECT e.name, d.department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
WHERE d.department_name IN ('HR', 'Sales');
```

یا

```
SELECT e.name, d.department_name
FROM employees e
INNER JOIN departments d
  ON e.department_id = d.department_id
  AND d.department_name IN ('HR', 'Sales');
```

خروجی:

```
name  department_name
Sara  HR
Hamed HR
Zahra Sales
```

مثال ۴: نام و دپارتمان کارمندانی را نمایش بده که در دپارتمان "IT" یا "Finance" هستند و نام آن‌ها با حرف "A" شروع می‌شود.

```
SELECT e.name, d.department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
WHERE d.department_name IN ('IT', 'Finance') AND e.name LIKE 'A%';
```

خروجی:

```
name  department_name
Ali   IT
```

مثال ۵: دپارتمان‌هایی را نمایش بده که حداقل دو کارمند دارند.

```
SELECT d.department_name, COUNT(e.employee_id) AS employee_count
FROM employees e
INNER JOIN departments d ON e.department_id = d.department_id
GROUP BY d.department_name
HAVING COUNT(e.employee_id) >= 2;
```

خروجی:

```
department_name  employee_count
Finance          2
IT               2
HR               2
```

LEFT JOIN

فرض کنید می‌خواهید ردیف‌های جدول X و Y را با استفاده از **LEFT JOIN** ادغام کنید:

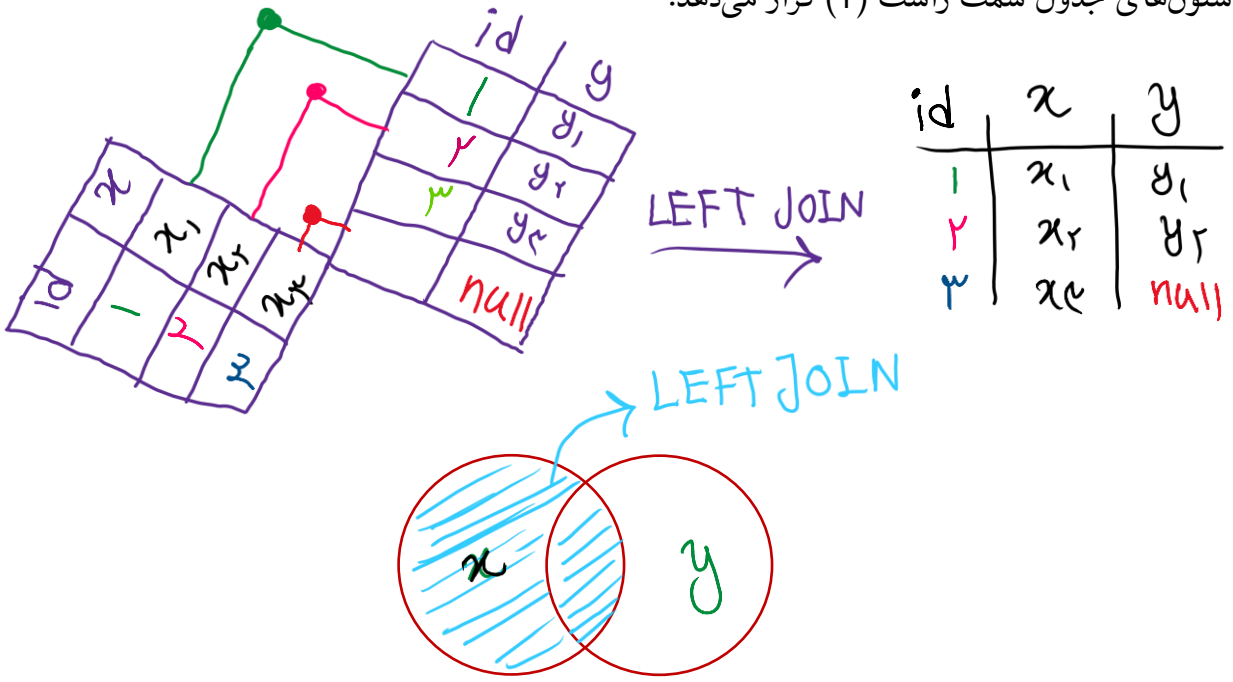
جدول X دو ستون دارد: id (کلید) و x

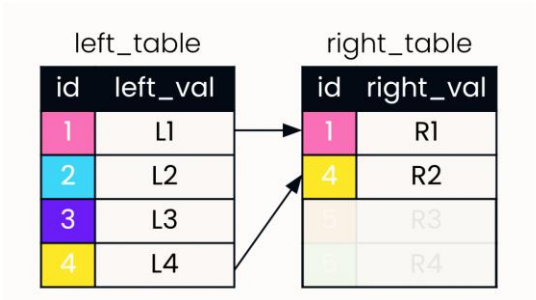
جدول Y نیز دو ستون دارد: id (کلید) و y

id	x
۱	x_1
۲	x_2
۳	x_3

id	y
۱	y_1
۲	y_2
۴	y_4

LEFT JOIN تمام ردیف‌های جدول سمت چپ (X) و ردیف‌های مطابق از جدول سمت راست (Y) را دربر می‌گیرد؛ اگر ردیفی در جدول سمت چپ وجود داشته باشد که در جدول سمت راست تطابقی نداشته باشد، **LEFT JOIN** مقدار **NULL** را برای ستون‌های جدول سمت راست (Y) قرار می‌دهد.





دستور LEFT JOIN ردیف‌هایی از دو جدول را با یکدیگر ترکیب می‌کند و تمام ردیف‌های جدول سمت چپ و ردیف‌های مطابق از جدول سمت راست را بازمی‌گرداند.

نحو دستور **LEFT JOIN**:

```
SELECT
  left_table.column1,
  right_table.column2,
  ...
FROM
  left_table
  LEFT JOIN right_table ON right_table.column1 = left_table.column1;
```

در این نحو:
ابتدا، نام جدول سمت چپ (left_table) را در بخش FROM مشخص می‌کنید:

```
FROM left_table
```

دوم، نام جدول سمت راست (right_table) را که می‌خواهید به جدول سمت چپ ملحق کنید، در بخش LEFT JOIN مشخص می‌کنید:

```
LEFT JOIN right_table
```

سوم، از یک شرط در بخش ON استفاده می‌کنید تا ردیف‌های جدول سمت چپ را با ردیف‌های جدول سمت راست مطابقت دهید:

```
ON left_table.column1 = right_table.column1;
```

PostgreSQL همیشه ردیف جدول سمت چپ را به همراه ردیف مطابق از جدول سمت راست در خروجی قرار می‌دهد.
اگر ردیفی از جدول سمت چپ، ردیف مطابقی در جدول سمت راست نداشته باشد، PostgreSQL اقدامات زیر را انجام می‌دهد:

- یک ردیف "ساختگی" از ستون‌های جدول سمت راست ایجاد می‌کند.
- تمام ستون‌های این ردیف را با مقدار NULL پر می‌کند.
- ردیف ساختگی را با ردیف واقعی از جدول سمت چپ ترکیب می‌کند.

در نهایت، ستون‌هایی را که در بخش SELECT مشخص شده‌اند بازمی‌گرداند:

```
SELECT
  left_table.column1,
  right_table.column2,
  ...
```

😉**توجه داشته باشید که جدولی که در بخش FROM مشخص می‌کنید، جدول سمت چپ است، و جدولی که در بخش LEFT JOIN تعیین می‌کنید، جدول سمت راست محسوب می‌شود.**

مثال ۱: لیست همه‌ی کارمندان را با نام دپارتمان‌شان نمایش بده، حتی اگر دپارتمانی نداشته باشند.

```
SELECT e.name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id;
```

خروجی:

```
name  department_name
Ali   IT
Sara  HR
Reza  IT
Neda  Finance
Mina  None
Omid  Marketing
Hamed HR
Zahra Sales
Maryam None
Javad Finance
```

مثال ۲: نام کارمندانی را پیدا کن که دپارتمان ندارند (فقط کارمندانی که department_id آن‌ها NULL است).

```
SELECT e.name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id
WHERE d.department_id IS NULL;
```

خروجی:

```
name  department_name
Mina        None
Maryam      None
```

نکته: استفاده از عبارت USING در دستورهای JOIN در SQL

همانطور که تا این‌جا مشاهده گردید، در SQL، برای ترکیب داده‌ها از دو یا چند جدول، معمولا از دستورات JOIN استفاده می‌شود. یکی از روش‌های پرکاربرد برای تعیین شرط اتصال بین جدول‌ها استفاده از کلید خارجی است **که در بسیاری از موارد، نام ستون‌هایی که برای اتصال استفاده می‌شوند، در هر دو جدول یکسان است.** در چنین شرایطی، SQL امکانی را فراهم کرده است که با استفاده از عبارت USING بتوانیم شرط اتصال را ساده‌تر و خواناتر بنویسیم.

استفاده از عبارت USING در مثال ۱:

```
SELECT e.name, d.department_name
FROM employees e
LEFT JOIN departments d USING(department_id);
```

در این روش، چون ستون department_id در هر دو جدول با نام یکسان وجود دارد، می‌توانیم از USING(department_id) استفاده کنیم تا شرط اتصال را ساده‌تر و تمیزتر بنویسیم.

استفاده از عبارت USING در مثال ۲:

```
SELECT e.name, d.department_name
FROM employees e
LEFT JOIN departments d USING(department_id);
WHERE d.department_id IS NULL;
```

محدودیت در استفاده: USING فقط زمانی قابل استفاده است که نام ستون در هر دو جدول کاملا یکسان باشد. اگر نام ستون‌ها متفاوت باشد، باید از ON استفاده کرد.

مثال‌های بیشتر برای LEFT JOIN

```
CREATE TABLE courses_sbu (
    course_id SERIAL PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL
);
CREATE TABLE students_sbu (
    student_id SERIAL PRIMARY KEY,
    student_name VARCHAR(100) NOT NULL,
    course_id INTEGER REFERENCES courses_sbu(course_id)
);
```

```
INSERT INTO courses_sbu (course_name) VALUES
('ریاضیات'),
('برنامه‌نویسی'),
('پایگاه داده‌ها'),
('شبکه‌های کامپیوتری');

INSERT INTO students_sbu (student_name, course_id) VALUES
('سینا محمدی', ۱),
('الهام کاظمی', ۲),
('مهدی شریفی', ۲),
('نسترن رجبی', NULL);  -- دانشجو بدون درس
```

کوئری زیر، همه دانشجوها را نمایش می‌دهد. اگر دانشجویی درسی انتخاب نکرده باشد (مانند نسترن رجبی)، ستون نام_درس مقدار NULL خواهد داشت.

```
SELECT s.student_name AS نام_دانشجو, c.course_name AS نام_درس
FROM students_sbu s
LEFT JOIN courses_sbu c ON s.course_id = c.course_id;
```

خروجی:

نام_دانشجو	نام_درس
سینا محمدی	ریاضیات
الهام کاظمی	برنامه‌نویسی
مهدی شریفی	برنامه‌نویسی
نسترن رجبی	None

در این مثال، می‌خواهیم فقط آن دسته از دانشجوها را ببینیم که هنوز درسی انتخاب نکرده‌اند. برای این کار از LEFT JOIN به‌همراه WHERE course_name IS NULL استفاده می‌کنیم:

```
SELECT s.student_name AS نام_دانشجو
FROM students_sbu s
LEFT JOIN courses_sbu c ON s.course_id = c.course_id
WHERE c.course_name IS NULL;
```

خروجی:

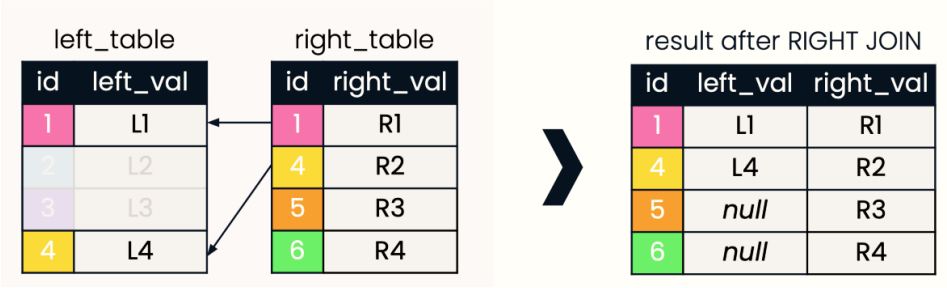
نام_دانشجو
نسترن رجبی

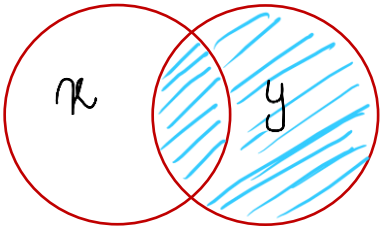
RIGHT JOIN

دستور RIGHT JOIN به شما این امکان را می‌دهد تمام ردیف‌های جدول سمت راست را به همراه ردیف‌های منطبق از جدول سمت چپ برمی‌گرداند و اگر ردیفی در جدول سمت راست بدون متناظر در جدول سمت چپ باشد، مقادیر جدول سمت چپ در آن ردیف NULL خواهد بود.

نحو پایه دستور RIGHT JOIN به شکل زیر است:

```sql
SELECT
  left_table.column1,
  right_table.column2,
  ...
FROM
  left_table
  RIGHT JOIN right_table ON right_table .column1 = left_table.column1;
```





نکته. در SQL، دو عبارت زیر از نظر عملکرد کاملا معادل هستند (معادل بودن RIGHT JOIN و LEFT JOIN با جابه‌جایی جدول‌ها):

A RIGHT JOIN B ≡ B LEFT JOIN A

یعنی اگر جدول‌ها را جابه‌جا کنیم، می‌توانیم RIGHT JOIN را به LEFT JOIN تبدیل کنیم و بالعکس. تفاوت فقط در ترتیب نوشتن است، نه در خروجی نهایی.

مثال: با کوئری زیر تمام دروس نمایش داده می‌شوند، حتی اگر دانشجویی در آن درس ثبت‌نام نکرده باشد:

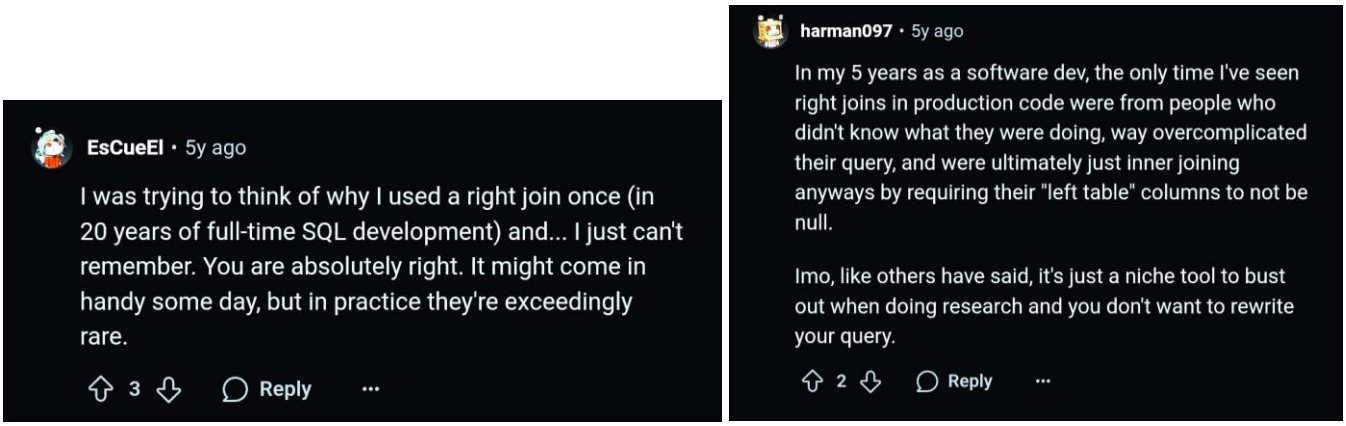
```sql
SELECT s.student_name AS نام_دانشجو, c.course_name AS نام_درس
FROM students_sbu s
RIGHT JOIN courses_sbu c ON s.course_id = c.course_id;
```

خروجی:

نام_درس	نام_دانشجو
ریاضیات	سینا محمدی
برنامه‌نویسی	الهام کاظمی
برنامه‌نویسی	مهدی شریفی
شبکه‌های کامپیوتری	None
پایگاه داده‌ها	None

معادل کوئری بالا با LEFT JOIN:

```
SELECT s.student_name AS نام_دانشجو, c.course_name AS نام_درس
FROM courses_sbu c
LEFT JOIN students_sbu s ON c.course_id = s.course_id;
```



FULL JOIN

FULL JOIN ردیف‌های دو جدول را با هم ادغام می‌کند و تمام ردیف‌ها را از هر دو جدول بازمی‌گرداند. علاوه بر این، FULL JOIN برای هر ستون از جدولی که ردیف متناظری ندارد، مقدار NULL قرار می‌دهد. به عبارت دیگر، FULL JOIN مجموعه نتایج حاصل از LEFT JOIN و RIGHT JOIN را ترکیب می‌کند. در زیر نحو پایه دستور FULL JOIN آمده است:

```
SELECT
  table1.column1,
  table2.column2,
  ...
FROM
  table1
  FULL JOIN table2 ON table2.column1 = table1.column1;
```

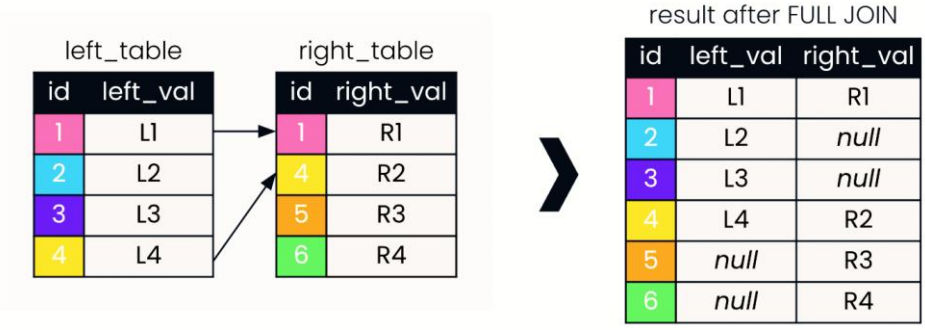
در این نحو:

اول، نام جدول اول (table1) را در FROM clause مشخص کنید.

دوم، نام جدول دوم (table2) که می‌خواهید full join روی آن انجام دهید را در FULL JOIN clause وارد کنید.

سوم، از یک شرط برای مطابقت ردیف‌ها از هر دو جدول در ON clause استفاده کنید. این شرط ردیف‌ها را با مقایسه مقادیر column1 در table1 با مقادیر column1 در table2 مطابقت می‌دهد.

در نهایت، ستون‌هایی که می‌خواهید داده‌های آنها را بازیابی کنید را در SELECT clause مشخص نمایید.



مثال ۱: نمایش تمام دانشجویان و تمام دروس، حتی اگر:

- دانشجویی هنوز انتخاب واحد نکرده باشد
- یا درسی باشد که هنوز هیچ دانشجویی آن را انتخاب نکرده

(نمایش تمام دانشجویان و تمام دروس به همراه مشخص کردن اینکه کدام دروس دانشجویانی ندارند و همچنین کدام دانشجویان در هیچ درسی ثبت‌نام نکرده‌اند).

```
SELECT s.student_name AS نام_دانشجو, c.course_name AS نام_درس
FROM students_sbu s
FULL JOIN courses_sbu c ON s.course_id = c.course_id;
```

خروجی:

نام_درس	نام_دانشجو
ریاضیات	سینا محمدی
برنامه‌نویسی	الهام کاظمی
برنامه‌نویسی	مهدی شریفی
None	نسترن رجبی
شبکه‌های کامپیوتری	None
پایگاه داده‌ها	None

CROSS JOIN

دستور CROSS JOIN هر سطر از جدول اول را با هر سطر از جدول دوم ترکیب می‌کند و تمام ترکیب‌های ممکن از ردیف‌ها را بازمی‌گرداند. بر خلاف سایر JOINها مانند INNER JOIN، LEFT JOIN و FULL JOIN، در CROSS JOIN هیچ شرطی برای تطبیق ردیف‌ها از دو جدول وجود ندارد. در زیر نحو دستور CROSS JOIN آمده است:

```
SELECT
  table1.column1,
  table2.column2,
  ...
FROM
  table1
  CROSS JOIN table2;
```

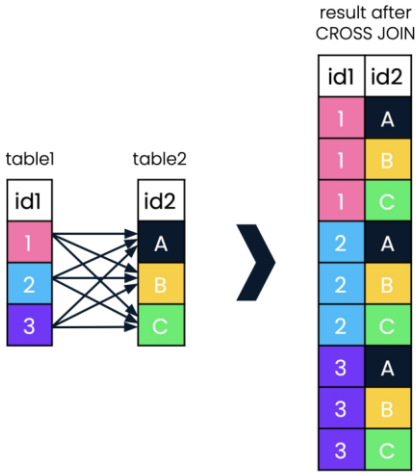
در این نحو:

اول، نام جدول اول (table1) را در بخش FROM مشخص کنید.

دوم، نام جدول دوم (table2) را در بخش CROSS JOIN وارد کنید.

سوم، ستون‌های هر دو جدول که می‌خواهید داده‌های آن‌ها را بازیابی کنید در بخش SELECT فهرست کنید.

اگر جدول اول دارای n ردیف و جدول دوم دارای m ردیف باشد، CROSS JOIN مجموعه نتایجی با n ضربدر m ردیف بازمی‌گرداند.



مثال: تولید همه تاریخ‌های ممکن برای سفارش‌های مشتریان ثبت‌شده

جدول‌ها:

- Customers (مشتری‌ها)
- PossibleDeliveryDates (تاریخ‌های ممکن برای تحویل سفارش)

```
CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);

CREATE TABLE possible_delivery_dates (
    delivery_date DATE PRIMARY KEY
);

-- مشتری‌ها
INSERT INTO customers (name) VALUES
('علی'),
('خشایار'),
('سارا');
-- تاریخ‌های تحویل
INSERT INTO possible_delivery_dates (delivery_date) VALUES
('2025-06-01'),
('2025-06-03');
```

```
SELECT
    c.name AS customer,
    d.delivery_date
FROM
    customers c
CROSS JOIN
    possible_delivery_dates d
ORDER BY
    c.name, d.delivery_date;
```

```
customer    delivery_date
خشایار  ۲۰۲۵-۰۶-۰۱
خشایار  ۲۰۲۵-۰۶-۰۳
سارا    ۲۰۲۵-۰۶-۰۱
سارا    ۲۰۲۵-۰۶-۰۳
علی     ۲۰۲۵-۰۶-۰۱
علی     ۲۰۲۵-۰۶-۰۳
```

زیرپرس‌وجو (Subquery)

در SQL، زیرپرس‌وجو یا پرس‌وجوی تو در تو، یک پرس‌وجو است که در درون یک پرس‌وجوی دیگر نوشته می‌شود. **زیرپرس‌وجوها معمولا برای استخراج اطلاعاتی به کار می‌روند که نیاز به مقایسه، فیلتر یا استفاده از نتایج یک پرس‌وجوی دیگر دارند.** یک زیرپرس‌وجو معمولا داخل پرانتز نوشته می‌شود و می‌تواند در بخش‌های مختلف یک کوئری مانند:

WHERE
FROM
SELECT

استفاده شود.

فرض کنید جدول‌های زیر را داریم (**در پیوست فصل**):

جدول دانشجو

student_id	first_name	last_name	major	entry_year	level
۱	آرتین	رمضانی	آمار	۱۴۰۰	کارشناسی
۲	علیرضا	جهانبخش	آمار	۱۳۹۹	کارشناسی
۳	سینا	قنبری	علوم کامپیوتر	۱۴۰۰	کارشناسی
۴	ریحانه	مهدوی	ریاضی	۱۳۹۹	کارشناسی
۵	مهسا	کریمی	بیم‌سنجی	۱۳۹۸	کارشناسی
۶	محمد	صادقی	علوم کامپیوتر	۱۴۰۱	کارشناسی

جدول درس‌ها

course_id	title	units	professor
۱۰۱	داده‌کاوی	۳	دکتر فراهانی
۱۰۲	یادگیری ماشین	۳	دکتر خردپیشه
۱۰۳	جبر خطی	۳	دکتر حجاریان
۱۰۴	ریاضی عمومی	۳	دکتر طوسی
۱۰۵	آمار و احتمال	۲	دکتر گنجعلی

student_id	course_id	term	grade
۱	۱۰۵	پاییز ۱۴۰۱	۱۷/۵
۱	۱۰۴	پاییز ۱۴۰۱	۱۶/۰
۲	۱۰۵	بهار ۱۴۰۰	۱۸/۵
۲	۱۰۳	پاییز ۱۳۹۹	۱۵/۰
۳	۱۰۱	پاییز ۱۴۰۱	۱۹/۰
۳	۱۰۲	پاییز ۱۴۰۱	۱۸/۵
۴	۱۰۴	پاییز ۱۳۹۹	۱۴/۰
۵	۱۰۵	بهار ۱۴۰۰	۱۷/۰
۶	۱۰۱	پاییز ۱۴۰۲	۱۶/۵
۶	۱۰۳	پاییز ۱۴۰۲	۱۷/۲

زیرپرس‌وجو در قسمت WHERE

مثال ۱: تصور کنید می‌خواهیم دانشجویانی را پیدا کنیم که نمره‌شان در درس آمار و احتمال بیشتر از میانگین نمره این درس بوده است.

مرحله اول: میانگین نمره درس آمار و احتمال را به‌دست می‌آوریم:

```
SELECT
  AVG(grade)
FROM
  student_course
WHERE
  course_id = 105;
```

خروجی:

```
avg
۱۷/۶
```

مرحله دوم: حالا دانشجویانی که نمره‌شان بالاتر از این میانگین بوده را پیدا می‌کنیم:

```
SELECT
  student_id,
  grade
FROM
  student_course
WHERE
  course_id = 105
  AND grade > 17.6;
```

خروجی:

```
 student_id   grade
 2            18.5
```

اما می‌توانیم این کار را با یک زیرپرس‌وجو در همان دستور انجام دهیم:

```
SELECT
  student_id,
  grade
FROM
  student_course
WHERE
  course_id = 105
  AND grade > (
    SELECT
      AVG(grade)
    FROM
      student_course
    WHERE
      course_id = 105
  );
```

خروجی:

```
 student_id   grade
 2            18.5
```

حالا بیایید اسامی دانشجویانی را که نمره‌شان در آمار و احتمال بالاتر از میانگین است نمایش دهیم:

```
SELECT first_name, last_name
FROM student
WHERE student_id IN (
    SELECT student_id
    FROM student_course
    WHERE course_id = 105
    AND grade > (
        SELECT AVG(grade)
        FROM student_course
        WHERE course_id = 105)
);
```

خروجی:

```
first_name last_name
جهانبخش  علیرضا
```

ساختار کلی کوئری بالا

```
SELECT first_name, last_name
FROM student
WHERE student_id IN ( ... );
```

این کوئری می‌گوید:

اسامی (first_name و last_name) از جدول student را بیاور **فقط برای آن دانشجویانی که شناسه‌شان در یک لیست خاص وجود دارد.** حالا سؤال این است، این لیست خاص چیست؟ در واقع زیرپرس‌وجو داخل IN (...) آن را می‌سازد.

زیرپرس‌وجو داخلی (زیرپرس‌وجو اول):

```
SELECT student_id
FROM student_course
WHERE course_id = 105
  AND grade > (...);
```

این بخش می‌گوید:

شماره دانشجویانی را پیدا کن که:

درس آمار و احتمال را گذرانده‌اند (course_id = 105)

و نمره‌شان از یک مقدار مشخص بیشتر بوده.

حال، آن مقدار خاص چیست؟ **زیرپرس‌وجوی داخلی‌تر آن را به ما می‌گوید.**

زیرپرس‌وجوی دوم (محاسبه میانگین)

```
SELECT AVG(grade)
FROM student_course
WHERE course_id = 105;
```

این بخش خیلی ساده است:

میانگین نمره درس آمار و احتمال (course_id = 105) را حساب کن.

به‌طور خلاصه:

ابتدا میانگین نمره درس آمار و احتمال محاسبه می‌شود.

سپس دانشجویانی که نمره‌شان بالاتر از آن میانگین است انتخاب می‌شوند.

در نهایت نام و نام خانوادگی این دانشجویان از جدول student گرفته می‌شود.

```sql
SELECT first_name, last_name
FROM student
WHERE student_id IN (
    SELECT student_id
    FROM student_course
    WHERE course_id = 105
    AND grade > (
        SELECT AVG(grade)
        FROM student_course
        WHERE course_id = 105)
);
```

۱۷٫۶۷

student_id	course_id	term	grade
۱	۱۰۵	پاییز ۱۴۰۱	۱۷/۵
۱	۱۰۴	پاییز ۱۴۰۱	۱۶/۰
۲	۱۰۵	بهار ۱۴۰۰	۱۸/۵
۲	۱۰۳	پاییز ۱۳۹۹	۱۵/۰
۳	۱۰۱	پاییز ۱۴۰۱	۱۹/۰
۳	۱۰۲	پاییز ۱۴۰۱	۱۸/۵
۴	۱۰۴	پاییز ۱۳۹۹	۱۴/۰
۵	۱۰۵	بهار ۱۴۰۰	۱۷/۰
۶	۱۰۱	پاییز ۱۴۰۲	۱۶/۵
۶	۱۰۳	پاییز ۱۴۰۲	۱۷/۲

AVG (۱۷٫۵, ۱۸٫۵, ۱۷٫۰) = ۱۷٫۶۷

```sql
SELECT first_name, last_name
FROM student
WHERE student_id IN (
    SELECT student_id
    FROM student_course
    WHERE course_id = 105
    AND grade > ۱۷/۶۷
);
```

۱) انتخاب می‌شود

student_id	course_id	term	grade
۱	۱۰۵	پاییز ۱۴۰۱	۱۷/۵
۱	۱۰۴	پاییز ۱۴۰۱	۱۶/۰
۲	۱۰۵	بهار ۱۴۰۰	۱۸/۵
۲	۱۰۳	پاییز ۱۳۹۹	۱۵/۰
۳	۱۰۱	پاییز ۱۴۰۱	۱۹/۰
۳	۱۰۲	پاییز ۱۴۰۱	۱۸/۵
۴	۱۰۴	پاییز ۱۳۹۹	۱۴/۰
۵	۱۰۵	بهار ۱۴۰۰	۱۷/۰
۶	۱۰۱	پاییز ۱۴۰۲	۱۶/۵
۶	۱۰۳	پاییز ۱۴۰۲	۱۷/۲

۱۷٫۶۷ ✗ ۱۷٫۶۷ ✓ ۱۷٫۶ ✗

```sql
SELECT first_name, last_name
FROM student
WHERE student_id IN (۲);
```

student_id	first_name	last_name	major	entry_year	level
۱	آرتین	رمضانی	آمار	۱۴۰۰	کارشناسی
۲	علیرضا	جهانبخش	آمار	۱۳۹۹	کارشناسی
۳	سینا	قنبری	علوم کامپیوتر	۱۴۰۰	کارشناسی
۴	ریحانه	مهدوی	ریاضی	۱۳۹۹	کارشناسی
۵	مهسا	کریمی	بیم‌سنجی	۱۳۹۸	کارشناسی
۶	محمد	صادقی	علوم کامپیوتر	۱۴۰۱	کارشناسی

مثال ۲: نام و نام خانوادگی دانشجویانی را نمایش بده که در هیچ درسی نمره کمتر از ۱۵ نگرفته‌اند.

- **می‌خواهیم دانشجویانی را پیدا کنیم که در هیچ کدام از دروسشان نمره‌ای کمتر از ۱۵ ندارند.**
- **پس باید بررسی کنیم که دانشجویانی که نمره کمتر از ۱۵ دارند، در لیست ما نباشند.**
- **یعنی student_idهایی که در student_course نمره < ۱۵ دارند، باید از لیست کلی حذف شوند.**

```
SELECT first_name, last_name
FROM student
WHERE student_id NOT IN (
  SELECT student_id
  FROM student_course
  WHERE grade < 15
);
```

خروجی:

first_name	last_name
آرتین	رمضانی
علیرضا	جهانبخش
سینا	قنبری
مهسا	کریمی
محمد	صادقی

student_id	first_name	last_name	major	entry_year	level
۱	آرتین	رمضانی	آمار	۱۴۰۰	کارشناسی
۲	علیرضا	جهانبخش	آمار	۱۳۹۹	کارشناسی
۳	سینا	قنبری	علوم کامپیوتر	۱۴۰۰	کارشناسی
۴	ریحانه	مهدوی	ریاضی	۱۳۹۹	کارشناسی
۵	مهسا	کریمی	بیم‌سنجی	۱۳۹۸	کارشناسی
۶	محمد	صادقی	علوم کامپیوتر	۱۴۰۱	کارشناسی

```
SELECT first_name, last_name
FROM student
WHERE student_id NOT IN (
  SELECT student_id
  FROM student_course
  WHERE grade < 15
);
```

NOT IN (۴)

۱-۲-۳-۵-۶

student_id	course_id	term	grade	
۱	۱۰۵	پاییز ۱۴۰۱	۱۷/۵	< ۱۵
۱	۱۰۴	پاییز ۱۴۰۱	۱۶/۰	< ۱۵
۲	۱۰۵	بهار ۱۴۰۰	۱۸/۵	< ۱۵
۲	۱۰۳	پاییز ۱۳۹۹	۱۵/۰	< ۱۵
۳	۱۰۱	پاییز ۱۴۰۱	۱۹/۰	< ۱۵
۳	۱۰۲	پاییز ۱۴۰۱	۱۸/۵	< ۱۵
۴	۱۰۴	پاییز ۱۳۹۹	۱۴/۰	< ۱۵ ✓
۵	۱۰۵	بهار ۱۴۰۰	۱۷/۰	< ۱۵
۶	۱۰۱	پاییز ۱۴۰۲	۱۶/۵	< ۱۵
۶	۱۰۳	پاییز ۱۴۰۲	۱۷/۲	< ۱۵

مثال ۳: نام دانشجویانی را نمایش بده که دروسی را گذرانده‌اند که توسط **دکتر فراهانی** تدریس شده است.

- ابتدا باید course_idهای دروسی که استاد آن‌ها دکتر فراهانی است را به‌دست آوریم.
- سپس دانشجویانی را که آن دروس را گذرانده‌اند از جدول student_course پیدا کنیم.
- در نهایت از جدول student اطلاعات آن‌ها را بگیریم.

```
SELECT first_name, last_name
FROM student
WHERE student_id IN (
  SELECT student_id
  FROM student_course
  WHERE course_id IN (
    SELECT course_id
    FROM course
    WHERE professor = 'دکتر فراهانی'
  )
);
```

خروجی:

first_name	last_name
محمد	صادقی
سینا	قنبری

student_id	first_name	last_name	major	entry_year	level
۱	آرتین	رمضانی	آمار	۱۴۰۰	کارشناسی
۲	علیرضا	جهانبخش	آمار	۱۳۹۹	کارشناسی
۳	سینا	قنبری	علوم کامپیوتر	۱۴۰۰	کارشناسی
۴	ریحانه	مهدوی	ریاضی	۱۳۹۹	کارشناسی
۵	مهسا	کریمی	بیم‌سنجی	۱۳۹۸	کارشناسی
۶	محمد	صادقی	علوم کامپیوتر	۱۴۰۱	کارشناسی

مرحله ۳

```
SELECT first_name, last_name
FROM student
WHERE student_id IN (
  SELECT student_id
  FROM student_course
  WHERE course_id IN (
    SELECT course_id
    FROM course
    WHERE professor = 'دکتر فراهانی'
  )
);
```

مرحله ۲

۱۰۱

مرحله ۱

course_id	title	units	professor
۱۰۱	داده‌کاوی	۳	دکتر فراهانی
۱۰۲	یادگیری ماشین	۳	دکتر خردپیشه
۱۰۳	جبر خطی	۳	دکتر حجاریان
۱۰۴	ریاضی عمومی	۳	دکتر طوسی
۱۰۵	آمار و احتمال	۲	دکتر گنجعلی

student_id	course_id	term	grade
۱	۱۰۵	پاییز ۱۴۰۱	۱۷/۵
۱	۱۰۴	پاییز ۱۴۰۱	۱۶/۰
۲	۱۰۵	بهار ۱۴۰۰	۱۸/۵
۲	۱۰۳	پاییز ۱۳۹۹	۱۵/۰
۳	۱۰۱	پاییز ۱۴۰۱	۱۹/۰
۳	۱۰۲	پاییز ۱۴۰۱	۱۸/۵
۴	۱۰۴	پاییز ۱۳۹۹	۱۴/۰
۵	۱۰۵	بهار ۱۴۰۰	۱۷/۰
۶	۱۰۱	پاییز ۱۴۰۲	۱۶/۵
۶	۱۰۳	پاییز ۱۴۰۲	۱۷/۲

زیرپرس‌وجو در قسمت SELECT

زیرپرس‌وجو در بخش SELECT معمولا زمانی استفاده می‌شود که می‌خواهید برای هر سطر یک مقدار محاسبه‌ای خاص را از یک جدول دیگر استخراج کنید.

برای نمایش یک ستون محاسبه‌ای اضافی در خروجی، مثل میانگین نمرات یک دانشجو، تعداد درس‌هایی که گرفته، یا بالاترین نمره‌ای که در هر درس گرفته.

ساختار کلی:

```
SELECT
  column1,
  (SELECT single_value FROM another_table WHERE condition) AS alias
FROM
  main_table;
```

نکته:

زیرپرس‌وجو در SELECT باید فقط یک مقدار برای هر سطر برگرداند.

اگر زیرپرس‌وجو بیش از یک مقدار برگرداند، خطای `more than one row returned` خواهی گرفت.

بنابراین معمولا از توابع تجمیعی استفاده می‌شود:

```
AVG(), COUNT(), MAX(), MIN(), SUM()
```

مثال ۱: تعداد درس‌هایی که هر دانشجو گذرانده است:

```
SELECT
  first_name, last_name,
  (
    SELECT
      COUNT(*)
    FROM
      student_course sc
    WHERE
      sc.student_id = s.student_id
  ) AS total_courses
FROM
  student s;
```

خروجی:

first_name	last_name	total_courses
آرتین	رمضانی	۲
علیرضا	جهانبخش	۲
سینا	قنبری	۲
ریحانه	مهدوی	۱
مهسا	کریمی	۱
محمد	صادقی	۲

```
FROM student s
```

از جدول student همه دانشجوها را انتخاب می‌کنیم:

student_id	first_name	last_name	major	entry_year	level
۱	آرتین	رمضانی	آمار	۱۴۰۰	کارشناسی
۲	علیرضا	جهانبخش	آمار	۱۳۹۹	کارشناسی
۳	سینا	قنبری	علوم کامپیوتر	۱۴۰۰	کارشناسی
۴	ریحانه	مهدوی	ریاضی	۱۳۹۹	کارشناسی
۵	مهسا	کریمی	بیم‌سنجی	۱۳۹۸	کارشناسی
۶	محمد	صادقی	علوم کامپیوتر	۱۴۰۱	کارشناسی

```
(SELECT
      COUNT(*)
    FROM
      student_course sc
    WHERE
      sc.student_id = s.student_id
  ) AS total_courses
```

این بخش یک زیرپرس‌وجو است که در ستون سوم از خروجی ظاهر می‌شود.

برای هر دانشجوی جدول student s، این زیرپرس‌وجو اجرا می‌شود.

این زیرپرس‌وجو به جدول student_course مراجعه می‌کند و می‌شمارد که چند ردیف مربوط به sc.student_id = s.student_id وجود دارد. **یعنی:**

برای هر دانشجو، زیرپرس‌وجو بررسی می‌کند که این دانشجو در چند درس ثبت‌نام کرده. مثلا اگر آرتین رمضانی دو درس دارد، مقدار total_courses برای او ۲ خواهد بود.

مثال ۲: نمایش نام و نام خانوادگی هر دانشجو به‌همراه میانگین نمرات دروسی که گذرانده است (معدل خام بدون وزن):

```
SELECT
  first_name,
  last_name,
  (
    SELECT
      AVG(grade)
    FROM
      student_course sc
    WHERE
      sc.student_id = s.student_id
  ) AS avg_grade
FROM
  student s;
```

first_name	last_name	avg_grade
آرتین	رمضانی	۱۶/۷۵
علیرضا	جهانبخش	۱۶/۷۵
سینا	قنبری	۱۸/۷۵
ریحانه	مهدوی	۱۴/۰
مهسا	کریمی	۱۷/۰
محمد	صادقی	۱۶/۸۵

برای هر دانشجو از جدول student، در جدول student_course به دنبال نمراتی می‌گردد که student_id آن‌ها با دانشجوی جاری (s.student_id) برابر باشد.

سپس با استفاده از AVG(grade) میانگین نمرات او محاسبه می‌شود.

زیرپرس‌وجو در قسمت FROM

عبارت **FROM** در یک پرس‌وجوی SQL برای مشخص کردن جدول یا جداولی استفاده می‌شود که داده‌ها باید از آن‌ها واکشی شوند. در حالت عادی، در بخش **FROM** نام جدول‌ها نوشته می‌شود، اما در برخی موارد، می‌توان به جای نام جدول، از یک زیرپرس‌وجو استفاده کرد. زمانی که از زیرپرس‌وجو در بخش **FROM** استفاده می‌کنیم، در واقع یک پرس‌وجوی داخلی اجرا می‌شود که نتیجه‌ی آن مانند یک جدول موقتی یا مجازی عمل می‌کند. به این نوع از زیرپرس‌وجوها اصطلاحا **جدول مشتق‌شده (Derived Table)** گفته می‌شود.

در این ساختار:

- ابتدا زیرپرس‌وجو در داخل بخش **FROM** اجرا می‌شود.
- نتیجه‌ی حاصل از آن مانند یک جدول موقت در نظر گرفته می‌شود.
- سپس کوئری اصلی می‌تواند روی این جدول مجازی عملیات انتخاب، فیلتر، گروه‌بندی یا مرتب‌سازی را انجام دهد، به گونه‌ای که گویی یک جدول معمولی است.

ساختار کلی:

```
SELECT column1, column2
FROM (
    SELECT ... FROM ...
) AS alias_table;
```

مثال ۱: اسامی دانشجویانی که درس "آمار و احتمال" را گذرانده‌اند همراه با نمره آن‌ها را نمایش بده.

```
SELECT s.first_name, s.last_name, t.grade
FROM (
    SELECT student_id, grade
    FROM student_course
    WHERE course_id = 105
) AS t,
student s
WHERE s.student_id = t.student_id;
```

خروجی:

first_name	last_name	grade
آرتین	رمضانی	۱۷/۵
علیرضا	جهانبخش	۱۸/۵
مهسا	کریمی	۱۷/۰

مثال ۲: نمایش نام و نام خانوادگی هر دانشجو به‌همراه میانگین نمرات دروسی که گذرانده است (معدل خام بدون وزن):

```
SELECT s.first_name, s.last_name, sub.avg_grade
FROM (
    SELECT student_id, AVG(grade) AS avg_grade
    FROM student_course
    GROUP BY student_id
) AS sub,
student s
WHERE s.student_id = sub.student_id;
```

خروجی:

first_name	last_name	avg_grade
آرتین	رمضانی	۱۶/۷۵
علیرضا	جهانبخش	۱۶/۷۵
سینا	قنبری	۱۸/۷۵
ریحانه	مهدوی	۱۴/۰
مهسا	کریمی	۱۷/۰
محمد	صادقی	۱۶/۸۵

مثال ۳: میانگین نمره دانشجویان آمار را در کنار میانگین نمره کل دانشجویان نمایش بده.

```
SELECT stat.avg_grade AS avg_stat_students, all_students.avg_grade AS avg_all_students
FROM (
    SELECT AVG(sc.grade) AS avg_grade
    FROM student_course sc, student s
    WHERE sc.student_id = s.student_id AND s.major = 'آمار'
) AS stat,
(
    SELECT AVG(grade) AS avg_grade
    FROM student_course
) AS all_students;
```

خروجی:

avg_stat_students	avg_all_students
16.75	16.919999999999998

یک سؤال SQL را می‌توان به روش‌های متفاوتی نوشت که همه آن‌ها خروجی یکسانی داشته باشند.

مثال ۴: نمایش نام، نام خانوادگی، رشته و تعداد درس‌های گذرانده‌ی دانشجویانی که سال ورودشان قبل از ۱۴۰۰ است.

روش ۱:

```
SELECT first_name, last_name, major,
       (
         SELECT COUNT(*)
         FROM student_course sc
         WHERE sc.student_id = s.student_id
       ) AS course_count
FROM student s
WHERE entry_year < 1400;
```

روش ۲:

```
SELECT s.first_name, s.last_name, s.major, sc.course_count
FROM student s,
     (
       SELECT student_id, COUNT(*) AS course_count
       FROM student_course
       GROUP BY student_id
     ) sc
WHERE s.student_id = sc.student_id
  AND s.entry_year < 1400;
```

روش ۳:

```
SELECT s.first_name, s.last_name, s.major, COUNT(sc.course_id) AS course_count
FROM student s
LEFT JOIN student_course sc ON s.student_id = sc.student_id
WHERE s.entry_year < 1400
GROUP BY s.student_id, s.first_name, s.last_name, s.major;
```

روش ۴:

```
SELECT s.first_name, s.last_name, s.major, sc.course_count
FROM student s
JOIN (
  SELECT student_id, COUNT(*) AS course_count
  FROM student_course
  GROUP BY student_id
) sc ON s.student_id = sc.student_id
WHERE s.entry_year < 1400;
```

خروجی:

first_name	last_name	major	course_count
مهسا	کریمی	بیم‌سنجی	۱
ریحانه	مهدوی	ریاضی	۱
علیرضا	جهانبخش	آمار	۲

مثال ۵: نام، نام خانوادگی، رشته، و میانگین نمرات دانشجویانی را نمایش بده که درسی با عنوان "آمار و احتمال" را گذرانده‌اند.

روش ۱:

```
SELECT first_name, last_name, major,
       (
         SELECT AVG(grade)
         FROM student_course sc
         WHERE sc.student_id = s.student_id
           AND sc.course_id = 105
       ) AS avg_grade
FROM student s
WHERE student_id IN (
  SELECT student_id
  FROM student_course
  WHERE course_id = 105
);
```

روش ۲:

```
SELECT s.first_name, s.last_name, s.major, t.avg_grade
FROM student s,
     (
       SELECT student_id, AVG(grade) AS avg_grade
       FROM student_course
       WHERE course_id = 105
       GROUP BY student_id
     ) t
WHERE s.student_id = t.student_id;
```

روش ۳:

```
SELECT s.first_name, s.last_name, s.major, AVG(sc.grade) AS avg_grade
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
WHERE sc.course_id = 105
GROUP BY s.student_id, s.first_name, s.last_name, s.major;
```

روش ۴:

```
SELECT s.first_name, s.last_name, s.major, AVG(sc.grade) AS avg_grade
FROM student s, student_course sc
WHERE s.student_id = sc.student_id
  AND sc.course_id = 105
GROUP BY s.student_id, s.first_name, s.last_name, s.major;
```

خروجی:

first_name	last_name	major	avg_grade
آرتین	رمضانی	آمار	۱۷/۵
علیرضا	جهانبخش	آمار	۱۸/۵
مهسا	کریمی	بیم‌سنجی	۱۷/۰

مثال ۶: نام، نام خانوادگی، و میانگین نمرات هر دانشجویی که حداقل دو درس گذرانده باشد را نمایش بده.

روش ۱:

```
SELECT s.first_name, s.last_name, AVG(sc.grade) AS avg_grade
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
GROUP BY s.student_id, s.first_name, s.last_name
HAVING COUNT(sc.course_id) >= 2;
```

روش ۲:

```
SELECT s.first_name, s.last_name, t.avg_grade
FROM student s,
     (
       SELECT student_id, AVG(grade) AS avg_grade, COUNT(*) AS course_count
       FROM student_course
       GROUP BY student_id
     ) t
WHERE s.student_id = t.student_id
  AND t.course_count >= 2;
```

روش ۳:

```
SELECT first_name, last_name,
       (SELECT AVG(grade)
        FROM student_course sc
        WHERE sc.student_id = s.student_id) AS avg_grade
FROM student s
WHERE (
    SELECT COUNT(*)
    FROM student_course sc
    WHERE sc.student_id = s.student_id
) >= 2;
```

خروجی:

first_name	last_name	avg_grade
آرتین	رمضانی	۱۶/۷۵
علیرضا	جهانبخش	۱۶/۷۵
سینا	قنبری	۱۸/۷۵
محمد	صادقی	۱۶/۸۵

مثال ۷: نام، نام خانوادگی و **بیشترین نمره‌ای** که هر دانشجو **در یک درس** گرفته را نمایش بده.

روش ۱:

```sql
SELECT s.first_name, s.last_name, MAX(sc.grade) AS max_grade
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
GROUP BY s.student_id, s.first_name, s.last_name;
```

روش ۲:

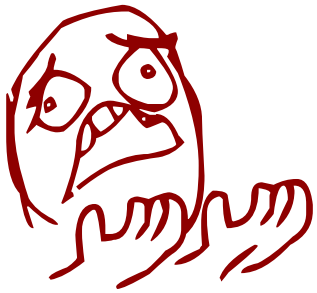
```sql
SELECT s.first_name, s.last_name, t.max_grade
FROM student s,
     (
       SELECT student_id, MAX(grade) AS max_grade
       FROM student_course
       GROUP BY student_id
     ) t
WHERE s.student_id = t.student_id;
```

روش ۳:

```sql
SELECT first_name, last_name,
       (
         SELECT MAX(grade)
         FROM student_course sc
         WHERE sc.student_id = s.student_id
       ) AS max_grade
FROM student s;
```

خروجی:

first_name	last_name	max_grade
آرتین	رمضانی	۱۷/۵
علیرضا	جهانبخش	۱۸/۵
سینا	قنبری	۱۹/۰
ریحانه	مهدوی	۱۴/۰
مهسا	کریمی	۱۷/۰
محمد	صادقی	۱۷/۲



عملگر EXISTS

عملگر EXISTS یکی از عملگرهای منطقی پیشرفته در SQL است که برای بررسی وجود داده‌ای خاص در خروجی یک زیرپرس‌وجو استفاده می‌شود. این عملگر به‌صورت بولی عمل می‌کند و تنها بررسی می‌نماید که آیا زیرپرس‌وجو حداقل یک سطر بازمی‌گرداند یا خیر. اگر زیرپرس‌وجو هیچ نتیجه‌ای برنگرداند، EXISTS مقدار FALSE را بازمی‌گرداند و در غیر این‌صورت، TRUE.

ساختار کلی:

```
SELECT ...
FROM ...
WHERE EXISTS (
    SELECT ...
    FROM ...
    WHERE ...
);
```

و برای منفی کردن نتیجه:

```
SELECT ...
FROM ...
WHERE NOT EXISTS (
    SELECT ...
    FROM ...
    WHERE ...
);
```

برخلاف IN، عملگر EXISTS معمولا برای داده‌های بزرگ و پویا عملکرد بهتری دارد، **زیرا اجرای آن می‌تواند پس از یافتن اولین ردیف متوقف شود.**

ستون‌هایی که در زیرپرس‌وجو SELECT شده‌اند اهمیتی ندارند؛ معمولاً از SELECT 1 استفاده می‌شود تا فقط ساختار حفظ شود (قراردادی رایج است بین برنامه‌نویس‌ها برای خوانایی بیشتر).

مثال ۱: نام و نام خانوادگی دانشجویانی را نمایش بده که حداقل یک درس را گذرانده‌اند.

```
SELECT first_name, last_name
FROM student s
WHERE EXISTS (
  SELECT 1
  FROM student_course sc
  WHERE sc.student_id = s.student_id
);
```

خروجی:

first_name	last_name
آرتین	رمضانی
علیرضا	جهانبخش
سینا	قنبری
ریحانه	مهدوی
مهسا	کریمی
محمد	صادقی

- برای هر دانشجو بررسی می‌شود که آیا ردیفی در جدول student_course برایش وجود دارد یا نه.
- اگر حداقل یک درس داشته باشد، EXISTS مقدار TRUE برمی‌گرداند و دانشجو در خروجی نمایش داده می‌شود.

مثال ۲: نام و نام خانوادگی دانشجویانی را نمایش بده که حداقل یک درس با نمره بالای ۱۸ گذرانده‌اند.

```
SELECT first_name, last_name
FROM student s
WHERE EXISTS (
    SELECT 1
    FROM student_course sc
    WHERE sc.student_id = s.student_id
      AND sc.grade > 18
);
```

خروجی:

first_name	last_name
علیرضا	جهانبخش
سینا	قنبری

مثال ۳: نمایش نام، نام خانوادگی دانشجویانی که درس "آمار و احتمال" (با course_id = 105) را نگذرانده‌اند.

```
SELECT first_name, last_name
FROM student s
WHERE NOT EXISTS (
    SELECT 1
    FROM student_course sc
    WHERE sc.student_id = s.student_id
      AND sc.course_id = 105
);
```

خروجی:

first_name	last_name
محمد	صادقی
ریحانه	مهدوی
سینا	قنبری

معادل‌های دیگر SQL برای **مثال ۲**:

استفاده از **IN**

```
SELECT first_name, last_name
FROM student
WHERE student_id IN (
    SELECT student_id
    FROM student_course
    WHERE grade > 18
);
```

استفاده از **INNER JOIN**

```
SELECT DISTINCT s.first_name, s.last_name
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
WHERE sc.grade > 18;
```

معادل‌های دیگر SQL برای مثال ۳

استفاده از LEFT JOIN و فیلتر کردن مقادیر NULL

```
SELECT s.first_name, s.last_name
FROM student s
LEFT JOIN student_course sc
  ON s.student_id = sc.student_id AND sc.course_id = 105
WHERE sc.student_id IS NULL;
```

استفاده از EXCEPT

```
-- همه دانشجویان
SELECT first_name, last_name
FROM student
EXCEPT
-- دانشجویانی که آمار و احتمال را گذرانده‌اند
SELECT s.first_name, s.last_name
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
WHERE sc.course_id = 105;
```

عملگر ANY

در SQL، عملگر ANY به شما این امکان را می‌دهد که یک مقدار را با مجموعه‌ای از مقادیر که توسط یک زیرپرس‌وجو برگردانده می‌شود، مقایسه کنید. **این عملگر در شرایطی که بخواهید بررسی کنید آیا یک مقدار با حداقل یکی از مقادیر موجود در یک لیست مطابقت دارد یا نه، بسیار کاربردی است.**

ساختار کلی:

```
value operator ANY (subquery)
```

در این ساختار:

- **value** مقداری است که می‌خواهید آن را مقایسه کنید. این مقدار می‌تواند یک ستون، متغیر، یا یک عبارت محاسباتی باشد.
- **Operator** عملگر مقایسه‌ای است مثل =، !=، <، >، <= یا >= که مشخص می‌کند چگونه مقایسه انجام شود.
- **subquery** یک زیرپرس‌وجو است که باید فقط **یک ستون** را برگرداند. این ستون حاوی مجموعه‌ای از مقادیر برای مقایسه با مقدار اصلی است.

نحوه‌ی عملکرد:

- عملگر ANY زمانی مقدار TRUE برمی‌گرداند که **حداقل یکی از مقایسه‌ها صحیح** باشد.
- اگر **تمام مقایسه‌ها غلط** باشند، نتیجه FALSE خواهد بود.

نکته:

در PostgreSQL، واژه‌های SOME و ANY معادل هم هستند و می‌توانید از هر کدام استفاده کنید.

مثال ۱: کدام دانشجویان درسی گذرانده‌اند که نمره‌شان بیشتر از نمره‌ی دانشجوی با شماره ۴ بوده است؟

```
SELECT DISTINCT s.first_name, s.last_name
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
WHERE sc.grade > ANY (
    SELECT grade
    FROM student_course
    WHERE student_id = 4
);
```

گام ۱: زیرپرس‌وجو اجرا می‌شود

```
SELECT grade
FROM student_course
WHERE student_id = 4;
```

و جدول student_course را فیلتر می‌کند تا فقط نمرات دانشجوی شماره ۴ را بیاورد. در جدول ما دانشجوی شماره ۴ فقط یک نمره دارد: ۱۴/۰

پس خروجی زیرپرس‌وجو:

[۱۴/۰]

گام ۲: اتصال بین student و student_course

```
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
```

این کار باعث می‌شود که به ازای هر درس گذرانده‌شده توسط هر دانشجو، داده‌های دانشجو هم در دسترس باشد. یعنی به‌ازای هر ردیف از student_course، نام و نام‌خانوادگی دانشجو نیز اضافه شود.

گام ۳: شرط WHERE با ANY

در این مرحله، برای هر ردیف از جدول ترکیبی (student + student_course)، شرط بررسی می‌شود:

- آیا sc.grade (نمره این دانشجو) در این درس **بزرگ‌تر از هر کدام از مقادیر زیرپرس‌وجو** هست؟ چون فقط ۱۴,۰ داریم، عملا می‌پرسیم:
  - آیا نمره دانشجو بزرگ‌تر از ۱۴,۰ است؟ اگر بله، این ردیف پذیرفته می‌شود.

گام ۴: انتخاب نتایج

```
SELECT DISTINCT s.first_name, s.last_name
```

حالا فقط نام و نام خانوادگی آن دانشجویانی را می‌خواهیم که در حداقل یکی از درس‌هایشان نمره‌ای بزرگ‌تر از ۱۴ گرفته‌اند. کلمه‌ی DISTINCT تضمین می‌کند که هر دانشجو فقط یک‌بار در خروجی نمایش داده شود، حتی اگر در چند درس نمره بالای ۱۴ داشته باشد.

خروجی:

first_name	last_name
مهسا	کریمی
آرتین	رمضانی
علیرضا	جهانبخش
سینا	قنبری
محمد	صادقی

مثال ۲: کدام دانشجویان حداقل یک درس گذرانده‌اند که نمره‌شان برابر با یکی از نمرات دانشجوی شماره ۲ باشد؟ (و خود دانشجوی ۲ نباشند)

```
SELECT DISTINCT s.first_name, s.last_name
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
WHERE sc.grade = ANY (
    SELECT grade
    FROM student_course
    WHERE student_id = 2
)
AND s.student_id != 2;
```

خروجی:

first_name	last_name
سینا	قنبری

اجرای زیرپرس‌وجو:

```
SELECT grade
FROM student_course
WHERE student_id = 2;
```

این بخش نمرات دانشجوی شماره ۲ را از جدول student_course می‌گیرد:

student_id	grade
2	18.5
2	15.0

پس لیست نمرات برابر با:

```
[18.5, 15.0]
```

شرط اصلی:

```
WHERE sc.grade = ANY ([18.5, 15.0])
```

برای هر دانشجو، بررسی می‌شود که آیا حداقل یکی از نمره‌هایش برابر با یکی از این دو عدد است؟ اگر بله، آن دانشجو وارد خروجی می‌شود.

حذف دانشجوی شماره ۲:

```
AND s.student_id != 2
```

برای اینکه دانشجوی اصلی سؤال، یعنی دانشجوی شماره ۲، خودش در خروجی نیاید.

معادل‌های دیگر SQL برای مثال ۲

روش ۱: استفاده از IN

```
SELECT DISTINCT s.first_name, s.last_name
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
WHERE sc.grade IN (
    SELECT grade
    FROM student_course
    WHERE student_id = 2
)
AND s.student_id != 2;
```

روش ۲: استفاده از EXISTS

```
SELECT DISTINCT s.first_name, s.last_name
FROM student s
JOIN student_course sc1 ON s.student_id = sc1.student_id
WHERE EXISTS (
    SELECT 1
    FROM student_course sc2
    WHERE sc2.student_id = 2
    AND sc1.grade = sc2.grade
)
AND s.student_id != 2;
```

در کوئری بالا:

sc1 اشاره به نمره‌های دانشجوی فعلی دارد.

sc2 اشاره به نمره‌های دانشجوی شماره ۲ دارد.

بخش اصلی: WHERE EXISTS (...)

```
WHERE EXISTS (
    SELECT 1
    FROM student_course sc2
    WHERE sc2.student_id = 2
    AND sc1.grade = sc2.grade
)
```

این بخش برای هر ردیف از sc1 بررسی می‌کند:

آیا در جدول student_course رکوردی برای دانشجوی شماره ۲ (sc2) وجود دارد که نمره‌اش با sc1.grade برابر باشد؟

اگر بله، پس EXISTS برقرار است. یعنی:

"آیا دانشجوی فعلی (sc1) نمره‌ای دارد که دانشجوی شماره ۲ هم دقیقا همان نمره را گرفته باشد؟"

روش ۳: استفاده از INTERSECT برای پیدا کردن نمرات مشترک

```
SELECT DISTINCT s.first_name, s.last_name
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
WHERE sc.grade IN (
    SELECT grade
    FROM student_course
    WHERE student_id = 2
    INTERSECT
    SELECT grade
    FROM student_course
    WHERE student_id != 2
) AND s.student_id != 2;
```

ابتدا INTERSECT پیدا می‌کند کدام نمرات بین دانشجوی شماره ۲ و بقیه مشترک‌اند، بعد دانشجویانی که این نمره‌ها را گرفته‌اند انتخاب می‌شوند.

INTERSECT بین دو مجموعه از نمرات را می‌گیرد:

مجموعه ۱: نمرات دانشجوی شماره ۲

مجموعه ۲: نمرات همه دانشجویان دیگر (student_id != 2)

پس وقتی می‌نویسیم:

```
WHERE sc.grade IN (نتیجه‌ی INTERSECT)
```

داریم بررسی می‌کنیم آیا نمره‌ی این دانشجو (sc.grade) یکی از آن نمرات مشترک هست یا نه.

عملگر ALL

در SQL، عملگر ALL به شما این امکان را می‌دهد که یک مقدار را با مجموعه‌ای از مقادیر برگردانده شده توسط یک زیرپرس‌وجو مقایسه کنید. اگر مقایسه برای همه مقادیر مجموعه درست باشد عملگر ALL مقدار true را برمی‌گرداند.

ساختار کلی:

```
value operator ALL (subquery)
```

در این ساختار:

بخش	توضیح
value	یک مقدار، ستون یا عبارت که می‌خواهید آن را مقایسه کنید
operator	یک عملگر مقایسه‌ای مثل =, <, >, <=, >=, !=
subquery	یک زیرپرس‌وجو که **یک ستون** برمی‌گرداند (یعنی یک لیست از مقادیر)

برای مثال اگر بنویسیم:

```
5 > ALL (SELECT نمره‌ها...)
```

یعنی:

- آیا ۵ بزرگ‌تر از تمام نمرات زیرپرس‌وجو هست؟
  - اگر حتی یک نمره پیدا شود که ۵ بزرگ‌تر از آن نباشد، نتیجه FALSE می‌شود.

کاربردهای رایج:

- بررسی اینکه یک مقدار از همه مقادیر یک ستون بیشتر است.
- بررسی اینکه یک مقدار با همه مقادیر برابر نیست (!= ALL(...)) یعنی هیچ‌کدام مساوی آن نیستند.
- بررسی اینکه یک دانشجو در همه امتحانات نمره‌ای بیشتر از مقدار خاصی گرفته یا نه.

تفاوت با ANY:

ویژگی	ANY	ALL
شرط	اگر برای **حتی یک مقدار** برقرار باشد، TRUE	اگر برای **همه مقادیر** برقرار باشد، TRUE
استفاده‌ی معمول	برای آزمایش "**وجود حداقل یک مورد**"	برای آزمایش "**برقرار بودن برای همه**"

مثال ۱: درس‌هایی را نمایش بده که نمره‌ی همه دانشجویان در آن بالای ۱۶ بوده است.

```
SELECT c.title
FROM course c
WHERE 16 < ALL (
    SELECT sc.grade
    FROM student_course sc
    WHERE sc.course_id = c.course_id
);
```

خروجی:

```
title
```

داده‌کاوی

یادگیری ماشین

آمار و احتمال

برای هر درس بررسی می‌شود که آیا همه‌ی دانشجویانی که آن را گذرانده‌اند نمره‌شان بالاتر از ۱۶ بوده یا نه. اگر حتی یک دانشجو نمره‌ی کمتر یا مساوی ۱۶ داشته باشد، آن درس در خروجی نخواهد بود.

معادل مثال ۱ با NOT EXISTS

```
SELECT c.title
FROM course c
WHERE NOT EXISTS (
    SELECT 1
    FROM student_course sc
    WHERE sc.course_id = c.course_id
    AND sc.grade <= 16
);
```

در این حالت می‌گوییم:
"هیچ نمره‌ای برای آن درس وجود ندارد که کمتر یا مساوی ۱۶ باشد"
یعنی همه نمرات بزرگ‌تر از ۱۶ هستند.

مثال ۲: بررسی کن آیا دانشجوی شماره ۳ در همه درس‌هایش نمره‌ای بالاتر از ۱۷ گرفته یا نه:

```
SELECT 'بله'
WHERE 17 < ALL (
    SELECT grade
    FROM student_course
    WHERE student_id = 3
);
```

اگر همه‌ی نمرات دانشجوی ۳ از ۱۷ بیشتر باشد، خروجی 'بله' خواهد بود. در غیر این صورت، خروجی‌ای نخواهیم داشت.

خروجی:

```
?column?
بله
```

مثال ۳: کدام دانشجویان همه‌ی نمرات‌شان از همه‌ی نمرات دانشجوی شماره ۵ بیشتر است؟

```
SELECT s.first_name, s.last_name
FROM student s
WHERE s.student_id != 5
AND (
    SELECT MIN(sc.grade)
    FROM student_course sc
    WHERE sc.student_id = s.student_id
) > ALL (
    SELECT grade
    FROM student_course
    WHERE student_id = 5
);
```

خروجی:

```
 first_name   last_name
 سینا         قنبری
```

- ابتدا کمترین نمره هر دانشجو را با `SELECT MIN(sc.grade) ...` می‌گیریم.
- شرط می‌گوید این کمترین نمره باید بزرگ‌تر از همه نمرات دانشجوی ۵ باشد.
- یعنی چون کمترین نمره‌اش بزرگ‌تر است، پس همه نمراتش بزرگ‌تر هستند.

عبارت جدول مشترک (Common Table Expression) یا CTE

در بخش قبل، با زیرپرس‌جو آشنا شدیم، روشی که به ما امکان می‌داد تا نتایج یک کوئری را در دل کوئری دیگری قرار دهیم. این رویکرد در بسیاری از موارد کاربردی و کارآمد است؛ اما با بزرگ‌تر شدن مسئله، زیرپرس‌جوها، به‌مرور ناخوانا، تودرتو و محدودکننده می‌شوند. برای حل این مشکل، وارد دنیای جدیدی در SQL می‌شویم؛ جایی که می‌توانیم از قدرتی به‌نام CTE یا Common Table Expression بهره ببریم. قدرتی که هم خوانایی کوئری‌ها را به‌طرز چشمگیری افزایش می‌دهد، هم کدنویسی ما را منظم‌تر و حرفه‌ای‌تر می‌کند، و هم می‌تواند محدودیت‌هایی را که با زیرپرس‌جوها روبه‌رو هستیم، برطرف نماید.

عبارت "Common Table Expression" را اگر بخواهیم به زبان ساده ترجمه کنیم، یعنی "**جدولی موقتی و مشترک که برای استفاده در کوئری اصلی تعریف می‌شود**". اما چرا "مشترک"؟ چون ما می‌توانیم یک CTE را تعریف کنیم و سپس در بخش‌های مختلف کوئری اصلی بارها از آن استفاده کنیم. انگار داریم یک جدول مجازی یا نمای موقتی از داده‌ها می‌سازیم که فقط در همان لحظه اجرا موجود است و بعد از آن از بین می‌رود. برخلاف View که در پایگاه‌داده ذخیره می‌شود، CTE هیچ ردی از خود باقی نمی‌گذارد و این ویژگی‌اش آن را به ابزاری بی‌نظیر برای تحلیل‌های موقتی، محاسبات پیچیده و تکه‌تکه کردن منطق سنگین کوئری‌ها تبدیل می‌کند.

CTE به SQL اجازه داد تا مفهومی مشابه با توابع در زبان‌های برنامه‌نویسی داشته باشد: ما یک "بخش" از منطق کوئری را جدا می‌کنیم، به آن یک نام می‌دهیم، و بعد از آن، هر کجا که خواستیم از آن استفاده می‌کنیم. این مفهوم از نظر ذهنی بسیار قوی است، چون ما را از نگرش خطی به SQL به سمت نگرش ماژولار و بلوک‌محور سوق می‌دهد.

CTE یک ابزار نیست؛ **یک طرز تفکر جدید برای طراحی کوئری‌های SQL است. یادگیری آن شما را از یک نویسنده‌ی ساده‌ی کوئری، به یک تحلیل‌گر حرفه‌ای پایگاه داده تبدیل می‌کند که می‌داند چگونه داده‌ها را به شکل ساخت‌یافته و معنادار مدیریت کند.**

آیا CTE جایگزین زیرپرس‌وجو است؟

نه همیشه، ولی در بسیاری از مواقع بهتر است.

CTE و زیرپرس‌وجو هر دو ابزارهای مهمی در SQL هستند و در برخی موارد می‌توان از هر دو استفاده کرد. اما وقتی:

- کوئری شما بیش از حد پیچیده می‌شود
- می‌خواهید کوئری‌هایتان خواناتر باشند
- یا نیاز به استفاده‌ی چندباره از یک کوئری دارید

در این شرایط، CTE بهترین انتخاب است.

نحو پایه یک CTE:

```
WITH cte_name(column_list) AS (
   -- CTE query
   SELECT ...
)
-- Main query
SELECT select_list
FROM cte_name;
```

اجزای دستور:

- کلمه‌ی کلیدی WITH برای تعریف یک CTE استفاده می‌شود. می‌توان CTE را مانند یک جدول موقتی درون کوئری در نظر گرفت.
- cte_name نامی است که برای CTE انتخاب می‌کنید. بعدا در کوئری اصلی می‌توانید این نام را درست مانند یک جدول عادی استفاده کنید.
- column_list فهرستی از نام ستون‌های CTE است که به‌صورت اختیاری و با کاما از هم جدا شده‌اند. اگر این بخش را ننویسید، CTEبه‌طور خودکار از نام ستون‌هایی که کوئری درون آن برمی‌گرداند استفاده خواهد کرد.
- کوئری CTE (بخشی که در پرانتز آمده است) ساختار CTE را تعریف می‌کند. این کوئری می‌تواند هر نوع عبارتی باشد که نتیجه‌ای بازمی‌گرداند؛ مانند SELECT، یا حتی دستوراتی مانند INSERT، UPDATE یا DELETE.
- کوئری اصلی (Main Query) کوئری‌ای است که از CTE استفاده می‌کند.

مثال ۱: نمایش نام دانشجویانی که نمره‌شان در درس "آمار و احتمال" بالای ۱۸ بوده است.

```
WITH high_score AS (
    SELECT student_id
    FROM student_course
    WHERE course_id = 105 AND grade > 18
)
SELECT s.first_name, s.last_name
FROM high_score hs
JOIN student s ON s.student_id = hs.student_id;
```

خروجی:

first_name	last_name
علیرضا	جهانبخش

بخش CTE:

```
WITH high_score AS (
    SELECT student_id
    FROM student_course
    WHERE course_id = 105 AND grade > 18
)
```

این بخش با استفاده از WITH یک CTE به نام high_score تعریف می‌کند.

- جدول student_course را بررسی می‌کند.
  - فقط سطرهایی را نگه می‌داره که:
    - مربوط به درس آمار و احتمال باشن (course_id = 105)
    - نمره‌شان بالاتر از ۱۸ باشه (grade > 18)
    - فقط student_id را از این سطرها برمی‌دارد.

نتیجه‌ی این بخش: لیستی از شناسه‌های دانشجویانی که در این درس نمره‌ی بالای ۱۸ گرفتن.

بخش اصلی کوئری:

```
SELECT s.first_name, s.last_name
FROM high_score hs
JOIN student s ON s.student_id = hs.student_id;
```

high_score را به جدول student وصل می‌کنیم.
شرط JOIN این است که student_idها با هم برابر باشند.
سپس نام (first_name) و نام خانوادگی (last_name) دانشجویان را استخراج می‌کنیم.

معادل مثال ۱ با زیرپرس‌وجو:

```
SELECT s.first_name, s.last_name
FROM student s
WHERE s.student_id IN (
    SELECT student_id
    FROM student_course
    WHERE course_id = 105 AND grade > 18
);
```

مثال ۲: میانگین نمره‌ی هر دانشجو و نام او

```
WITH avg_grades AS (
    SELECT student_id, AVG(grade) AS avg_grade
    FROM student_course
    GROUP BY student_id
)
SELECT s.first_name, s.last_name, a.avg_grade
FROM avg_grades a
JOIN student s ON s.student_id = a.student_id;
```

خروجی:

first_name	last_name	avg_grade
آرتین	رمضانی	۱۶/۷۵
علیرضا	جهانبخش	۱۶/۷۵
سینا	قنبری	۱۸/۷۵
ریحانه	مهدوی	۱۴/۰
مهسا	کریمی	۱۷/۰
محمد	صادقی	۱۶/۸۵

معادل‌های مثال ۲ با زیرپرس‌وجو

روش ۱:

```
SELECT
    first_name, last_name,
    (
        SELECT AVG(grade)
        FROM student_course sc
        WHERE sc.student_id = s.student_id
    ) AS avg_grade
FROM student s;
```

روش ۲:

```
SELECT s.first_name, s.last_name, ag.avg_grade
FROM student s
JOIN (
    SELECT student_id, AVG(grade) AS avg_grade
    FROM student_course
    GROUP BY student_id
) ag ON s.student_id = ag.student_id;
```

مثال ۳: نمایش عنوان درس‌هایی که حداقل یک دانشجوی "علوم کامپیوتر" آن‌ها را گذرانده است.

```
WITH cs_students AS (
    SELECT student_id
    FROM student
    WHERE major = 'علوم کامپیوتر'
),
cs_courses AS (
    SELECT DISTINCT sc.course_id
    FROM student_course sc
    JOIN cs_students cs ON sc.student_id = cs.student_id
)
SELECT c.title
FROM cs_courses cc
JOIN course c ON c.course_id = cc.course_id;
```

خروجی:

title
جبر خطی
داده‌کاوی
یادگیری ماشین

معادل‌های مثال ۳

روش ۱:

```
SELECT DISTINCT c.title
FROM course c
JOIN student_course sc ON c.course_id = sc.course_id
JOIN student s ON s.student_id = sc.student_id
WHERE s.major = 'علوم کامپیوتر';
```

به‌طور مستقیم **student_course** را به **student** و سپس به **course** وصل می‌کنیم.
فقط رشته‌ی تحصیلی را فیلتر می‌کنیم.
DISTINCT برای حذف تکرار عنوان دروس استفاده شده.

روش ۲:

```
SELECT DISTINCT c.title
FROM course c
WHERE EXISTS (
    SELECT 1
    FROM student_course sc
    JOIN student s ON s.student_id = sc.student_id
    WHERE sc.course_id = c.course_id AND s.major = 'علوم کامپیوتر'
);
```

روش ۳:

```
SELECT DISTINCT c.title
FROM course c
WHERE c.course_id IN (
    SELECT sc.course_id
    FROM student_course sc
    WHERE sc.student_id IN (
        SELECT student_id
        FROM student
        WHERE major = 'علوم کامپیوتر'
    )
);
```

- زیرپرس‌جو داخلی دانشجویان علوم کامپیوتر را مشخص می‌کند.
- سپس زیرپرس‌جو بیرونی درس‌هایی را برمی‌گرداند که این دانشجویان انتخاب کرده‌اند.
- در نهایت، از جدول **course** عنوان آن درس‌ها را می‌گیریم.

مثال ۴: برای هر دانشجویی که در ترم "پاییز ۱۴۰۱" درس گرفته، تعداد درس‌های انتخابی‌اش رو همراه با نام و نام خانوادگی‌اش نمایش بده.

```
WITH fall_1401 AS (
    SELECT student_id, COUNT(*) AS course_count
    FROM student_course
    WHERE term = 'پاییز ۱۴۰۱'
    GROUP BY student_id
)
SELECT s.first_name, s.last_name, f.course_count
FROM fall_1401 f
JOIN student s ON s.student_id = f.student_id;
```

خروجی:

first_name	last_name	course_count
آرتین	رمضانی	۲
سینا	قنبری	۲

معادل‌های مثال ۴

روش ۱:

```
SELECT s.first_name, s.last_name, COUNT(*) AS course_count
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
WHERE sc.term = 'پاییز ۱۴۰۱'
GROUP BY s.student_id, s.first_name, s.last_name;
```

روش ۲:

```
SELECT s.first_name, s.last_name, f.course_count
FROM student s
JOIN (
    SELECT student_id, COUNT(*) AS course_count
    FROM student_course
    WHERE term = 'پاییز ۱۴۰۱'
    GROUP BY student_id
) f ON s.student_id = f.student_id;
```

روش ۳:

```
SELECT first_name, last_name,
    (
        SELECT COUNT(*)
        FROM student_course sc
        WHERE sc.student_id = s.student_id
          AND sc.term = 'پاییز ۱۴۰۱'
    ) AS course_count
FROM student s
WHERE (
    SELECT COUNT(*)
    FROM student_course sc
    WHERE sc.student_id = s.student_id
      AND sc.term = 'پاییز ۱۴۰۱'
) > 0;
```

روش ۴:

```
SELECT DISTINCT s.first_name, s.last_name,
       (
           SELECT COUNT(*)
           FROM student_course sc
           WHERE sc.student_id = s.student_id AND sc.term = 'پاییز ۱۴۰۱'
       ) AS course_count
FROM student s
WHERE EXISTS (
    SELECT 1
    FROM student_course sc
    WHERE sc.student_id = s.student_id AND sc.term = 'پاییز ۱۴۰۱'
);
```

مثال ۵ برای نشان دادن خواناتر بودن CTE نسبت به زیرپرس‌وجو.

نام دانشجویانی را نمایش بده که:

- درسی با عنوان "آمار و احتمال" را گذرانده‌اند
- و میانگین نمراتشان از میانگین کل دانشجویان بالاتر است

مرحله ۱: محاسبه میانگین نمره کل دانشجویان

```
WITH avg_all_students AS (
    SELECT AVG(grade) AS overall_avg
    FROM student_course
),
```

مرحله ۲: میانگین نمره هر دانشجو

```
student_avg AS (
    SELECT student_id, AVG(grade) AS avg_grade
    FROM student_course
    GROUP BY student_id
),
```

مرحله ۳: انتخاب دانشجویانی که آمار و احتمال پاس کرده‌اند

```
students_with_stats AS (
    SELECT DISTINCT student_id
    FROM student_course
    WHERE course_id = 105
)
```

مرحله نهایی: فیلتر دانشجویانی که هر دو شرط را دارند

```
SELECT s.first_name, s.last_name, a.avg_grade
FROM student_avg a
JOIN students_with_stats sws ON a.student_id = sws.student_id
JOIN avg_all_students aa ON a.avg_grade > aa.overall_avg
JOIN student s ON s.student_id = a.student_id;
```

خروجی:

first_name	last_name	avg_grade
مهسا	کریمی	۱۷/۰

`avg_all_students`: فقط یک عدد تولید می‌کند: میانگین نمرات همه.

`student_avg`: میانگین نمره هر دانشجو را محاسبه می‌کند.

`students_with_stats`: فقط کسانی که درس "آمار و احتمال" را پاس کرده‌اند.

مرحله نهایی: فقط کسانی که در هر دو شرط بالا هستند را نمایش می‌دهیم.

معادل‌های مثال ۵ با زیرپرس‌وجو:

روش ۱:

```
SELECT s.first_name, s.last_name, a.avg_grade
FROM (
    SELECT student_id, AVG(grade) AS avg_grade
    FROM student_course
    GROUP BY student_id
) a
JOIN student s ON s.student_id = a.student_id
WHERE a.student_id IN (
    SELECT student_id
    FROM student_course
    WHERE course_id = 105
)
AND a.avg_grade > (
    SELECT AVG(grade) FROM student_course
);
```

روش ۲:

```
SELECT s.first_name, s.last_name, AVG(sc.grade) AS avg_grade
FROM student s
JOIN student_course sc ON s.student_id = sc.student_id
WHERE s.student_id IN (
    SELECT student_id FROM student_course WHERE course_id = 105
)
GROUP BY s.student_id, s.first_name, s.last_name
HAVING AVG(sc.grade) > (
    SELECT AVG(grade) FROM student_course
);
```

روش ۳:

```
WITH avg_all AS (
    SELECT AVG(grade) AS overall_avg FROM student_course
),
student_avg AS (
    SELECT student_id, AVG(grade) AS avg_grade FROM student_course GROUP BY
student_id
)
SELECT s.first_name, s.last_name, sa.avg_grade
FROM student_avg sa
JOIN student s ON s.student_id = sa.student_id
JOIN student_course sc ON sc.student_id = sa.student_id AND sc.course_id = 105
JOIN avg_all a ON sa.avg_grade > a.overall_avg
GROUP BY s.first_name, s.last_name, sa.avg_grade;
```

پیوست: پایگاه داده مورد استفاده در این فصل - دانلود

```
CREATE TABLE student (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    major VARCHAR(50),
    entry_year INT,
    level VARCHAR(20)
);

INSERT INTO student VALUES
(1, 'کارشناسی', ۱۴۰۰, 'آمار', 'رمضانی', 'آرتین'),
(2, 'کارشناسی', ۱۳۹۹, 'آمار', 'جهانبخش', 'علیرضا'),
(3, 'کارشناسی', ۱۴۰۰, 'علوم کامپیوتر', 'قنبری', 'سینا'),
(4, 'کارشناسی', ۱۳۹۹, 'ریاضی', 'مهدوی', 'ریحانه'),
(5, 'کارشناسی', ۱۳۹۸, 'بیم‌سنجی', 'کریمی', 'مهسا'),
(6, 'کارشناسی', ۱۴۰۱, 'علوم کامپیوتر', 'صادقی', 'محمد');

CREATE TABLE course (
    course_id INT PRIMARY KEY,
    title VARCHAR(50),
    units INT,
    professor VARCHAR(50)
);

INSERT INTO course VALUES
(101, 'داده‌کاوی', ۳, 'دکتر فراهانی'),
(102, 'یادگیری ماشین', ۳, 'دکتر خردپیشه'),
(103, 'جبر خطی', ۳, 'دکتر حجاریان'),
(104, 'ریاضی عمومی', ۳, 'دکتر طوسی'),
(105, 'آمار و احتمال', ۲, 'دکتر گنجعلی');

CREATE TABLE student_course (
    student_id INT,
    course_id INT,
    term VARCHAR(20),
    grade FLOAT,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES student(student_id),
    FOREIGN KEY (course_id) REFERENCES course(course_id)
);

INSERT INTO student_course VALUES
(1, 105, '۱۷/۵', پاییز ۱۴۰۱),
(1, 104, '۱۶/۰', پاییز ۱۴۰۱),
(2, 105, '۱۸/۵', بهار ۱۴۰۰),
(2, 103, '۱۵/۰', پاییز ۱۳۹۹),
(3, 101, '۱۹/۰', پاییز ۱۴۰۱),
(3, 102, '۱۸/۵', پاییز ۱۴۰۱),
(4, 104, '۱۴/۰', پاییز ۱۳۹۹),
(5, 105, '۱۷/۰', بهار ۱۴۰۰),
(6, 101, '۱۶/۵', پاییز ۱۴۰۲),
(6, 103, '۱۷/۲', پاییز ۱۴۰۲);
```