**گزارش کار پروژه پایانی پایگاه داده**

-یلدا جعفری 40219463

-ریحانه افشارزاده 40216403

ابزار های مورد استفاده:

* طراحی دیاگرام: drawio
* مدیریت دیتابیس و ابزار های گرافیکی: MSSQL , pgAdmin

**الف) ERD**

**نحوه‌ی مشارکت:**

* منطق طراحی: مشترک
* پیاده سازی ERD و دیاگرام ها: یلدا جعفری

در ابتدا سیستم کرایه خودرو با منطق زیر پیاده شد:

هر خودرو شامل یک کارخانه سازنده است که به آن به صورت یک رابطه‌ی چند به یک مرتبط می‌شود. هر خودرو تنها یک کارخانه سازنده دارد و هر کارخانه می‌تواند به تعداد دلخواهی خودرو مرتبط شود.

مشتریان در این سیستم می‌توانند حساب بانکی داشته باشند که اطلاعات حساب آنها را شامل می شود.

هر مشتری میتواند چندین حساب داشته باشد و هر حساب تنها متعلق به یک کاربر است.

**رزرو خودرو (Reservation)**

* هر رزرو توسط یک مشتری انجام می‌شود : رابطه‌ی یک به چند بین Customer و Reservation
* هر رزرو برای یک خودرو مشخص است رابطه‌ی یک به چند بین Car و Reservation
* بنابراین جدول Reservation شامل CustomerID و CarID است.
* ویژگی‌های مهم رزرو: تاریخ شروع (StartDate) و تاریخ پایان (EndDate).

**پرداخت‌ها (Payment)**

* هر رزرو می‌تواند چندین پرداخت داشته باشد (مثلاً قسطی یا چند مرحله‌ای).
* هر پرداخت مربوط به یک حساب از مشتری است.
* بنابراین Payment شامل ReservationID و AccountID می‌شود.
* ویژگی‌ها: مبلغ (Amount)، تاریخ پرداخت (PaymentDate).

**تعمیرات (Repairment)**

* هر خودرو ممکن است چند بار تعمیر شود.
* هر تعمیر توسط یک کارمند انجام می‌شود : رابطه‌ی چند به یک بین Repairment و Employee.
* هر تعمیر برای یک خودرو خاص است : رابطه‌ی چند به یک بین Repairment و Car.
* ویژگی‌ها: تاریخ تعمیر (RepairDate)، هزینه (Cost)

**کارمندان (Employee)**

* هر کارمند می‌تواند چند تعمیر انجام دهد.
* اطلاعات شخصی مثل نام، شماره تماس، ایمیل و تاریخ استخدام ذخیره می‌شود.

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

**ب)طراحی ساختار فیزیکی پایگاه داده**

**نحوه‌ی مشارکت:**

* ساخت جداول SQL: ریحانه افشارزاده

**Manufacture**

* دارای کلید اصلی ID از نوع INT.
* ستون Name اجباری (NOT NULL).
* ستون Country اختیاری.

**Car**

* کلید اصلی ID.
* ستون PlateNumber یکتا (UNIQUE) و غیرقابل خالی بودن.
* ستون‌های متنی برای مدل، رنگ و وضعیت.
* Year از نوع عددی INT.
* شامل کلید خارجی ManufactureID برای ارجاع به جدول Manufacture.

**Customer**

* کلید اصلی ID.
* ستون FullName اجباری.
* ستون NationalID یکتا و اجباری.
* سایر ستون‌ها مثل شماره تلفن، ایمیل و آدرس اختیاری.

**Account**

* کلید اصلی ترکیبی از AccountNumber و CustomerID.
* این ترکیب تضمین می‌کند که یک مشتری می‌تواند چند حساب داشته باشد ولی شماره حساب برای هر مشتری یکتا بماند. توضیح: در سناریو های واقعی این امکان وجود دارد که چند نفر از یک شماره کارت خرید کنند برای مثال در یک خانه اعضای خانواده از شماره کارت پدر پرداخت می‌کنند لذا هر شماره کارت برای مشتری یکتا است ولی به طور کلی یکتا نیست.
* BankName اختیاری.
* CustomerID به جدول Customer اشاره دارد.

**Reservation**

* کلید اصلی ID.
* شامل کلیدهای خارجی CustomerID و CarID.
* ستون‌های StartDate و EndDate از نوع DATE.

**Payment**

* کلید اصلی ID.
* ستون Amount از نوع (10,2)DECIMAL برای دقت مالی.
* PaymentDate از نوع DATE.
* شامل کلید خارجی ترکیبی (AccountNumber, CustomerID) که به جدول Account اشاره می‌کند.

**Employee**

* کلید اصلی ID.
* ستون FullName اجباری.
* NationalID یکتا.
* Salary از نوع (10,2)DECIMAL .
* سایر ستون‌ها مثل شماره تماس و ایمیل اختیاری.

**Repairment**

* کلید اصلی ID.
* ستون Cost از نوع (10,2)DECIMAL .
* RepairDate از نوع DATE.
* شامل کلید خارجی CarID و EmployeeID.

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

**پ) درج داده نمونه(Data Sample)**

**نحوه‌ی مشارکت:**

* درج دادگان نمونه: مشترک

ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

**ت) پیاده سازی دستورات پیشرفتهSQL**

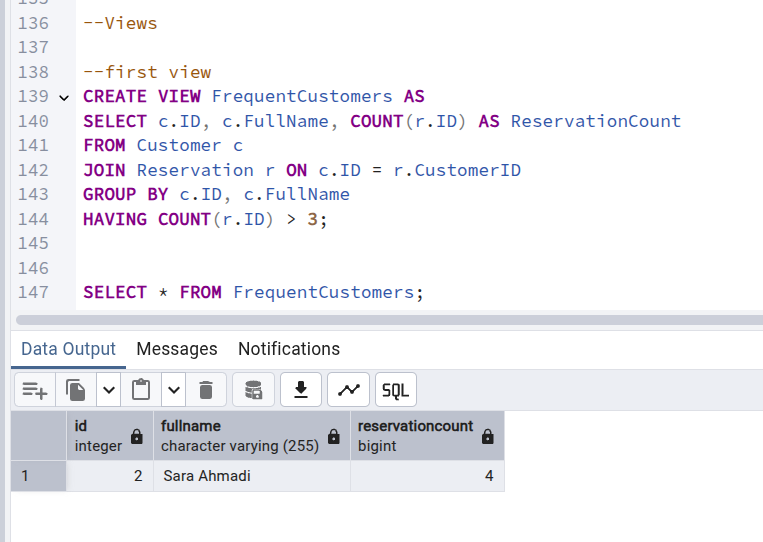
**نحوه‌ی مشارکت:**

* ویو: یلدا جعفری - ۴ مورد
* تریگر: یلدا جعفری - ۴ مورد
* فانکشن: ریحانه افشارزاده - ۴ مورد
* پروسیجر: ریحانه افشارزاده - ۴ مورد

1.**View**

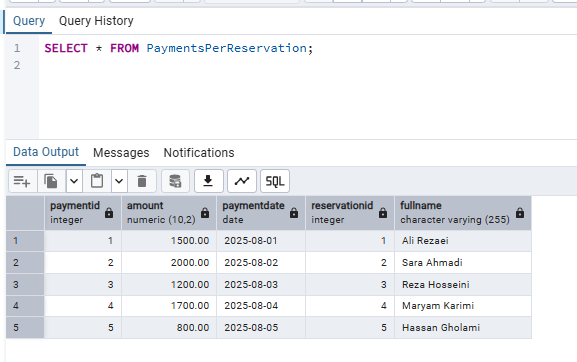
مشتریانی که بیش از سه رزرو دارند:

ویوی **FrequentCustomers** مشتریانی را نمایش می‌دهد که بیش از سه بار رزرو داشته‌اند. در این ویو با استفاده از **JOIN** بین جدول‌ مشتری و رزرو، تعداد رزروهای هر مشتری محاسبه شده و با **GROUP BY** روی شناسه و نام مشتری گروه‌بندی می‌شود. در نهایت شرط **HAVING COUNT(r.ID) > 3** باعث می‌شود فقط مشتریانی نمایش داده شوند که تعداد رزروشان بیشتر از سه است.



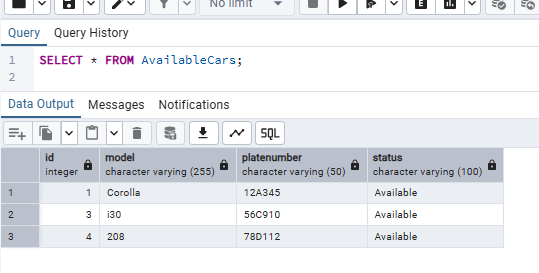
نمایش پرداختهای انجام شده برای هر رزرو:

ویوی **PaymentsPerReservation** اطلاعات پرداخت‌ها را همراه با جزئیات رزرو و نام مشتری نمایش می‌دهد. در این ویو جدول پرداخت با جدول رزرو بر اساس شناسه رزرو **JOIN** شده و سپس با جدول مشتری نیز ارتباط داده می‌شود. نتیجه شامل شناسه پرداخت، مبلغ و تاریخ پرداخت، شناسه رزرو و نام مشتری است.



نمایش خودروهای در دسترس برای رزرو:

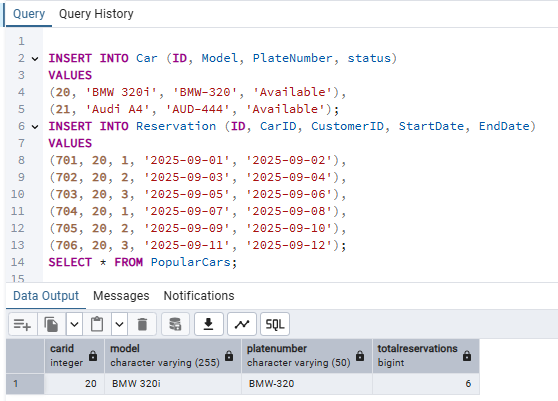
ویوی **AvailableCars** اطلاعات ماشین‌های در دسترس شامل آیدی، مدل، شماره پلاک و وضعیت ماشین‌ را نمایش می‌دهد.



نمایش خودروهای پرکاربرد (ماشین‌هایی که بیش از ۵ بار رزرو شده‌اند):

ویوی **PopularCars** اطلاعات ماشین‌هایی که بیش از 5 بار رزرو شده‌اند (ماشین‌های پرکاربرد) را نمایش می‌دهد. این اطلاعات شامل آیدی، مدل، شماره پلاک و تعداد دفعات اجاره است.

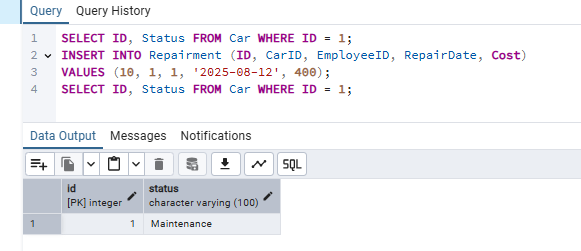
که از **JOIN** شدن جداول رزرو و ماشین این اطلاعات بدست می‌آید.



2. **Trigger**

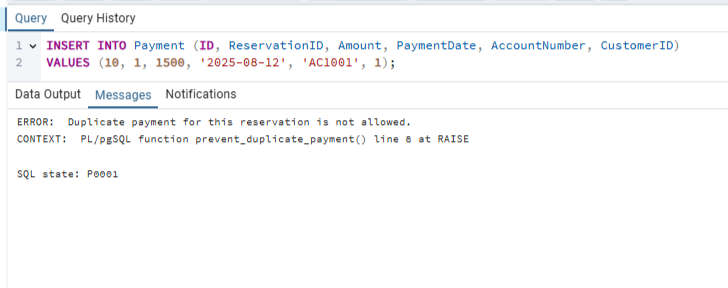
به‌روزرسانی خودکار وضعیت خودرو هنگام ثبت تعمیر جدید:

وقتی تعمیر جدیدی برای یک خودرو ثبت می‌شود، تریگر **update-car-status-on-repair** وضعیت خودرو را به صورت خودکار روی **Maintenance** قرار می‌دهد تا مشخص شود خودرو در تعمیر است و برای رزرو در دسترس نیست. **AFTER INSERT**: بعد از اینکه رکورد جدید تعمیر ثبت شد، این تریگر اجرا می‌شود.  
**FOR EACH ROW**: برای هر ردیف جدیدی که اضافه می‌شود، فانکشن اجرا می‌شود.  
**عملکرد:** شناسه خودرو (CarID) از رکورد تعمیر گرفته می‌شود و وضعیت خودرو در جدول Car به Maintenance تغییر می‌کند.



بررسی و جلوگیری از پرداخت های تکراری:

جلوگیری از ثبت پرداخت تکراری برای یک رزرو مشخص، تا هر رزرو فقط یک پرداخت داشته باشد.  
**BEFORE INSERT**: قبل از اینکه رکورد پرداخت جدید اضافه شود، تریگر اجرا می‌شود.  
**IF EXISTS:** بررسی می‌کند آیا برای همان ReservationID از قبل رکورد پرداخت موجود است یا نه.  
**RAISE EXCEPTION**: اگر پرداخت قبلاً ثبت شده باشد، یک خطا صادر می‌شود و از ثبت رکورد تکراری جلوگیری می‌شود.



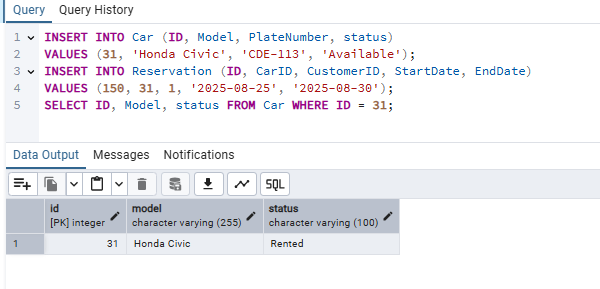
به‌روزرسانی خودکار وضعیت خودرو هنگام ثبت رزرو:

وقتی رزروی جدید برای یک خودرو ثبت می‌شود، وضعیت خودرو به صورت خودکار به **Rented** تغییر کند تا نشان دهد خودرو در حال اجاره است.

**AFTER INSERT**: بعد از ثبت رکورد رزرو جدید، اجرا می‌شود.

CarID از رزرو گرفته می‌شود و ستون status خودرو در جدول Car به Rented تغییر پیدا می‌کند.

با این کار، خودرو نمی‌تواند در همان بازه دوباره رزرو شود و وضعیت واقعی آن همواره به‌روز است.

****

بررسی و جلوگیری از ثبت رزروهای همزمان برای یک خودرو:

جلوگیری از رزرو همزمان یک خودرو در بازه‌های زمانی تداخل‌دار، تا هر خودرو در یک زمان مشخص فقط یک رزرو داشته باشد.

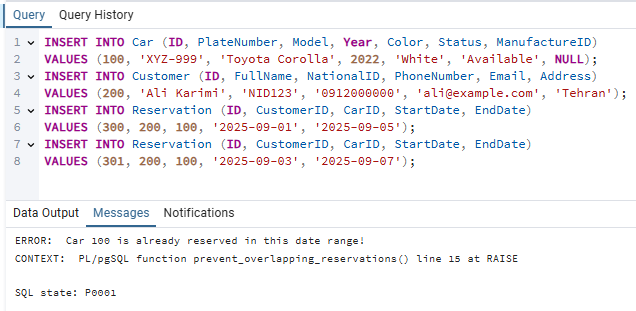
**BEFORE INSERT**: قبل از اضافه شدن رزرو جدید اجرا می‌شود.

**IF EXISTS**: بررسی می‌کند که آیا رزروی از قبل برای همان خودرو و در بازه زمانی تداخل‌دار وجود دارد یا خیر.

**شرایط تداخل:**

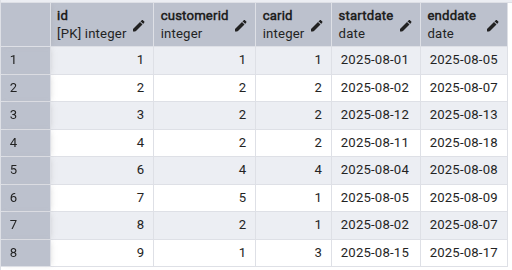
1. شروع رزرو جدید بین تاریخ شروع و پایان رزرو قبلی باشد
2. پایان رزرو جدید بین تاریخ شروع و پایان رزرو قبلی باشد
3. شروع رزرو قبلی بین تاریخ شروع و پایان رزرو جدید باشد

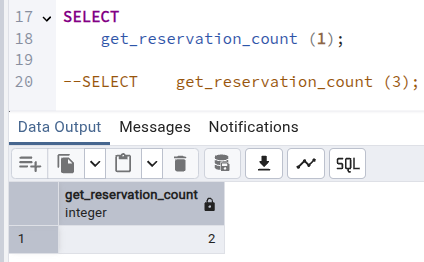
**RAISE EXCEPTION**: اگر تداخل باشد، ثبت رزرو متوقف می‌شود و پیام خطای مناسب نمایش داده می‌شود.

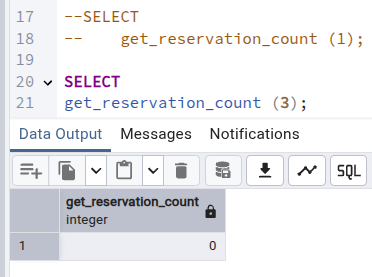


3. **Function**

محاسبه تعداد رزروهای انجامشده توسط یک مشتری:

فانکشن **get\_reservation\_count** برای محاسبه تعداد رزروهای ثبت‌شده برای یک مشتری خاص طراحی شده است. این فانکشن یک پارامتر ورودی به نام **p\_customer\_id** از نوع عدد صحیح دریافت می‌کند که نشان‌دهنده شناسه مشتری است. در داخل فانکشن، یک متغیر محلی به نام **res\_count** از نوع عدد صحیح تعریف شده است. با استفاده از دستور **SELECT COUNT(\*)**، تعداد رکوردهای موجود در جدول **Reservation** که مربوط به مشتری مشخص‌شده هستند، محاسبه و در متغیر **res\_count** ذخیره می‌شود. در نهایت، مقدار **res\_count** به عنوان خروجی فانکشن بازگردانده می‌شود.

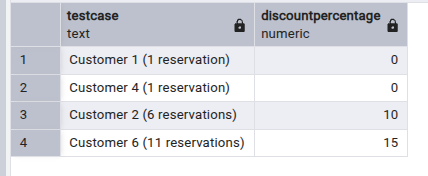




ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

-بازگرداندن درصد تخفیف بر اساس تعداد رزروها:

فانکشن **get\_discount\_percentage** برای تعیین درصد تخفیف یک مشتری بر اساس تعداد رزروهای او طراحی شده است. این فانکشن یک پارامتر ورودی به نام **p\_customer\_id** از نوع عدد صحیح دریافت می‌کند. در داخل فانکشن، دو متغیر محلی تعریف شده‌اند: **res\_count** از نوع عدد صحیح برای ذخیره تعداد رزروها و **discount** از نوع عددی برای ذخیره درصد تخفیف. ابتدا با استفاده از دستور **SELECT** **COUNT(\*)**، تعداد رزروهای مشتری از جدول **Reservation** محاسبه و در **res\_count** ذخیره می‌شود. سپس، با استفاده از ساختار شرطی **IF-ELSIF-ELSE**، درصد تخفیف بر اساس تعداد رزروها تعیین می‌شود: اگر تعداد رزروها کمتر از 3 باشد، تخفیف 0 درصد؛ بین 3 تا 5 رزرو، 5 درصد؛ بین 6 تا 10 رزرو، 10 درصد؛ و برای بیش از 10 رزرو، 15 درصد اعمال می‌شود. در نهایت، مقدار **discount** به عنوان خروجی فانکشن بازگردانده می‌شود.

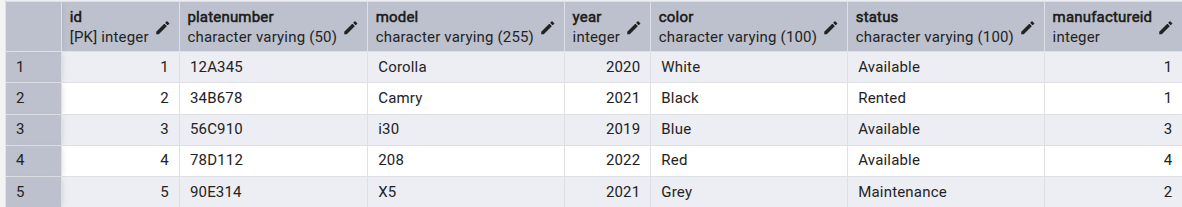


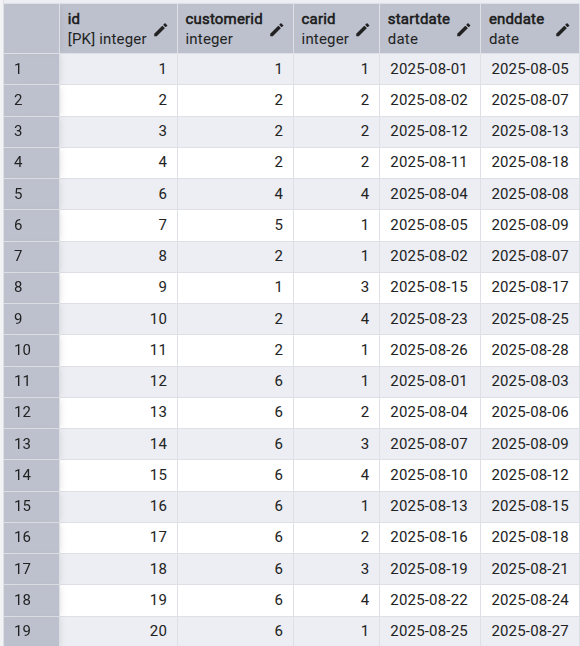
ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

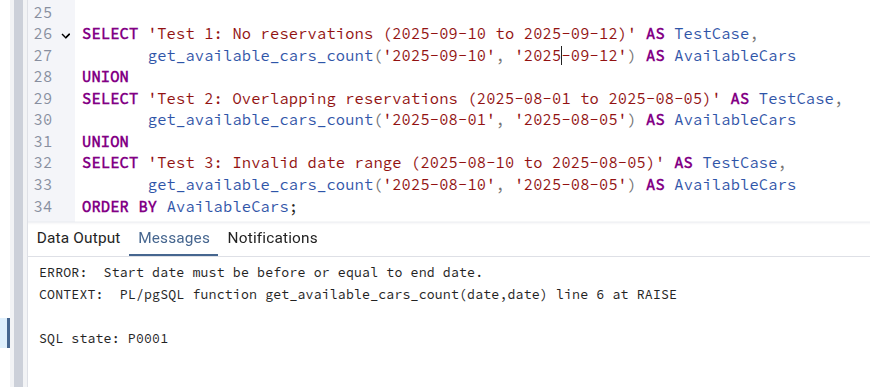
-خودرو های موجود در یک بازه‌ی زمانی

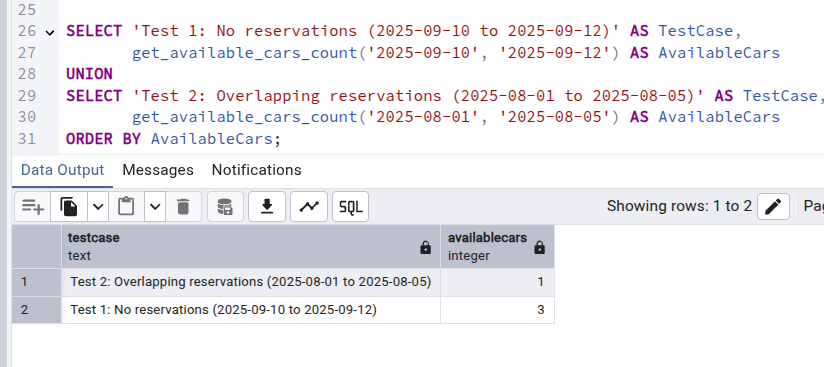
فانکشن تعداد خودروهای موجود برای رزرو در یک بازه زمانی مشخص را محاسبه می‌کند. فانکشن دو پارامتر ورودی از نوع تاریخ به نام‌های **p\_start\_date** و **p\_end\_date** دریافت می‌کند که نشان‌دهنده بازه زمانی موردنظر هستند. در داخل فانکشن، ابتدا بررسی می‌شود که آیا تاریخ شروع قبل یا برابر با تاریخ پایان است؛ در غیر این صورت، خطایی با پیام "تاریخ شروع باید قبل یا برابر با تاریخ پایان باشد" تولید می‌شود. سپس، یک متغیر محلی به نام **available\_count** از نوع عدد صحیح تعریف شده و با استفاده از کوئری **SELECT** **COUNT(\*)**، تعداد خودروهایی که در جدول Car وضعیت "**Available**" دارند و در بازه زمانی مشخص‌شده رزرو نشده‌اند (با استفاده از عملگر **NOT** **EXISTS** برای بررسی عدم همپوشانی رزروها) محاسبه می‌شود. این مقدار به عنوان خروجی فانکشن بازگردانده می‌شود.

توجه شود که این فانکشن با ویوی AvailableCars متفاوت است زیرا در اینجا ما رزرو ها را بررسی میکنیم و در آن قسمت فقط در دسترس بودن خودرو را.





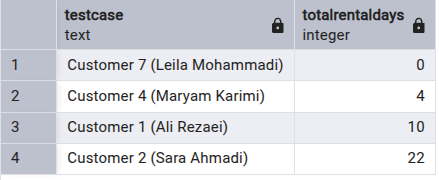




ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

#### تعداد کل روز هایی که یک مشتری خودرو کرایه کرده است:

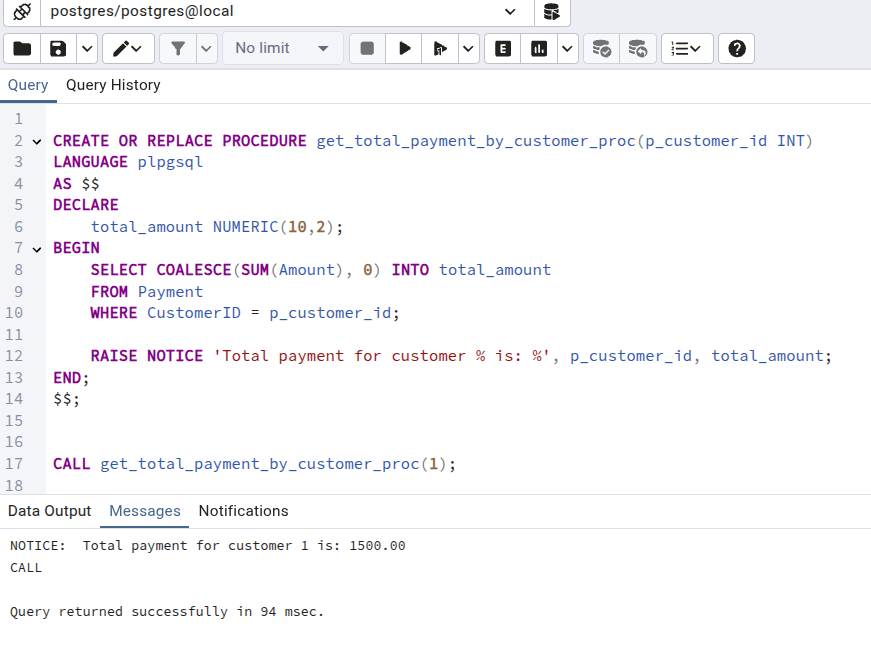
فانکشن **get\_customer\_total\_rental\_days** مجموع روزهای اجاره یک مشتری خاص را محاسبه می‌کند. فانکشن یک پارامتر ورودی به نام **p\_customer\_id** از نوع عدد صحیح دریافت می‌کند. در داخل فانkشن، یک متغیر محلی به نام **total\_days** از نوع عدد صحیح تعریف شده است. با استفاده از کوئری **SELECT** **SUM**(**EXTRACT**(**EPOCH** **FROM** (**EndDate** - **StartDate**)) / **86400**)، مجموع روزهای رزرو (تفاوت بین تاریخ پایان و شروع هر رزرو) برای مشتری مشخص‌شده از جدول **Reservation** محاسبه شده و در **total\_days** ذخیره می‌شود. تابع **COALESCE** تضمین می‌کند که اگر مشتری رزرو نداشته باشد، مقدار 0 برگردانده شود. این مقدار به عنوان خروجی فانکشن بازگردانده می‌شود.



4. **Procedure**

- محاسبه مبلغ کل پرداختی یک مشتری خاص:

پروسیجر **get\_total\_payment\_by\_customer\_proc** برای محاسبه و نمایش مجموع مبالغ پرداختی یک مشتری خاص طراحی شده است. این پروسیجر یک پارامتر ورودی به نام **p\_customer\_id** از نوع عدد صحیح دریافت می‌کند که نشان‌دهنده شناسه مشتری است. در داخل پروسیجر، یک متغیر محلی به نام **total\_amount** از نوع عددی با دقت 10 رقم و 2 رقم اعشار تعریف شده است. با استفاده از دستور **(SELECT COALESCE(SUM(Amount), 0**، مجموع مبالغ پرداختی از جدول **Payment** برای مشتری مشخص‌شده محاسبه شده و در متغیر **total\_amount** ذخیره می‌شود. تابع **COALESCE** تضمین می‌کند که در صورت نبود پرداخت برای مشتری، مقدار صفر برگردانده شود. در نهایت، با استفاده از دستور **RAISE NOTICE**، مجموع مبلغ پرداختی به همراه شناسه مشتری در قالب یک پیام نمایش داده می‌شود.

  
ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

-ثبت یک رزرو جدید با اعتبارسنجی اولیه:

پروسیجر **create\_reservation\_proc** برای ایجاد یک رزرو جدید در سیستم طراحی شده است و چهار پارامتر ورودی دریافت می‌کند: **p\_customer\_id** (شناسه مشتری)، **p\_car\_id** (شناسه خودرو)، **p\_start\_date** (تاریخ شروع رزرو) و **p\_end\_date** (تاریخ پایان رزرو)، که همگی از نوع مناسب (عدد صحیح برای شناسه‌ها و تاریخ برای بازه زمانی) هستند.

در داخل پروسیجر، ابتدا یک متغیر محلی به نام **conflicting\_count** از نوع عدد صحیح تعریف شده است. این پروسیجر ابتدا بررسی می‌کند که آیا تاریخ شروع رزرو (**p\_start\_date**) از تاریخ پایان (**p\_end\_date**) عقب‌تر است یا خیر. اگر تاریخ شروع بعد از تاریخ پایان باشد، با استفاده از دستور **RAISE EXCEPTION** خطایی با پیام "تاریخ شروع باید قبل یا برابر با تاریخ پایان باشد" تولید می‌شود و اجرای پروسیجر متوقف می‌گردد.

سپس، پروسیجر بررسی می‌کند که آیا خودرو در بازه زمانی درخواستی قبلاً رزرو شده است یا خیر. این کار با استفاده از دستور **SELECT** **COUNT(\*)** انجام می‌شود که تعداد رزروهای موجود در جدول **Reservation** را برای خودرو مشخص‌شده (**p\_car\_id**) که بازه زمانی آن‌ها با بازه درخواستی (**p\_start\_date** و **p\_end\_date**) همپوشانی دارد (با استفاده از عملگر **OVERLAPS**)، محاسبه می‌کند و نتیجه در متغیر **conflicting\_count** ذخیره می‌شود. اگر **conflicting\_count** بزرگتر از صفر باشد، خطایی با پیام "خودرو در بازه زمانی درخواستی قبلاً رزرو شده است" تولید شده و پروسیجر متوقف می‌شود.

در صورت نبود مشکل، پروسیجر یک رکورد جدید در جدول **Reservation** درج می‌کند. برای تعیین شناسه رزرو (**ID**)، از **COALESCE(MAX(ID), 0)** **+** **1** استفاده می‌شود تا بزرگ‌ترین شناسه موجود در جدول به علاوه یک به عنوان شناسه جدید انتخاب شود. سپس مقادیر **p\_customer\_id،** **p\_car\_id،** **p\_start\_date** و **p\_end\_date** به ترتیب در ستون‌های مربوطه درج می‌شوند.

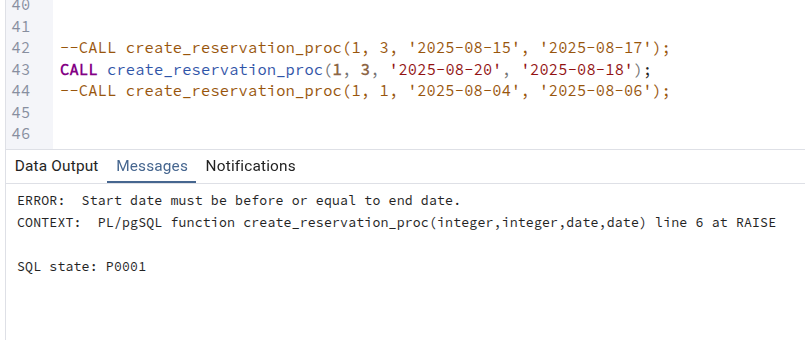
در نهایت، با استفاده از دستور **RAISE** **NOTICE،** پیامی مبنی بر "رزرو با موفقیت ایجاد شد" نمایش میدهیم.

**تست ها:**

حالتی که رزرو به درستی انجام می‌پذیرد:



حالتی که تاریخ پایان قبل از تاریخ شروع قرار دارد:



حالتی که خودرو قبلا در آن بازه زمان رزرو شده است:



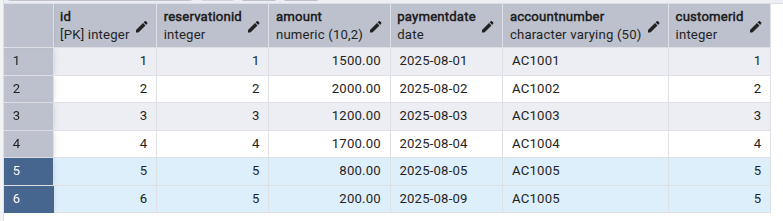
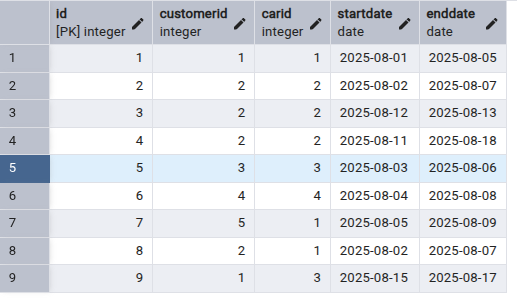
ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

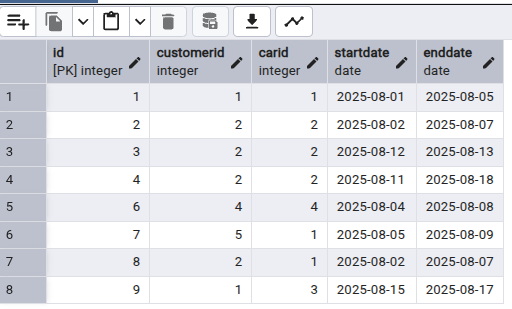
-کنسل کردن یک رزرو

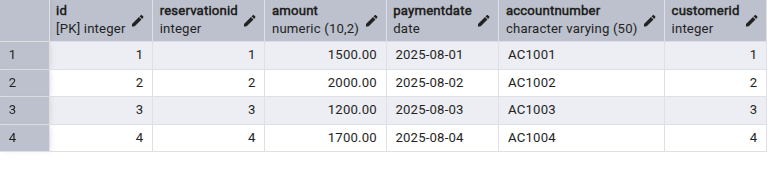
پروسیجر **cancel\_reservation\_proc** برای لغو یک رزرو موجود طراحی شده است و یک پارامتر ورودی به نام **p\_reservation\_id** از نوع عدد صحیح دریافت می‌کند که نشان‌دهنده شناسه رزرو است. در داخل پروسیجر، ابتدا بررسی می‌شود که آیا رزرو با شناسه مشخص‌شده وجود دارد یا خیر. اگر رزرو وجود نداشته باشد، با استفاده از دستور **RAISE** **EXCEPTION** خطایی با پیام "رزرو مورد نظر یافت نشد" تولید شده و پروسیجر متوقف می‌شود. در صورت وجود رزرو، رکورد مربوطه از جدول **Reservation** حذف می‌شود. همچنین، اگر پرداخت مرتبط با این رزرو وجود داشته باشد، رکورد پرداخت نیز از جدول **Payment** حذف می‌گردد تا یکپارچگی داده‌ها حفظ شود. در نهایت، با استفاده از دستور **RAISE** **NOTICE،** پیامی مبنی بر "رزرو با موفقیت لغو شد" نمایش داده می‌شود. این پروسیجر با حذف ایمن رزرو و پرداخت مرتبط، از ایجاد داده‌های ناسازگار جلوگیری کرده و مدیریت رزروها را تسهیل می‌کند.

تست ها:

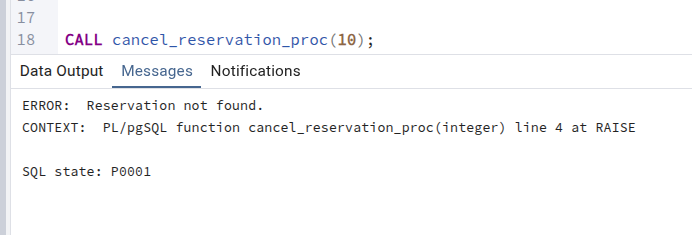
حالتی که رزرو و پرداخت های مرتبط وجود دارد:







حالتی که رزرو وجود ندارد:



ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

-محاسبه هزینه نگهداری های یک خودرو

پروسیجر **get\_car\_repair\_cost\_proc** برای محاسبه و نمایش مجموع هزینه‌های تعمیرات یک خودرو خاص طراحی شده است و یک پارامتر ورودی به نام **p\_car\_id** از نوع عدد صحیح دریافت می‌کند که نشان‌دهنده شناسه خودرو است. ابتدا بررسی می‌شود که آیا خودرویی با شناسه مشخص‌شده در جدول **Car** وجود دارد یا خیر. اگر خودرو وجود نداشته باشد، با استفاده از دستور **RAISE** **EXCEPTION** خطایی با پیام "خودرو مورد نظر یافت نشد" تولید شده و پروسیجر متوقف می‌شود. در صورت وجود خودرو، یک متغیر محلی به نام **total\_cost** از نوع عددی با دقت 10 رقم و 2 رقم اعشار تعریف شده و با استفاده از دستور **SELECT** **COALESCE**(**SUM**(**Cost**), 0 ، مجموع هزینه‌های تعمیرات از جدول **Repairment** برای خودرو مشخص‌شده محاسبه شده و در متغیر total\_cost ذخیره می‌شود. تابع **COALESCE** تضمین می‌کند که در صورت نبود تعمیر برای خودرو، مقدار صفر برگردانده شود. در نهایت، با استفاده از دستور **RAISE** **NOTICE،** مجموع هزینه تعمیرات به همراه شناسه خودرو در قالب یک پیام نمایش داده می‌شود.

تست برای ماشین موجود و ماشین غیر موجود



ــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــــ

**ث) پیاده سازی اصول طراحی منطقی**

**نحوه‌ی مشارکت:**

* نرمال سازی تا سطح سوم: ریحانه افشارزاده

در اسکیمای اولیه، جدول Account شامل ستون BankName بود که فقط به AccountNumber وابسته بود و نه به کلید مرکب (AccountNumber, CustomerID). این موضوع نشان‌دهنده‌ی وجود وابستگی جزئی و نقض 2NF بود. برای رفع این مشکل، ستون BankName به جدول مستقل Bank منتقل شد که کلید اصلی آن AccountNumber است.

با این تغییر، وابستگی‌های جزئی حذف شد (رسیدن به 2NF) و هیچ ویژگی غیرکلیدی نیز به ویژگی غیرکلیدی دیگری وابسته نیست (رسیدن به 3NF). بنابراین اسکیمای ثانویه در **سطح سوم نرمال‌سازی )3(NF** قرار دارد.