

## 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

نحوه ی 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 رابطه ی چند به یک مرتبط می شود. هر خودرو تنها یک کارخانه سازنده دارد و هر کارخانه می تواند به تعداد دلخواهی خودرو مرتبط شود. مشتریان در این سیستم می توانند 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 می شود : رابطه ی یک به چند بین Reservation و Customer
- هر رزرو برای یک خودرو مشخص است رابطه ی یک به چند بین Reservation و Car
- بنابراین جدول Reservation شامل CustomerID و CarID است.
- ویژگی های مهم رزرو: تاریخ شروع (StartDate) و تاریخ پایان (EndDate).

### پرداخت ها (Payment)

- هر رزرو می تواند چندین پرداخت داشته باشد (مثلاً قسطی یا چند مرحله ای).

- 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.

### 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 DECIMAL(10,2) 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 DECIMAL(10,2).
- سایر ستون‌ها مثل شماره تماس و ایمیل اختیاری.

### Repairment

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

---

پ) درج داده نمونه (Data Sample)

نحوه‌ی مشارکت:

- درج دادگان نمونه: مشترک
-

## ت) پیاده سازی دستورات پیشرفته SQL

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.

```
138 --first view
139 CREATE VIEW FrequentCustomers AS
140 SELECT c.ID, c.FullName, COUNT(r.ID) AS ReservationCount
141 FROM Customer c
142 JOIN Reservation r ON c.ID = r.CustomerID
143 GROUP BY c.ID, c.FullName
144 HAVING COUNT(r.ID) > 3;
145
146
147 SELECT * FROM FrequentCustomers;
```

Data Output Messages Notifications

	id integer	fullname character varying (255)	reservationcount bigint
1	2	Sara Ahmadi	4

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.

	paymentid integer	amount numeric (10,2)	paymentdate date	reservationid integer	fullname character varying (255)
1	1	1500.00	2025-08-01	1	Ali Rezaei
2	2	2000.00	2025-08-02	2	Sara Ahmadi
3	3	1200.00	2025-08-03	3	Reza Hosseini
4	4	1700.00	2025-08-04	4	Maryam Karimi
5	5	800.00	2025-08-05	5	Hassan Gholami

نمایش خودروهای در دسترس برای رزرو:

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

	id integer	model character varying (255)	platenumber character varying (50)	status character varying (100)
1	1	Corolla	12A345	Available
2	3	i30	56C910	Available
3	4	208	78D112	Available

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.

```

1
2  v INSERT INTO Car (ID, Model, PlateNumber, status)
3  VALUES
4  (20, 'BMW 320i', 'BMW-320', 'Available'),
5  (21, 'Audi A4', 'AUD-444', 'Available');
6  v INSERT INTO Reservation (ID, CarID, CustomerID, StartDate, EndDate)
7  VALUES
8  (701, 20, 1, '2025-09-01', '2025-09-02'),
9  (702, 20, 2, '2025-09-03', '2025-09-04'),
10 (703, 20, 3, '2025-09-05', '2025-09-06'),
11 (704, 20, 1, '2025-09-07', '2025-09-08'),
12 (705, 20, 2, '2025-09-09', '2025-09-10'),
13 (706, 20, 3, '2025-09-11', '2025-09-12');
14 SELECT * FROM PopularCars;
15

```

Data Output
Messages
Notifications

+
File
Dropdown
Clipboard
Dropdown
Trash
Database
Download
Line Graph
SQL

	carid integer	model character varying (255)	platenumber character varying (50)	totalreservations bigint
1	20	BMW 320i	BMW-320	6

## 2. Trigger

به روزرسانی 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.

Query Query History

```

1 SELECT ID, Status FROM Car WHERE ID = 1;
2 INSERT INTO Repairment (ID, CarID, EmployeeID, RepairDate, Cost)
3 VALUES (10, 1, 1, '2025-08-12', 400);
4 SELECT ID, Status FROM Car WHERE ID = 1;

```

Data Output Messages Notifications

	id [PK] integer	status character varying (100)
1	1	Maintenance

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.

Query Query History

```

1 INSERT INTO Payment (ID, ReservationID, Amount, PaymentDate, AccountNumber, CustomerID)
2 VALUES (10, 1, 1500, '2025-08-12', 'AC1001', 1);

```

Data Output Messages Notifications

ERROR: Duplicate payment for this reservation is not allowed.  
CONTEXT: PL/pgSQL function prevent\_duplicate\_payment() line 8 at RAISE  
SQL state: P0001

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.

Query Query History

```
1 INSERT INTO Car (ID, Model, PlateNumber, status)
2 VALUES (31, 'Honda Civic', 'CDE-113', 'Available');
3 INSERT INTO Reservation (ID, CarID, CustomerID, StartDate, EndDate)
4 VALUES (150, 31, 1, '2025-08-25', '2025-08-30');
5 SELECT ID, Model, status FROM Car WHERE ID = 31;
```

Data Output Messages Notifications

+

📄

▼

📋

▼

🗑️

🔍

⬇️

📈

SQL

	Id [PK] integer	model character varying (255)	status character varying (100)
1	31	Honda Civic	Rented

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.

Query
Query History

```

1  INSERT INTO Car (ID, PlateNumber, Model, Year, Color, Status, ManufactureID)
2  VALUES (100, 'XYZ-999', 'Toyota Corolla', 2022, 'White', 'Available', NULL);
3  INSERT INTO Customer (ID, FullName, NationalID, PhoneNumber, Email, Address)
4  VALUES (200, 'Ali Karimi', 'NID123', '09120000000', 'ali@example.com', 'Tehran');
5  INSERT INTO Reservation (ID, CustomerID, CarID, StartDate, EndDate)
6  VALUES (300, 200, 100, '2025-09-01', '2025-09-05');
7  INSERT INTO Reservation (ID, CustomerID, CarID, StartDate, EndDate)
8  VALUES (301, 200, 100, '2025-09-03', '2025-09-07');

```

Data Output
Messages
Notifications

```

ERROR: Car 100 is already reserved in this date range!
CONTEXT: PL/pgSQL function prevent_overlapping_reservations() line 15 at RAISE

SQL state: P0001

```

### 3. Function

محاسبه تعداد رزروهای انجامشده توسط یک مشتری:

فانکشن **get\_reservation\_count** برای محاسبه تعداد رزروهای ثبت شده برای یک مشتری خاص طراحی شده است. این فانکشن یک پارامتر ورودی به نام **p\_customer\_id** از نوع عدد صحیح دریافت می کند که نشان دهنده 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 مشخص شده هستند، محاسبه و در متغیر **res\_count** ذخیره می شود. در نهایت، مقدار **res\_count** به عنوان خروجی فانکشن بازگردانده می شود.

	Id [PK] integer	customerid integer	carid integer	startdate date	enddate date
1	1	1	1	2025-08-01	2025-08-05
2	2	2	2	2025-08-02	2025-08-07
3	3	2	2	2025-08-12	2025-08-13
4	4	2	2	2025-08-11	2025-08-18
5	6	4	4	2025-08-04	2025-08-08
6	7	5	1	2025-08-05	2025-08-09
7	8	2	1	2025-08-02	2025-08-07
8	9	1	3	2025-08-15	2025-08-17

```

17 SELECT
18     get_reservation_count (1);
19
20 --SELECT     get_reservation_count (3);

```

Data Output   Messages   Notifications

get_reservation_count	
integer	
1	2

```

17 --SELECT
18 --     get_reservation_count (1);
19
20 SELECT
21     get_reservation_count (3);

```

Data Output   Messages   Notifications

get_reservation_count	
integer	
1	0

-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.





	testcase text	discountpercentage numeric
1	Customer 1 (1 reservation)	0
2	Customer 4 (1 reservation)	0
3	Customer 2 (6 reservations)	10
4	Customer 6 (11 reservations)	15

#### -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 مشخص شده رزرو نشده‌اند (با استفاده از عملگر **NOT EXISTS** برای بررسی عدم همپوشانی رزورها) 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.

	id [PK] integer	platenumber character varying (50)	model character varying (255)	year integer	color character varying (100)	status character varying (100)	manufactureid integer
1	1	12A345	Corolla	2020	White	Available	1
2	2	34B678	Camry	2021	Black	Rented	1
3	3	56C910	i30	2019	Blue	Available	3
4	4	78D112	208	2022	Red	Available	4
5	5	90E314	X5	2021	Grey	Maintenance	2

	id [PK] integer 	customerid integer 	carid integer 	startdate date 	enddate date 
1	1	1	1	2025-08-01	2025-08-05
2	2	2	2	2025-08-02	2025-08-07
3	3	2	2	2025-08-12	2025-08-13
4	4	2	2	2025-08-11	2025-08-18
5	6	4	4	2025-08-04	2025-08-08
6	7	5	1	2025-08-05	2025-08-09
7	8	2	1	2025-08-02	2025-08-07
8	9	1	3	2025-08-15	2025-08-17
9	10	2	4	2025-08-23	2025-08-25
10	11	2	1	2025-08-26	2025-08-28
11	12	6	1	2025-08-01	2025-08-03
12	13	6	2	2025-08-04	2025-08-06
13	14	6	3	2025-08-07	2025-08-09
14	15	6	4	2025-08-10	2025-08-12
15	16	6	1	2025-08-13	2025-08-15
16	17	6	2	2025-08-16	2025-08-18
17	18	6	3	2025-08-19	2025-08-21
18	19	6	4	2025-08-22	2025-08-24
19	20	6	1	2025-08-25	2025-08-27

```

25
26 ✓ SELECT 'Test 1: No reservations (2025-09-10 to 2025-09-12)' AS TestCase,
27         get_available_cars_count('2025-09-10', '2025-09-12') AS AvailableCars
28 UNION
29 SELECT 'Test 2: Overlapping reservations (2025-08-01 to 2025-08-05)' AS TestCase,
30         get_available_cars_count('2025-08-01', '2025-08-05') AS AvailableCars
31 UNION
32 SELECT 'Test 3: Invalid date range (2025-08-10 to 2025-08-05)' AS TestCase,
33         get_available_cars_count('2025-08-10', '2025-08-05') AS AvailableCars
34 ORDER BY AvailableCars;

```

Data Output Messages Notifications

ERROR: Start date must be before or equal to end date.  
 CONTEXT: PL/pgSQL function get\_available\_cars\_count(date,date) line 6 at RAISE


SQL state: P0001

```

25
26 ✓ SELECT 'Test 1: No reservations (2025-09-10 to 2025-09-12)' AS TestCase,
27         get_available_cars_count('2025-09-10', '2025-09-12') AS AvailableCars
28 UNION
29 SELECT 'Test 2: Overlapping reservations (2025-08-01 to 2025-08-05)' AS TestCase,
30         get_available_cars_count('2025-08-01', '2025-08-05') AS AvailableCars
31 ORDER BY AvailableCars;

```

Data Output Messages Notifications

Showing rows: 1 to 2 		
	testcase text	availablecars integer
1	Test 2: Overlapping reservations (2025-08-01 to 2025-08-05)	1
2	Test 1: No reservations (2025-09-10 to 2025-09-12)	3

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 فانکشن، یک

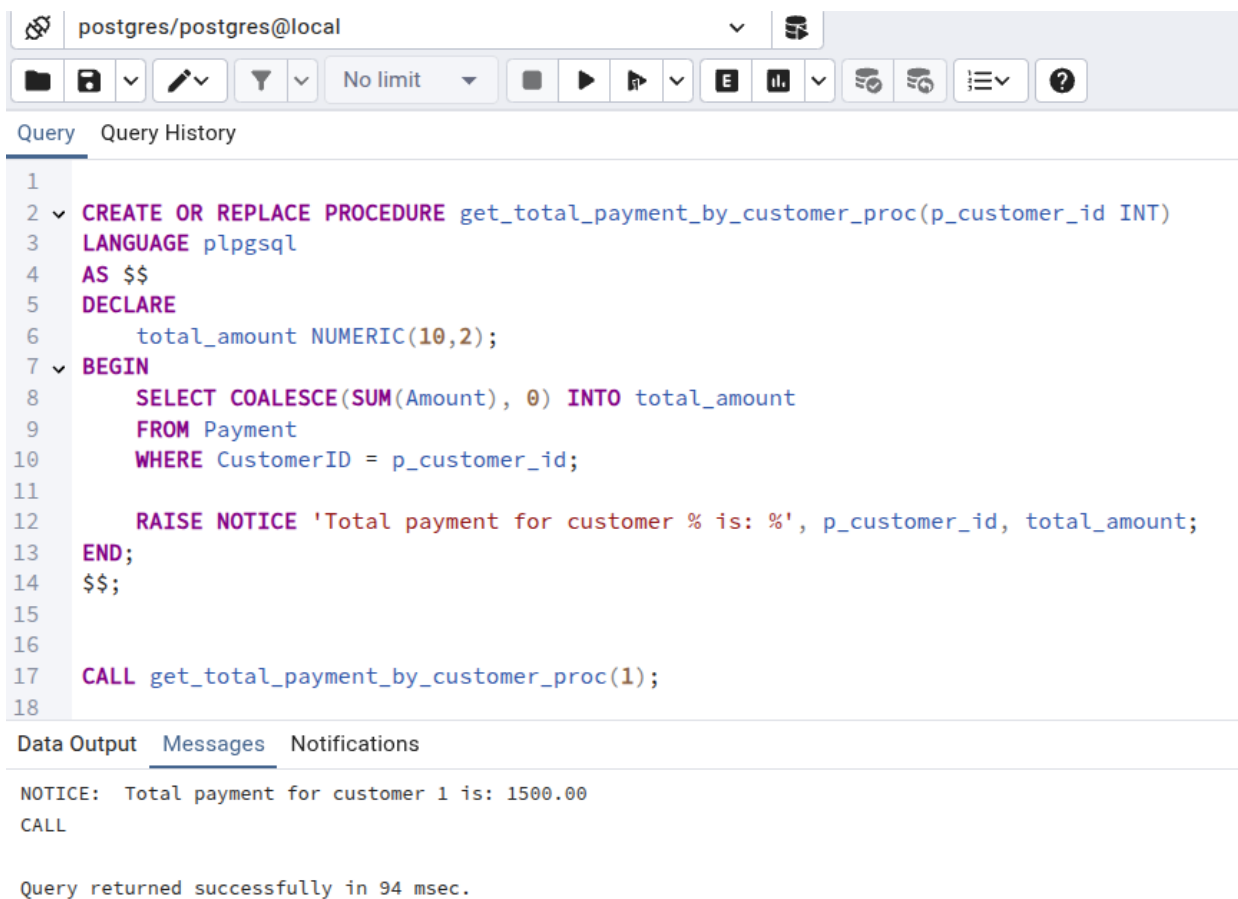
متغیر محلی به نام **total\_days** از نوع عدد صحیح تعریف شده است. با استفاده از کوئری **SELECT** **86400 / (SUM(EXTRACT(EPOCH FROM (EndDate - StartDate)))**، مجموع روزهای رزرو (تفاوت بین تاریخ پایان و شروع هر رزرو) برای مشتری مشخص شده از جدول **Reservation** محاسبه شده و در **total\_days** ذخیره می‌شود. تابع **COALESCE** تضمین می‌کند که اگر مشتری رزرو نداشته باشد، مقدار 0 برگردانده شود. این مقدار به عنوان خروجی فانکشن بازگردانده می‌شود.

	testcase text	totalrentaldays integer
1	Customer 7 (Leila Mohammadi)	0
2	Customer 4 (Maryam Karimi)	4
3	Customer 1 (Ali Rezaei)	10
4	Customer 2 (Sara Ahmadi)	22

#### 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 مشخص شده 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.



```
1
2 CREATE OR REPLACE PROCEDURE get_total_payment_by_customer_proc(p_customer_id INT)
3 LANGUAGE plpgsql
4 AS $$
5 DECLARE
6     total_amount NUMERIC(10,2);
7 BEGIN
8     SELECT COALESCE(SUM(Amount), 0) INTO total_amount
9     FROM Payment
10    WHERE CustomerID = p_customer_id;
11
12     RAISE NOTICE 'Total payment for customer % is: %', p_customer_id, total_amount;
13 END;
14 $$;
15
16
17 CALL get_total_payment_by_customer_proc(1);
18
```

Data Output   Messages   Notifications

NOTICE: Total payment for customer 1 is: 1500.00  
CALL

Query returned successfully in 94 msec.

-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 مشخص شده (**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.

تست‌ها:

حالتی که رزرو به درستی انجام می‌پذیرد:

```
41
42 CALL create_reservation_proc(1, 3, '2025-08-15', '2025-08-17');
43 --CALL create_reservation_proc(1, 3, '2025-08-20', '2025-08-18');
44 --CALL create_reservation_proc(1, 1, '2025-08-04', '2025-08-06');
45
46
```

Data Output Messages Notifications

NOTICE: Reservation created successfully.  
CALL

Query returned successfully in 36 msec.

حالتی که تاریخ پایان قبل از تاریخ شروع قرار دارد:

```

40
41
42 --CALL create_reservation_proc(1, 3, '2025-08-15', '2025-08-17');
43 CALL create_reservation_proc(1, 3, '2025-08-20', '2025-08-18');
44 --CALL create_reservation_proc(1, 1, '2025-08-04', '2025-08-06');
45
46

```

Data Output Messages Notifications

```

ERROR: Start date must be before or equal to end date.
CONTEXT: PL/pgSQL function create_reservation_proc(integer,integer,date,date) line 6 at RAISE

SQL state: P0001

```

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

```

41
42 --CALL create_reservation_proc(1, 3, '2025-08-15', '2025-08-17');
43 --CALL create_reservation_proc(1, 3, '2025-08-20', '2025-08-18');
44 CALL create_reservation_proc(1, 1, '2025-08-04', '2025-08-06');
45
46

```

Data Output Messages Notifications

```

ERROR: Car is already reserved during the requested period.
CONTEXT: PL/pgSQL function create_reservation_proc(integer,integer,date,date) line 15 at RAISE

SQL state: P0001

```

-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:

	id [PK] integer	customerid integer	carid integer	startdate date	enddate date
1	1	1	1	2025-08-01	2025-08-05
2	2	2	2	2025-08-02	2025-08-07
3	3	2	2	2025-08-12	2025-08-13
4	4	2	2	2025-08-11	2025-08-18
5	5	3	3	2025-08-03	2025-08-06
6	6	4	4	2025-08-04	2025-08-08
7	7	5	1	2025-08-05	2025-08-09
8	8	2	1	2025-08-02	2025-08-07
9	9	1	3	2025-08-15	2025-08-17

	id [PK] integer	reservationid integer	amount numeric (10,2)	paymentdate date	accountnumber character varying (50)	customerid integer
1	1	1	1500.00	2025-08-01	AC1001	1
2	2	2	2000.00	2025-08-02	AC1002	2
3	3	3	1200.00	2025-08-03	AC1003	3
4	4	4	1700.00	2025-08-04	AC1004	4
5	5	5	800.00	2025-08-05	AC1005	5
6	6	5	200.00	2025-08-09	AC1005	5

Query Query History

```
1 CREATE OR REPLACE PROCEDURE cancel_reservation_proc(p_reservation_id INT)
2 LANGUAGE plpgsql
3 AS $$
4 BEGIN
5     IF NOT EXISTS (SELECT 1 FROM Reservation WHERE ID = p_reservation_id) THEN
6         RAISE EXCEPTION 'Reservation not found.';
7     END IF;
8
9     DELETE FROM Payment WHERE ReservationID = p_reservation_id;
10
11     DELETE FROM Reservation WHERE ID = p_reservation_id;
12
13     RAISE NOTICE 'Reservation cancelled successfully.';
14 END;
15 $$;
16
17
18 CALL cancel_reservation_proc(5);
```

Data Output Messages Notifications

NOTICE: Reservation cancelled successfully.  
CALL

Query returned successfully in 43 msec.

	id [PK] integer	customerid integer	carid integer	startdate date	enddate date
1	1	1	1	2025-08-01	2025-08-05
2	2	2	2	2025-08-02	2025-08-07
3	3	2	2	2025-08-12	2025-08-13
4	4	2	2	2025-08-11	2025-08-18
5	6	4	4	2025-08-04	2025-08-08
6	7	5	1	2025-08-05	2025-08-09
7	8	2	1	2025-08-02	2025-08-07
8	9	1	3	2025-08-15	2025-08-17

	id [PK] integer	reservationid integer	amount numeric (10,2)	paymentdate date	accountnumber character varying (50)	customerid integer
1	1	1	1500.00	2025-08-01	AC1001	1
2	2	2	2000.00	2025-08-02	AC1002	2
3	3	3	1200.00	2025-08-03	AC1003	3
4	4	4	1700.00	2025-08-04	AC1004	4

حالتی که رزرو وجود ندارد:

```
17
18 CALL cancel_reservation_proc(10);
```

Data Output Messages Notifications

ERROR: Reservation not found.

CONTEXT: PL/pgSQL function cancel\_reservation\_proc(integer) line 4 at RAISE

SQL state: P0001

-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 مشخص شده 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 مشخص شده 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

```
14 CALL get_car_repair_cost_proc(5);  
15 CALL get_car_repair_cost_proc(999);
```

Data Output   Messages   Notifications

NOTICE: Total repair cost for car 5 is: 1100.00

ERROR: Car not found.

CONTEXT: PL/pgSQL function get\_car\_repair\_cost\_proc(integer) line 6 at RAISE

SQL state: P0001

## 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 NF2 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 NF2) و هیچ ویژگی غیرکلیدی نیز به ویژگی غیرکلیدی دیگری وابسته نیست (رسیدن به NF3). بنابراین اسکیمای ثانویه در سطح سوم نرمال‌سازی (NF3) قرار دارد.