
# اصول طراحی پایگاه داده

**نام استاد :دکتر پوربهمن**

**پروژه بانکداری**

**حسین زارعی نژاد ۴۰۱۳۱۴۰۹**

**پاییز ۱۴۰۳**

**مقدمه**

در فاز دوم این پروژه، هدف اصلی طراحی و پیاده‌سازی قابلیت‌های پیشرفته پایگاه داده و آزمایش عملی آن‌ها بوده است. این فاز شامل اجرای موارد زیر است:

- **پیاده‌سازی عملیات:CRUD**
  - طراحی و پیاده‌سازی متدهایInsert ، Select، Updateو Delete در پایگاه داده، به منظور مدیریت داده‌ها با استفاده از زبان‌های برنامه‌نویسی استاندارد.
- **اجرای نمونه کدهای:CRUD**
  - تست کدهای عملیاتی برای بررسی عملکرد متدهای CRUD و اطمینان از صحت عملکرد آن‌ها، مانند عملیاتSelect ، Insertو.Update
- **کوئری‌های پیشرفته:**
  - طراحی و اجرای کوئری‌هایی جهت استخراج و تحلیل داده‌های ذخیره‌شده، به‌منظور پشتیبانی از تصمیم‌گیری‌های داده‌محور.
- **محدودیت‌های کاربری:**
  - تعریف سطوح دسترسی برای کاربران مختلف پایگاه داده، به‌گونه‌ای که برخی کاربران فقط مجاز به خواندن داده‌ها باشند و از تغییر آن جلوگیری شود.
- **ایجادTrigger ها:**
  - تعریف رویدادهای خودکار جهت اطمینان از جامعیت و هماهنگی داده‌ها در هنگام انجام عملیات خاص.
- **ایجاد توابع:**
  - طراحی و استفاده از توابع سفارشی به منظور ساده‌سازی و بهبود عملیات پیچیده.
- **ایجادView ها:**
  - طراحیView هایی برای ارائه ساختار ساده‌تر و قابل فهم‌تر از داده‌های پایگاه داده به کاربران.

## ایجاد و تست CRUD ها:

هدف این بخش از پروژه، پیاده‌سازی و آزمایش متدهای مدیریت داده‌ها (CRUD) در پایگاه داده است. این عملیات شامل **Insert**، **Select**، **Update** و **Delete** می‌شود. در این مرحله، تمرکز بر پیاده‌سازی متدهایی داینامیک و انعطاف‌پذیر است که امکان تعامل آسان و کارآمد با پایگاه داده را فراهم کنند.

در اینجا ما با استفاده از زبان پایتون،متد های ذکر شده را ساختیم و در فایل db_operations.py قرار داده ایم و در فایل main.py فراخوانی کرده ایم.

```python
import psycopg2
from psycopg2 import sql


class DatabaseOperations:
    def __init__(self, connection):
        self.connection = connection

    def insert(self, table: str, queries: dict):···

    def update(self, table: str, updates: dict, conditions: dict):···

    def delete(self, table: str, conditions: dict):···

    def select(self, table: str, columns: list, conditions: dict):···

```

سپس عملیات های تست را پیاده سازی کردیم که با توجه به خروجی جداول،درست انجام میشد.

## اجرای کوئری‌هایSQL

در این بخش از پروژه، تمرکز بر تمرین مهارت‌های کوئری‌نویسی SQL برای مدیریت و استخراج اطلاعات از پایگاه داده است. هدف این بخش، آشنایی عملی با مفاهیم کوئری‌نویسی و بهبود تسلط بر استفاده از دستورات SQLاست.

**بخش اجباری**

1. **اضافه کردن فرد جدید** :توضیح دهید که چگونه یک فرد جدید در جدول Personبا مشخصات دلخواه وارد می‌شود.
2. **افزودن حساب جدید** :نحوه افزودن حساب برای فردی موجود در جدول Accountرا توضیح دهید.
3. **انتخاب تراکنش‌ها** :نمایش و انتخاب تمام تراکنش‌های مربوط به یک حساب خاص از جدول Transaction.
4. **نمایش وام‌های فعال** :نمایش تمام وام‌های فعال از جدول .Loan
5. **نمایش حساب‌های با موجودی مشخص** :انتخاب حساب‌هایی که موجودی آنها از یک مقدار مشخص بیشتر است.
6. **محاسبه موجودی کل حساب‌ها** :نحوه محاسبه و نمایش موجودی کل حساب‌های هر فرد و تفکیک حساب‌ها بر اساس نوع آنها.
7. **نمایش اطلاعات وام‌ها** :انتخاب نام، نام خانوادگی و مقدار وام کارمندانی که وام‌های فعال دارند.
8. **نمایش تعداد حساب‌ها** :نمایش نام، نام خانوادگی و تعداد حساب‌های مشتریانی که بیش از یک حساب دارند.

---

**بخش امتیازی**

1. **نمایش مشتریان با بیشترین وام‌های فعال** :توضیح نحوه انتخاب مشتریانی که بیشترین تعداد وام‌های فعال را دارند.
2. **انتخاب وام با کمترین تعداد اقساط پرداخت‌نشده** :کوئری انتخاب وامی که کمترین تعداد اقساط پرداخت‌نشده را دارد.
3. **اطلاعات وام‌های ناتمام** :نمایش نام مشتری، شناسهٔ وام و مقدار وامی که مشتریان هنوز پرداخت نکرده‌اند.

4. **نمایش موجودی‌های بالا** :انتخاب نام و نام خانوادگی ۵ مشتری با بیشترین موجودی در حساب‌های خود.

همه این کوییری ها اجرا شده اند و در فایل queries.sql قرار داده شده اند.

```sql
--اضافه کردن یک فرد جدید در جدول Person:
▷ Run
INSERT INTO Person (FirstName, LastName, DateOfBirth, PhoneNumber, Email, Address)
VALUES ('Ali', 'Rezaei', '1990-03-15', '09123456789', 'ali.rezaei@example.com', 'Tehran, Iran');

--افزودن یک حساب جدید برای یک فرد موجود در جدول Account:
▷ Run | + Tab | JSON
SELECT CustomerID FROM Customer WHERE PersonID = [PersonID];--بادی پر کنیم

▷ Run
INSERT INTO Account (OwnerID, AccountNumber, AccountType, Balance, DateOpened, Status)
VALUES ([CustomerID], '123456789012', 'Checking', 1000, CURRENT_DATE, 'Active');--باید پر کنیم جاشو


--انتخاب و نمایش تمام تراکنش‌های مرتبط با یک حساب خاص از جدول Transaction:
▷ Run | + Tab | JSON
SELECT * FROM Transaction WHERE SourceID = [AccountID] OR DestinationID = [AccountID];--اکانت ایدی میزاریم اونجا


--نمایش تمام وام‌های فعال از جدول Loan:
▷ Run | + Tab | JSON
SELECT * FROM Loan WHERE Status = 'Active';


--مایش تمام حساب‌هایی که موجودی آنها از یک مقدار مشخص بیشتر است:
▷ Run | + Tab | JSON
SELECT * FROM Account WHERE Balance > 5000;


--محاسبه مجموع کل حساب‌های هر فرد و نمایش نام و نام خانوادگی صاحب حساب به تفکیک نوع حساب:
▷ Run | + Tab | JSON | Select
SELECT P.FirstName, P.LastName, A.AccountType, SUM(A.Balance) AS TotalBalance
FROM Person P
JOIN Customer C ON P.PersonID = C.PersonID
JOIN Account A ON C.CustomerID = A.OwnerID
GROUP BY P.FirstName, P.LastName, A.AccountType;
```

```sql
--انتخاب نام، نام خانوادگی و مقدار وام کارمندانی که دارای وام‌های فعال هستند:
▷ Run | + Tab | JSON | Select
SELECT P.FirstName, P.LastName, L.LoanAmount
FROM Person P
JOIN Employee E ON P.PersonID = E.PersonID
JOIN Loan L ON E.EmployeeID = L.ApplicantID
WHERE L.Status = 'Active';


--نمایش نام، نام خانوادگی و تعداد حساب‌هایی که مشتریانی با بیش از یک حساب دارند:
▷ Run | + Tab | JSON | Select
SELECT P.FirstName, P.LastName, COUNT(A.AccountID) AS NumberOfAccounts
FROM Person P
JOIN Customer C ON P.PersonID = C.PersonID
JOIN Account A ON C.CustomerID = A.OwnerID
GROUP BY P.FirstName, P.LastName
HAVING COUNT(A.AccountID) > 1;

-----------------------------------------------------BOUNES-----------------------------------------------

--نمایش مشتریانی که بیشترین تعداد وام‌های فعال را دارند:
▷ Run | + Tab | JSON | Select
SELECT P.FirstName, P.LastName, COUNT(L.LoanID) AS ActiveLoans
FROM Person P
JOIN Customer C ON P.PersonID = C.PersonID
JOIN Loan L ON C.CustomerID = L.ApplicantID
WHERE L.Status = 'Active'
GROUP BY P.FirstName, P.LastName
ORDER BY ActiveLoans DESC
LIMIT 1;
```

```sql
--انتخاب وامی که کمترین تعداد اقساط پرداخت‌شده را داشته است:
SELECT L.LoanID, COUNT(LP.LoanPaymentID) AS PaidInstallments
FROM Loan L
LEFT JOIN LoanPayment LP ON L.LoanID = LP.LoanID AND LP.PaidDate IS NOT NULL
GROUP BY L.LoanID
ORDER BY PaidInstallments ASC
LIMIT 1;



--نمایش نام، نام خانوادگی، شناسه وام و مقدار وام مشتریانی که اقساط وام خود را به موقع پرداخت نکرده‌اند:
SELECT P.FirstName, P.LastName, L.LoanID, L.LoanAmount
FROM Person P
JOIN Customer C ON P.PersonID = C.PersonID
JOIN Loan L ON C.CustomerID = L.ApplicantID
JOIN LoanPayment LP ON L.LoanID = LP.LoanID
WHERE LP.ScheduledPaymentDate < LP.PaidDate;


--نمایش نام، نام خانوادگی و موجودی ۵ مشتری که بالاترین موجودی را در حساب‌های خود دارند:
SELECT P.FirstName, P.LastName, SUM(A.Balance) AS TotalBalance
FROM Person P
JOIN Customer C ON P.PersonID = C.PersonID
JOIN Account A ON C.CustomerID = A.OwnerID
GROUP BY P.FirstName, P.LastName
ORDER BY TotalBalance DESC
LIMIT 5;
```

## ایجاد view:

در این بخش باید این ویو ها را پیاده سازی کنیم:

- **customer_accounts view:** نمایش اطلاعات مشتریان همراه با حساب‌ها، شامل:
  - اطلاعات شخصی مشتری (نام، نام خانوادگی، شماره تماس).
  - شماره حساب.
  - نوع حساب.
  - موجودی حساب.
- **bank_transactions view:** اطلاعات مربوط به تراکنش‌ها، شامل:
  - نام بانک.
  - شناسه تراکنش.
  - شماره حساب مبدأ و مقصد.
  - مبلغ تراکنش.

- تاریخ تراکنش.

- **bank_member view:** نمایش اطلاعات کارکنان و مشتریان بانک، شامل:
  - نام بانک.
  - مشخصات شخصی (نام، نام خانوادگی، شناسه شخص).
  - نقش فرد در بانک (کارمند یا مشتری).
  - اطلاعات تماس (ایمیل، شماره تلفن).

در فایل views.sql قرار دادیم:

```sql
--customer_accounts view
CREATE VIEW customer_accounts AS
SELECT
    P.FirstName AS CustomerFirstName,
    P.LastName AS CustomerLastName,
    P.PhoneNumber AS CustomerPhone,
    A.AccountNumber AS AccountNumber,
    A.AccountType AS AccountType,
    A.Balance AS AccountBalance
FROM
    Person P
JOIN Customer C ON P.PersonID = C.PersonID
JOIN Account A ON C.CustomerID = A.OwnerID;


--bank_transactions view
CREATE VIEW bank_transactions AS
SELECT
    T.TransactionID AS TransactionID,
    A1.AccountNumber AS SourceAccount,
    A2.AccountNumber AS DestinationAccount,
    T.Amount AS TransactionAmount,
    T.TransactionDate AS TransactionDate
FROM
    Transaction T
JOIN Account A1 ON T.SourceID = A1.AccountID
JOIN Account A2 ON T.DestinationID = A2.AccountID;
```

```sql
--bank_member view
CREATE VIEW bank_member AS
SELECT
    P.FirstName AS MemberFirstName,
    P.LastName AS MemberLastName,
    CASE
        WHEN E.EmployeeID IS NOT NULL THEN 'Employee'
        WHEN C.CustomerID IS NOT NULL THEN 'Customer'
        ELSE 'Unknown'
    END AS Role,
    P.Email AS MemberEmail,
    P.PhoneNumber AS MemberPhone
FROM
    Person P
LEFT JOIN Employee E ON P.PersonID = E.PersonID
LEFT JOIN Customer C ON P.PersonID = C.PersonID;
```

## تعریف محدودیت‌های دسترسی در پایگاه داده

یکی از جنبه‌های مهم در طراحی پایگاه داده، ایجاد محدودیت‌های دسترسی برای کاربران است. این محدودیت‌ها با هدف حفاظت از داده‌ها، اطمینان از امنیت کاربران و جلوگیری از دسترسی‌های غیرمجاز طراحی می‌شوند. در این پروژه، تمرکز اصلی بر پیاده‌سازی اصول دسترسی مبتنی بر نقش (Role-Based Access Control)بود.

**هدف این بخش:**

1. ایجاد کاربری جدید با دسترسی‌های محدود.
2. تعریف نقش مشخص برای کاربر.
3. اعطای مجوزهای لازم برای انجام عملیات خاص (مانند خواندن داده‌ها) و محدود کردن دسترسی به سایر عملیات (مانند نوشتن و حذف داده‌ها).

**جزئیات پیاده‌سازی:**

- در این مرحله، کاربری با نام **William1939**و رمز عبور **Blazkowicz**ایجاد شد.
- این کاربر صرفاً اجازه انجام عملیات **Read**بر روی تمام جداول و **View**ها را دارد.
- دسترسی‌های **Write**، **Update**، **Delete**از این کاربر گرفته شده است تا از دسترسی غیرمجاز به داده‌ها جلوگیری شود.

**مراحل پیاده‌سازی:**

1. **ایجاد کاربر جدید** :با استفاده از دستوراتSQL ، کاربر مورد نظر در سیستم تعریف شد و یک رمز عبور ایمن به آن اختصاص داده شد.
2. **اعطای مجوزهای لازم:**
    - دسترسی‌های پیش‌فرض از کاربر حذف شد.
    - دسترسی **SELECT**به تمام جداول و **View**ها به کاربر اعطا شد.
    - اطمینان حاصل شد که کاربر دسترسی **INSERT**، **UPDATE**و **DELETE** ندارد.

### 3. تست محدودیت‌ها:

- تلاش برای اجرای کوئری‌های نوشتن و حذف داده‌ها با این کاربر انجام شد و سیستم به درستی این عملیات‌ها را رد کرد.
- کوئری‌های خواندن داده‌ها با موفقیت اجرا شدند.

در فایل access.sql قرار دادیم:

```
-- ایجاد کاربر
▷ Run
CREATE USER "William1939" WITH PASSWORD 'Blazkowicz';


-- لغو تمام دسترسی‌های پیشفرض
▷ Run
REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA public FROM William1939;
💡
-- اعطای دسترسی فقط خواندن
▷ Run
GRANT SELECT ON ALL TABLES IN SCHEMA public TO William1939;

-- اطمینان از دسترسی SELECT برای جداول جدید
▷ Run
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO William1939;
```

تست نهایی:

```
PS C:\Users\Hossein> docker exec -it db_test psql -U William1939 -d postgres
psql (14.15)
Type "help" for help.

postgres=> SELECT * FROM Person;
 personid | lastname  | firstname | dateofbirth | phonenumber |           email            |    address
----------+-----------+-----------+-------------+-------------+----------------------------+---------------
        1 | Mills     | David     | 1985-11-03  | 214555123   | davidmills8@gmail.com      | 12 Main St
        2 | Doe       | John      | 1990-01-01  | 1234567890  | johndoe@example.com        | 123 Main St
        3 | Smith     | Jane      | 1988-05-15  | 9876543210  | janesmith@example.com      | 456 Elm St
        4 | Rezaei    | Ali       | 1992-03-22  | 09121234567 | ali.rezaei@example.com     | Tehran, Iran
        5 | Ahmadi    | Sara      | 1980-07-19  | 09129876543 | sara.ahmadi@example.com    | Isfahan, Iran
        6 | Brown     | Michael   | 1982-04-12  | 09134567890 | michael.brown@example.com  | 456 Market St


postgres=> INSERT INTO Person (FirstName, LastName, DateOfBirth, PhoneNumber, Email, Address)
postgres-> VALUES ('Test', 'User', '2000-01-01', '1234567890', 'test.user@example.com', 'Test Address');
ERROR:  permission denied for table person
```

میبینیم که دستور select را به درستی انجام میدهد ولی بدلیل محدودیت های موجود،نمی تواند دستور insert را انجام دهد.

## در این بخش به پیاده سازی تریگر ها و توابع می پردازیم:

**تریگر های مورد نیاز:**

ثبت تاریخ ایجاد حساب جدید

جلوگیری از حذف مشتری با وام‌های فعال

به‌روزرسانی موجودی حساب پس از انجام تراکنش

بررسی موجودی کافی قبل از انجام تراکنش

**توابع مورد نیاز:**

محاسبه موجودی کل حساب‌های یک مشتری

بررسی وضعیت یک وام خاص

محاسبه تعداد وام‌های فعال یک مشتری

محاسبه مجموع پرداخت‌های انجام‌شده برای یک وام

دریافت نام مشتری بر اساس شناسه

همه این ها در فایل های triggers.sql و functions.sql نوشته شده اند.