گزارش‌کار جلسه ۶

سوال ۱) در این سوال باید جدول‌ها را با کمک ERD بکشیم؛

یک جدول به نام Employee داریم با مشخصات زیر:

```
CREATE TABLE Employee (
    employee_id INT PRIMARY KEY NOT NULL,
    dependent_name VARCHAR(50),
    employment_length INT,
    start_date DATE,
    telephone_number VARCHAR(20),
    employee_name VARCHAR(50),
    manager_id INT FOREIGN KEY REFERENCES Employee(employee_id)
)
```

یک جدول به نام Customer داریم که رابطه‌ی آن با Employee به صورت Many-to-One است و دارای مشخصات زیر است:

```
CREATE TABLE Customer (
    customer_id INT PRIMARY KEY NOT NULL,
    customer_name VARCHAR(50),
    customer_street VARCHAR(50),
    customer_city VARCHAR(50) ,
    employee_id INT FOREIGN KEY REFERENCES Employee(employee_id)
)
```

یک جدول به نام Branch داریم با مشخصات زیر:

```
CREATE TABLE Branch (
    branch_name VARCHAR(50) PRIMARY KEY NOT NULL,
    branch_city VARCHAR(50) NOT NULL,
    assets INT NOT NULL
)
```

یک جدول به نام Loan داریم که رابطه‌ی آن با Branch و Customer به صورت Many-to-One است و دارای مشخصات زیر است:

```
CREATE TABLE Loan (
    loan_number INT PRIMARY KEY NOT NULL,
    amount INT,
    branch_name VARCHAR(50) FOREIGN KEY REFERENCES Branch(branch_name),
    customer_id INT FOREIGN KEY REFERENCES Customer(customer_id)
)
```

یک جدول به نام Payment داریم که رابطه‌ی آن با Loan به صورت Many-to-One است و دارای مشخصات زیر است:

```
CREATE TABLE Payment (
    payment_number INT PRIMARY KEY NOT NULL,
    payment_date DATE,
    payment_amount INT,
    loan_number INT FOREIGN KEY REFERENCES Loan(loan_number)
)
```

جدول‌های زیر انواع Account هستند که به نوعی ارث‌بری در آن‌ها دیده می‌شود:

```
CREATE TABLE Account (
    account_number INT PRIMARY KEY NOT NULL,
    balance INT
)

CREATE TABLE SavingAccount (
    interest_rate INT,
    account_number INT FOREIGN KEY REFERENCES Account(account_number)
)

CREATE TABLE CheckingAccount (
    overdraft_amount INT,
    account_number INT FOREIGN KEY REFERENCES Account(account_number)
)
```

و در نهایت هم یک جدول به نام Depositor داریم با مشخصات زیر:

```
CREATE TABLE Depositor (
    depositor_id INT PRIMARY KEY NOT NULL,
    customer_id INT FOREIGN KEY REFERENCES Customer(customer_id),
    account_number INT FOREIGN KEY REFERENCES Account(account_number),
    access_date DATE
)
```

در انتها نیز این جداول را با تعدادی داده پر می‌کنیم.

تمامی این دستورات sql به پیوست ارسال می‌گردند.

سوال ۲) در این سوال نیز پرس و جوهای مورد نظر صورت گرفته اند:

الف) در اینجا ویویی از گیرندگان وام داریم:

```
-- CREATE VIEW borrower_view AS
--SELECT Customer.customer_name, Customer.customer_id, Loan.amount, Loan.branch_name
--FROM Customer JOIN Loan ON Loan.customer_id = Customer.customer_id

SELECT * FROM borrower_view;
```

100 %

Results | Messages

	customer_name	customer_id	amount	branch_name
1	Hasan	1	400	Vanak

ب) توجه شود که در پرس و جوی زیر قسمت سود به گفته استاد حذف شد.

```
SELECT Account.account_number, balance, access_date
FROM Account JOIN Depositor ON Depositor.account_number = Account.account_number
WHERE Depositor.access_date >=  '2009-01-01';
```

100 %

Results | Messages

	account_number	balance	access_date
1	1	100	2017-02-21
2	2	150	2018-02-21

ج) شماره بازپرداخت وام‌های تهران

```
SELECT payment_number
FROM Payment JOIN Loan ON Payment.loan_number = Loan.loan_number
JOIN Branch on Loan.branch_name = Branch.branch_name
WHERE Branch.branch_city = 'Tehran'
```

100 %

Results | Messages | Client Statistics

	payment_number
1	1

سوال ۲ دومی)

الف) در اینجا هم شماره حساب و بالانس را به کمک SP به دست می‌آوریم:

```
-- CREATE PROCEDURE get_customer_balance
-- @name VARCHAR(50)
-- AS
-- SELECT Account.account_number, Account.balance
-- FROM Customer JOIN Depositor ON Depositor.customer_id = Customer.customer_id
-- JOIN Account ON Depositor.account_number = Account.account_number
-- WHERE Customer.customer_name = @name

EXEC get_customer_balance 'Hasan'
```

100 %

Results | Messages | Client Statistics

	account_number	balance
1	1	100

ب) در اینجا هم نام شعبه‌ی ورودی SP را خروجی می‌دهیم:

```
-- CREATE PROCEDURE get_payment_branch
-- @payment_number INT, @branch_name VARCHAR(50) OUTPUT
-- AS
-- SET @branch_name = (SELECT Branch.branch_name
-- FROM Payment JOIN Loan ON Payment.loan_number = Loan.loan_number
-- JOIN Branch ON Branch.branch_name = Loan.branch_name
-- WHERE Payment.payment_number = @payment_number)

DECLARE @Res VARCHAR(50)
EXEC get_payment_branch 1, @branch_name = @Res OUTPUT;
PRINT @Res
```

100 %

Messages | Client Statistics

```
Vanak
```

ج)

```
-- CREATE PROCEDURE get_customer_data
-- @account_number INT
-- AS
-- BEGIN
-- WAITFOR DELAY '00:00:10'
-- SELECT Account.account_number, Account.balance, Customer.customer_id,
--  Customer.customer_name, Customer.customer_street, Customer.customer_city, Depositor.access_date
-- FROM Account JOIN Depositor on Account.account_number = Depositor.account_number
-- JOIN Customer ON Depositor.customer_id = Customer.customer_id
-- WHERE Account.account_number = @account_number
-- END

EXEC get_customer_data 1
```

100 %

Results | Messages | Client Statistics

	account_number	balance	customer_id	customer_name	customer_street	customer_city	access_date
1	1	100	1	Hasan	Artesh	Esfahan	2017-02-21

علی نظری - ۹۶۳۱۰۷۵

سوال ۳)

الف) شماره حسابِ حسابی که بیشترین سود را دارد:

```
-- CREATE FUNCTION get_highest_interest()
-- RETURNS INT
-- AS
-- BEGIN
--  DECLARE @tmp AS INT = (SELECT AVG(Account.account_number)
--                         FROM Account JOIN SavingAccount
--                         ON Account.account_number = SavingAccount.account_number
--                         WHERE interest_rate = (SELECT MAX(interest_rate)
--                                                FROM Account JOIN SavingAccount
--                                                ON Account.account_number = SavingAccount.account_number))
--  RETURN @tmp
-- END

DECLARE @temp INT
EXEC @temp = get_highest_interest
PRINT @temp
```

100 %

Messages | Client Statistics

```
2

Completion time: 2021-05-20T02:21:37.8054899+04:30
```

ب) در اینجا نیز دپارتمان کارمند ورودی را می‌گیریم:

```
-- CREATE FUNCTION get_employee_dep (@employee_id INT)
-- RETURNS VARCHAR(50)
-- AS
-- BEGIN
--     DECLARE @tmp AS VARCHAR(50) = (SELECT DISTINCT(dependent_name)
--                                    FROM Employee
--                                    WHERE employee_id = @employee_id)
--   RETURN @tmp
-- END

DECLARE @temp VARCHAR(50)
EXEC @temp = get_employee_dep @employee_id = 1
PRINT @temp
```

100 %

Messages | Client Statistics

```
Dep1

Completion time: 2021-05-20T02:29:31.3130842+04:30
```

علی نظری - ۹۶۳۱۰۷۵

سوال ۴)

الف) در این‌جا یک جدول برای تهیه‌ی لاگ‌های اضافه کردن و حذف نمونه از جدول پرداخت‌ها می‌سازیم و می‌بینیم که با اضافه کردن یک نمونه، دو سطر تغییر در دیتابیس داشتیم؛

```
CREATE TABLE PaymentLogs (
    payment_number INT NOT NULL,
    change_date DATETIME DEFAULT GETDATE() NOT NULL,
    command VARCHAR(6) NOT NULL,
    payment_date DATE,
    payment_amount INT,
    loan_number INT
)
GO

CREATE TRIGGER payment_change
ON Payment
AFTER INSERT, DELETE
AS
BEGIN
    DECLARE @command VARCHAR(6)
        SET @command = CASE
            WHEN EXISTS(SELECT * FROM INSERTED) AND EXISTS(SELECT * FROM DELETED)
                THEN 'Update'
            WHEN EXISTS(SELECT * FROM INSERTED)
                THEN 'Insert'
            WHEN EXISTS(SELECT * FROM DELETED)
                THEN 'Delete'
            ELSE NULL
        END
    IF @command = 'Delete'
        INSERT INTO PaymentLogs (payment_number, change_date, command, payment_date, payment_amount, loan_number)
        SELECT d.payment_number, GETDATE(), @command, d.payment_date, d.payment_amount, d.loan_number
        FROM DELETED d

    IF @command = 'Insert'
        INSERT INTO PaymentLogs (payment_number, change_date, command, payment_date, payment_amount, loan_number)
        SELECT i.payment_number, GETDATE(), @command, i.payment_date, i.payment_amount, i.loan_number
        FROM INSERTED i
END
GO

INSERT Payment (payment_number, payment_date, payment_amount, loan_number) VALUES (2, '2018-02-05', 200, 1)
```

```
83 %
Messages

(1 row affected)

(1 row affected)
```

ب) در اینجا از آپدیت branch_name جلوگیری می‌کنیم:

```
CREATE TRIGGER avoid_change_branch_name
ON Branch
AFTER UPDATE
AS
    if UPDATE(branch_name)
    BEGIN
        ROLLBACK
        RAISERROR('Change on branch_name is not allowed', 16, 1)
    END
GO
```

```
100 %
Messages  Client Statistics

Commands completed successfully.

Completion time: 2021-05-20T02:47:25.9966351+04:30
```