ΑΣΚΗΣΕΙΣ
(Οψεις – Διαδικασίες – Πυροδότες)

Άσκηση 1 – Όψεις (views).
Με δεδομένο το λογικό σχήμα της βάσης bank, όπου καταχωρούνται διάφορες τραπεζικές συναλλαγές για πελάτες και υποκαταστήματα ζητείται:

1. Να δημιουργήσετε μία όψη **LoanPerCustomer** που θα περιλαμβάνει το όνομα και το επώνυμο των πελατών, καθώς και το συνολικό ποσό των δανείων που έχει ο καθένας (αν ένας πελάτης έχει παραπάνω από ένα δάνεια το συνολικό ποσό προκύπτει ως το άθροισμα των ποσών των επιμέρους δανείων). Στην συνέχεια χρησιμοποιώντας την όψη LoanPerCustomer να εμφανίσετε τα στοιχεία των πελατών που τα δάνειά τους ξεπερνούν τα 10000 ευρώ.

   Δημιουργία όψης
   ```
   CREATE VIEW LoanPerCustomer AS
   SELECT firstname,lastname,sum(amount) as total
   FROM customers,borrowers,loans
   WHERE customers.cid=borrowers.cid AND
         borrowers.lnum=loans.lnum
        group by firstname, lastname;
   ```

   Επερώτηση
   ```
   SELECT firstname,lastname,total
     FROM LoanPerCustomer
   WHERE amount>10000;
   ```

2. Να δημιουργήσετε μία όψη **TotalLoanPerBranch** που θα περιλαμβάνει τα ονόματα των υποκαταστημάτων καθώς και το συνολικό ποσό των δανείων που έχει χορηγήσει κάθε υποκατάστημα (άθροισμα). Η όψη να περιλαμβάνει και τα ονόματα των υποκαταστημάτων που δεν έχουν χορηγήσει κανένα δάνειο. Στην συνέχεια να εκτελεστεί ένα ερώτημα πάνω στην όψη που δημιουργήσατε, το οποίο θα επιστρέφει τα υποκαταστήματα και τα δάνεια ταξινομημένα σε φθίνουσα σειρά με βάση το συνολικό ποσό των δανείων.

   Δημιουργία όψης
   ```
   CREATE VIEW TotalLoanPerBranch AS
   SELECT bname,SUM(loan.amount) AS totalloans
   FROM branches left join loans on branches.bcode=loans.bcode
   GROUP BY bname
   ```

   Επερώτηση
   ```
   SELECT bname,totalloans
     FROM TotalLoanPerBranch
   ORDER BY TotalLoans DESC
   ```
3. Να δημιουργήσετε μία όψη **TotalLoanAndDepositsPerBranch** που θα περιλαμβάνει για κάθε υποκατάστημα το όνομα του υποκαταστήματος, το συνολικό ποσό των δανείων

που το υποκατάστημα έχει χορηγήσει, καθώς επίσης και το συνολικό ποσό των καταθέσεων που διαθέτει (άθροισμα). Η όψη να περιλαμβάνει και τα υποκαταστήματα τα οποία μπορεί να μην έχουν χορηγήσει δάνεια ή να μην έχουν καταθέσεις. Στην συνέχεια να εκτελέσετε ένα ερώτημα στην όψη που δημιουργήσατε το οποίο θα εμφανίζει τα υποκαταστήματα που το συνολικό ποσό των δανείων είναι μεγαλύτερο από το συνολικό ποσό των καταθέσεων.

Δημιουργία όψης

```
CREATE VIEW TotalLoanAndDepositsPerBranch AS
SELECT bname,SUM(loans.amount) AS"TotalLoans",
SUM(accounts.balance) AS "Deposits"
FROM branches left join loans on branches.bcode=loans.bcode
            left join accounts on branches.bcode=accounts.bcode
GROUP BY bname
```

Επερώτηση

```
SELECT bname, TotalLoans, Deposits
FROM TotalLoanAndDepositsPerBranch
WHERE TotalLoans>Deposits
```

4. Να δημιουργήσετε μία όψη **BranchesCondition** που θα περιλαμβάνει το όνομα, την πόλη και τα αποθεματικά του κάθε υποκαταστήματος (αντιγραφή του πίνακα branch). Παρατηρήστε τα αποθεματικά του υποκαταστήματος “Σταδίου” από τον πίνακα Branch ότι είναι 10.000.000. Στην συνέχεια να ενημερώσετε την όψη που δημιουργήσατε έτσι ώστε τα αποθεματικά του υποκαταστήματος "Σταδίου" να είναι 5.000.000. Κατόπιν εμφανίστε τα περιεχόμενα του πίνακα branch. Τι παρατηρείτε;

Δημιουργία όψης

```
CREATE VIEW BranchesCondition AS
SELECT bname,city,assets
FROM branches
```

Επερώτηση

```
UPDATE BranchesCondition
SET assets=5000000
WHERE bname = 'Σταδίου'
```

Προσοχή: Η ενημέρωση της όψης έχει αλλάξει τις εγγραφές και στον ίδιο τον πίνακα branch για το υποκατάστημα Σταδίου.

Άσκηση 2 – Διαδικασίες

Με δεδομένο το λογικό σχήμα της βάσης bank, όπου καταχωρούνται διάφορες τραπεζικές συναλλαγές για πελάτες και υποκαταστήματα ζητείται:

1. Να δημιουργήσετε μια διαδικασία **Get_Customer_Info** που θα δέχεται ως παράμετρο το cid ενός πελάτη και θα εμφανίζει τα στοιχεία του (όνομα, επίθετο, πόλη). Στην συνέχεια να εκτελεστεί το procedureγια τον πελάτη με cid=1.

   Δημιουργία διαδικασίας

```
CREATE PROCEDURE Get_Customer_Info
@customer_id INT
AS
BEGIN
SELECT firstname, lastname, city FROM customers WHERE cid=@customer_id
END
```

   Εκτέλεση διαδικασίας

```
EXECUTE Get_Customer_Info
@customer_id = 1
```

2. Να δημιουργήσετε μια διαδικασία **Get_Customer_Name** που θα δέχεται ως παράμετρο το cid ενός πελάτη και θα εμφανίζει το όνομα και το επίθετο του. Το όνομα και το επίθετο του πελάτη να επιστρέφεται από μια παράμετρο εξόδου που θα οριστεί. Στην συνέχεια να εκτελεστεί η διαδικασία για τον πελάτη με cid=2.

   Δημιουργία διαδικασίας

```
CREATE PROCEDURE Get_Customer_Name
@customer_id INT,
@customer_name VARCHAR(100) OUT
AS
BEGIN
  SELECT @customer_name= firstname+' '+lastname FROM customers
      WHERE cid = @customer_id
END
```

   Εκτέλεση διαδικασίας

```
DECLARE @customer_name VARCHAR(100)
EXECUTE Get_Customer_Name
@customer_id = 2, @customer_name = @customer_name OUTPUT
SELECT @customer_name
```

3. Να δημιουργήσετε μια διαδικασία **Insert_Customer** που θα δέχεται τα στοιχεία ενός πελάτη (cid, firstname, lastname, city) και να τον καταχωρεί στην βάση μόνο αν δεν υπάρχει άλλος πελάτης με το ίδιο όνομα και το ίδιο επίθετο. Η διαδικασία σε κάθε περίπτωση θα πρέπει να εμφανίζει αντίστοιχο μήνυμα. Στην συνέχεια να εκτελέσετε την διαδικασία για την εισαγωγή του πελάτη με cid=22, firstname='Μάριος', lastname='Αβραμίδης', city='Αθήνα'.

Δημιουργία διαδικασίας

```
CREATE PROCEDURE Insert_Customer
@cid int, @firstname Varchar (30), @lastname Varchar (30), @city Varchar
(30)
AS
DECLARE @count int
SELECT @count=0
SELECT @count=COUNT(*)
  FROM customers
WHERE firstname=@firstname AND lastname=@lastname
IF (@count=0)
BEGIN
   INSERT INTO customers VALUES (@cid, @firstname, @lastname, @city)
   PRINT 'Customer Record Inserted'
END
ELSE
   PRINT 'Customer Record already exists'
```

Εκτέλεση διαδικασίας

```
EXECUTE Insert_Customer
@cid=22, @firstname='Μάριος', @lastname='Αβραμίδης', @city='Αθήνα'
```

4. Να δημιουργήσετε μια διαδικασία **Rate_Customers** που θα αξιολογεί όλους τους πελάτες με βάση το ποσό των καταθέσεών τους. Συγκεκριμένα οι πελάτες με συνολικές καταθέσεις <15.000 θα αξιολογούνται ως '**3 (Low)'**, οι πελάτες με συνολικές καταθέσεις <50.000 θα αξιολογούνται ως '**2 (Medium)'**, και οι υπόλοιποι ως '**1 (High)'.** Στην συνέχεια να εκτελέσετε την διαδικασία για να εμφανιστεί η λίστα με την αξιολόγηση όλων των πελατών.

Δημιουργία διαδικασίας

```
CREATE PROCEDURE Rate_Customers AS
DECLARE @customer varchar (30)
DECLARE @TotalDeposits numeric
DECLARE @minCid int

SELECT @minCid=min(cid) FROM customers

WHILE @minCid is NOT NULL
BEGIN
   SET @customer= (SELECT lastname FROM customers WHERE customers.cid=@minCID)
   SET @TotalDeposits= (SELECT SUM (accounts.balance)
                        FROM customers, depositors, accounts
                         WHERE customers.cid=@minCid AND
                              customers.cid=depositors.cid AND
                              depositors.accno=accounts.accno)
IF (@TotalDeposits is NOT NULL)
BEGIN
  --Υλοποίηση με χρήση case
  SELECT CASE
    WHEN @TotalDeposits<15000 THEN @customer+': '+CAST(@TotalDeposits AS
       VARCHAR(12)) + ' - 3 (Low)'
    WHEN @TotalDeposits<50000 THEN @customer+': '+CAST(@TotalDeposits AS
       VARCHAR(12)) + ' – 2 (Medium)'
    ELSE @customer+': '+CAST(@TotalDeposits AS VARCHAR(12)) + ' – 1 (High)'
  END
END
SELECT @minCid = min(cid)FROM customers WHERE @minCid < cid
END
```

Εκτέλεση διαδικασίας

```
EXECUTE Rate_Customers
```

5. Να τροποποιήσετε τη διαδικασία **Rate_Customers** σε **Rate_Customers_Table** ώστε να επιστρέφει έναν πίνακα με την αξιολόγηση του κάθε πελάτη. Τα στοιχεία που πρέπει να ορισθούν στον πίνακα είναι το cid και το lastname του πελάτη καθώς και το rate που υπολογίζεται από το procedure **1(High), 2 (Medium), 3 (Low)**. Στην συνέχεια να εκτελέσετε τη διαδικασία για να εμφανισθούν τα στοιχεία του πίνακα που δημιουργήθηκε. Παρατηρήστε ότι ο πίνακας που δημιουργήθηκε δεν υπάρχει στους πίνακες της βάσης.

Δημιουργία διαδικασίας

```
CREATE PROCEDURE Rate_Customers_Table AS
DECLARE @customer varchar (30)
DECLARE @TotalDeposits numeric
DECLARE @Customer_Rate TABLE (cid int, lastname varchar (30), rate varchar
(10))
DECLARE @rate varchar (10)
DECLARE @minCid int
SELECT @minCid=min (cid) FROM customers

WHILE @minCid is NOT NULL
BEGIN
    SET @customer= (SELECT lastname FROM customers WHERE customer.cid=@minCID)
    SET @TotalDeposits= (SELECT SUM (accounts.balance)
                          FROM customers, depositors,accounts
                            WHERE customers.cid=@minCid AND
                                  customers.cid=depositors.cid AND
                                   depositors.accno=accounts.accno)
    IF (@TotalDeposits is NOT NULL)
    BEGIN
       IF (@TotalDeposits<15000) SET @rate='3 (Low)'
       ELSE IF    (@TotalDeposits<50000)SET @rate='2 (Medium)'
       ELSE SET @rate='1 (High)'
       INSERT @Customer_Rate (cid,lastname,rate)
                 VALUES (@minCid,@customer,@rate)
   END
   SELECT @minCid = min(cid)FROM customers WHERE @minCid < cid
END
SELECT * FROM @Customer_Rate
```

Εκτέλεση διαδικασίας

```
EXECUTE Rate_Customers_Table
```

Άσκηση 3 – Πυροδότες (Triggers)
Με δεδομένο το λογικό σχήμα της βάσης bank, όπου καταχωρούνται διάφορες τραπεζικές συναλλαγές για πελάτες και υποκαταστήματα ζητείται:

1. Να δημιουργήσετε έναν πυροδότη **trigger_customers** που θα ενεργοποιείται με κάθε update, insert, delete στον πίνακα customer και θα εμφανίζει το μήνυμα 'You made one DML operation'. Στην συνέχεια να εισάγετε έναν καινούργιο πελάτη με το όνομά σας για να ενεργοποιηθεί ο πυροδότης.

   Δημιουργία πυροδότη
   ```
   CREATE TRIGGER trigger_customers
   ON customers
   AFTER INSERT, UPDATE, DELETE
   AS PRINT ('You made one DML operation');
   ```

   Ενεργοποίηση πυροδότη
   ```
   INSERT INTO   customers   (cid,firstname,lastname,city)
   VALUES (21, 'Χρυσόστομος', 'Καπέτης', 'Αθήνα');
   ```

2. Να δημιουργήσετε έναν πυροδότη **BorrowerLoanInfo** που θα ενεργοποιείται μετά από μία καταχώρηση στον πίνακα borrower η οποία θα αναθέτει σε έναν υπάρχων πελάτη (cid) έναν νέο αριθμό δανείου (lnum)). Κάθε φορά που ένας πελάτης παίρνει ένα νέο δάνειο ο πυροδότης θα ενημερώνει για το συνολικό ποσό των δανείων που έχει πάρει ο συγκεκριμένος πελάτης. Στην συνέχεια αρχικά να εισάγεται ένα νέο δάνειο στον πίνακα loan με τιμές: lnum='L480', bcode=550, amount=3300 και μετά μία πλειάδα στον πίνακα borrower για αυτό το δάνειο στον πελάτη με cid=2 για να ενεργοποιηθεί ο πυροδότης.

   Δημιουργία πυροδότη
   ```
   CREATE TRIGGER BorrowerLoanInfo ON borrower
   AFTER INSERT AS
   BEGIN
     DECLARE @id int, @loans numeric, @customer varchar (30)
     SET @id= (SELECT cid FROM INSERTED)
     SET @customer= (SELECT lastname FROM customers WHERE
   customers.cid=@id)
     SET @loans= (SELECT sum (loans.amount) FROM loans, borrowers
        WHERE borrowers.cid=@id AND borrowers.lnum=loans.lnum)
     PRINT 'The new loan is accepted'
     PRINT'The total loans of '+@customer+ ': '+CAST(@loans as
   varchar(12))
   END
   GO
   ```

   Ενεργοποίηση πυροδότη
   ```
   INSERT INTO   loans   (lnum,bcode,amount) VALUES ('L480', 550,
   3300);
   INSERT INTO   borrowers   (cid,lnum) VALUES (2, 'L480');
   ```

3. Να δημιουργήσετε έναν πυροδότη **BorrowerLoanApproval** που θα ενεργοποιείται μετά από μία καταχώρηση στον πίνακα borrower η οποία θα αναθέτει σε έναν υπάρχων πελάτη ένα νέο δάνειο. Κάθε φορά που ένας πελάτης παίρνει ένα νέο δάνειο θα γίνεται ο έλεγχος:
   - Αν τα συνολικά του δάνεια ξεπερνούν τις 55000, τότε το νέο δάνειο απορρίπτεται.
   - Αλλιώς το δάνειο εγκρίνεται δίνοντας ενημέρωση για την συνολική δανειακή κατάσταση του πελάτη.

   Στην συνέχεια αρχικά να εισάγεται ένα νέο δάνειο στον πίνακα loanμε τιμές: lnum='L1100', bcode=550, amount=20000 και στη συνέχεια να αναθέσετε αυτό το δάνειο στον πελάτη με cid=2 για να ενεργοποιηθεί ο πυροδότης. Τι παρατηρείτε;

Δημιουργία πυροδότη

```
CREATE TRIGGER BorrowerLoanApproval ON borrowers
AFTER INSERT AS
BEGIN
  DECLARE @id int, @lnum varchar(10), @loans numeric, @customer
varchar(30)
  SET @id= (SELECT cid FROM INSERTED)
  SET @lnum= (SELECT lnum FROM INSERTED)
  SET @customer=(SELECT lastname FROM customers WHERE
customers.cid=@id)
  SET @loans=(SELECT sum(loans.amount) FROM loans,borrowers WHERE
borrowers.cid=@id AND borrowers.lnum=loans.lnum)
  IF (@loans>55000)
   BEGIN
     PRINT ' ';
     PRINT'Total loans of '+@customer+ ': '+CAST(@loans as
varchar(12));
     PRINT 'The new loan is denied';
     PRINT 'Deleting new loan attached to this customer ...'
     DELETE FROM borrowers WHERE borrowers.lnum=@lnum and
borrowers.cid=@id;
     DELETE FROM loans WHERE loans.lnum=@lnum;
    END
  ELSE
   BEGIN
   PRINT 'The new loan is accepted';
   PRINT' Total loans of '+@customer+ ': '+CAST(@loans as
varchar(12));
   END
END
```

Ενεργοποίηση πυροδότη

```
INSER INTO   loans   (lnum,bcode,amount) VALUES ('L1100', 550, 20000);
INSERT INTO   borrowers   (cid,lnum) VALUES (2, 'L1100')
```

4. Να δημιουργήσετε έναν πυροδότη **Open_Loan** που θα ενεργοποιείται μετά από μία ενημέρωση στον πίνακα accounts η οποία έχει ως αποτέλεσμα την δημιουργία αρνητικού υπολοίπου σε έναν λογαριασμό. Κάθε φορά που συμβαίνει μια ανάληψη χωρίς αντίκρισμα πρέπει να γίνονται οι εξής λειτουργίες:
    - Μηδενίζεται το υπόλοιπο του λογαριασμού.
    - Δημιουργείται ένα δάνειο με το επιπλέον ποσό.
    - Το νέο δάνειο φέρει ως αριθμό δανείου τον αριθμό του λογαριασμού.

   Στην συνέχεια να ενημερώσετε τον λογαριασμό 'A901' μειώνοντας το υπόλοιπό του κατά 2000. Παρατηρήστε το υπόλοιπο του λογαριασμού μετά την ενεργοποίηση του πυροδότη. Αναζητήστε το νέο δάνειο που δημιουργήθηκε με κωδικό 'LA901'.

Δημιουργία πυροδότη

```
CREATE TRIGGER Open_Loan ON accounts
AFTER  UPDATE AS
BEGIN
  SET NOCOUNT OFF
  DECLARE @accnum varchar (10), @customerID int, @newbalance
numeric, @oldbalance numeric
  DECLARE @bcode int, @lnum varchar (10), @balanceAfterLoan
numeric
  SET @accnum = (SELECT accno FROM INSERTED)

/* Προσοχή πρέπει ο λογαριασμός να ανήκει μόνο σε έναν πελάτη.
Διαφορετικά το παρακάτω ερώτημα θα επιστρέψει περισσότερες από
μία τιμές και θα προκύψει σφάλμα χρόνου εκτέλεσης */
  SET @customerID = (SELECT cid FROM depositors WHERE
depositors.accno=@accnum)  SET @newbalance = (SELECT balance FROM
INSERTED)

  SET @oldbalance = (SELECT balance FROM DELETED)
  SET @bcode = (SELECT bcode FROM INSERTED)
  SET @lnum='L'+@accnum
IF (@newbalance>=0)
   BEGIN
       PRINT 'The withdrawal: '+ CAST((@oldbalance-@newbalance)
AS varchar(15))+' was successful.'
   END
ELSE --(@newbalance<0)
   BEGIN
   PRINT 'The withdrawal is greater than balance'
    PRINT 'The account is charged with a loan: '+ CAST(-
@newbalance AS varchar(15))
    INSER INTO   loans   (lnum,bcode,amount) VALUES (@lnum,
@bcode, -@newbalance);
   INSER INTO  borrowers  (cid,lnum) VALUES (@customerID, @lnum);
   UPDATE accounts SET balance=0 WHERE accounts.accno=@accnum
END
SET @balanceAfterLoan=(SELECT balance FROM accounts WHERE
accounts.accno=@accnum)
PRINT 'The new balance is  ' +CAST(@balanceAfterLoan AS
varchar(15))
END
```

Ενεργοποίηση πυροδότη

```
UPDATE accounts SET balance= (balance - 50000)
    WHERE accounts.accno = 'A906'
```

5. Να δημιουργήσετε έναν πυροδότη για την υλοποίηση μιας διαδρομής παρακολούθησης της δραστηριότητας στον πίνακα accounts. Συγκεκριμένα να δημιουργήσετε έναν πίνακα audit_account, που αποθηκεύει όλες τις τροποποιήσεις της στήλης balance του πίνακα accounts. Για να ενεργοποιήσετε τον πυροδότη χρησιμοποιείστε την εντολή UPDATE για να ενημερώσετε το υπόλοιπο ενός λογαριασμού. Τέλος εμφανίστε τα περιεχόμενα του πίνακα audit_account. Τι παρατηρείτε;

Δημιουργία πίνακα audit_account

```
CREATE TABLE audit_account
( accno VARCHAR(10),
  bcode int,
  usr_name VARCHAR(30),
  utime DATETIME NULL,
  balance_old numeric(18,0),
  balance_new numeric(18,0)
  );
```

Δημιουργία πυροδότη

```
CREATE TRIGGER modify_balance
 ON accounts AFTER UPDATE
  AS IF UPDATE (balance)
  BEGIN
    DECLARE @balance_old NUMERIC(18,0)
       DECLARE @balance_new NUMERIC(18,0)
       DECLARE @accno VARCHAR(10)
       DECLARE @bcode INT
       SELECT @balance_old =(SELECT balance FROM deleted)
       SELECT @balance_new =(SELECT balance FROM inserted)
       SELECT @accno =(SELECT accno FROM deleted)
       SELECT @bcode = (SELECT bcode FROM deleted)

       INSERT INTO audit_account
         VALUES (@accno, @bcode, USER_NAME(),
                      GETDATE(), @balance_old,@balance_new)

  END
```

Ενημέρωση του υπολοίπου του λογαριασμού A906.

```
UPDATE accounts
  SET balance=100000
      WHERE accno='A906'
```