ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Όψεις (VIEWS)
Διαδικασίες (Stored Procedures)
Πυροδότες (Triggers)

Χρυσόστομος Καπέτης

Όψεις (Views)

- Μια **όψη** είναι ένας **ιδεατός πίνακας** ο οποίος σχηματίζεται με πεδία τα οποία μπορεί να προέρχονται από έναν η περισσότερους πίνακες της βάσης, ενώ παράλληλα οι αντίστοιχες εγγραφές πληρούν ένα δεδομένο κριτήριο αναζήτησης.
- Μοιάζει πολύ με ένα πραγματικό πίνακα διότι περιέχει στήλες και εγγραφές.
- Μπορούμε να χρησιμοποιήσουμε μία όψη σαν να είναι πίνακας για να εκτελέσουμε επερωτήσεις (εντολή select).
- Η όψη δεν απαιτείται να είναι αποθηκευμένη σε φυσική μορφή.
- Υπάρχουν και τα materialized views (indexed views στον sql server) όπου μια όψη υλοποιείται με έναν προσωρινό πραγματικό πίνακα στον οποίο αποθηκεύονται τα δεδομένα της όψης.

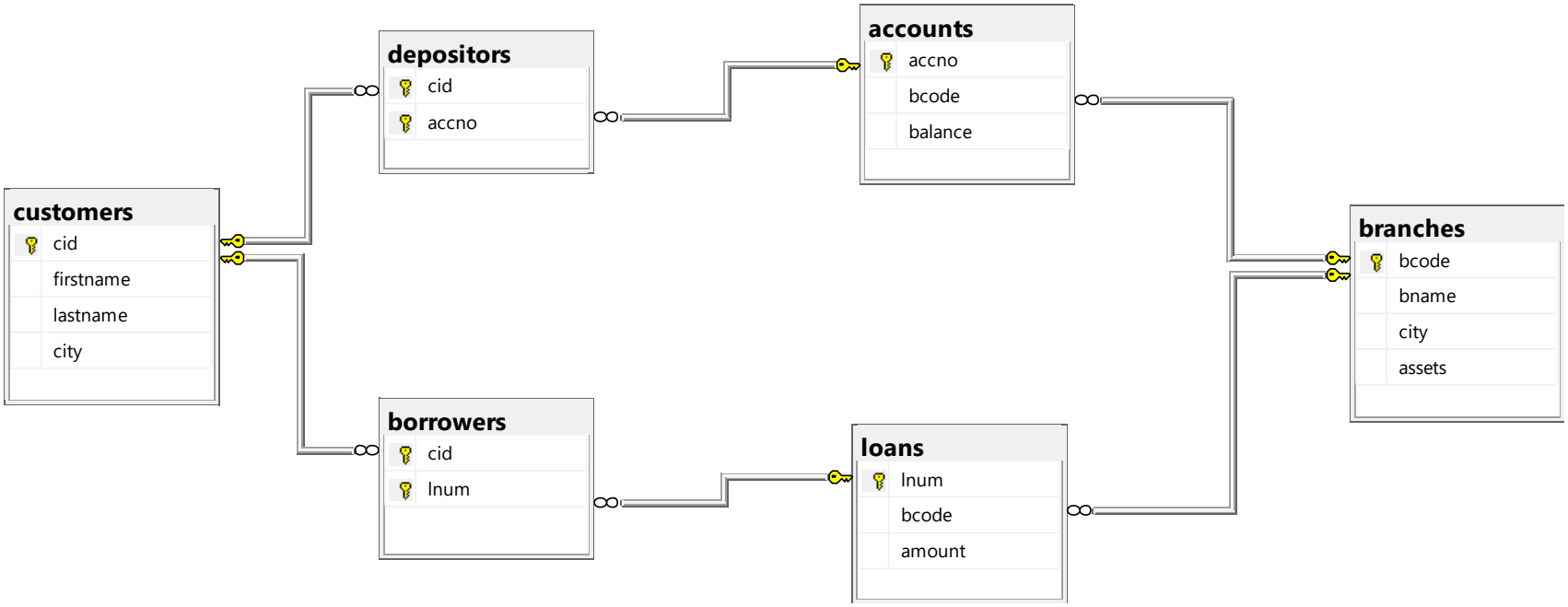
Δημιουργία όψης: Create View


```
CREATE VIEW ViewName AS

SELECT...
```

- Οι όψεις δημιουργούνται με την εντολή CREATE VIEW.
- Οι όψεις πρέπει να έχουν ένα μοναδικό όνομα στην βάση δεδομένων.
- Μπορούν να ορισθούν με οποιοδήποτε έγκυρο ερώτημα SELECT.

Σχήμα Παραδειγμάτων



Δημιουργία Όψης: Παράδειγμα 1


```
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;
```

- Δημιουργείται η όψη **LoanPerCustomer**.
- Η όψη περιλαμβάνει το όνομα και το επώνυμο των πελατών καθώς και το **συνολικό ποσό** των δανείων που έχουν.
- Πάνω σε αυτή την όψη μπορούν να εκτελεστούν ερωτήματα σαν να είναι ένας πίνακας.

Χρήση Όψης: Παράδειγμα 1


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

firstname	lastname	total
Μαρία	Αγγελίδου	40000
Ελένη	Αγγελοπούλου	13000
Γεώργιος	Γεωργίου	11000
Παναγιώτης	Κολιάτσος	59000
Ιωάννης	Μαρκόπουλος	50000

Δημιουργία Όψης: Παράδειγμα 2


```
CREATE VIEW TotalLoanPerBranch AS

SELECT bname,SUM(loan.amount) AS totalloans

FROM branches left join loans on
branches.bcode=loans.bcode

GROUP BY bname
```

- Δημιουργείται η όψη**TotalLoanPerBranch**.
- Η όψη περιλαμβάνει το όνομα κάθε υποκαταστήματος και το σύνολο των δανείων που το υποκατάστημα έχει χορηγήσει. Η όψη περιλαμβάνει και τα ονόματα των υποκαταστημάτων που δεν έχουν χορηγήσει κανένα δάνειο.
- Πάνω σε αυτή την όψη μπορούν να εκτελεστούν ερωτήματα σαν να είναι ένας πίνακας.

Χρήση Όψης: Παράδειγμα 2


```
SELECT bname,totalloans
    FROM TotalLoanPerBranch
ORDER BY totalloans DESC
```

bname	totalloans
Σταδίου	63000
Κηφισίας	36000
Πατησίων	16000
Τσιμισκή	12000
Πειραιώς	9000
Γούναρη	8000
Δωδώνης	3000
Νίκης	2000
Πανεπιστημίου	NULL
Αμαλίας	NULL

Χρήση Όψης: Παράδειγμα 3


```
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
```

- Δημιουργείται η όψη TotalLoanAndDepositPerBranch.
- Η όψη περιλαμβάνει το όνομα του υποκαταστήματος και το σύνολο των δανείων και των καταθέσεων που έχει. Η όψη περιέχει και τα ονόματα των υποκαταστημάτων που δεν έχουν χορηγήσει δάνειο ή δεν έχουν καταθέσεις.

Παράδειγμα Χρήση View 3


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

bname	TotalLoans	Deposits
Κηφισίας	144000	48000
Πατησίων	48000	22000
Πειραιώς	18000	11000
Σταδίου	252000	225000

Τροποποίηση – Διαγραφή Όψεων (Alter-Drop Views)


```
ALTER VIEW ViewName AS

SELECT ...
```

- Οι όψεις μπορούν να τροποποιηθούν δίχως να χρειαστεί να διαγραφούν πρώτα.

```
Drop VIEW ViewName
```

- Η όψεις μπορούν να διαγραφούν χωρίς να επηρεάζουν τα ίδια τα δεδομένα της βάσης.

Ενημέρωση Όψεων (Update Views)


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

- Δημιουργούμε μια όψη με τα δεδομένα τον υποκαταστημάτων(όνομα, πόλη, αποθεματικό).

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

- Ενημερώνοντας την όψη **αλλάζουμε** και τα αποθεματικά (assets) του καταστήματος Σταδίου και στον **πραγματικό πίνακα**.

Ενημέρωση Όψεων

- Η ενημέρωση των όψεων δεν επιτρέπεται όταν:
  - Στην όψη συμμετέχουν περισσότεροι από ένας πίνακες.
  - Η όψη χρησιμοποιεί συνοπτικές συναρτήσεις (aggregate functions).
  - Στην όψη υπάρχει ο προσδιοριστής DISTINCT.
  - Στην όψη γίνεται χρήση των προσδιοριστών GROUP BY ή HAVING.
- Γενικότερα ενημερώσεις σε πολύπλοκες όψεις είναι δύσκολο έως και αδύνατο να μεταφραστούν σωστά σε ενημερώσεις των αντίστοιχων πινάκων. Για το λόγο αυτό δεν επιτρέπονται.

Παρατηρήσεις

- Η όψη μπορεί να χρησιμοποιηθεί ως πίνακας σε ερωτήματα SELECT.
- Οι πίνακες αντιστοιχούν σε βασικές σχέσεις, οι όψεις αντιστοιχούν σε παράγωγες σχέσεις.
- Οι όψεις δεν αποθηκεύουν δεδομένα.
- Η διαγραφή μιας όψης δεν σημαίνει διαγραφή δεδομένων από την βάση.
- Οποιαδήποτε διαγραφή, τροποποίηση, εισαγωγή δεδομένων γίνει στην βάση σημαίνει αυτόματη ενημέρωση των περιεχομένων μιας όψης.

Πλεονεκτήματα Όψεων

- Η χρήση των όψεων απλοποιεί μερικά σύνθετα ερωτήματα.
- Κάθε χρήστης μπορεί να έχει διαφορετική εξουσιοδότηση για τον χειρισμό δεδομένων.
- Δεν υπάρχει ανάγκη ο κάθε χρήστης να γνωρίζει το σχήμα της βάσης δεδομένων.
- Οι όψεις μορφοποιήσουν τα δεδομένα με πιο φιλικό τρόπο για τους χρήστες.
- Ενοποιούν δεδομένα από πολλούς πίνακες (οι οποίοι μπορεί να βρίσκονται και σε διαφορετικούς εξυπηρετητές σε περίπτωση κατανεμημένων βάσεων) για χρήση στην παραγωγή αναφορών.

ΔΙΑΔΙΚΑΣΙΕΣ (STORED PROCEDURES)

Διαδικασίες (Stored Procedures)

- Μία διαδικασία αποτελείται από ένα σύνολο εντολών SQL με σκοπό την εκτέλεση μιας συγκεκριμένης λειτουργίας.
- Οι διαδικασίες είναι τμήματα προγράμματος που αποθηκεύονται μόνιμα και εκτελούνται στην βάση δεδομένων.
- Καλούνται με το όνομα τους από οποιαδήποτε εφαρμογή συνδεθεί με τη Βάση.
- Περιέχουν προγραμματιστικές εντολές (IF, LOOPS) αλλά ταυτόχρονα μπορούν να χειριστούν και εντολές SQL.

Διαδικασίες (Stored Procedures) (2)

- Δέχονται παραμέτρους από την εφαρμογή που τις καλεί και μπορούν να επιστρέψουν αποτελέσματα.
- Μια διαδικασία μπορεί να καλέσει εσωτερικά μια άλλη διαδικασία.
- Αποθηκεύονται σαν αντικείμενα της βάσης.

Χρήση Διαδικασιών

- Οι διαδικασίες μπορούν να χρησιμοποιηθούν για:
  - τον έλεγχο πολύπλοκων περιορισμών.
  - την απόκρυψη του κώδικα της διαδικασίας.
  - τη δημιουργία βιβλιοθήκης έτοιμων συναρτήσεων και διαδικασιών στην βάση.

Πλεονεκτήματα Διαδικασιών

- Βελτιώνουν την ασφάλεια της βάσης.
  - Ο κώδικας δεν είναι ορατός στους χρήστες και με σωστό σχεδιασμό και συστηματική χρήση τους είναι δυνατόν να εξαφανιστεί η δομή της βάσης από τους χρήστες.
  - Προσθέτουν ένα επιπλέον επίπεδο ασφάλειας σχετικά με την χορήγηση δικαιωμάτων.
- Βελτιώνουν την ταχύτητα απόκρισης του εξυπηρετητή της Βάσης. Αυτό οφείλεται στο ότι είναι προμεταγλωττισμένες (precompiled) και έτσι εκτελούνται γρηγορότερα.
- Επίσης, αντικαθιστώντας ένα σύνολο SQL εντολών με την κλήση μιας αντίστοιχης διαδικασίας «ελαφραίνει» η κίνηση στο δίκτυο μεταξύ του πελάτη και του εξυπηρετητή.

Σύνταξη Διαδικασιών


```
CREATE PROCEDURE PROC_Name
Input Parameters @Name type
Output parameters @Name type OUT
AS
BEGIN
--OUR LOGIC GOES HERE
END
```

→ Proc Name

→ Input

→ Output

→ Logic of Procedure

Σύνταξη Διαδικασιών (2)

- Το σώμα μιας διαδικασίας μπορεί να περιλαμβάνει τα παρακάτω:
  - Δηλώσεις DECLARE <variable>
  - Δομές ελέγχου (IF)
  - Δομές επανάληψης (WHILE)
  - Εντολές CALL, GOTO, RETURN

Μεταβλητές (Variables)

- Μία μεταβλητή μπορούμε να την δηλώσουμε με την εντολή DECLARE:

  DECLARE @variable data_type [= value]

- Παραδείγματα

  DECLARE @lastname VARCAR(50)='Γεωργίου'
  DECLARE @accno VARCHAR(10)

- Όπως βλέπουμε στο πρώτο παράδειγμα μπορούμε κατά τον ορισμό να αποδώσουμε στην μεταβλητή μία τιμή.

- Στις μεταβλητές μπορούμε να εκχωρούμε τιμές με τις εντολές **SET** και **SELECT.**

  SET @variable = value
  SELECT @variable = value

- Με την εντολή SELECT μπορούμε να εκχωρήσουμε σε μεταβλητές τιμές από τις εγγραφές ενός πίνακα.

  SELECT @lastname = lastname **FROM** customers **WHERE** cid=10

Μεταβλητές (Variables)

- Με την εντολή SELECT μπορούμε να εκχωρήσουμε σε μεταβλητές τιμές από τις εγγραφές ενός πίνακα.

```
SELECT  @lastname = lastname FROM customers WHERE cid=10
```

- Στο παραπάνω παράδειγμα εντολή SELECT επιστρέφει μόνο μία εγγραφή (υπάρχει μόνο ένας πελάτης με κωδικό cid=10). Αν η εντολή SELECT επιστρέφει περισσότερες από μία εγγραφές τότε η μεταβλητή θα πάρει την αντίστοιχη τιμή από την τελευταία εγγραφή του αποτελέσματος της εντολής SELECT.
- Για να εμφανίσουμε την τιμή μίας μεταβλητής χρησιμοποιούμε την εντολή **PRINT.** Με την εντολή PRINT εκτός από τις τιμές των μεταβλητών μπορούμε να εμφανίσουμε και συμβολοσειρές (strings).

```
PRINT msg_str | @variable
```

- **Παράδειγμα**

```
DECLARE @lastname VARCAR(30)
SET @lastname='Γεωργίου'
PRINT @lastname
PRINT 'Το επώνυμο είναι '  +  @lastname
```

Συνένωση συμβολοσειρών

- Ο τελεστής **+** μπορεί να χρησιμοποιηθεί για την συνένωση συμβολοσειρών.
- **Παράδειγμα**

  DECLARE @lastname **VARCHAR(30)**, @firstname **VARCHAR(30)**
  SET @lastname='Γεωργίου'
  SET @firstname='Γεώργιος'
  PRINT 'Ονοματεπώνυμο: ' + @firstname+ ' ' + @lastname

  Θα εμφανίσει: Ονοματεπώνυμο: Γεώργιος Γεωργίου

- Δύο ή περισσότερες συμβολοσειρές μπορούμε να τις ενώσουμε και με την συνάρτηση **CONCAT(string1, string2, …., string_n)**

  PRINT CONCAT ('Ονοματεπώνυμο: ',@firstname, ' ', @lastname)

Οι συναρτήσεις CONVERT() και CAST()

- Η συνάρτηση **CONVERT()** μετατρέπει μια τιμή από έναν τύπο δεδομένων σε έναν άλλον τύπο δεδομένων.

  CONVERT(*data_type(length), expression*)

  - data_type: ο τύπος στον οποίο θέλουμε να μετατρέψουμε την τιμή.
  - expression: η τιμή προς μετατροπή.

- **Παράδειγμα**

```
DECLARE @X VARCHAR(5)
SET @X=CONVERT(VARCHAR, 30.12)
```

- Η μετατροπή μιας τιμής από έναν τύπο δεδομένων σε έναν άλλον μπορεί να γίνει και με την συνάρτηση CAST()

  CAST (*expression* ***AS*** *data_type(length*))

- **Παράδειγμα**

```
DECLARE @X VARCHAR(5)
SET @X=CAST(30.12 AS VARCHAR(5))
```

Η εντολή IF..ELSE

- Σύνταξη:

```
IF Boolean_expression
    { sql_statement | statement_block }
[ ELSE
    { sql_statement | statement_block } ]
```

- Αν ισχύει η συνθήκη εκτελούνται οι εντολές που ακολουθούν τον όρο **IF**, διαφορετικά εκτελούνται οι εντολές που ακολουθούν τον όρο **ELSE.**
- Αν μετά τον όρο **IF** (ή ELSE) ακολουθούν περισσότερες από μία εντολές τότε αυτές σχηματίζουν μια ομάδα εντολών (block).
- Μπορούμε να δηλώσουμε την αρχή και το τέλος μίας ομάδας εντολών (block) με τις εντολές **BEGIN** και **END** αντίστοιχα.

Η εντολή IF..ELSE - Παράδειγα


```
DECLARE @Number INT;
SET @Number = 50;
IF @Number > 100
        PRINT 'The number is large.';
ELSE
   BEGIN
      IF @Number < 10
              PRINT 'The number is small.';
      ELSE
            PRINT 'The number is medium.';
   END;
```

Η εντολή WHILE

- Σύνταξη:

  WHILE Boolean_expression
  { sql_statement | statement_block | **BREAK** | **CONTINUE** }

- Όσο ισχύει η συνθήκη εκτελούνται οι εντολές που ακολουθούν τον όρο **WHILE.**
- Αν μετά τον όρο **WHILE** ακολουθούν περισσότερες από μία εντολές τότε αυτές σχηματίζουν μια ομάδα εντολών (block).
- Μπορούμε να δηλώσουμε την αρχή και το τέλος μίας ομάδας εντολών (block) με τις εντολές **BEGIN** και **END** αντίστοιχα.
- Ό όρος **BREAK** προκαλεί την έξοδο από τον βρόχο της επανάληψης.
- Ο όρος **CONTINUE** προκαλεί την επανεκκίνηση του βρόχου αγνοώντας τυχόν εντολές που ακολουθούν τον όρο CONTINUE.

Η εντολή WHILE - Παράδειγμα

- Ο παρακάτω κώδικας εμφανίζει την τιμή της μεταβλητής counter 10 φορές.

```
DECLARE @Counter INT
SET @Counter=1
WHILE ( @Counter <= 10)
BEGIN
   PRINT 'The counter value is = ' + CONVERT(VARCHAR, @Counter)
   SET @Counter  = @Counter  + 1
END
```

- Στο ακόλουθο παράδειγμα μόλις η τιμή του μετρητή (Counter) γίνει 7 θα εκτελεστεί η εντολή **BREAK** και θα τερματιστεί ο βρόχος της επανάληψης.

```
DECLARE @Counter INT
SET @Counter=1
WHILE ( @Counter <= 10)
BEGIN
  PRINT 'The counter value is = ' + CONVERT(VARCHAR, @Counter)
  IF @Counter >=7
  BEGIN
    BREAK
  END
   SET @Counter  = @Counter  + 1
END
```

Εκτέλεση Διαδικασιών

- Για να καλέσουμε και να εκτελέσουμε μια διαδικασία χρησιμοποιούμε την εντολή EXECUTE:

```
EXECUTE(EXEC) PROC_Name
   Input Parameters,
   Output parameters
```

Έυρεση-Τροποποίηση-Διαγραφή Διαδικασιών


```
SELECT *
FROM sys.procedures
WHERE is_ms_shipped = 0
```

- Η παραπάνω εντολή sql επιστρέφει όλες τις διαδικασίες της βάσης.

```
DROP PROCEDURE proc_name;
```

- Διαγράφει την διαδικασία με όνομα proc_name.

```
ALTER PROCEDURE proc_name;
```

- Τροποποιεί την διαδικασία proc_name.

Δημιουργία διαδικασίας (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.

```
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
```

Δημιουργία διαδικασίας (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.

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

Δημιουργία διαδικασίας (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'.

```
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'
```

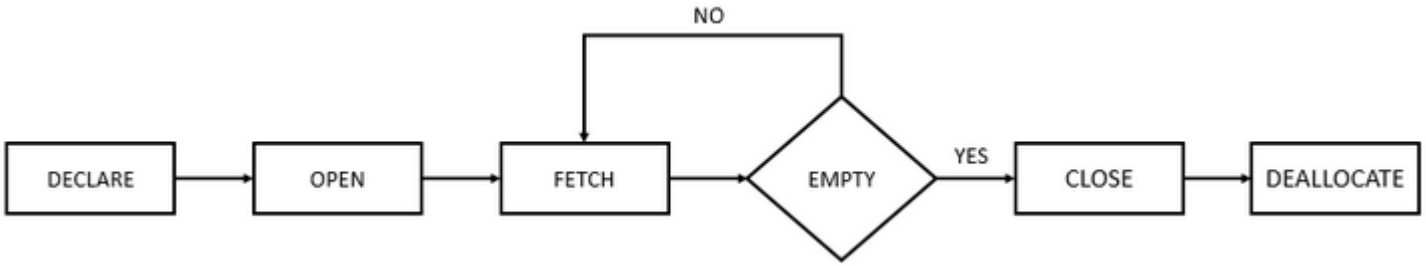
Εκτέλεση διαδικασίας

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

Δρομείς (Cursors)

Δρομείς (Cursors)

- Ένας δρομέας (cursor) είναι ένα αντικείμενο το οποίο μας δίνει την δυνατότητα να επεξεργαστούμε ξεχωριστά κάθε εγγραφή του συνόλου αποτελεσμάτων μιας εντολής select.
- Η χρήση ενός δρομέα απαιτεί συγκεκριμένα βήματα.



Δρομείς (2)

- **Δήλωση Δρομέα:** για να δηλώσουμε έναν δρομέα χρησιμοποιούμε την εντολή **DECLARE:**

  DECLARE cursor_name **CURSOR**
  FOR select_statement;

- **Άνοιγμα και τροφοδότηση δρομέα :** σε αυτό το βήμα εκτελείται η εντολή **select** και τροφοδοτείται ο δρομέας με τα αποτελέσματα. Το άνοιγμα του δρομέα γίνεται με την εντολή **OPEN:**

  OPEN cursor_name;

- **Επεξεργασία αποτελεσμάτων:** η εντολή **FETCH** μας επιτρέπει να προσπελάσουμε μια εγγραφή (γραμμή) του δρομέα. Μπορούμε να αποθηκεύσουμε τα δεδομένα της εγγραφής σε μία ή περισσότερες μεταβλητές.

  FETCH NEXT FROM cursor_name **INTO variable_list;**

Δρομείς (3)

- Ο SQL Server παρέχει την συνάρτηση **@@FETCH_STATUS** η οποία επιστρέφει την κατάσταση της πιο πρόσφατης εντολής FETCH. Αν η συνάρτηση **@@FETCH_STATUS** επιστρέψει την τιμή 0 τότε η εντολή FETCH εκτελέστηκε επιτυχώς.
- Μπορούμε να χρησιμοποιήσουμε την εντολή **WHILE** για να προσπελάσουμε διαδοχικά όλες τις εγγραφές ενός δρομέα:

```
WHILE @@FETCH_STATUS = 0
 BEGIN
    FETCH NEXT FROM cursor_name;
 END;
```

- **Κλείσιμο δρομέα:** Η εντολή CLOSE κλείνει ένα ανοικτό δρομέα.

```
CLOSE cursor_name;
```

- Η εντολή **DEALLOCATE** απελευθερώνει τις δομές δεδομένων που περιλαμβάνουν τον δρομέα:

```
DEALLOCATE cursor_name;
```

Δρομείς - Παράδειγμα

- Ο δρομέας που ακολουθεί εμφανίζει έναν κατάλογο με το ονοματεπώνυμο, τον αριθμό και το υπόλοιπο των λογαριασμών των πελατών (καταθετών) όλων των υποκαταστημάτων.

Δρομείς – Παράδειγμα (1)


```
DECLARE @fname VARCHAR(50), @lname VARCHAR(50), @accno VARCHAR(10),
@balance NUMERIC(18,0);

DECLARE cursor_customer CURSOR

FOR SELECT firstname, lastname, accounts.accno, balance

                FROM customers, depositors,accounts

            WHERE customers.cid=depositors.cid and depositors.accno=accounts.accno

          ORDER BY firstname;

OPEN cursor_customer;

FETCH NEXT FROM cursor_customer INTO @fname, @lname, @accno,@balance;

WHILE @@FETCH_STATUS = 0

  BEGIN

    PRINT @lname +' ' + @fname + ' ' + @accno + ' ' + CAST(@balance AS varchar);

    FETCH NEXT FROM cursor_customer INTO @fname, @lname, @accno,@balance;

  END;

CLOSE cursor_customer;

DEALLOCATE cursor_customer;
```

Πυροδότες (Triggers)

Πυροδότες (Triggers)

- Ένας πυροδότης είναι ένας μηχανισμός που καλείται όταν συμβαίνει μια συγκεκριμένη ενέργεια σε έναν συγκεκριμένο πίνακα. Κάθε πυροδότης έχει τρία γενικά μέρη:
  - **Ένα όνομα**
  - **Μία ενέργεια**
  - **Την εκτέλεση**
- Η **ενέργεια** μπορεί να είναι μια εντολή INSERT, UPDATE ή DELETE.
- Το **τμήμα εκτέλεσης** του πυροδότη περιέχει συνήθως μια διαδικασία (stored procedure) ή μια δέσμη.

Πυροδότες

- Οι πυροδότες είναι ουσιαστικά μικρά προγράμματα σε SQL τα οποία καλούνται **ΑΥΤΟΜΑΤΑ** μόλις εκτελεστεί μια ενέργεια (ερέθισμα).
- Τους πυροδότες **ΔΕΝ** μπορούμε να τους καλέσουμε αυτόνομα.

Χρήση πυροδοτών

- Οι πυροδότες χρησιμοποιούνται για την επιβολή περιορισμών διαδικαστικής ακεραιότητας.
- Οι πυροδότες μπορούν να χρησιμοποιηθούν για να εκτελέσουν μεταξύ άλλων και τις παρακάτω ενέργειες:
  - Να δημιουργήσουν μία διαδρομή παρακολούθησης σε έναν ή περισσότερους πίνακες της βάσης δεδομένων.
  - Να υλοποιήσουν επιχειρησιακούς κανόνες.
  - Να επιβάλουν σχεσιακή ακεραιότητα.

Σύνταξη Πυροδότη


```
CREATE TRIGGER TriggerName
ON TableName
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
--OUR LOGIC GOES HERE
END
```

Trigger Name

Table Name

On which operation this trigger should work

Logic of Trigger

Λειτουργία Πυροδότη

- Όταν δημιουργείτε μία ενέργεια πυροδότη, συνήθως πρέπει να δηλώσετε αν αναφέρεστε στην τιμή μιας στήλης, πριν ή μετά την επίδραση των ενεργειών του πυροδότη. Για τον λόγο αυτό δύο ειδικοί εικονικοί πίνακες χρησιμοποιούνται για να ελέγξουν την επίδραση των ενεργειών του πυροδότη:
  - **Inserted:** Περιέχει αντίγραφα των γραμμών που εισάγονται στον πίνακα. Χρησιμοποιείται αν ο όρος **INSERT** ή **UPDATE** καθορίζεται στην πρόταση **CREATE TRIGGER**.
  - **deleted:** Περιέχει αντίγραφα των γραμμών που διαγράφονται από τον πίνακα. Χρησιμοποιείται αν ο όρος **DELETE** ή **UPDATE** καθορίζεται στην πρόταση **CREATE TRIGGER**.

Παράδειγμα 1: Δημιουργία και ενεργοποίηση Πυροδότη

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

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

Ενεργοποίηση

```
INSERT INTO customers   ( cid,  firstname,  lastname,  city  )
VALUES (21, 'Καπέτης', 'Χρυσόστομος', 'Αθήνα');
You made one DML operation
(1 row(s) affected)
```

Παράδειγμα 2: Πυροδότης BorrowerLoanInfo

- Δημιουργούμε έναν πυροδότη ο οποίος κάθε φορά που ένας πελάτης παίρνει ένα νέο δάνειο ενημερώνει για το συνολικό ποσό των δανείων που έχει.
- Η συνθήκη ενεργοποίησης του πυροδότη είναι μία καταχώρηση στον πίνακα Borrower η οποία αναθέτει σε έναν υπάρχων πελάτη (cid) έναν νέο αριθμό δανείου (lnum).

Παράδειγμα 2: Δημιουργία πυροδότη BorrowerLoanInfo

Δημιουργία πυροδότη

```
CREATE TRIGGER BorrowerLoanInfo ON borrowers
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'Total loans of '+@customer+ ': '+CAST(@loans as
varchar(12))
END
```

Παράδειγμα 2: Πυροδότης BorrowerLoanInfo

Ενεργοποίηση πυροδότη

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

- Με την εισαγωγή ενός νέου δανείου στον Borrower με id=2, ενεργοποιείται ο πυροδότης και τυπώνει το συνολικό ποσό των δανείων του πελάτη.

(1 row(s) affected)

The new loan is accepted

Total loans of Αγγελίδου : 54100

(1 row(s) affected)

Παράδειγμα 4: Πυροδότης Open_Loan

- Δημιουργούμε έναν πυροδότη που απαγορεύει τα αρνητικά υπόλοιπα λογαριασμών. Η τράπεζα χειρίζεται τις αναλήψεις χωρίς αντίκρισμα ως εξής:
  - Θέτει το υπόλοιπο του λογαριασμού σε μηδέν.
  - Δημιουργεί ένα δάνειο με το επιπλέον ποσό.
  - Δίνει σε αυτό τα δάνειο έναν αριθμό δανείου ίδιο με τον αριθμό του λογαριασμού.
- Η συνθήκη ενεργοποίησης του πυροδότη είναι μία ενημέρωση στον πίνακα a*ccount* η οποία έχει ως αποτέλεσμα την δημιουργία αρνητικού υπολοίπου για ένα λογαριασμό.

Παράδειγμα 4: Πυροδότης Open_Loan

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))
    INSERT INTO   loans   (lnum,bcode,amount)
      VALUES (@lnum, @bcode, -@newbalance);
  INSERT 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
```

Παράδειγμα 4: Ενεργοποίηση πυροδότη Open_Loan

Ενεργοποίηση πυροδότη

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

Παράδειγμα 5: Πυροδότης modify_balance

- Να δημιουργήσετε έναν πυροδότη για την υλοποίηση μιας διαδρομής παρακολούθησης της δραστηριότητας στον πίνακα **account**. Συγκεκριμένα να δημιουργήσετε έναν πίνακα **audit_account**, που αποθηκεύει όλες τις τροποποιήσεις της στήλης **balance** του πίνακα **accounts**.

Παράδειγμα 5: Δημιουργία πυροδότη modify_balance

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
```

Παράδειγμα 5: Ενεργοποίηση πυροδότη modify_balance

Ενημέρωση του υπολοίπου του λογαριασμού A906.

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

Εμφάνιση/Διαγραφή Πυροδοτών

- Η παρακάτω εντολή επιστρέφει όλους τους ενεργούς πυροδότες της βάσης.

```
SELECT *
FROM sys.triggers
WHERE is_disabled = 0
```

- Οι πυροδότες μπορούν να διαγραφούν με την εντολή DROP.

```
DROP TRIGGER trigger_customers;
```

Απενεργοποίηση/Τροποποίηση Πυροδότη

DISABLE/ENABLE TRIGGER trigger_customers

ON customers

- Η παραπάνω εντολή απενεργοποιεί/ενεργοποιεί έναν πυροδότη.

ALTER TRIGGER trigger_customers

ON customers

- Οι πυροδότες μπορούν να τροποποιηθούν με την εντολή ALTER.

Ορισμός Προτεραιότητας Πυροδότη

- Σε έναν συγκεκριμένο πίνακα μπορεί να υπάρχουν περισσότεροι από ένας πυροδότες.
- Ο sql server σας δίνει τη δυνατότητα να ορίσετε την σειρά εκτέλεσης πολλαπλών πυροδοτών που ορίζονται για ένα δεδομένο συμβάν.
- Χρησιμοποιώντας την διαδικασία συστήματος sp_settriggerorder, μπορείτε να καθορίσετε ότι ένας από τους πυροδότες που σχετίζονται με έναν πίνακα μπορεί να εκτελεστεί πρώτος (first) ή τελευταίος (lats) για κάθε ενέργεια. Π.χ.

  execute sp_settriggerorder @triggername='modify_balance'

  @order = 'first' @smttype='update'

- Η παραπάνω εντολή ορίζει ότι ο πυροδότης 'modify_balance' θα εκτελεστεί πρώτος μεταξύ όλων των πυροδοτών που έχουν δημιουργηθεί για τον πίνακα account και ενεργοποιούνται μετά την εκτέλεση μιας εντολής update στον συγκεκριμένο πίνακα.

Παρατηρήσεις

- Ένα βασικό πρόβλημα με τους πυροδότες είναι ότι οι χρήστες πολλές φορές ενεργοποιούν διαδικασίες με τα δεδομένα χωρίς να το γνωρίζουν.
- Γενικότερα οι διαδικασίες (stored procedures) μπορούν να αντικαταστήσουν τους πυροδότες και είναι ένας πιο ενδεδειγμένος τρόπος υλοποίησης για διασφάλιση της ακεραιότητας και της ασφάλειας των δεδομένων.
- Οι πυροδότες καθυστερούν την διαδικασία μιας συναλλαγής, ιδιαίτερα στις περιπτώσεις που απαιτούν πρόσβαση και σε άλλους πίνακες.
- Η εκτέλεση ενός πυροδότη μπορεί να προκαλέσει την αλυσιδωτή εκτέλεση πυροδοτών με απρόσμενα αποτέλεσμα.