

TRIGGEREK, HIBAKEZELÉS ÉS SORMUTATÓK

ADATBÁZISOK SEPSISZENTGYÖRGY

Dóka - Molnár Andrea

andrea.molnar@ubbcluj.ro



BABEŞ-BOLYAI TUDOMÁNYEGYETEM
Matematika és Informatika Kar



Feladatok

<https://bit.ly/3fBme85>

- Eddig: az adatbázisba tárolásra kerülő adatok helyességének az ellenőrzése \leftrightarrow megszorítások (SQL2)

- Eddig: az adatbázisba tárolásra kerülő adatok helyességének az ellenőrzése ↔ **megszorítások** (SQL2)
 - Megszorítás: kapcsolat adatelemek között, melyet az ABKR köteles érvényesíteni/ellenőrizni. ↔ **adat változtatásakor**

Triggererek

- Eddig: az adatbázisba tárolásra kerülő adatok helyességének az ellenőrzése \leftrightarrow megszorítások (SQL2)
 - Megszorítás: kapcsolat adatelemek között, melyet az ABKR köteles érvényesíteni/ellenőrizni. \leftrightarrow adat változtatásakor
- Ezentúl: megszorítások + triggererek \leftrightarrow *aktív elemek*

Triggerek

- = (SQL3) „speciális tárolt eljárás”, jelentése: elsüt, kivált.
 - Automatikusan végrehajtódik/elindul „elsütő események” hatására.
 - A felhasználó meghatározhatja, *mikor* történjen az ellenőrzés.
- 3 típusú trigger:
 - DML trigger
 - DDL trigger
 - (Bejelentkezési) Logon trigger

DML triggerek

- Trigger részei:
 - **esemény (event)** – bekövetkezése a trigger automatikus kioldását eredményezi
 - DML triggerek esetén: egy táblára vonatkozó adatmódosító művelet(ek) (**INSERT, DELETE, UPDATE utasítás**).
 - ↔ Egy trigger mindig egy táblára vonatkozik.
- **művelet (action)** – egy eljárás, melyet végrehajt a rendszer, amikor a trigger aktiválva van.

Innentől: DML trigger ↔ trigger

Triggererek osztályozása

- „Esemény” típusa szerint - beszúrás, módosítás, törlés
⇒ 3 típusú trigger: `INSERT`, `UPDATE`, `DELETE` trigger
- Az „esemény” bekövetkezéséhez képest a kioldódás megtörténtének ideje szerint:
 - 2 típusú trigger: `INSTEAD OF`, `AFTER/FOR`
 - Mindegyik definiálható minden típusú eseménynél.
 - `INSTEAD OF`: max 1 db/tábla/esemény
 - `AFTER/FOR`: több db/tábla/esemény
- Oracle, MySQL: + `BEFORE` trigger

Trigger részei (folyt.)

- művelet (action) – egy eljárás, melyet végrehajt a rendszer, amikor a trigger aktiválva van.

Trigger részei (folyt.)

- **művelet (action)** – egy eljárás, melyet végrehajt a rendszer, amikor a trigger aktiválva van.
 - Egy lehetséges művelet az esemény hatásának valamilyen módon történő megváltoztatásra.
 - Megakadályozhatja a kiváltó esemény megtörténtét VAGY meg nem történtté teheti azt (pl. kitörölheti az épp felvitt sorokat).

Trigger részei (folyt.)

- művelet (action) – egy eljárás, melyet végrehajt a rendszer, amikor a trigger aktiválva van.
 - A trigger művelet része hivatkozhat: a triggert kiváltó parancs által módosított sorok régi és új értékeire:

Trigger részei (folyt.)

- **művelet (action)** – egy eljárás, melyet végrehajt a rendszer, amikor a trigger aktiválva van.
 - A trigger művelet része hivatkozhat a trigger kiváltó parancs által módosított sorok **régi** és **új** értékeire:
 - **INSERT trigger**: a műveletet kiváltó esemény által beszűrt sorok új értékeire – **inserted** tábla/sorváltozó
 - **DELETE trigger**: törölt sorok régi értékeire – **deleted** tábla/sorváltozó
 - **UPDATE trigger**: módosított sorok régi ill. új értékeire – **inserted**- és **deleted** tábla/sorváltozó
 - Végrehajthat új lekérdezéseket, változtathatja az adatbázist (más táblákat is!).

Triggerek létrehozása és módosítása

- **Létrehozás/módosítás:**

```
CREATE/ALTER TRIGGER <trigger_név>  
ON <táblanév>  
INSTEAD OF|AFTER|FOR  
INSERT|[, ]UPDATE|[, ]DELETE  
AS  
[BEGIN]  
    T-SQL utasítások  
[END]
```

Trigger törlése

■ Törlés: DROP

Ellenőrzéssel:

```
IF EXISTS (SELECT * FROM sys.objects
  WHERE object_id = OBJECT_ID(N'<nev>' )
    AND type IN (N'TR' ) )
DROP TRIGGER <nev>
```

SQL 2016-tól: IF EXISTS argumentum használatával

```
DROP {TRIGGER} [IF EXISTS]
{ [schema_name.]<nev> } [ , ...n ]
```

Példa: DROP TRIGGER IF EXISTS <nev>

Triggerek letiltása, engedélyezése

- Két állapot lehetséges:
 - **Engedélyezett** - alapértelmezett
 - **Tiltott**: ALTER TABLE segítségével (ON záradékban levő tábla)

```
ALTER TABLE <táblanév>
```

```
    ENABLE | DISABLE TRIGGER <trigger_név> / ALL
```

Pl. ALTER TABLE Szallit

```
    DISABLE TRIGGER <trigger_név>
```

```
ALTER TABLE Szallit
```

```
    ENABLE TRIGGER ALL
```

Példa

Nyaralok(NyaraloID, ..., Ar, Osszbevetel)

Berlesek(BerlesID, *NyaraloID*, BerloNev, KDatum, Idotart, Ertek)

Naplo(ID, Datum, esemeny) – ID-identity típusú

Naplózzuk a Nyaralak táblán történt módosításokat, a naplózott események nyomon követésére használjuk a Naplo táblát!

Példa

Nyaralok(NyaraloID, ..., Ar, Osszbevetel)

Berlesek(BerlesID, *NyaraloID*, BerloNev, KDatum, Idotart, Ertek)

Naplo(ID, Datum, esemeny) – **ID-identity** típusú

```
CREATE TRIGGER trNyaraloBeszur_ForI
ON Nyaralok
FOR INSERT
AS BEGIN
SET NOCOUNT ON
DECLARE @esemeny VARCHAR(50), @NyID INT
SELECT @NyID = NyaraloID
FROM INSERTED
SET @esemeny = 'Beszúrtuk a ' +
              CAST(@NyID AS VARCHAR(30)) +
              ' ID-jú nyaralót.'
INSERT INTO Naplo VALUES
              (GETDATE(), @esemeny)
END
```

Példa (folyt.)

Nyaralok(NyaraloID, ..., Ar, Osszbevetel)

Berlesek(BerlesID, *NyaraloID*, BerloNev, KDatum, Idotart, Ertek)

Naplo(ID, Datum, esemeny) – ID-identity típusú

```
CREATE TRIGGER trNyaraloTorol_Ford
ON Nyaralok
FOR DELETE
AS BEGIN
SET NOCOUNT ON
DECLARE @esemeny VARCHAR(50), @NyID INT
SELECT @NyID = NyaraloID
FROM DELETED
SET @esemeny = 'Töröltük a ' +
              CAST(@NyID AS VARCHAR(30)) +
              ' ID-jú nyaralót.'
INSERT INTO Naplo VALUES
              (GETDATE(), @esemeny)
END
```

Lehetséges-e egyetlen trigger írása a Naplo tábla feltöltésére?

Példa

Nyaralok(NyaraloID, ..., Ar, Osszbevetel)

Berlesek(BerlesID, *NyaraloID*, BerloNev, KDatum, Idotart, Ertek)

Naplo(ID, Datum, esemeny) – **ID-identity** típusú

```
CREATE TRIGGER trNyaraloBeszur_ForID
ON Nyaralok
FOR INSERT, DELETE
AS BEGIN
SET NOCOUNT ON
```

Példa

Nyaralok(NyaraloID, ..., Ar, Osszbevetel)

Berlesek(BerlesID, *NyaraloID*, BerloNev, KDatum, Idotart, Ertek)

Naplo(ID, Datum, esemeny) – **ID-identity** típusú

```
CREATE TRIGGER trNyaraloBeszur_ForID
ON Nyaralok
FOR INSERT, DELETE
AS BEGIN
SET NOCOUNT ON
DECLARE @esemeny VARCHAR(50),
        @NyIDi INT, @NyIDd INT
IF (SELECT COUNT(*) FROM deleted) = 1
BEGIN
    PRINT 'Törlés.'
    SELECT @NyID = NyaraloID
    FROM DELETED
    SET @esemeny = 'Töröltük a ' +
        CAST(@NyIDd AS VARCHAR(30)) +
        ' ID-jú nyaralót.'
END
...
```

Példa

Nyaralok(NyaraloID, ..., Ar, Osszbevetel)

Berlesek(BerlesID, *NyaraloID*, BerloNev, KDatum, Idotart, Ertek)

Naplo(ID, Datum, esemeny) – ID-identity típusú

...

ELSE

IF (SELECT COUNT(*) FROM inserted) = 1

BEGIN

PRINT 'Beszúrás.'

SELECT @NyID = NyaraloID

FROM INSERTED

SET @esemeny = 'Beszúrtuk a ' +
CAST(@NyIDd AS VARCHAR(30)) +
' ID-jú nyaralót.'

END

INSERT INTO Naplo VALUES (GETDATE(), @esemeny)

END --(vége) trigger

Hogyan járnánk el, ha a Nyaralok táblában történő módosításokat is szeretnénk naplózni?

Példa

Nyaralok(NyaraloID, ..., Ar, Osszbevetel)

Berlesek(BerlesID, *NyaraloID*, BerloNev, KDatum, Idotart, Ertek)

Naplo(ID, Datum, esemeny) – ID-identity típusú

```
CREATE TRIGGER trNyaraloBeszur_ForID
```

```
ON Nyaralok
```

```
FOR INSERT, DELETE
```

```
AS
```

```
BEGIN
```

```
SET NOCOUNT ON
```

Mi a teendő, ha a beszúrandó/törlendő

sorok száma > 1?

```
...  
IF (SELECT COUNT(*) FROM deleted) = 1
```

```
    BEGIN ... END
```

```
...  
ELSE
```

```
IF (SELECT COUNT(*) FROM inserted) = 1
```

```
    BEGIN ... END
```

```
INSERT INTO Naplo VALUES (GETDATE(), @esemeny)
```

```
END
```

Visszatekintés: Globális változók

- Rendszer által definiált függvények (nem kell őket deklarálni).
- Minden globális változó a szerverről vagy az aktuális felhasználó session-jéről tárol információkat.
- Változó beazonosítása: @@valtozonev;
- Példák:
 - @@ERROR – legutolsó T-SQL utasítás hibakódja (0-nem történt hiba)
 - @@IDENTITY, SCOPE_IDENTITY() – legutolsó beszúrt sor IDENTITY típusú mezőjének értéke (különböző értékek lehetnek hatásköröktől függően)
 - @@ROWCOUNT – legutolsó T-SQL utasítás által feldolgozott sorok száma (!SET NOCOUNT ON!)

Példa

Nyaralok(NyaraloID, ..., Ar, Osszbevetel)

Berlesek(BerlesID, NyaraloID, BerloNev, KDatum, Idotart, Ertek)

Feltételezzük, hogy meg szeretnénk oldani, hogy a Berlesek táblába való beszúrás esetén a rendszer számolja ki a bérlés értékét, majd automatikusan aktualizálja a Nyaralok tábla Osszbevetel mezőjének értékét (Berlesek.Ertek és Nyaralok.Ar attribútumok értékét figyelembe véve)! Hogyan járnánk el?

Példa

```
Nyaralok(NyaraloID, ..., Ar, Osszbevetel)
```

```
Berlesek(BerlesID, NyaraloID, BerloNev, KDatum, Idotart, Ertek)
```

```
CREATE TRIGGER trBerlesErtek_ForI
```

```
ON Berlesek
```

```
FOR INSERT
```

```
AS BEGIN
```

```
IF @@ROWCOUNT=0 RETURN -- no rows affected
```

```
IF @@ROWCOUNT>1
```

```
  BEGIN
```

```
    PRINT 'Egyszerre csak egy sort szűrj be!'
```

```
    RETURN
```

```
  END
```

```
SET NOCOUNT ON
```

Trigger esetén használható RETURN,
de érték nem téríthető vissza.



Példa

```
Nyaralok(NyaraloID, ..., Ar, Osszbevetel)
```

```
Berlesek(BerlesID, NyaraloID, BerloNev, KDatum, Idotart, Ertek)
```

...

```
DECLARE @ertek int, @Ar INT, @Idot INT,  
        @NyID int, @KD date
```

```
SELECT @NyID=NyaraloID, @KD=KDatum  
FROM inserted I
```

```
SELECT @Ar=Ar  
FROM Nyaralok
```

```
WHERE NyaraloID =@NyID
```

```
SET @idot = (SELECT Idotartam  
             FROM inserted)
```

```
SET @ertek= @Ar*@idot
```

...

Példa

```
Nyaralok(NyaraloID, ..., Ar, Osszbevetel)
```

```
Berlesek(BerlesID, NyaraloID, BerloNev, KDatum, Idotart, Ertek)
```

...

```
UPDATE Berlesek
```

```
  SET Ertek=@ertek
```

```
  WHERE NyaraloID=@NyID AND KDatum=@KD
```

```
  IF @@ERROR<>0
```

```
    RAISERROR('Hiba a bérlés  
               módosításánál.',17,1)
```

```
  ELSE
```

```
    PRINT 'Bérlés módosítása sikeres!'
```

...

Példa

```
Nyaralok(NyaraloID, ..., Ar, Osszbevetel)
```

```
Berlesek(BerlesID, NyaraloID, BerloNev, KDatum, Idotart, Ertek)
```

...

```
UPDATE Nyaralok
```

```
  SET Osszbevetel+=@ertek
```

```
  WHERE NyaraloID=@NyID
```

```
IF @@ERROR<>0
```

```
  RAISERROR('Hiba a nyaraló  
            módosításánál.',17,2)
```

```
ELSE
```

```
  PRINT 'Nyaraló módosítása sikeres!'
```

```
END --(vége) trigger
```

Példa (TRY-CATCH blokkal)

```
Nyaralok(NyaraloID, ..., Ar, Osszbevetel)
```

```
Berlesek(BerlesID, NyaraloID, BerloNev, KDatum, Idotart, Ertek)
```

```
CREATE TRIGGER trBerlesErtek_ForI
ON Berlesek
FOR INSERT
AS BEGIN
    IF @@ROWCOUNT=0 RETURN
    IF @@ROWCOUNT>1 BEGIN
        PRINT 'Egyszerre csak egy sort szúrj be!'
        RETURN
    END
    SET NOCOUNT ON
    BEGIN TRY
        DECLARE @ertek int, @NyID int, @KD date
        SELECT @NyID=NyaraloID, @KD=KDatum
        FROM inserted i
        SELECT @ertek= Ar*Idotartam
        FROM Nyaralok ny JOIN inserted i
            ON ny.NyaraloID=i.NyaraloID
        WHERE NyaraloID =@NyID
```

```
...
```

Példa (TRY-CATCH blokkal)

```
Nyaralok(NyaraloID, ..., Ar, Osszbevetel)
```

```
Berlesek(BerlesID, NyaraloID, BerloNev, KDatum, Idotart, Ertek)
```

```
...
```

```
UPDATE Berlesek
```

```
  SET Ertek=@ertek
```

```
  WHERE NyaraloID=@NyID AND KDatum=@KD
```

```
PRINT 'Bérlés módosítása sikeres!'
```

```
UPDATE Nyaralok
```

```
  SET Osszbevetel+=@ertek
```

```
  WHERE NyaraloID=@NyID
```

```
PRINT 'Nyaraló módosítása sikeres!'
```

```
END TRY
```

```
BEGIN CATCH
```

```
  THROW
```

```
END CATCH
```

```
END
```

Példa (TRY-CATCH blokkal)

```
Nyaralok(NyaraloID, ..., Ar, Osszbevetel)
```

```
Berlesek(BerlesID, NyaraloID, BerloNev, KDatum, Idotart, Ertek)
```

```
...
```

```
UPDATE Berlesek
```

```
SET Ertek=@ertek
```

```
WHERE NyaraloID=@NyID AND KDatum=@KD
```

```
PRINT 'Bérlés módosítása sikeres!'
```

```
UPDATE Nyaralok
```

```
SET Osszbevetel=ISNULL(Osszbevetel, 0) + @ertek
```

```
WHERE NyaraloID=@NyID
```

```
PRINT 'Nyaraló módosítása sikeres!'
```


```
END TRY
```

```
BEGIN CATCH
```

```
THROW
```

```
END CATCH
```

```
END
```



Speciális esetre is működik: ha a bevétel értéke NULL.

Triggerek segítségével megoldható feladatok

Triggerek segítségével megoldható feladatok

- származtatott oszlopértékek automatikus generálása;
- események naplózása (átlátszó megoldás);
- statisztikák készítése: pl. a táblához való hozzáférésről; ki milyen táblákon végzett valamilyen műveletet → segítség biztonság ellenőrzésében;
- hivatkozási épség megszorítások osztott adatbázis esetén;
- másolt táblák osztott adatbázisok esetén.

Triggerek sajátosságai

- Trigger - erős mechanizmus az adatbázisban végzett módosítások könnyebb feldolgozására, **DE: elővigyázatosan kell használni őket.**
- Egy trigger mindig egy táblára vonatkozik.
- Aktív adatbázis - melyben triggerek is implementálva vannak.
- Triggerek halmazának hatása lehet nagyon komplex → aktív adatbázis karbantartása nehezzé válhat.
- Tábla törlése ↔ a hozzá tartozó triggerek is törlődnek.
- Nem-naplózott (non-logged) műveletek (pl. TRUNCATE TABLE) nem aktiválják a triggereket.

Triggerok sajátosságai

- Ha egy parancs egynél több triggert aktivál, az ABKR mindegyiket végrehajtja, **tetszőleges sorrendben**.
 - Megj. **sp_settriggerorder** – rendszerszintű tárolt eljárás triggerok végrehajtási sorrendjének megadására
- Sokszor nem tudhatjuk, a trigger mely program mellékhatásaként lett végrehajtva.
- Egy trigger művelet része aktiválhat egy másik triggert.
 - Sajátos eset: előfordulhat, hogy ismét aktiválja az előző triggert → rekurzív triggerok.
 - Az ilyen lánc aktiválásakor mivel nem tudjuk, hogy az ABKR milyen sorrendben hajtja őket végre, nehezen lehet eldönteni, hogy mi lesz az összhatásuk.

Triggerek és megszorítások kapcsolata

További példa

```
Nyaralok(NyaraloID, NyaraloNev, TulajID, ...)
```

```
CREATE TRIGGER trNyaralok_InsteadOfUpdate
ON Nyaralok
INSTEAD OF UPDATE AS
  IF @@ROWCOUNT=0 RETURN
  SET NOCOUNT ON
  BEGIN TRY
    IF (UPDATE(NyaraloID))
      RAISERROR('ID nem módosítható.',16,1)
  DECLARE @uNyID INT,
          @rNyNev VARCHAR(30), @uNyNev VARCHAR(30),
          @rTID INT, @uTID INT [,...] -- a Nyaralok
                                     tábla többi attribútumának régi és
                                     új értékeinek tárolására
  SELECT @rNyNev=NyaraloNev, @rTID=TulajID, ...
  FROM deleted
  SELECT @uNyID=NyaraloID, @uNyNev=NyaraloNev,
         @uTID=TulajID, ...
  FROM inserted
```

...

További példa

Tulajok(TulajID, Nev,...)-Nev NOT NULL megszorítással ellátva
Nyaralok(NyaraloID, NyaraloNev, *TulajID*, ...)

...

```
IF (@rTID<>@uTID)
BEGIN
    DECLARE @tNev VARCHAR(30)
    SELECT @tNev=Nev
    FROM Tulajdonosok
    WHERE TulajID=@uTID
    IF @tNev IS NULL
        RAISERROR('Nemlétező TulajID-ra szeretnéd
                    módosítani.',14,1)
    UPDATE Nyaralok
    SET TulajID=@uTID
    WHERE NyaraloID=@uNyID
END
```

További példák

```
Nyaralok(NyaraloID, NyaraloNev, TulajID, ...)
```

```
...
```

```
IF (@uNyNev<>@rNyNev)
    UPDATE Nyaralok
    SET NyaraloNev=@uNyNev
    WHERE NyaraloID=@uNyID
```

... --a többi attribútumra is ellenőrizzük mindezeket

```
END TRY
BEGIN CATCH
    THROW
END CATCH
```

■ **Megj.: Megj.:** Az UPDATE() függvény **igazat térít vissza** akkor is, ha **ugyanarra az értékre módosítjuk** az attribútumot. \Rightarrow INSERTED és DELETED tábla megfelelő attribútumainak összehasonlítása sok esetben hatékonyabb.

Feladat

Országok (OrszágID, OrszágNev)
Tulajdonosok (TulajID, Nev, Cim, Email)
Nyaralók (NyaraloID, NyaraloNev, NyaraloCim,
OrszágID, TulajID, Ferohely, Ar,...)
Felszerelések (FelszerelesID, FelszerelesNev)
Felszerelései (Felszerelesei, NyID, FID)
Bérlések (BerlesID, NyaraloID, BerloNev,
KezdoDatum, Idotartam, Ertek)

Tr1) Egy adott tulajdonos szeretné törölni adatait az adatbázisunkból. Írjunk megfelelő trigger-t, mely segítségével ki tudjuk őrölni! Az integritási megszorításokat vegyük figyelembe!

További példák

Egy software cég projektjeit tároló adatbázis sémája:

Employees (EmployeeID, FirstName, LastName, ...)

Customers (CustomerID, Name, Address, ...)

[megrendelők]

ProjStates (ProjStateID, Name) (Name-projekt állapota, pl. 1-tervezett, 2-aktív, 3-megszakítva, 4-befejezett stb.)

Projects (ProjectID, Title, CustomerID, ProjManID, ProjStateID, PlannedStartDate, PlannedEndDate, RealStartDate, RealEndDate, ...) [rendelt projektek]

Tasks (TaskID, Name, ProjectID, Finished, EmployeeID, PlannedStartDate, PlannedEndDate, RealStartDate, RealEndDate, ...) [feladatok]

Activities (ActivID, Text, TaskID, Finished, PlannedStartDate, PlannedEndDate, RealStartDate, RealEndDate, ...) [tevékenységek]

További példák (Proj_upd trigger)

ProjStates (ProjStateID, Name)

Projects (ProjectID,..., *ProjStateID*,...)

Tasks (TaskID, Name, *ProjectID*, Finished, *EmployeeID*,...)

```
CREATE TRIGGER Proj_upd ON Projects
    FOR UPDATE
```

```
AS
```

```
declare @vProjID int, @vProjStID as int
/* Tests if the ProjStateID is modified to
be finished, if there are some tasks
belonging to the project, which has Finished
on 0 */
set nocount on
select @vProjStID = i.ProjStateID,
       @vProjID = i.ProjectID from inserted i
```

További példa (Proj_upd trigger)

ProjStates (ProjStateID, Name)

Projects (ProjectID,..., ProjStateID,...)

Tasks (TaskID, Name, ProjectID, Finished, EmployeeID,...)

```
if @vProjStID = 4          (ProjStateID=4-projekt állapota
    begin                  befejezett)
        if exists (select TaskID from Tasks
            where ProjectID = @vProjID and
            Finished = 0)
            begin
                raiserror ('There are tasks
                belonging to the project, which are
                not finished.',16,1)
                ROLLBACK TRANSACTION
            end
        end
    end
set nocount off
```

További példák (Task_upd trigger)

Projects (ProjectID,..., *ProjStateID*,...)

Tasks (TaskID, Name, *ProjectID*, Finished, *EmployeeID*,...)

Activities (ActivID, Text, *TaskID*, Finished,...)

```
CREATE TRIGGER Task_upd ON Tasks
    FOR INSERT, UPDATE
    AS
```

```
declare @task_finished tinyint,
        @vTaskID int, @vfinished tinyint,
        @vProjID int, @proj_state int
```

```
/* Tests if the Finished field is modified
to 1, if there are some activities belonging
to the task, which has Finished on 0 */
```

```
set nocount on
```

```
select @task_finished = i.Finished,
        @vTaskID=i.TaskID,
        @vProjID = i.ProjectID from inserted i
```

További példák (Task_upd trigger)

Projects (ProjectID,..., *ProjStateID*,...)

Tasks (TaskID, Name, *ProjectID*, Finished, *EmployeeID*,...)

Activities (ActivID, Text, *TaskID*, Finished,...)

```
if @task_finished = 1
begin
    if exists (select ActivID from
        Activities where TaskID = @vTaskID and
        Finished = 0)
        begin
            raiserror ('There are activities
                belonging to the task, which are not
                finished.',16,1)
            ROLLBACK TRANSACTION
        end
    end
end
```

További példák (Task_upd trigger)

Projects (ProjectID,..., ProjStateID,...)

Tasks (TaskID, Name, ProjectID, Finished, EmployeeID,...)

Activities (ActivID, Text, TaskID, Finished,...)

```
/* Tests if the Finished field is modified  
to 0, the project which it belongs has been  
marked Finished (ProjStateID= 4) */
```

```
if @task_finished = 0
```

```
begin
```

```
    select @proj_state = ProjStateID from  
        Projects where ProjectID = @vProjID
```

```
    if @proj_state = 4
```

```
        begin
```

```
            raiserror ('The project from  
which the task belongs was marked  
finished.',16,1)
```

```
            ROLLBACK TRANSACTION
```

```
        end
```

```
    end
```

```
set nocount off
```

Tervezési irányelvek

- Amikor csak lehet, megszorításokkal dolgozzunk.
- Trigger korlátozása max 60 sorra.
 - Több művelet esetén: tárolt eljárás írása + meghívása triggerből.
- Rekurzív triggerek kerülése.
- Triggerek mértékkel történő használata: minden egyes adatkezelési művelet esetén meghívásra kerülnek, és nagyon sok munkát igényelhetnek.
- Csak globális műveletek elvégzésére használjuk.
 - Globális művelet \leftrightarrow minden felhasználó esetében szükségesek

További infók

- Kurzorok
- Változók
- Procedurális szerkezetek
- Tárolt eljárások: SP1
- Függvények: F1
- Triggerek: Tr1