VEZÉRLÉSI SZERKEZETEK & HIBAKEZELÉS

ADATBÁZISOK 1

Dóka - Molnár Andrea

andrea.molnar@math.ubbcluj.ro

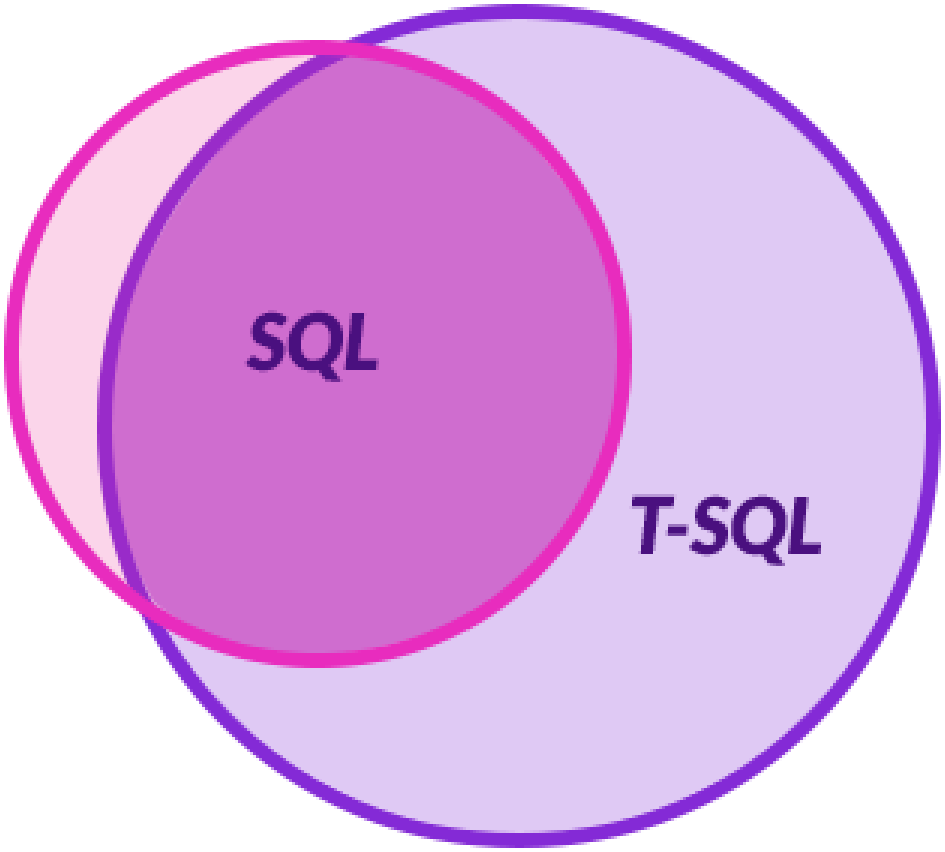
BABEŞ-BOLYAI TUDOMÁNYEGYETEM

Matematika és Informatika Kar

Áttekintés/Tematika

1. Adatbázis-kezelő rendszerek általános jellemzői.
2. Adatmodellezés.
3. A relációs algebra műveletei, használata.
4. Az SQL nyelv részei (MSSQL specifikusan):
   - **Vezérlési szerkezetek**
   - **Tárolt eljárások**
   - **Függvények**
5. Relációs adatbázisok tervezése
6. NoSQL adatbázisok

Transact-SQL



SQL - ANSI/ISO standard programozási nyelv

Kiterjesztések relációs ABKR-k esetén:

MS SQL – Transact SQL

Oracle – PL/SQL

Változók SQL-ben

- Változó beazonosítása:
  - Lokális (felhasználó által definiált) változó: @változónév
  - Globális változó: @@változónév
- Lokális változók deklarálása:

```
DECLARE @<változónév> <típus>
```

Pl. `DECLARE @v INT`

Változók SQL-ben

- Értékadás lokális változók esetén:

1.mód:

```
SET @<változónév>=<kifejezés>
```

- ahol `<kifejezés>` lehet: konstans vagy egyetlen értéket visszaadó lekérdezés

2.mód:

```
SELECT @<változónév1>=<Ai>,
       @<változónév2>=<Aj>, …
FROM R
```

- ahol `R(A1,A2,...,Aj, ..., Ai,...)` sémájú reláció; `Ai, Aj,… ∈ R` - típusuk egyezik `@<változónév1>, @<változónév2>,…` típusával
- több változónak is adhatunk ilyen formában értéket.

3.mód:

```
DECLARE @<változónév> <típus>=<kifejezés>
```

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`– legutolsó beszúrt sor IDENTITY típusú mezőjének értéke (különböző értékek lehetnek hatásköröktől függően) (lsd még: `SCOPE_IDENTITY()`)
  - `@@ROWCOUNT – !SET NOCOUNT ON!` – legutolsó T-SQL utasítás által feldolgozott sorok száma

Értékek kiíratása

- PRINT vagy SELECT utasítások segítségével.
  - SELECT - Results; PRINT - Messages
- Példák:

```
DECLARE @v INT
SELECT @v=AVG(Ar) AtlagAr
FROM Szallit
SELECT @v --1.mód
PRINT @v  --2.mód
PRINT 'Szállított áruk átlagára: '+
                CONVERT(VARCHAR(2),@v)  --3.mód
```

Vezérlési szerkezetek

- Összetett utasítás (blokk): begin … end
- Feltételes utasítások (if…else)
- CASE szerkezet

Vezérlési szerkezetek

- Összetett utasítás (blokk): begin … end
  - Több SQL utasítás **begin** és **end** között.
  - Lokális változók deklarálása megengedett blokkon belül.

Vezérlési szerkezetek

- Összetett utasítás(blokk): begin … end
  - Több SQL utasítás **begin** és **end** között.
  - Lokális változók deklarálása megengedett blokkon belül.

- Feltételes utasítás (if-else)
  - `IF [NOT] EXISTS` ([több sort/oszlopot visszatérítő `SELECT` utasítás])
  - `IF [NOT]` ([skalárt visszatérítő `SELECT` utasítás])

Feltételes utasítás

- Egyszerű `IF` utasítás

```
IF DATEPART(weekday, GETDATE()) = 6
[BEGIN]
  PRINT 'Ma péntek van.'
[END]
```

```
DECLARE @alksz INT
  SET @alksz = (SELECT COUNT(*) FROM
                Alkalmazottak
                WHERE ReszlegID = 1)
  IF @alksz > 100
      PRINT 'Túl sok alkalmazott az 1-es
részlegen.'
```

Alkalmazottak(SzemSzám, Név, Fizetés, Cím, *RészlegID*)

Feltételes utasítás

- IF/ELSE szerkezet *(lok. változó használatával)*

```
DECLARE @alksz INT
SET @alksz = (SELECT COUNT(*)
              FROM Alkalmazottak
              WHERE ReszlegID = 1)
IF @alksz > 100
   [BEGIN]
    PRINT 'Túl sok alkalmazott az 1-es
           részlegen.'
   [END]
ELSE
  [BEGIN]
   PRINT 'Az alkalmazottak száma megfelelő.'
  [END]
```

Feltételes utasítás

- `IF/ELSE` szerkezet *(lok. változó és SET parancs nélkül)*

```
IF (SELECT COUNT(*)
     FROM Alkalmazottak
     WHERE ReszlegID = 1 ) > 100
   [BEGIN]
    PRINT 'Túl sok alkalmazott az 1-es
                részlegen.'
   [END]
ELSE
  [BEGIN]
   PRINT 'Az alkalmazottak száma megfelelő.'
  [END]
```

Mikor van szükség lokális változóra és mikor nincs?

Feltételes utasítás

- IF EXISTS szerkezet

```
IF EXISTS (SELECT *
           FROM Alkalmazottak
           WHERE ReszlegID = 15)
  [BEGIN]
   PRINT 'Találtam alkalmazotta(ka)t
          a 15-ös részlegben.'
  [END]
```

Feltételes utasítás

- IF NOT szerkezet

```
IF NOT
  (SELECT COUNT(*)
   FROM Szerzodesek
   WHERE Datum BETWEEN '1/1/2016' AND
                       '12/31/2016') > 0
         --VAGY egyszerűen: YEAR(Datum)=2016
   [BEGIN]
    PRINT 'Nem található szerződés 2016-ból.'
   [END]
```

Szerződések(SzerződID, Dátum, Részletek, *VevőID*)

CASE utasítás

- **SELECT, UPDATE, SET utasítások esetén.**

Példa:

```
SELECT Nev, Kategoria = CASE
  WHEN AVG(Ar) BETWEEN 10 AND 24
            THEN 'Átlagos szállító.'
  WHEN AVG(Ar) BETWEEN 25 AND 50
            THEN 'Drága szállító.'
  WHEN AVG(Ar) > 50 THEN 'Nagyon drága szállító.'
  ELSE 'Legjobb fogás.'
  END
 FROM Szallit sz JOIN Szallitok s
    ON sz.SzallKod=s.SzallKod
 GROUP BY s.SzallKod, Nev
```

Szallitok (SzallKod, Nev, Cim), SzallKod-identity típusú
Szallit(*SzallKod, AruKod*, Ar)

CASE utasítás

Példa:

Szallitok (SzallKod, Nev, Cim), SzallKod-identity típusú
Szallit(*SzallKod, AruKod*, Ar)

```
SELECT Nev, Kategoria = CASE
 WHEN AVG(Ar) BETWEEN 10 AND 24 THEN 'Átlagos szállító'
 WHEN AVG(Ar) BETWEEN 25 AND 50 THEN 'Drága szállító.'
 WHEN AVG(Ar) > 50 THEN 'Nagyon drága szállító.'
 ELSE 'Legjobb fogás.'
END
FROM Szallit sz JOIN Szallitok s
   ON sz.SzallKod=s.SzallKod
GROUP BY s.SzallKod, Nev
```

Kimenet:

	Nev	Kategoria
1	Rolicom SA	Atlagos szallito
2	Metro SA	Nagyon draga szallito.
3	Colirom SA	Draga szallito

Vezérlési szerkezetek (folyt.)

- Ismétlő ciklusok
- Vezérlésátadó utasítások
  - return – T-SQL batch-ből/eljárásból való kilépés (lsd. később)
  - goto – feltétel nélküli vezérlésátadás (nem igazán használjuk)

Ciklusok

- For? – SQL-ben nincs
- Do-while:

```
DECLARE @X INT=1;
WAY:  --> Here the  DO statement
   PRINT @X;
   SET @X += 1;
IF @X<=10 GOTO WAY; --> Here the WHILE @X<=10
```

- Repeat:

```
DECLARE @X INT = 1;
WAY:  -- Here the REPEAT statement
   PRINT @X;
   SET @X += 1;
IF NOT(@X>10 ) GOTO WAY; -- Here the  UNTIL @X>10
```

Ciklusok

- Inkább while-t használjunk!

```
DECLARE @X INT=1;
WHILE @X<=10  --> Here the  DO statement
  BEGIN
   PRINT @X;
   SET @X += 1;
  END
```

Sormutatók (kurzorok)

- Vannak szituációk, amikor egy eredményhalmazt hatékonyabb soronként bejárni/feldolgozni. → kurzorok
- Kurzor deklarációja:

```
DECLARE <kurzornév> [SCROLL] CURSOR FOR
    <lekérdezés>;
```

- alapértelmezés szerinti bejárás: előlről a vége felé

- SCROLL - ha a lekérdezés által visszaadott sorokat más sorrendben (is) be kívánjuk járni

Sormutatók (kurzorok)

- Kurzor megnyitása: `OPEN <kurzornév>`
  - Hatására a rendszer kiértékeli a lekérdezést.
  - Lekérdezés eredménye hozzáférhetővé válik.
- Kurzor bezárása: `CLOSE <kurzornév>`
- Kurzor által lefoglalt memória felszabadítása:
  `DEALLOCATE < kurzornév>`

Hogyan olvashatunk ki sorokat a kurzorból? Mi lesz a kurzort bejáró ciklus leállási feltétele?

Kurzor bejárása

- **Kurzor eredményhalmazából sorok kiválasztása -** `FETCH` utasítás segítségével.

```
FETCH [NEXT|PRIOR|FIRST|LAST]
   FROM <kurzornév>
    INTO @v1,…,@vn
```

  - <lekérdezés> foka: `n (n>=1)`.
- INTO záradék a FETCH utasításon belül: **kivett értékek eltárolására lokális változókban**
  - `@vi` – a kiolvasott sor i. komponensének értékét tároló változó (`i={1,…n}`).

Kurzor bejárása

- `FETCH` utasítás - lehetőségek sorok kiválasztására:
  - `NEXT`: következő sor (alapértelmezett);
  - CSAK `SCROLL` kurzorok esetén:
    - `PRIOR`: előző sor;
    - `FIRST,LAST`: első ill. utolsó sor.

Kurzor bejárása

- `FETCH` utasítás - lehetőségek sorok kiválasztására:
  - `NEXT:` következő sor (alapértelmezett);
  - CSAK `SCROLL` kurzorok esetén:
    - `PRIOR`: előző sor;
    - `FIRST,LAST`: első ill. utolsó sor.
    - `FETCH ABSOLUTE n:`
      - Ha n pozitív egész – előlről az n.sor
      - Ha n negatív egész – kurzor s-n. sora, ahol s-kurzor sorainak száma.
      - Ha n=0 – nincs sor kivéve a kurzorból.
    - `FETCH RELATIVE n:`
      - Ha n pozitív egész – m+n.sor, ahol m az utoljára kivett sor.
      - Ha n negatív egész – m-n.sor, ahol m az utoljára kivett sor.
      - Ha n=0 – ugyanazt a sort vesszük ki újból.

Kurzor bejárása – példa

```
DECLARE c SCROLL CURSOR FOR
     SELECT *
      FROM Nyaralok    -- sorok száma: 16
OPEN c

FETCH c   –- Megkapjuk az első sort.
FETCH ABSOLUTE 4 FROM c     -- 4. sor
FETCH ABSOLUTE -4 FROM c    -- 13. sor
FETCH RELATIVE -1 FROM c    -- 12. sor
FETCH LAST FROM c   -- utolsó sor
FETCH FIRST FROM c -- első sor
```

Globális változók (kurzorok)

- `@@CURSOR_ROWS` – a kurzor sorainak számát adja vissza.
- `@@FETCH_STATUS` – az utolsó FETCH utasítás állapotát téríti vissza bármely kurzorral szemben, mely a kapcsolat által meg van nyitva.

Return value	Description
0	The FETCH statement was successful.
-1	The FETCH statement failed or the row was beyond the result set.
-2	The row fetched is missing.
-9	The cursor is not performing a fetch operation.

Forrás: https://docs.microsoft.com/en-us/sql/t-sql/functions/fetch-status-transact-sql?view=sql-server-ver15

Kurzorok viselkedése

- Adatmódosítás szempontjából: pl. read only
- Bejárás szempontjából: forward only (alapért.), scrool
- Láthatóság szempontjából: lokális, globális

```
DECLARE cursor_name CURSOR [LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
[TYPE_WARNING]
FOR select_statement
[FOR UPDATE [OF column_name [ ,...n ] ] ]
```

Példa kurzor használatára

```
DECLARE Alkalmazottak_kurzor CURSOR FOR
  SELECT AlkID, Munkakor
  FROM Alkalmazottak
OPEN Alkalmazottak_kurzor
DECLARE @AlkID INT, @MK NVARCHAR(50)
FETCH NEXT FROM Alkalmazottak_kurzor
     INTO @AlkID, @MK
WHILE @@FETCH_STATUS = 0 -- amig van ki nem
                         olvasott sor a kurzorban
  BEGIN
   ... -- műveletek
   FETCH NEXT FROM Alkalmazottak_kurzor
          INTO @AlkID, @MK
  END
CLOSE Alkalmazottak_kurzor
DEALLOCATE Alkalmazottak_kurzor
```

Alkalmazottak(AlkID,AlkNev, Munkakor,…, *ReszlegID*)

Példa kurzor használatára

```
DECLARE MaxFizAlkKurzor SCROLL CURSOR FOR
   SELECT AlkID, AlkNev, Munkakor, ReszlegNev
   FROM Alkalmazottak A
     JOIN Reszlegek R ON A.ReszlegID=R.RID
   WHERE Fizetes = (SELECT MAX(Fizetes)
                    FROM Alkalmazottak)
   FOR READ ONLY
OPEN MaxFizAlkKurzor
DECLARE @AlkID INT, @ANev NVARCHAR(30),
        @MK NVARCHAR(30), @RNev NVARCHAR(30)
FETCH LAST FROM MaxFizAlkKurzor
          INTO @AlkID, @ANev, @MK, @RNev
...
```

Reszlegek(RID, RNev)

Alkalmazottak(AlkID,AlkNev,Munkakor,Fizetes,…,*ReszlegID*)

Példa kurzor használatára (folyt.)

```
...
WHILE @@FETCH_STATUS = 0
   BEGIN
    PRINT @ANev+N' a(z) '
      +@MK+ N' munkakörben dolgozik a(z) '
      +@RNev+ N'részlegen.';
    UPDATE Alkalmazottak -- hibát fog adni
    SET Fizetes+=Fizetes*0.1
    WHERE CURRENT OF MaxFizAlkKurzor
    FETCH PRIOR FROM MaxFizAlkKurzor
                INTO @AlkID, @ANev, @MK, @RNev
   END
CLOSE MaxFizAlkKurzor
DEALLOCATE MaxFizAlkKurzor
```

Reszlegek(RID, RNev)
Alkalmazottak(AlkID,AlkNev, Munkakor,Fizetes …,*ReszlegID*)

Példa kurzor használatára (folyt.)

Megj. Előző példában:

Nem fogja végrehajtódni a módosítás:

```
UPDATE Alkalmazottak
SET Fizetes+=Fizetes*0.1
WHERE CURRENT OF MaxFizAlkKurzor
```

A módosítás hiba nélkül végrehajtódik:

```
UPDATE Alkalmazottak
SET Fizetes+=Fizetes*0.1
WHERE AlkID = @AlkID
```

```
Reszlegek(RID, RNev)
Alkalmazottak(AlkID,AlkNev,Munkakor, Fizetes …,ReszlegID)
```

Hibakezelés

- `@@ERROR` – rendszerváltozó, annak ellenőrzésére, hogy az előző(!) SQL utasítás sikeresen végrehajtódott-e vagy sem.
  - `@@ERROR = 0` ↔ nem történt hiba
  - `@@ERROR <> 0` ↔ hiba lépett fel

- Példa:

```
INSERT INTO Berlesek VALUES
       (1, ,Berlo Janos', '2009-6-24', 4, 0)
IF(@@ERROR <> 0)
  PRINT 'Hiba a beszúrásnál.'
ELSE
  PRINT 'Sikeres beszúrás.'
SELECT * FROM Berlesek
```

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

Hibakezelés

- `@@ERROR` – rendszerváltozó, annak ellenőrzésére, hogy az előző(!) SQL utasítás sikeresen végrehajtódott-e vagy sem.
  - `@@ERROR = 0` ↔ nem történt hiba
  - `@@ERROR <> 0` ↔ hiba lépett fel

- Példa (lokális változó használatával):

```
DECLARE @hiba INT
INSERT INTO Berlesek VALUES
        (1, ,Berlo Janos', '2009-6-24', 4, 0)
SET @hiba = @@ERROR
SELECT * FROM Berlesek
IF (@hiba <> 0)
   PRINT 'Hiba a beszúrásnál.'
ELSE
   PRINT 'Sikeres beszúrás.'
```

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

Hibakezelés

- `TRY...CATCH` blokk
  - SQL Server 2005-től

```
BEGIN TRY
    <Transact-SQL utasítások>
END TRY
BEGIN CATCH
    <Transact-SQL utasítások>
END CATCH
```

  - Minden olyan végrehajtási hibát elkap, melynek a severity-je nagyobb, mint 10.
  - Severity?

Hibakezelés

- `TRY...CATCH` blokk
  - Léteznek rendszerfüggvények a fellépett hibákból történő információk kinyerésére:
    - `NULL`-t térítenek vissza, ha `CATCH` blokkon kívül hívjuk meg őket.

```
BEGIN TRY
    SELECT 1/0; -- Generate a divide-by-zero error.
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber
        ,ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_STATE() AS ErrorState
        ,ERROR_PROCEDURE() AS ErrorProcedure
        ,ERROR_LINE() AS ErrorLine
        ,ERROR_MESSAGE() AS ErrorMessage;
END CATCH
```

Hibakezelés - példa

```
BEGIN TRY
    SELECT 1/0; -- Generate a divide-by-zero error.
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber
        ,ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_STATE() AS ErrorState
        ,ERROR_LINE() AS ErrorLine
        ,ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
GO
```

ErrorNumber	ErrorSeverity	ErrorState	ErrorLine	ErrorMessage
8134	16	1	3	Divide by zero error encountered.

Hibakezelés

- `RAISERROR` függvény

```
RAISERROR ( { msg_id | msg_str | @local_var}
                { ,severity ,state } )
```

  - Hibaüzenetet generál + kezdeményezi a hiba feldolgozását.
  - Severity – a hibát jellemzi
    - 0-10 között: warning (nem adódik át az irányítás a CATCH blokknak)
    - 11-18 között: hibaüzenet
    - 19-25 között: csak bizonyos jogosultságú userek használhatják (pl. sysadmin).
    - Severity <0 => severity := 0, Severity >25 => severity := 25
  - State – általuk a hibák megkülönböztethetőek

Hibakezelés – példa

Nyaralok(NyaraloID, …, Ar, Bevetel)-NyID – NyID nem IDENTITY típusú
Berlesek(BerlesID,*NyID*,BerloNev,KDatum,Idotart,Ertek)

```
1 IF NOT EXISTS (SELECT NyaraloID FROM Nyaralok
2                  WHERE Nyaraloid=3)
3 BEGIN
4   RAISERROR ('Nincs ilyen nyaralo.',8,1);
5   DECLARE @NyID INT = (SELECT MAX(NyaraloID)+1
                           FROM Nyaralok)
6   INSERT INTO Nyaralok(NyaraloID, NyaraloNev) VALUES
      (@NyID, 'UjNyaralo')
7   IF @@ERROR <> 0 BEGIN
8       RAISERROR ('Hiba a beszurasnal.',16,1);
9          RETURN;  END
10  INSERT INTO Berlesek VALUES
         (@NyID, 'Balazs Csongor', '2009-6-24', 4, 0)
11  IF @@ERROR <> 0 BEGIN
12           RAISERROR ('Hiba a beszurasnal.',16,2);
13         RETURN; END
14 END
15 ELSE … -- 10-13.sorok (@NyID helyett: 3)
```

Hibakezelés - példa

Nyaralok(NyaraloID, …, Ar, Bevetel) – NyID nem IDENTITY típusú
Berlesek(BerlesID,*NyID*,BerloNev,KDatum,Idotart,Ertek)

```
5     DECLARE @MaxNyID INT
5.1     SELECT @MaxNyID = CASE
5.2        WHEN MAX(NyaraloID) IS NULL THEN 0
5.3         ELSE MAX(NyaraloId)
5.4        END
5.5       FROM Nyaralok
5.6   SET @MaxNyID = @MaxNyID + 1
6     INSERT INTO
         Nyaralok(NyaraloID,NyaraloNev)
            VALUES (@NyID, 'UjNyaralo’)
```

Hibakezelés - példa

```
BEGIN TRY
    ...
  RAISERROR ('Warning message.',8,1);
    ...
  RAISERROR ('Error raised in TRY block.',16,1);
        --átadódik az irányítás a CATCH blokknak
END TRY
BEGIN CATCH
...
END CATCH
```

Hibakezelés

- `THROW` utasítás
  - SQL SERVER 2012-től
  - `TRY...CATCH` blokkon belül kivételt vált ki/dob tovább.
  - Nincs severity szint: 16 az értéke.
  - `THROW`-t megelőző utasítást (;) kell kövesse (kivéve, ha nincs más utasítás a `CATCH`-en belül).

```
THROW [ { error_number | @local_variable },
  { message | @local_variable },
  { state | @local_variable } ] [ ; ]
```

Hibakezelés – előző példa

```
BEGIN TRY
    ...
    --hiba fellépése
    ...
END TRY
BEGIN CATCH
   THROW
END CATCH
```

THROW vs. RAISERROR

RAISERROR függvény

- Severity paraméter meghatározza a kivétel „súlyosságát”.
- Az utolsó (egyetlen) hibaüzenetet és annak részleteit téríti vissza.
- A raiserror függvény hívásának helyét téríti vissza.
- Nem fejeződik be az utasítás-sorozat a függvényhíváskor.

THROW záradék

- Nincs severity paraméter, a kivétel súlyossága = 16.
- Az összes hibaüzenetet és azok részleteit visszatéríti.
- Helyesen téríti vissza a hiba sorszámát (error number) és a sort (line number), ahol a probléma keletkezett.
- TRY-CATCH blokk hiányában az utasítás-sorozat befejeződik.

- Újabban a THROW-t részesítik előnyben a RAISERROR-ral szemben.

Forrás: https://sqlhints.com/2013/06/30/differences-between-raiserror-and-throw-in-sql-server/

Példa (throw-val)

```
CREATE PROCEDURE spUjSzallitoT
     (@pSzallNev VARCHAR(30),
      @pSzallCim VARCHAR(30))
AS BEGIN
 SET NOCOUNT ON
 BEGIN TRY
  INSERT INTO Szallitok VALUES
              (@pSzallNev, @pSzallCim)
 END TRY
 ...
```

Szallitok (SzallKod, Nev, Cim), SzallKod-identity típusú

Példa (throw-val) (folyt.)

```
  ...
  BEGIN CATCH
   DECLARE @ERR_MSG NVARCHAR(4000),
           @ERR_STA SMALLINT
   SELECT @ERR_MSG = ERROR_MESSAGE(),
          @ERR_STA = ERROR_STATE()
   SET @ERR_MSG='Hiba a Szallito beszurasanal: '
               + @ERR_MSG;
   THROW 50001, @ERR_MSG, @ERR_STA;
  END CATCH
END
```

Példa (raiserror-ral)

```
CREATE PROCEDURE spUjSzallitoR
     (@pSzallNev VARCHAR(30),
      @pSzallCim VARCHAR(30))
AS BEGIN
 SET NOCOUNT ON
 BEGIN TRY
  INSERT INTO Szallitok VALUES
              (@pSzallNev, @pSzallCim)
 END TRY
 ...
```

Szallitok (SzallKod, Nev, Cim), SzallKod-identity típusú

Példa (raiserror-ral) (folyt.)

```
...
BEGIN CATCH
 DECLARE @ERR_MSG NVARCHAR(4000),
         @ERR_SEV SMALLINT,
         @ERR_STA SMALLINT
 SELECT @ERR_SEV = ERROR_SEVERITY(),
        @ERR_MSG = ERROR_MESSAGE(),
        @ERR_STA = ERROR_STATE()
 SET @ERR_MSG='Hiba az adat kiolvasásakor: ' +
                @ERR_MSG;
 RAISERROR (@ERR_MSG,@ERR_SEV,@ERR_STA)
   WITH NOWAIT
 END CATCH
PRINT 'T.e. folytatása (mert nem akartunk
        explicit kilépni).'
END
```

További infók

- Kurzorok
- Változók
- Procedurális szerkezetek