Témakörök

- Tárolt eljárások
- MySQL tárolt rutinok
- Az SPL nyelv elemei
- Bolt: Tárolt eljárás példák
- Triggerek
- Bolt: Trigger példák

Tárolt eljárások (1)

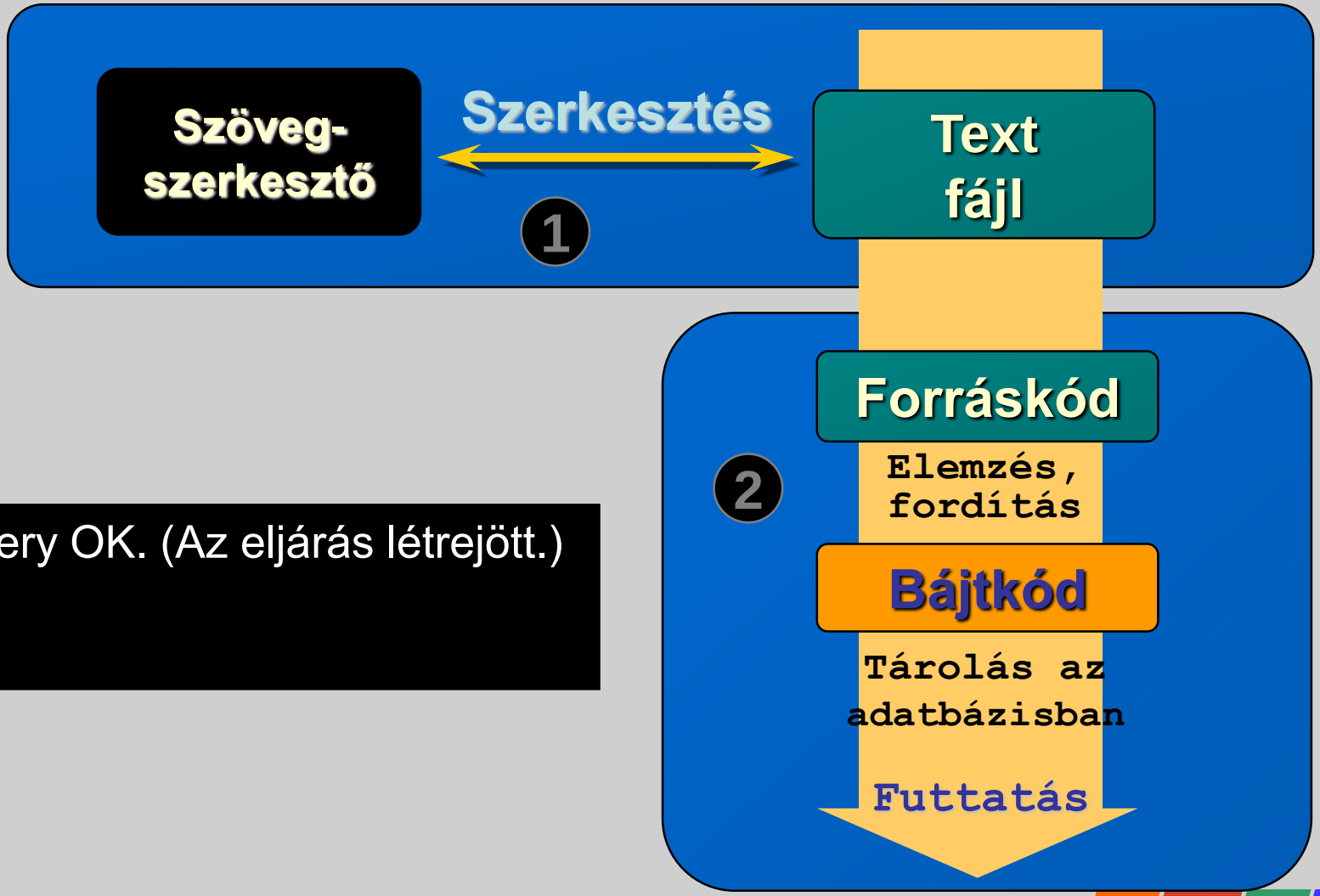
- *Tárolt eljárás* = PSM = Persistent Stored Module (tartósan tárolt modul)*:* adatbázis-objektumként tárolt algoritmikus program, amely SQL utasításokat is tartalmazhat.
- Szintaxisa *az SQL algoritmikus kiterjesztésének* tekinthető.
- A tárolt eljárásokra rendszerenként más-más elnevezést használnak, és szintaxisuk is többé-kevésbé eltér:
  - *SQL:1999 szabvány:* PSM = Persistent Stored Modules
  - *Oracle:* PL/SQL = Procedural Language extension to SQL
  - *SyBase, Microsoft SQL Server:* Transact-SQL
  - *Informix:* SPL = Stored Procedural Language
  - MySQL: MySQL Stored Routines (5.1 verziótól), Stored Program Language (SPL)

Tárolt eljárások (2)

Tárolt eljárások előnyei:

- Az eljárások a szerveren tárolódnak, így nem kell üzeneteket küldözgetni az SQL utasítások végrehajtásakor a kliens és a szerver között.
- Az eljárások elemzése egyszer történik meg, helyes szintaxis esetén lefordítva (futásra kész állapotban) tárolódnak.
- Az eljárásokra ugyanolyan biztonsági intézkedések vonatkoznak, mint az adatbázisban tárolt többi adatra.

Adatbázis-objektumként tárolt rutinok




```
Query OK. (Az eljárás létrejött.)
```

Témakörök

- Tárolt eljárások
- MySQL tárolt rutinok
- Az SPL nyelv elemei
- Bolt: Tárolt eljárás példák
- Triggerek
- Bolt: Trigger példák

MySQL tárolt rutinok

- Kétféle tárolt rutin van a MySQL-ben:
  - Eljárás – PROCEDURE,
  - Függvény – FUNCTION.

- A FUNCTION mindig visszaad egy értéket a hívó félnek, míg a PROCEDURE nem ad vissza értéket, hanem paraméterek átadásával kommunikál a környezetével.
- A hívás során alkalmazhatunk paramétereket, így ezekkel vezérelhetjük a tárolt rutinok működését.

MySQL tárolt eljárás (1)

CREATE PROCEDURE *eljárásnév* (paraméterek)
Begin
  eljárástörzs
End

```
CREATE PROCEDURE aki()
BEGIN
  DECLARE a INT Default 5;
  select a;
END;
```

```
mysql> CREATE PROCEDURE aki()
    -> BEGIN
    ->    DECLARE a INT Default 5;
    ->    select a;
    -> END;
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> call aki()
+------+
| a    |
+------+
|    5 |
+------+
1 row in set (0.00 sec)
```

CALL aki();

CALL aki;

BITMAN

MySQL tárolt eljárás (2)

- A tárolt rutinok mindig adatbázishoz tartoznak, tehát előtte kell: **use bolt;**
- A parancssor sorvégjele a **;** Ha ezzel találkozik a parancssor értelmezője, egyből végrehajtja az addig beírtakat. Ezt át kell állítani:
- DELIMITER␣**//** Kell bele a szóköz is!!

- A tárolt rutin megírása után célszerű visszaállítani, mert egyébként könnyen megbolondulunk!
- DELIMITER␣;

MySQL tárolt eljárás (3)

A tényleges kód:

```
DELIMITER //
CREATE PROCEDURE aki()
BEGIN
   DECLARE a INT Default 5;
   select a;
END; //
DELIMITER ;
```

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE aki()
    -> BEGIN
    ->    DECLARE a INT Default 5;
    ->    select a;
    -> END; //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> call aki();
+------+
| a    |
+------+
|    5 |
+------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

```
CALL aki();
```


BITMAN

MySQL tárolt függvény (1)

CREATE FUNCTION *függvénynév* (paraméterek)
RETURNS adattípus
Begin
  Függvénytörzs
  RETURN adat;
End;

```
mysql> CREATE FUNCTION SQR (a int) RETURNS int
    -> BEGIN
    -> RETURN a*a;
    -> END;//
Query OK, 0 rows affected (0.00 sec)
```

```
CREATE FUNCTION SQR (a int) RETURNS int
BEGIN
RETURN a*a;
END;
```

```
mysql> select sqr(4);//
+--------+
| sqr(4) |
+--------+
|     16 |
+--------+
1 row in set (0.01 sec)
```

```
SELECT SQR (4);
```

BITMAN

Tárolt rutinok hívása (elindítása)

- Eljárás: CALL eljárásnév (paraméterek);
- Függvény: SELECT függvénynév (paraméterek);
- Eljárásnak lehet paraméterként visszaadott értéke:

```
CREATE PROCEDURE N2 (inout a int)
BEGIN
SET a = a*a;
END;
```

```
mysql> CREATE PROCEDURE N2 (inout a int)
    -> BEGIN
    -> SET a = a*a;
    -> END;//
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> set @a = 4;//
Query OK, 0 rows affected (0.00 sec)

mysql> call n2(@a);//
Query OK, 0 rows affected (0.00 sec)

mysql> select @a;//
+------+
| @a   |
+------+
|   16 |
+------+
1 row in set (0.00 sec)
```

Tárolt rutinok kezelése

- Drop procedure eljárásnév; - Törlés
- Drop function függvénynév; - Törlés

- Nincs Create or replace parancs!
- Helyette:

DROP PROCEDURE IF EXISTS eljárásnév;

CREATE PROCEDURE eljárásnév . . .

- Show procedure status; - Eljárások listája
- Show function status; - Függvények listája

- Show create procedure eljárásnév; - Kiíratás
- Show create function függvénynév; - Kiíratás

Eljárás vagy függvény?

Parancssor

```
>create database prb;
Query OK.

>use prb;
Database changed

create table t1 (nev char(20), ar int);
Query OK.

>insert into t1 values('kifli', 20);
>insert into t1 values('tej', 200);
>insert into t1 values('kenyér', 250);
Query OK.
```

Parancssor

```
>select * from t1;
+---------+------+
| nev     | ar   |
+---------+------+
| kifli   |   20 |
| tej     |  200 |
| kenyér  |  250 |
+---------+------+
3 rows in set
```

Eljárás vagy függvény?

Parancssor

```
>delimiter //
>create procedure atlagar1()
  ->declare a float;
  ->select avg(ar) into a from t1;
  ->select a;
  ->end; //
Query OK.
>delimiter ;
```

Parancssor

```
>call atlagar1;
+---------+
| a       |
+---------+
| 156.667 |
+---------+
1 row in set
```


BITMAN

Eljárás vagy függvény?

Parancssor

```
>drop procedure atlagar1;
Query OK.

>delimiter //
>create procedure atlagar2()
  ->declare a float;
  ->select avg(ar) into a from t1;
  ->select a Átlagár;
  ->end; //
Query OK.
>delimiter ;
```

Parancssor

```
>call atlagar2;
+----------+
| Átlagár  |
+----------+
| 156.667  |
+----------+
1 row in set
```


BITMAN

Eljárás vagy függvény?

Parancssor

```
>delimiter //
>create function atlagar() returns float
  ->declare a float;
  ->select avg(ar) into a from t1;
  ->return a;
  ->end; //
Query OK.
>delimiter ;
```

Parancssor

```
>select atlagar();
+--------------------+
| atlagar()          |
+--------------------+
| 156.6666717529297  |
+--------------------+
1 row in set
```


BITMAN

Eljárás vagy függvény?

Parancssor

```
>select * from t1 where ar > atlagar();
+--------+------+
| nev    | ar   |
+--------+------+
| tej    |  200 |
| kenyér |  250 |
+--------+------+
2 rows in set
```

Parancssor

```
>drop database prb;
Query OK:
```


BITMAN

Témakörök

- Tárolt eljárások
- MySQL tárolt rutinok
- Az SPL nyelv elemei
- Bolt: Tárolt eljárás példák
- Triggerek
- Bolt: Trigger példák

Változók a tárolt rutinokban (1)

A változókat deklarálni kell:

DECLARE változó_neve adattípus(méret) DEFAULT kezdő érték;

Pl:

DECLARE i, j INT DEFAULT 0;

DECLARE név VARCHAR(50);

DECLARE atlagfizetes DOUBLE;

Használhatók az ékezetes karakterek, de nem érdemes használni őket!

Változók a tárolt rutinokban (2)

Értékadás (csak deklarálás után lehet!):

```
SET i = 10;
SET i := 10;
SET név = 'Kis Pista';
```

Értékadás SELECT INTO paranccsal:

```
SELECT AVG(fizetes) INTO atlagfizetes FROM dolgozok;
```

Kiíratás:

```
SELECT név, atlagfizetes;
```

Session változók

- Bárhol létrehozhatók:
  - parancssorból,
  - tárolt eljárásokban.
- Szuper globálisak, bárhol elérhetők, értéküket a teljes session alatt megőrzik.

- Létrehozásuk:

```
SET @a = 13;
SET @n = 'Béla';
```

Paraméterek típusai

- Háromféle funkciójú paraméter létezik:
  - IN – Bemenő (Alapértelmezett típus, elmaradhat!)
  - OUT – Visszaadott értéket tartalmazó
  - INOUT – A kettő kombinációja (kétirányú adatforgalom)

  - Az IN paraméterek az alprogramra nézve konstansok.

IN, OUT paraméter példa

```
DELIMITER //
create procedure sum (IN x1 INT, IN x2 int, OUT x3 int)
begin
   set x3 := x1+x2;
end; //
DELIMITER ;
```

```
call sum (2, 5, @vv);
select @vv;
```

```
mysql> DELIMITER //
mysql> create procedure sum (IN x1 INT, IN x2 int, OUT x3 int)
    -> begin
    ->    set x3 := x1+x2;
    -> end; //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> set @vv = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> call sum (2, 5, @vv);
Query OK, 0 rows affected (0.00 sec)

mysql> select @vv;
+------+
| @vv  |
+------+
|    7 |
+------+
1 row in set (0.00 sec)
```

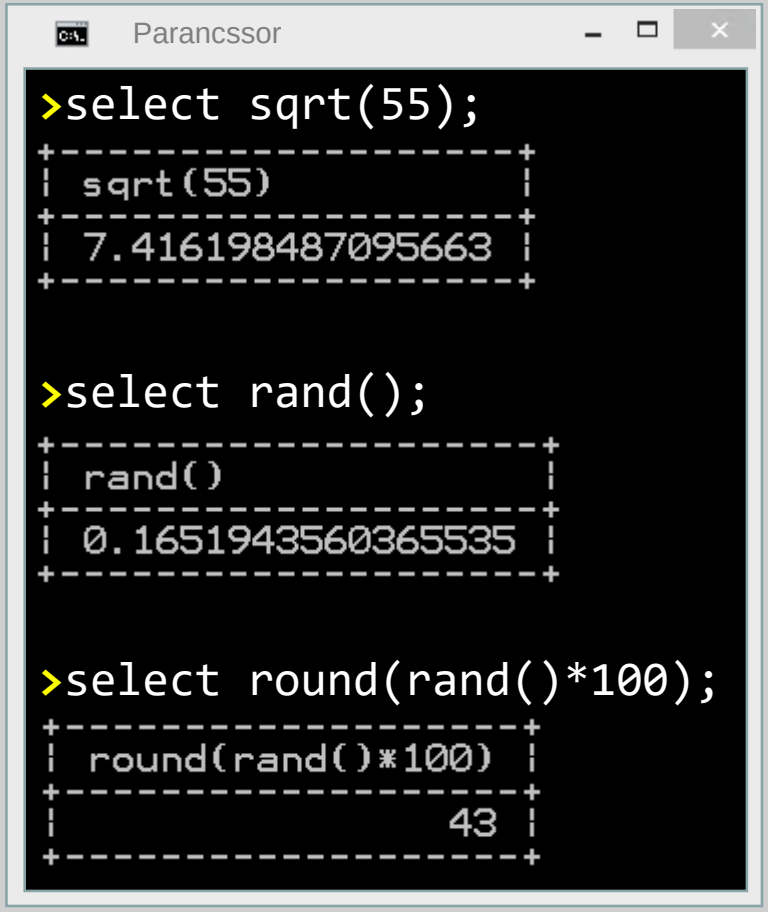
INOUT paraméter példa

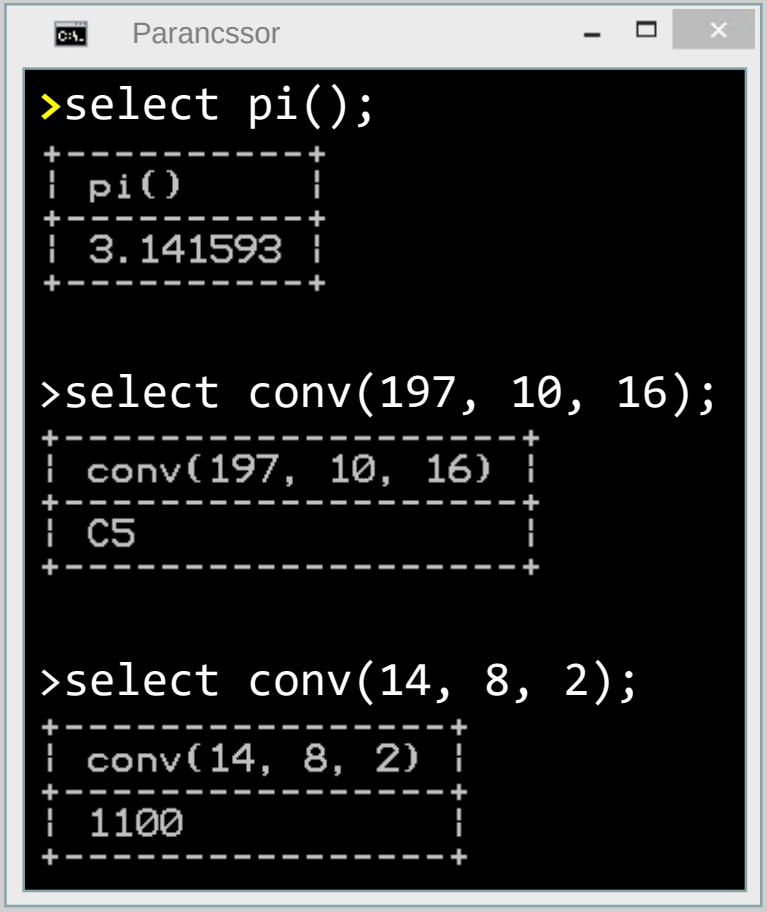
```
DELIMITER //
create procedure pow (INOUT a INT)
begin
   set a := a*a;
end; //
DELIMITER ;
```

```
set @b = 3;
call pow (@b);
select @b;
```

```
mysql> DELIMITER //
mysql> create procedure pow (INOUT a INT)
    -> begin
    ->    set a := a*a;
    -> end; //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql> set @b = 3;
Query OK, 0 rows affected (0.00 sec)

mysql> call pow(@b);
Query OK, 0 rows affected (0.00 sec)

mysql> select @b;
+------+
| @b   |
+------+
|    9 |
+------+
1 row in set (0.00 sec)
```

Beépített függvények





BITMAN

Elágazások a tárolt rutinokban (1)

```
IF kifejezés THEN
  utasitások;
ELSEIF kifejezes2 THEN
  utasitások2;
ELSE
  utasitások3;
END IF;
```

IF példa

```
DELIMITER //
DROP PROCEDURE IF EXISTS Nap;
CREATE PROCEDURE Nap(in d varchar(20))
BEGIN
IF DAYNAME(d) = 'Monday' THEN select d, 'Hétfő';
ELSEIF DAYNAME(d) = 'Tuesday' THEN select d, 'Kedd';
ELSEIF DAYNAME(d) = 'Wednesday' THEN select d, 'Szerda';
ELSEIF DAYNAME(d) = 'Thursday' THEN select d, 'Csütörtök';
ELSEIF DAYNAME(d) = 'Friday' THEN select d, 'Péntek';
ELSEIF DAYNAME(d) = 'Saturday' THEN select d, 'Szombat';
ELSE select d, 'Vasárnap';
END IF;
END ; //
DELIMITER ;
```

```
mysql> call Nap('2013-05-01');
+------------+--------+
| d          | Szerda |
+------------+--------+
| 2013-05-01 | Szerda |
+------------+--------+
1 row in set (0.00 sec)
```


BITMAN

Elágazások a tárolt rutinokban (2)

```
CASE változó
   WHEN érték1 THEN utasítás1;
   WHEN érték2 THEN utasítás2;
   WHEN érték3 THEN utasítás3;
   ELSE utasítás4;
END CASE;
```

CASE példa

```
DELIMITER //
DROP PROCEDURE IF EXISTS Nap;
CREATE PROCEDURE Nap(in d varchar(20))
BEGIN
CASE DAYNAME(d)
WHEN 'Monday' THEN select d as 'Dátum', 'Hétfő' as 'Napnév';
WHEN 'Tuesday' THEN select d as 'Dátum', 'Kedd' as 'Napnév';
WHEN 'Wednesday' THEN select d as 'Dátum', 'Szerda' as 'Napnév';
WHEN 'Thursday' THEN select d as 'Dátum', 'Csütörtök' as 'Napnév';
WHEN 'Friday' THEN select d as 'Dátum', 'Péntek' as 'Napnév';
WHEN 'Saturday' THEN select d as 'Dátum', 'Szombat' as 'Napnév';
ELSE select d as 'Dátum', 'Vasárnap' as 'Napnév';
END CASE;
END ; //
DELIMITER ;
```

```
mysql> call Nap('2015-12-24');
+------------+-----------+
| Dátum      | Napnév    |
+------------+-----------+
| 2015-12-24 | Csütörtök |
+------------+-----------+
1 row in set (0.00 sec)
```

BITMAN

Ciklusok a tárolt rutinokban (1)

```
WHILE kifejezés DO
utasítások;
END WHILE;
```

Elöl tesztelő, amíg igaz a feltétel, addig működik

While példa (1)

```
DELIMITER //
CREATE PROCEDURE Uj_teszt_user (IN darab INT)
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE v_nev, v_pw, v_email VARCHAR(20);
DECLARE v_kor INT;
WHILE i <= darab DO
   SET v_nev = CONCAT('TesztNév_', i);
   SET v_pw = CONCAT('TesztPw_', i);
   SET v_email= CONCAT('t_', i, '@teszt.hu');
   SET v_kor = FLOOR(18 + RAND() * 42);
   INSERT INTO user (User_id, Nev, Jelszo, Email, Kor)
     VALUES (i, v_nev, v_pw, v_email, v_kor);
   SET i = i + 1;
END WHILE;
END; //
DELIMITER ;
```


BITMAN

While példa (2)

```
Create table User(
User_id int primary key,
Nev VARCHAR(15),
Jelszo VARCHAR(15),
Email VARCHAR(20),
Kor int);
```

```
mysql> SELECT CEIL(11.256), FLOOR(11.256);
+--------------+---------------+
| CEIL(11.256) | FLOOR(11.256) |
+--------------+---------------+
|           12 |            11 |
+--------------+---------------+
```

```
DELIMITER //
CREATE PROCEDURE Uj_teszt_user (IN darab INT)
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE v_nev, v_pw, v_email VARCHAR(20);
DECLARE v_kor INT;
WHILE i <= darab DO
    SET v_nev = CONCAT('TesztNév_', i);
    SET v_pw = CONCAT('TesztPw_', i);
    SET v_email= CONCAT('t_', i, '@teszt.hu');
    SET v_kor = FLOOR(18 + RAND() * 42);
    INSERT INTO user (User_id, Nev, Jelszo, Email, Kor)
      VALUES (i, v_nev, v_pw, v_email, v_kor);
    SET i = i + 1;
END WHILE;
END; //
DELIMITER ;
```

BITMAN

While példa (3)

```
mysql> call Uj_teszt_user(10);
Query OK, 1 row affected (0.03 sec)

mysql> select * from user;
+---------+-------------+------------+----------------+------+
| User_id | Nev         | Jelszo     | Email          | Kor  |
+---------+-------------+------------+----------------+------+
|       1 | TesztNév_1  | TesztPw_1  | t_1@teszt.hu   |   33 |
|       2 | TesztNév_2  | TesztPw_2  | t_2@teszt.hu   |   45 |
|       3 | TesztNév_3  | TesztPw_3  | t_3@teszt.hu   |   24 |
|       4 | TesztNév_4  | TesztPw_4  | t_4@teszt.hu   |   51 |
|       5 | TesztNév_5  | TesztPw_5  | t_5@teszt.hu   |   38 |
|       6 | TesztNév_6  | TesztPw_6  | t_6@teszt.hu   |   22 |
|       7 | TesztNév_7  | TesztPw_7  | t_7@teszt.hu   |   20 |
|       8 | TesztNév_8  | TesztPw_8  | t_8@teszt.hu   |   59 |
|       9 | TesztNév_9  | TesztPw_9  | t_9@teszt.hu   |   50 |
|      10 | TesztNév_10 | TesztPw_10 | t_10@teszt.hu  |   57 |
+---------+-------------+------------+----------------+------+
```

```
mysql> call Uj_teszt_user(10);
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

While példa (4)

```
DELIMITER //
DROP PROCEDURE IF EXISTS Uj_teszt_user;
CREATE PROCEDURE Uj_teszt_user (IN darab INT)
BEGIN
DECLARE i INT DEFAULT 1;
DECLARE v_nev, v_pw, v_email VARCHAR(20);
DECLARE v_kor INT;
Select count(*) into i from user;
SET i=i+1;
SET darab = darab+i-1;
WHILE i <= darab DO
    SET v_nev = CONCAT('TesztNév_', i);
    SET v_pw = CONCAT('TesztPw_', i);
    SET v_email= CONCAT('t_', i, '@teszt.hu');
    SET v_kor = FLOOR(18 + RAND() * 42);
    INSERT INTO user (User_id, Nev, Jelszo, Email, Kor)
      VALUES (i, v_nev, v_pw, v_email, v_kor);
    SET i = i + 1;
END WHILE;
END; //
DELIMITER ;
```

Ciklusok a tárolt rutinokban (2)

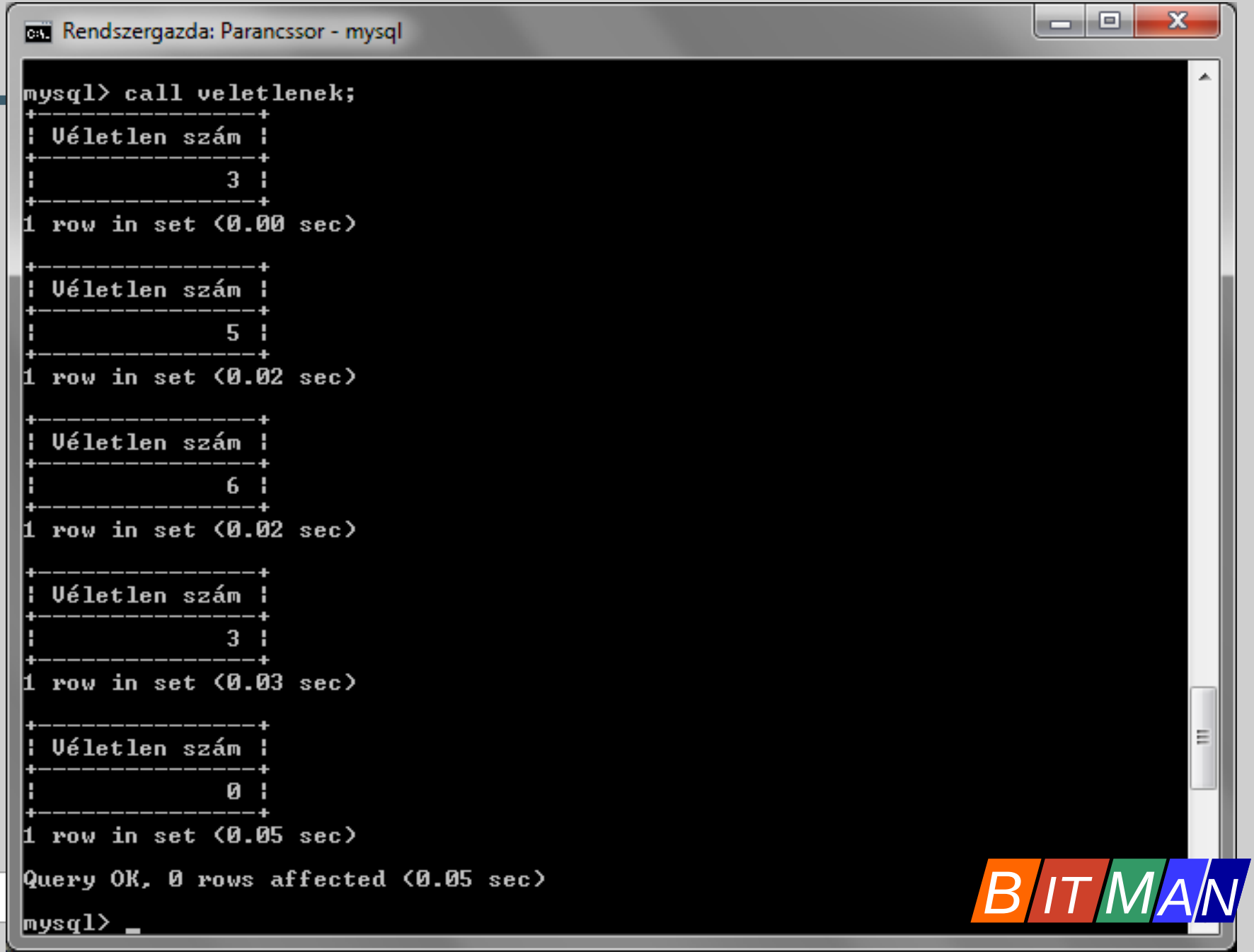
```
REPEAT
utasítások;
UNTIL kifejezés
END REPEAT;
```

Hátul tesztelő, amíg igaz nem lesz a feltétel, addig működik!

Repeat példa

```
DELIMITER //
DROP PROCEDURE IF EXISTS Veletlenek;
CREATE PROCEDURE Veletlenek()
BEGIN
DECLARE db INT DEFAULT 0;
REPEAT
   SELECT FLOOR(RAND() * 10) AS 'Véletlen szám';
   SET db = db + 1;
UNTIL db >=5 END REPEAT;
END //
DELIMITER ;
```

```
mysql> DELIMITER //
mysql> DROP PROCEDURE IF EXISTS Veletlenek;
    -> CREATE PROCEDURE Veletlenek()
    -> BEGIN
    -> DECLARE db INT DEFAULT 0;
    -> REPEAT
    ->    SELECT FLOOR(RAND() * 10) AS 'Véletlen szám';
    ->    SET db = db + 1;
    -> UNTIL db >=5 END REPEAT;
    -> END //
Query OK, 0 rows affected (0.00 sec)
```


Rendszergazda: Parancssor - mysql
mysql> call veletlenek;
+---------------+
| Véletlen szám |
+---------------+
| 3 |
+---------------+
1 row in set (0.00 sec)
+---------------+
| Véletlen szám |
+---------------+
| 5 |
+---------------+
1 row in set (0.02 sec)
+---------------+
| Véletlen szám |
+---------------+
| 6 |
+---------------+
1 row in set (0.02 sec)
+---------------+
| Véletlen szám |
+---------------+
| 3 |
+---------------+
1 row in set (0.03 sec)
+---------------+
| Véletlen szám |
+---------------+
| 0 |
+---------------+
1 row in set (0.05 sec)
Query OK, 0 rows affected (0.05 sec)
mysql> _

Ciklusok a tárolt rutinokban (3)

```
loop_cimke:  LOOP
IF vizsgálat THEN
   LEAVE  loop_cimke;
END  IF;
IF vizsgálat1 THEN
   utasítás1;
   ITERATE  loop_cimke;
ELSE utasítás2;
END  IF;
END LOOP loop_cimke;
```

LEAVE – kilépés a ciklusból

ITERATE – vissza a ciklus elejére

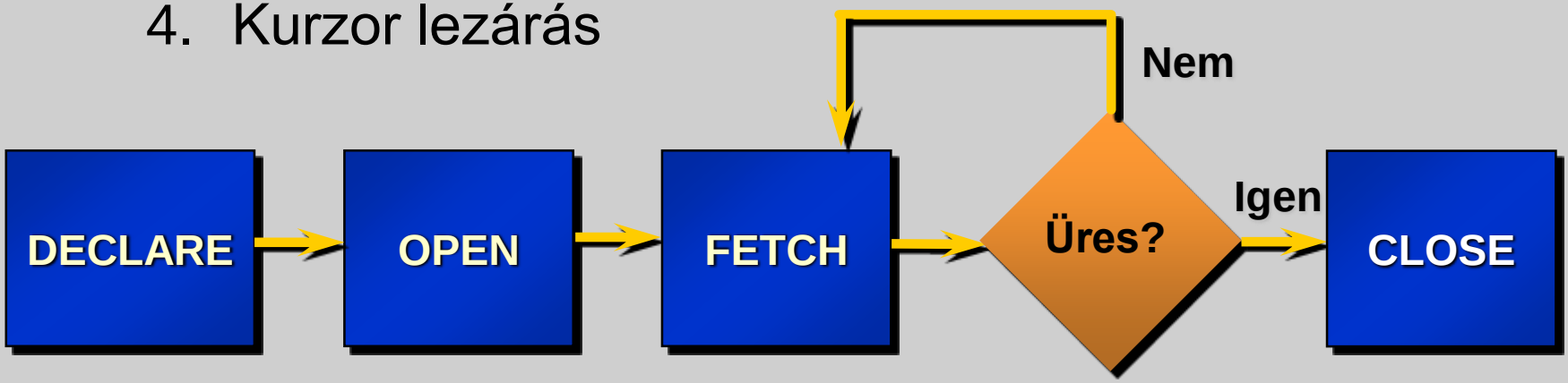
BITMAN

Kurzorok

- A kurzor egy lekérdezés eredményének átmeneti tárolási lehetősége.
- Akkor használjuk, ha a lekérdezés több rekordot ad eredményül.
- Mindig egy SELECT parancs eredményeképp jön létre
- A rekordok ciklus segítségével bejárhatók (kiíratható, módosíthatók)

Adatok kezelése – CURSOR

- Több rekordot visszaadó lekérdezés esetén használandó a kurzor (cursor):
- Lépések:
  1. Kurzor deklaráció (begin előtt!)
  2. Kurzor megnyitás
  3. Rekord kiolvasások ciklusa
  4. Kurzor lezárás



BITMAN

Kurzorok – MySQL sajátosságok

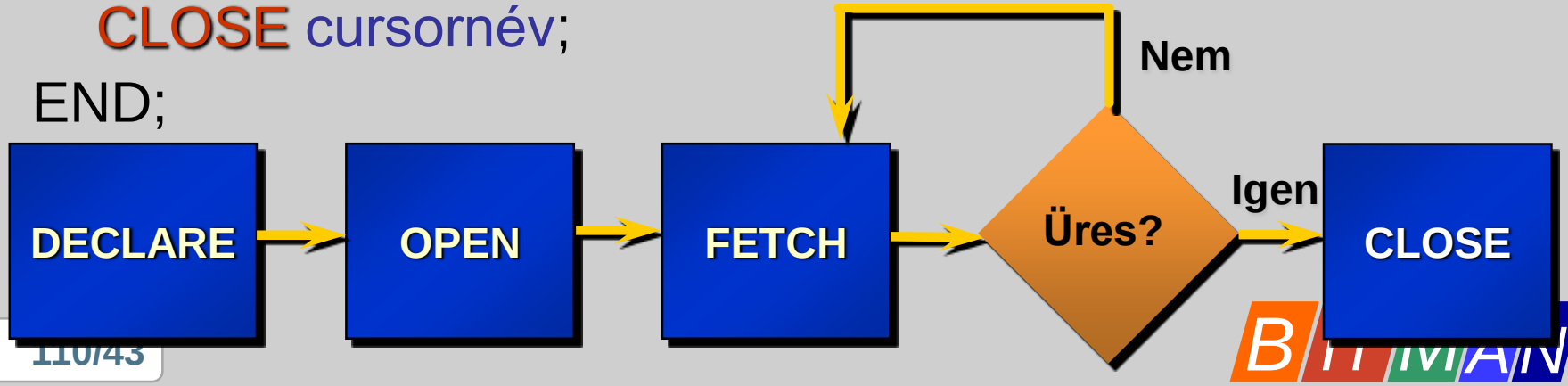
- Létrehozását a DECLARE kulcsszóval vezethetjük be, és csak a tárolt rutin határait jelző BEGIN-END blokkon belül érvényes.
- Ciklussal járjuk be (LOOP), a ciklus akkor ér véget, ha elfogynak az adatok a kurzorból.
- A kurzor kifogyását egy hibakezelővel figyeljük.

- Hibakezelő után nem deklarálhatunk kurzort!
- Kurzor után nem deklarálhatunk változót!
- Kötött deklarációs sorrend:
  - változók,
  - kurzorok,
  - hibakezelők.

BITMAN

Adatok kezelése – CURSOR (elvi példa)

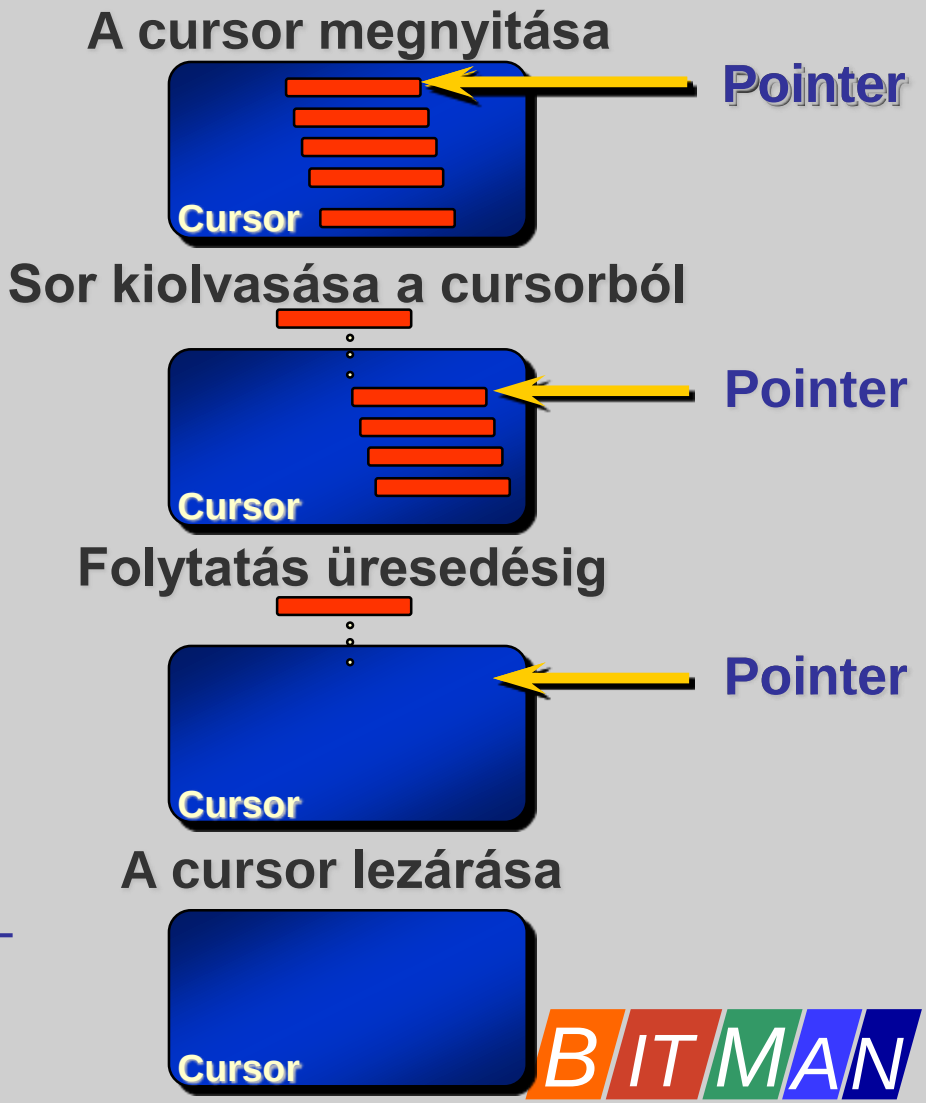
```
DECLARE cursornév CURSOR FOR SELECT … ;
BEGIN
    OPEN cursornév;
    ciklus: LOOP
        FETCH cursornév INTO változók;
        további műveletek … ;
        IF vizsgálat THEN LEAVE ciklus; END IF;
    END LOOP ciklus;
    CLOSE cursornév;
END;
```



CURSOR kiolvasása

```
OPEN cursornév;
ciklus: LOOP
  FETCH cursornév INTO változók;
  további műveletek … ;
  IF vizsgálat THEN
     LEAVE ciklus; END IF;
  END LOOP ciklus;
CLOSE cursornév;
```

Vizsgálat:
HANDLER FOR NOT FOUND SET



CURSOR példa

```
CREATE procedure curdemo()
BEGIN
DECLARE a int;
DECLARE b char(255);
DECLARE kesz INT DEFAULT 0;
DECLARE cur1 CURSOR FOR SELECT id,nev from automarkak;
DECLARE CONTINUE HANDLER FOR NOT FOUND set kesz = 1;
OPEN cur1;
REPEAT
  FETCH cur1 INTO a, b;
  IF NOT kesz THEN SELECT a as ID, b as Név;
  END IF;
UNTIL kesz END REPEAT;
CLOSE cur1;
END;
```

V

C

H

Amíg kesz=0 (hamis) a feltétel, addig működik a ciklus. Ha kifogy a cursor, kesz=1 lesz, igaz lesz a feltétel, és leáll a ciklus.

BITMAN

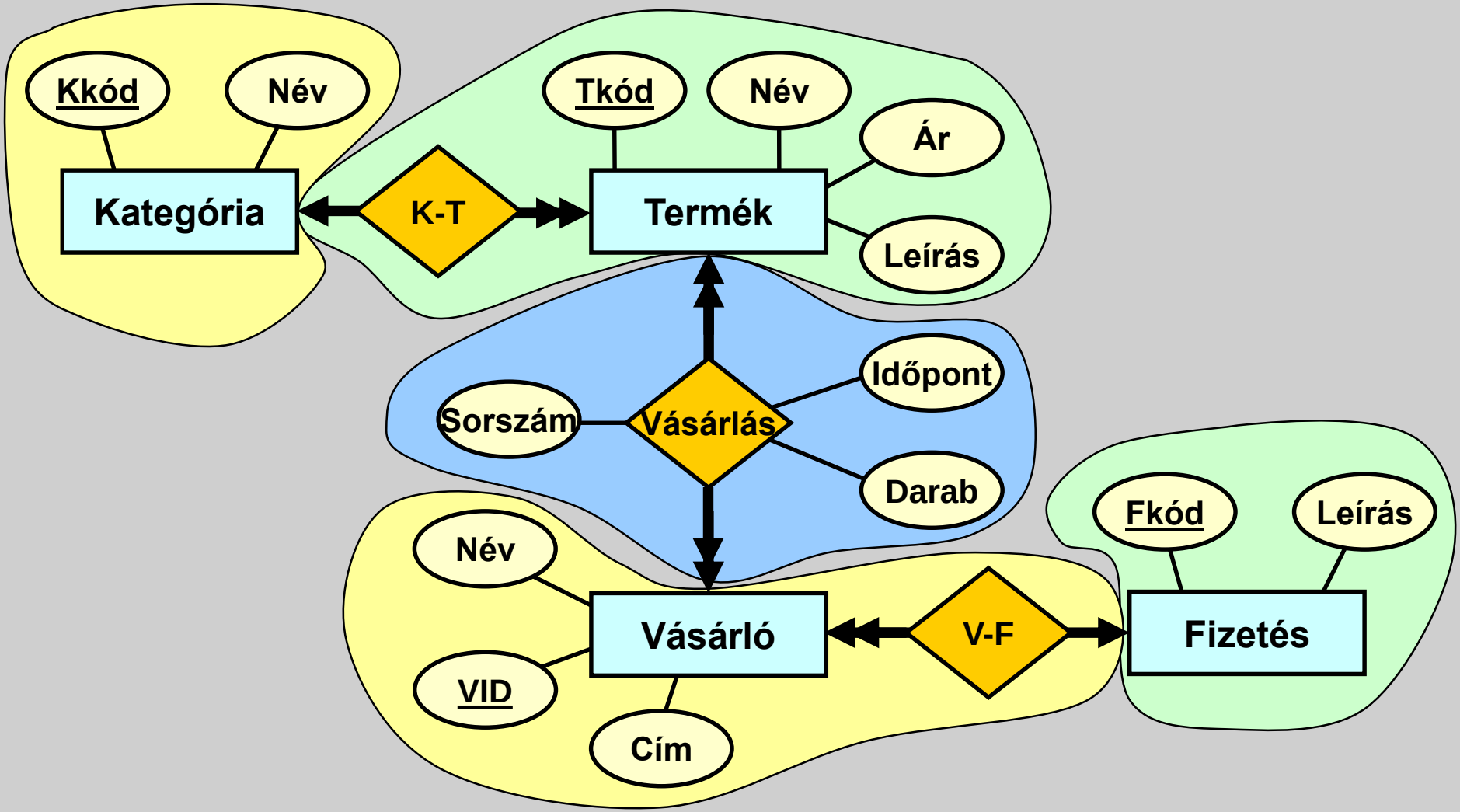
Handler – Hibakezelők

- Ezeket az elemeket kivételkezelésre (exception) használhatjuk, ahol a kivételt a MySQL kiszolgáló dobja egy hibakód formájában, mi pedig egy ilyen hibakezelő segítségével rögzíthetjük, hogy az adott hibakód felmerülése esetén milyen műveletet kell végrehajtani.
- Pl.:
  - DECLARE CONTINUE HANDLER FOR SQLWARNING
  - DECLARE EXIT HANDLER FOR SQLEXCEPTION
  - DECLARE CONTINUE HANDER FOR 1062 SELECT 'szöveg';
  - DECLARE CONTINUE HANDER FOR SQLSTATE '23000' SELECT 'szöveg‘;
  - DECLARE CONTINUE HANDLER FOR NOT FOUND

Témakörök

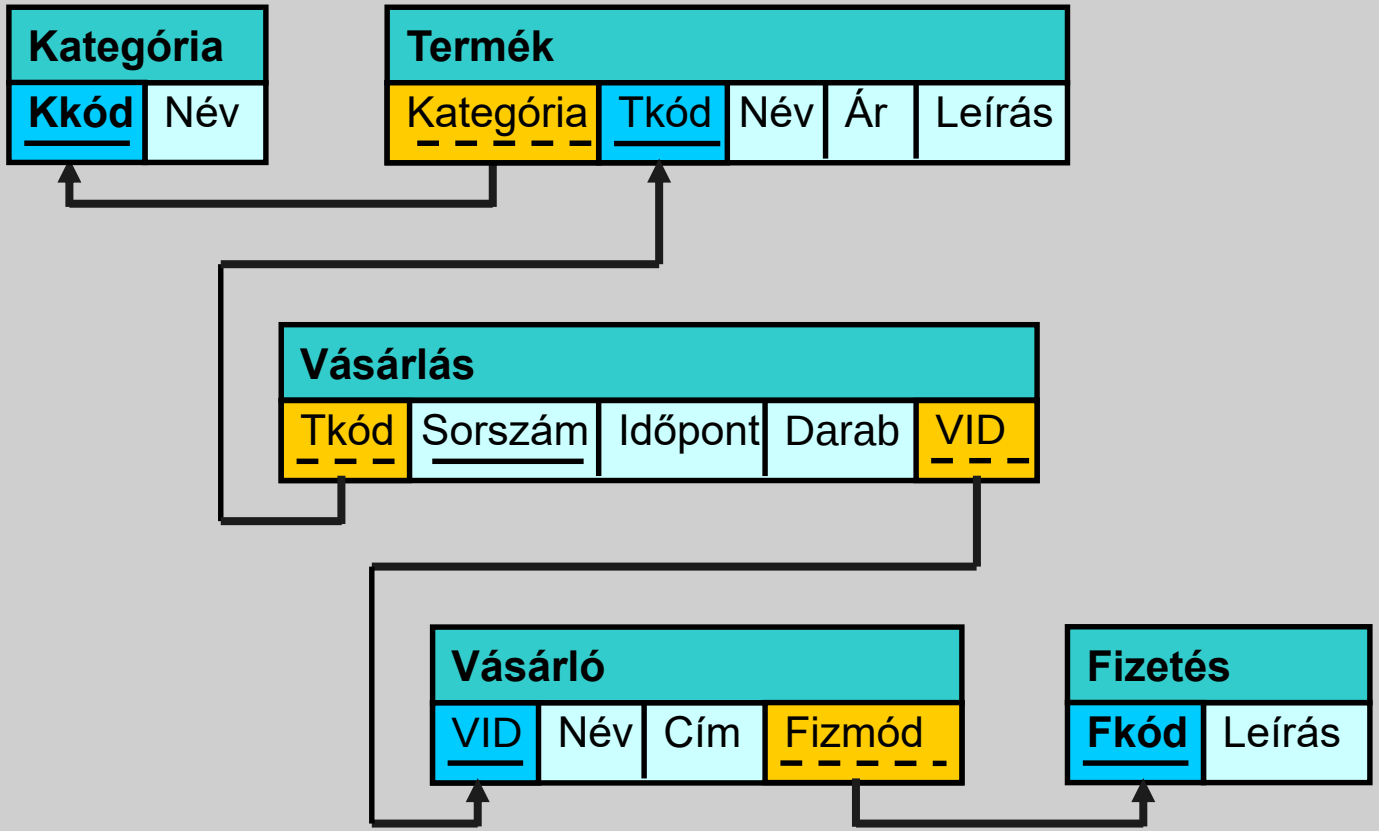
- Tárolt eljárások
- MySQL tárolt rutinok
- Az SPL nyelv elemei
- Bolt: Tárolt eljárás példák
- Triggerek
- Bolt: Trigger példák

Bolt adatbázis



BITMAN

Bolt – Struktúra



BITMAN

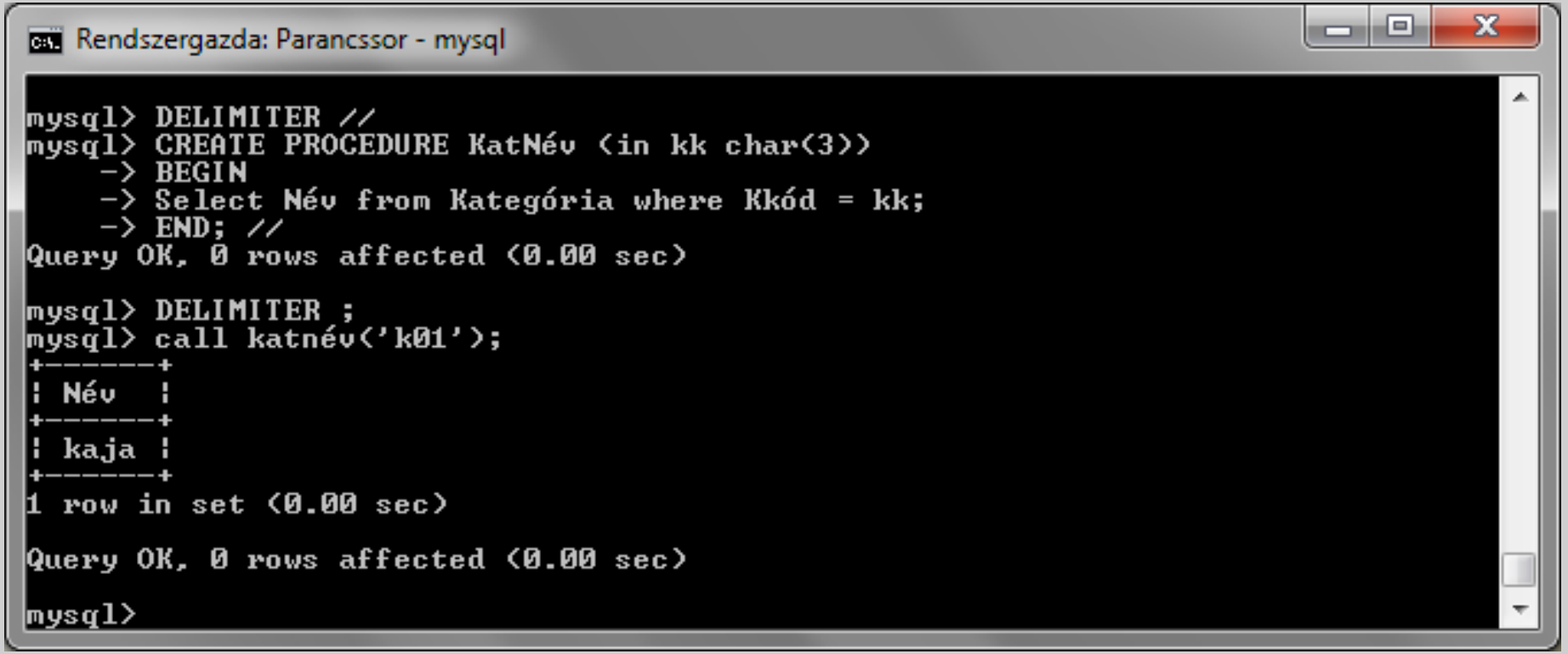
Bolt – Tárolt eljárások (1)

KatNév nevű tárolt eljárás, mely kiírja a képernyőre egy paraméterként megadott kódú kategória nevét.

```
DELIMITER //
CREATE PROCEDURE KatNév (in kk char(3))
BEGIN
   Select Név from Kategória where Kkód = kk;
END; //
DELIMITER ;
```

```
CALL KatNév('k01');
```

KatNév procedure



BITMAN

Bolt – Tárolt eljárások (2)

KatDb nevű tárolt eljárás, mely kiírja a képernyőre egy paraméterként megadott nevű kategória termékeinek darabszámát.

```
DELIMITER //
CREATE PROCEDURE KatDb (in kn varchar(20))
BEGIN
   DECLARE kk char(3);
   Select Kkód into kk from Kategória where Név = kn;
   Select Count(*) from Termék where Kategória = kk;
END; //
DELIMITER ;
```

```
CALL KatDb('Pia');
```

```
mysql> call katdb('pia');
+-----------+
| Count(*)  |
+-----------+
|         0 |
+-----------+
```

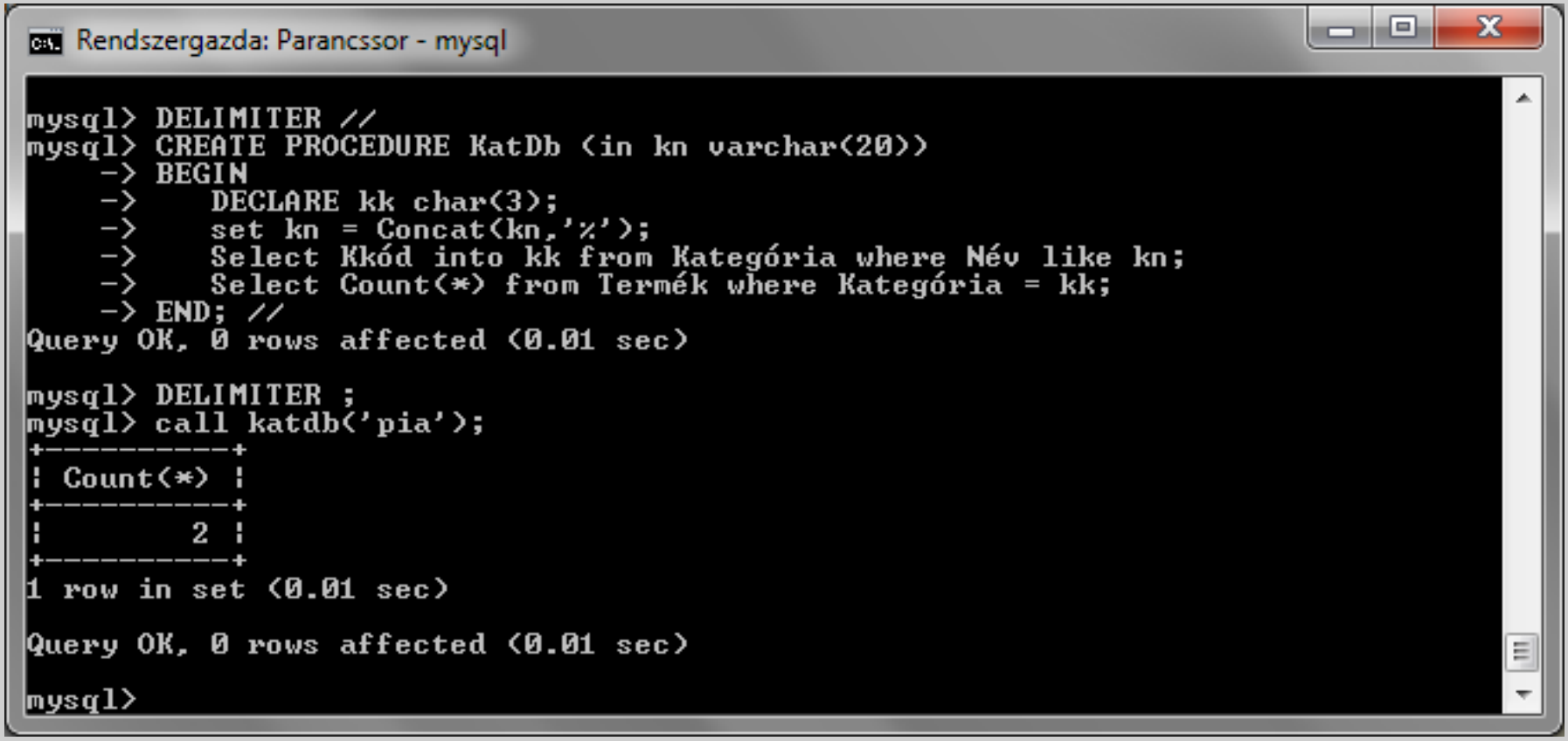
Bolt – Tárolt eljárások (3)

```
DELIMITER //
CREATE PROCEDURE KatDb (in kn varchar(20))
BEGIN
   DECLARE kk char(3);
   set kn = Concat(kn,'%');
   Select Kkód into kk from Kategória where Név like kn;
   Select Count(*) from Termék where Kategória = kk;
END; //
DELIMITER ;
```

```
CALL KatDb('Pia');
```


BITMAN

KatDb procedure



Bolt – Tárolt eljárások (4)

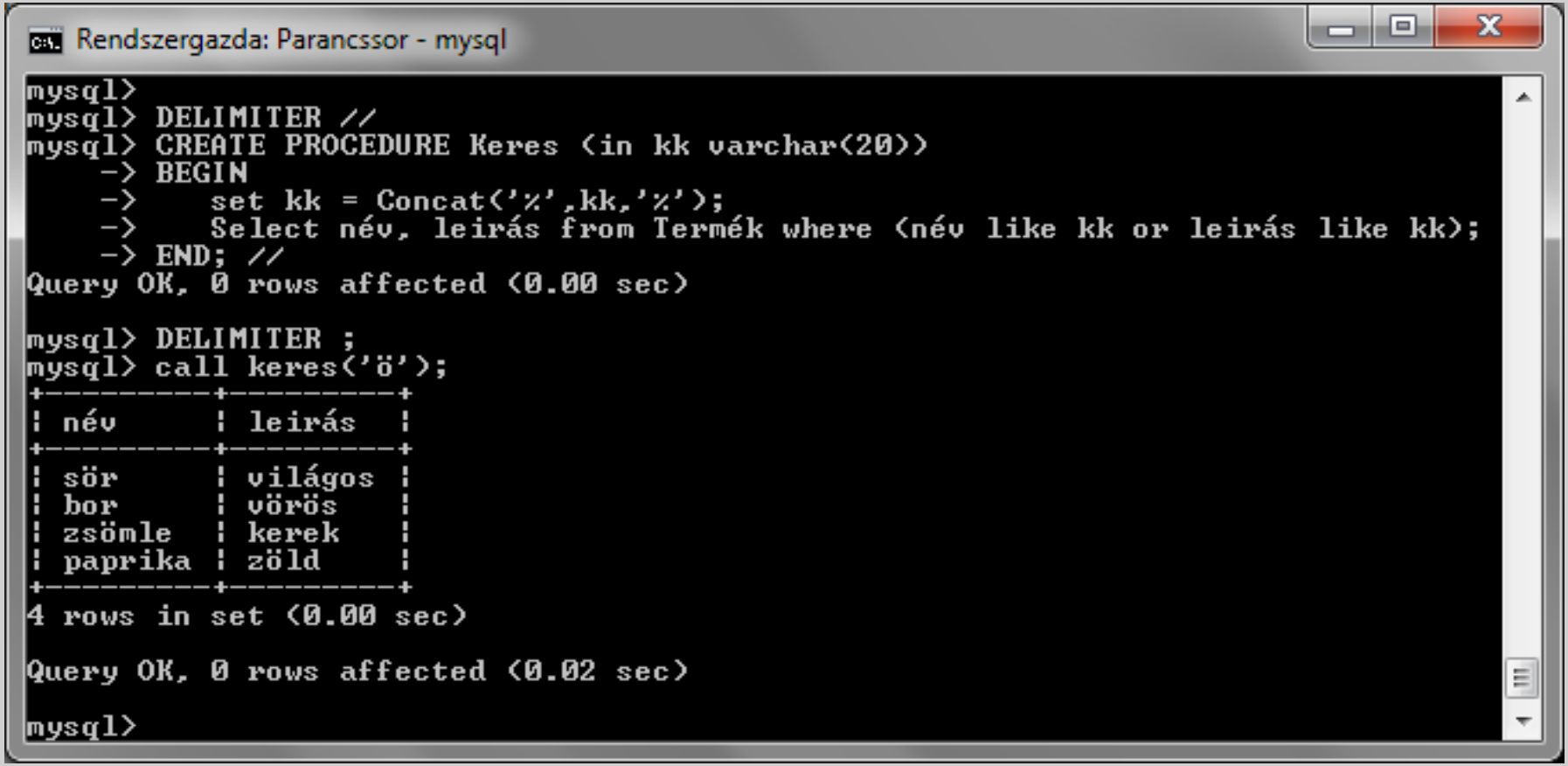
Keres nevű tárolt eljárás, mely megadott részletet keres a termék nevében és leírásában.

```
DELIMITER //
CREATE PROCEDURE Keres (in kk varchar(20))
BEGIN
   set kk = Concat('%',kk,'%');
   Select név, leirás from Termék where (név like kk or leirás like kk);
END; //
DELIMITER ;
```

```
CALL Keres('ö');
```


BITMAN

Keres procedure



BITMAN

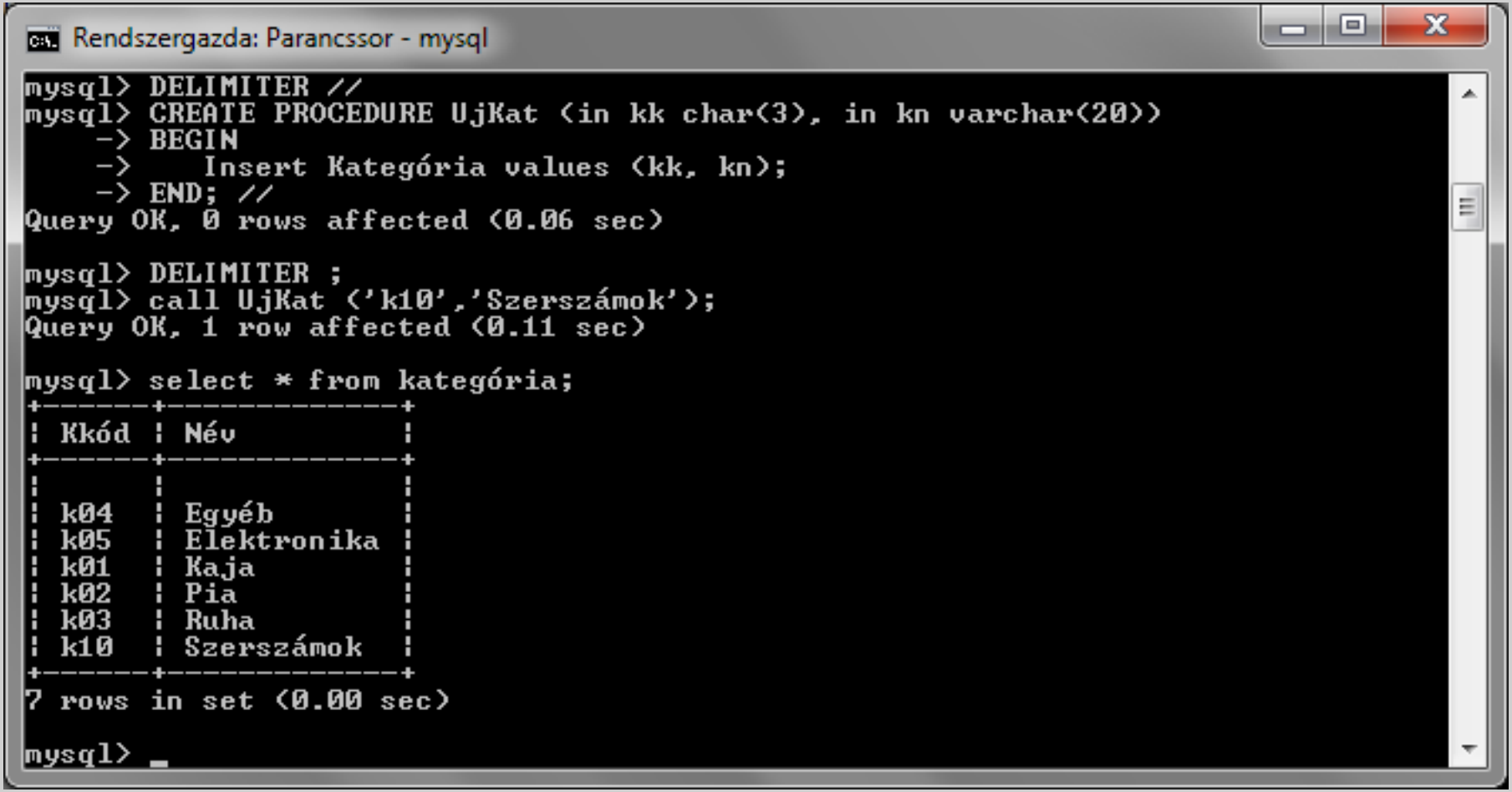
Bolt – Tárolt eljárások (5)

UjKat nevű tárolt eljárás, mely létrehoz egy új kategóriát.

```
DELIMITER //
CREATE PROCEDURE UjKat (in kk char(3), in kn
varchar(20))
BEGIN
   Insert Kategória values (kk, kn);
END; //
DELIMITER ;
```

```
CALL UjKat('k10', 'Szerszámok');
```

UjKat procedure



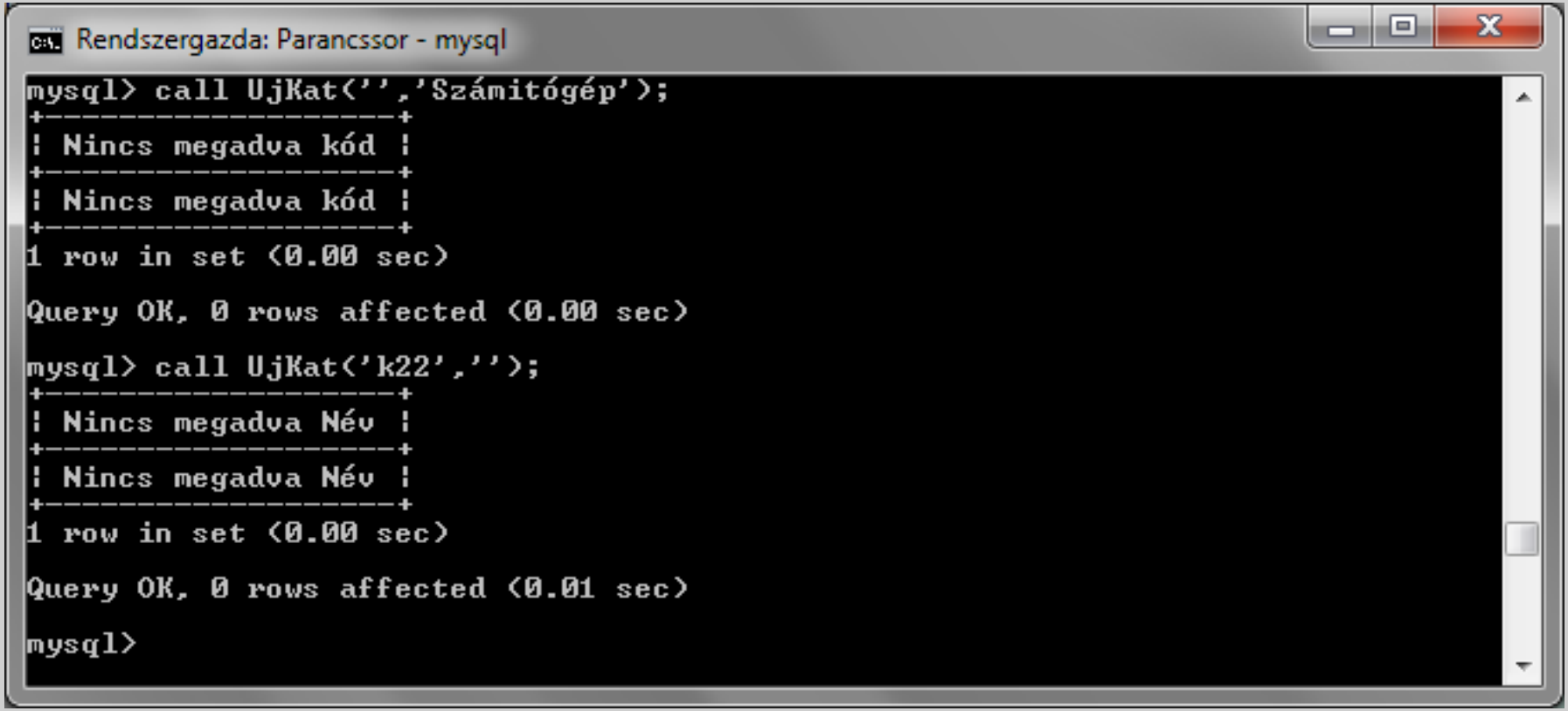
BITMAN

Bolt – Tárolt eljárások (6)

UjKat tárolt eljárás, némi hibakezeléssel.

```
DELIMITER //
CREATE PROCEDURE UjKat (in kk char(3), in kn
varchar(20))
BEGIN
DECLARE CONTINUE HANDLER FOR 1062 SELECT
'Már van ilyen kód vagy ilyen név!';
IF kk ='' THEN SELECT 'Nincs megadva kód';
ELSEIF kn ='' THEN SELECT 'Nincs megadva Név';
ELSE Insert Kategória values (kk, kn);
END IF;
END; //
DELIMITER ;
```

UjKat kipróbálása



BITMAN

Bolt – Tárolt eljárások (7/1)

KatList nevű tárolt eljárás, mely kiírja a képernyőre, hogy az egyes kategóriákban hány darab termék van.

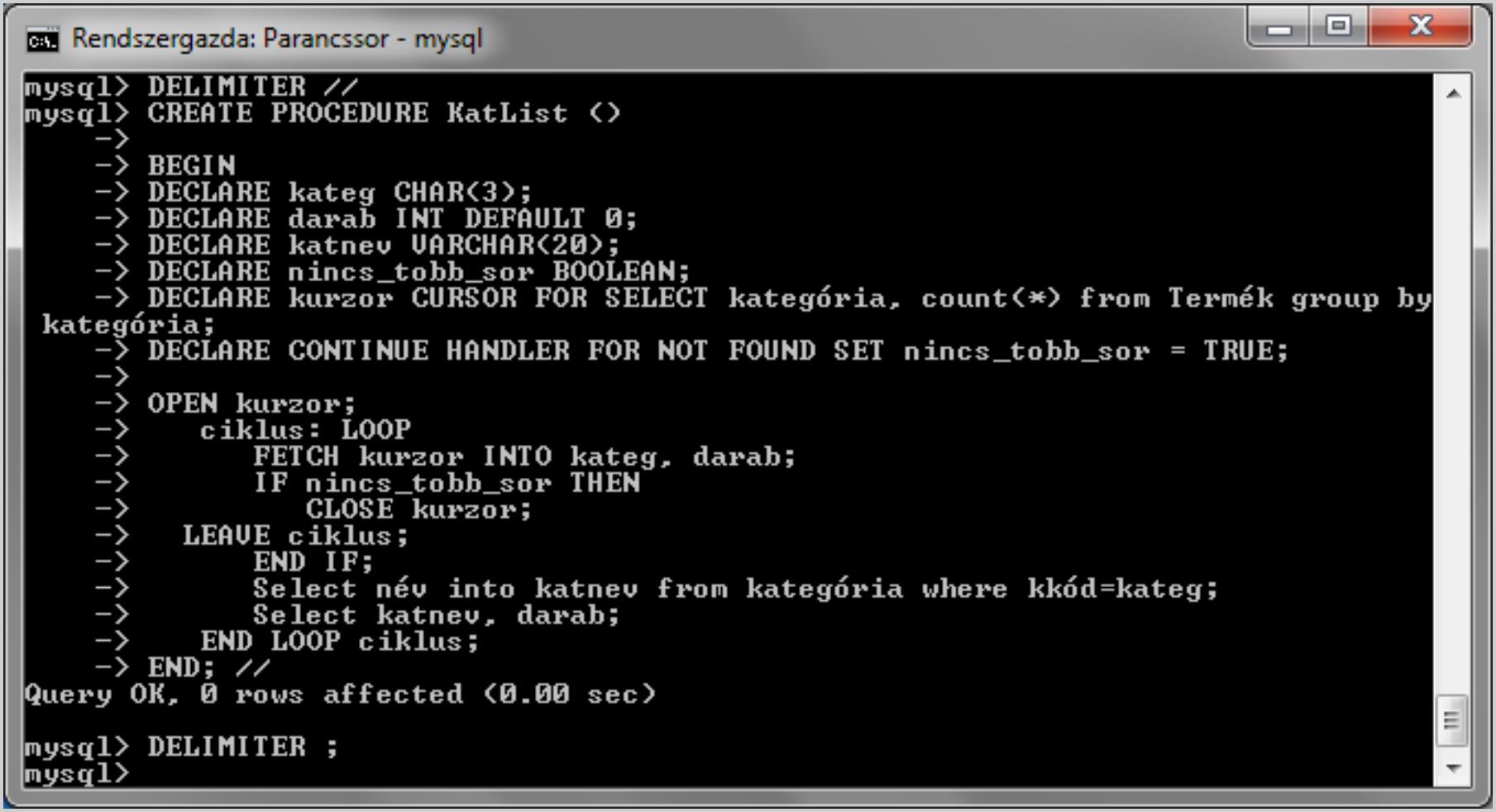
```
DELIMITER //
CREATE PROCEDURE KatList ()
BEGIN
DECLARE kateg CHAR(3);
DECLARE darab INT DEFAULT 0;
DECLARE katnev VARCHAR(20);
DECLARE nincs_tobb_sor BOOLEAN;
DECLARE kurzor CURSOR FOR SELECT kategória, count(*) from Termék group by kategória;
DECLARE CONTINUE HANDLER FOR NOT FOUND set nincs_tobb_sor = TRUE;
```

Bolt – Tárolt eljárások (7/2)

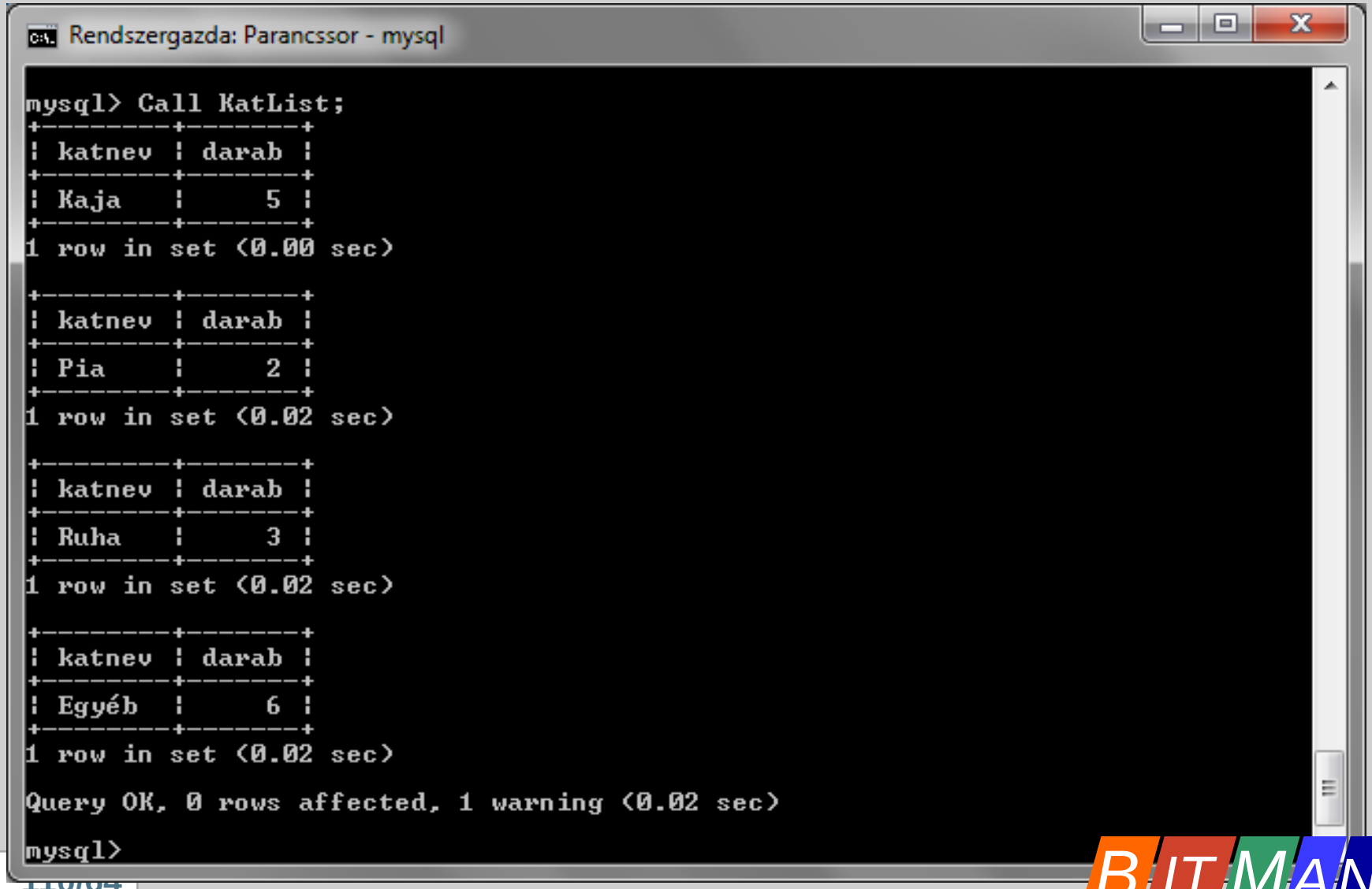
```
OPEN kurzor;
   ciklus: LOOP
        FETCH kurzor INTO kateg, darab;
        IF nincs_tobb_sor THEN
            CLOSE kurzor;
                LEAVE ciklus;
        END IF;
      Select név into katnev from kategória where kkód=kateg;
      Select katnev, darab;
   END LOOP ciklus;
END; //
DELIMITER ;
```


BITMAN

KatList eljárás létrehozása



KatList eljárás – Futtatás



BITMAN

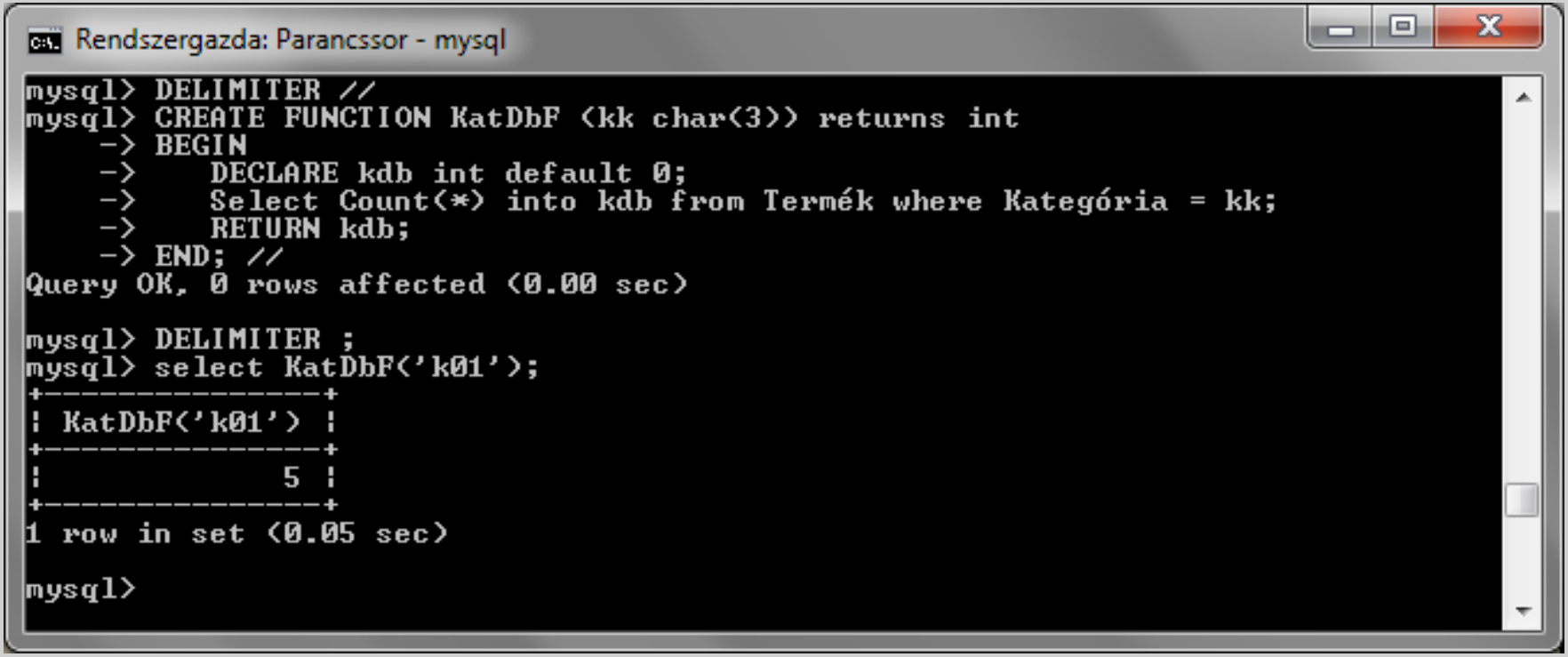
Bolt – Tárolt függvény (1)

KatDbF nevű tárolt függvény, mely megadott kategóriájú termékek darabszámát adja vissza.

```
DELIMITER //
CREATE FUNCTION KatDbF (kk char(3)) returns int
BEGIN
   DECLARE kdb int DEFAULT 0;
   Select Count(*) into kdb from Termék where Kategória = kk;
   RETURN kdb;
END; //
DELIMITER ;
```

```
SELECT KatDbF('k01');
```

KatDbF function



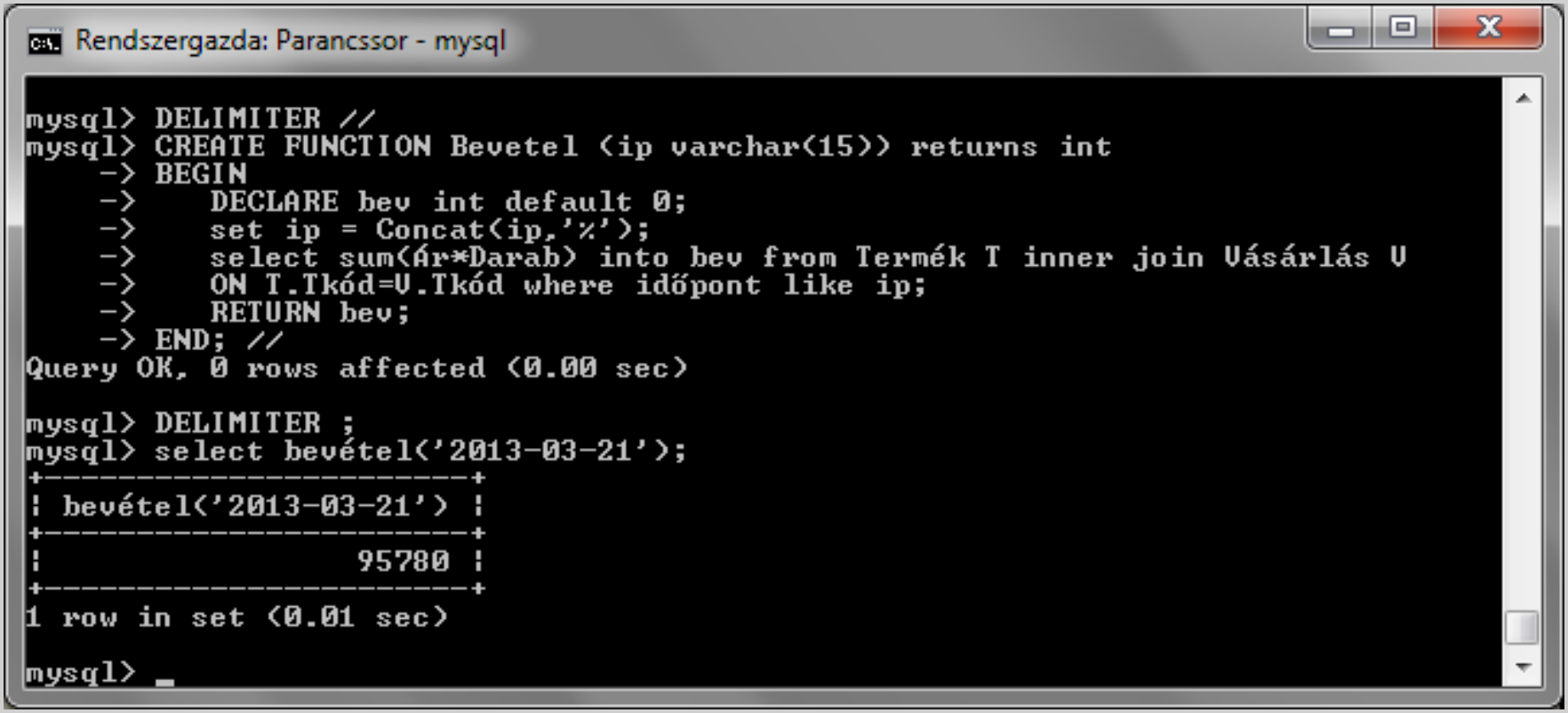
Bolt – Tárolt függvény (2)

Bevetel nevű tárolt függvény, mely megadott nap bevételét adja vissza.

```
DELIMITER //
CREATE FUNCTION Bevetel (ip varchar(15)) returns int
BEGIN
   DECLARE bev int default 0;
   set ip = Concat(ip,'%');
   select sum(Ár*Darab) into bev from Termék T inner join
Vásárlás V ON T.Tkód=V.Tkód where időpont like ip;
   RETURN bev;
END; //
DELIMITER ;
```

```
SELECT Bevetel('2013-03-21');
```

Bevétel function



BITMAN

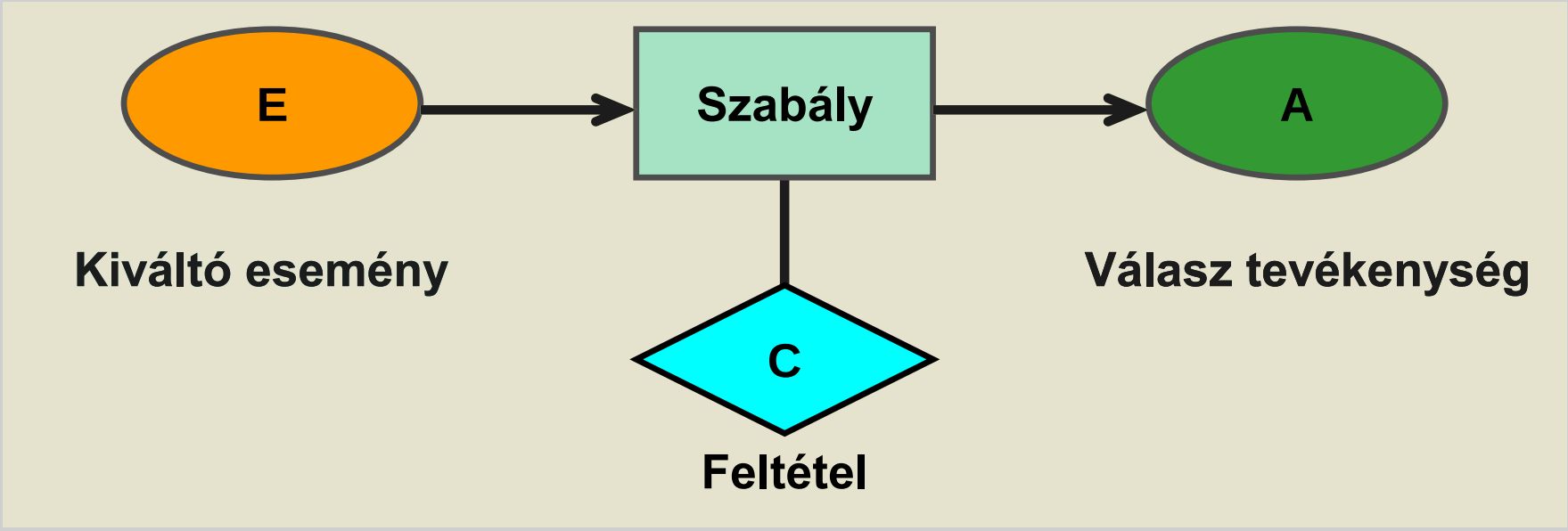
Témakörök

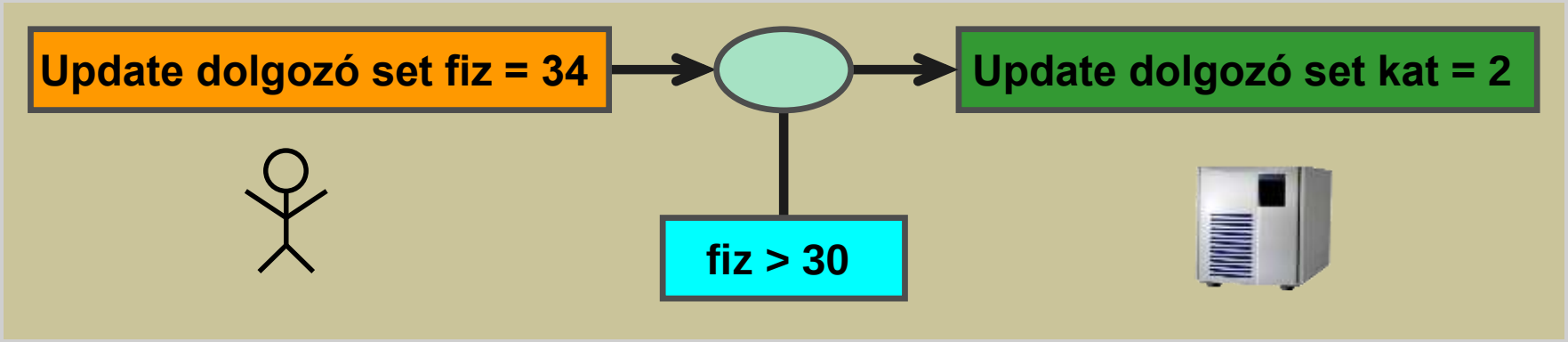
- Tárolt eljárások
- MySQL tárolt rutinok
- Az SPL nyelv elemei
- Bolt: Tárolt eljárás példák
- Triggerek
- Bolt: Trigger példák

Trigger (1)

- Olyan tárolt rutin, amely valamilyen DML művelet (insert, delete, update) bekövetkeztekor automatikusan meghívódik, és végrehajtja a benne megadott műveleteket.
- A trigger egy táblához kötődik, és csak az adott táblánál bekövetkező DML műveletek aktivizálják.
- Általában adatbázisba írás előtti ellenőrzésre, számított adatok kiszámítására, törlés előtt adatok mentésére, események naplózására használjuk.

ECA (Event, Condition, Action) modell





BITMAN

ECA modell (2)

- Events – Események: Műveletsor, melynek bekövetkezését figyeli a rendszer
  - Adatkezelő utasítások (DML):
    - INSERT, DELETE, UPDATE
  - Adatlekérdező utasítások (DQL):
    - SELECT
  - Időfigyelés:
    - Megadott időpontban aktivizálódik a tevékenység
  - Összetett események:
    - and, or, not
    - Műveletsorok
  - Esemény előtt, után, vagy helyett

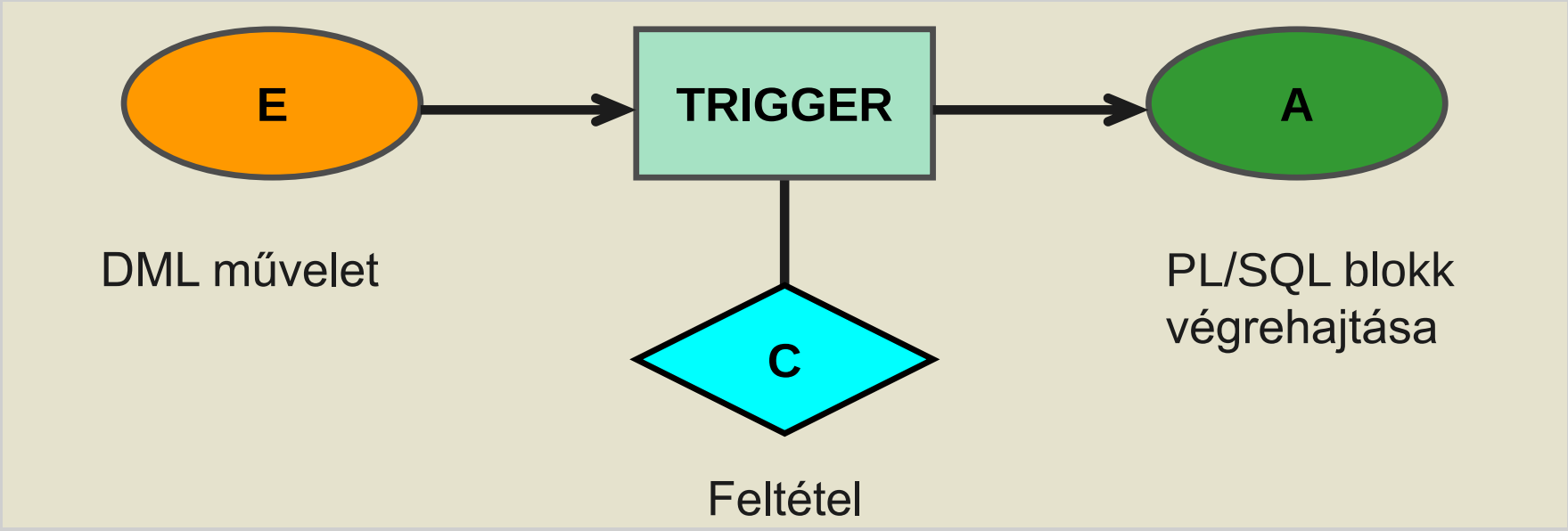
ECA modell (3)

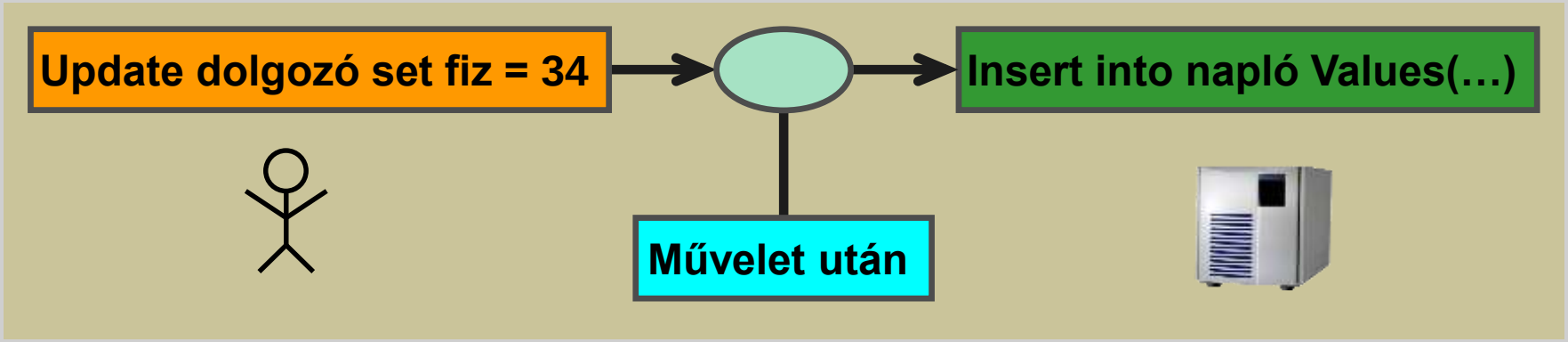
- Condition – Feltétel: logikai kifejezés, melynek teljesülni kell a választevékenység elindításához:
  - Adatbázis feltétel: where
  - Alkalmazás feltétel: az alkalmazásban tetszőlegesen megfogalmazható feltétel

ECA modell (4)

- Action – Akció (választevékenység):
  - adatbázis műveletsor
    - adatbázis-kezelő műveleteket tartalmaz a válasz
  - alkalmazás modul
    - egy alkalmazás modul meghívását jelenti a válasz
  - összetett választevékenységeket is írhatunk

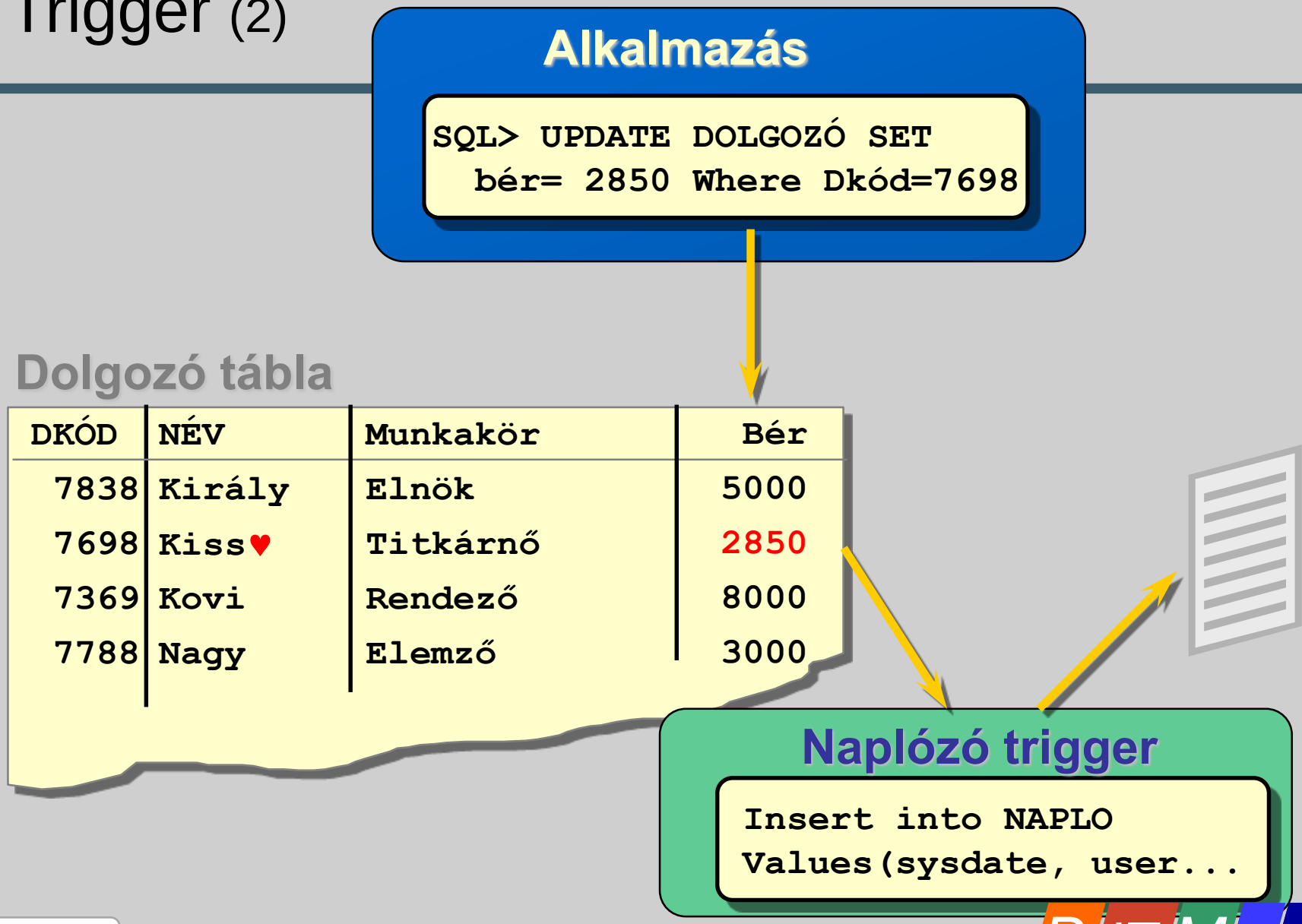
Trigger (kioldó, előidéző)





BITMAN

Trigger (2)



BITMAN

Trigger (3)

- Tökéletesen megvalósítja az ECA elvet:
  - Ha egy megadott tevékenység és feltétel bekövetkezik, akkor végrehajtódik a választevékenység.
  - Segítségével az figyelhető, hogy végrehajtásra kerül-e valamilyen kijelölt adatkezelő művelet
- Felhasználás:
  - Származtatott értékek kiszámítására
  - Érvénytelen adatmanipulációk kiszűrésére
  - Működési korlátozások biztosítására
  - Naplózásra
  - Statisztika-gyűjtésre

Trigger (4)

- Általános formátuma:

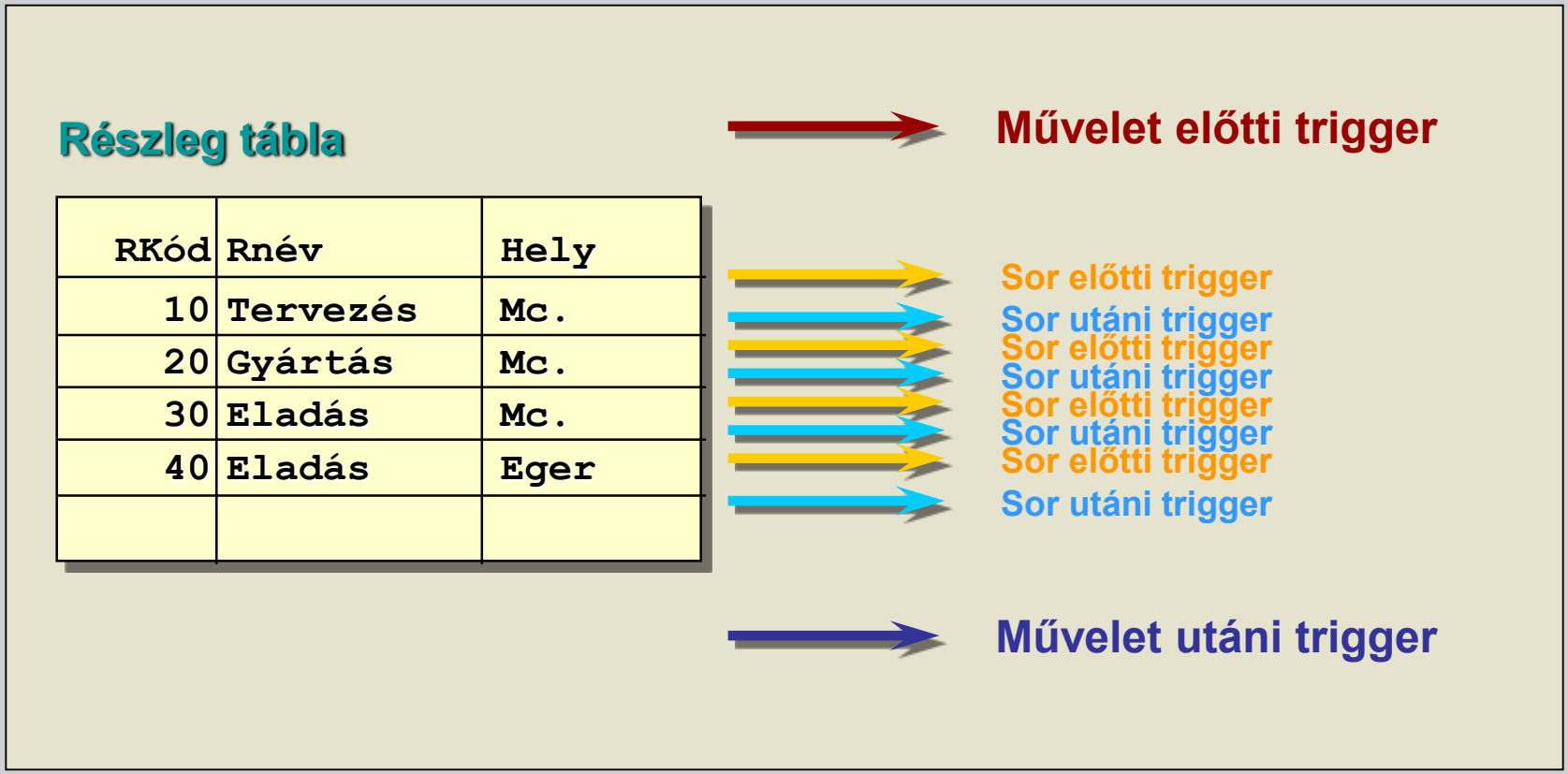
  CREATE TRIGGER triggernév kiváltó_ok akció_blokk;

- Kiváltó okok: DML műveletek
  - INSERT
  - UPDATE
  - DELETE

Trigger (4)

- Trigger hatásmevhanizmusok:
  - **Egyszer** fut le a trigger **egy DML művelet előtt vagy után** (alapértelmezés). Elnevezése: **műveleti trigger**.
  - **Minden egyes érintett rekord** esetén lefut a trigger, a művelet **előtt, vagy után.** Elnevezése: **sorszintű** vagy **rekordszintű trigger**.

Trigger hatásmechanizmus



BITMAN

Trigger (5)

- Általános formátuma:

```
CREATE TRIGGER triggernév BEFORE | AFTER
INSERT | UPDATE | DELETE
ON táblanév [ FOR EACH ROW ]
Begin
   utasítások;
End;
```

Trigger (6)

Megkötések:

- Egy eseményhez nem kapcsolhatunk több, ugyan arra az eseményre (INSERT, UPDATE, DELETE), ugyan abban az időben (BEFORE, AFTER) aktiválódó triggert.
- Pl.: Nem lehet két BEFORE UPDATE trigger egy táblára, de lehet egy BEFORE UPDATE és AFTER UPDATE.
- Átmeneti táblákhoz és nézetekhez nem kapcsolhatunk triggert.
- A triggerek nem módosíthatják azokat a táblákat, amikhez kapcsolva vannak.
- Pl.: ha a users tábla beszúrás utáni eseményére lefut egy trigger, az nem szúrhat be, nem frissíthet adatot, és nem végezhet törlést a users táblán.

Trigger (7)

Megkötések:

- Nem lehet benne tárolt eljárást, vagy függvényt meghívni,
- Nem lehet benne tranzakciót kezdeményezni, jóváhagyni vagy visszavonni,
- Nem lehet benne SELECT parancs INTO nélkül,
- Csak az aktuálisan érintett rekord mezőire hivatkozhatunk:
  - INSERT esetén: NEW.oszlopnév (írható)
  - DELETE esetén: OLD.oszlopnév (olvasható)
  - UPDATE esetén: NEW.oszlopnév (írható), és OLD.oszlopnév (olvasható)

Trigger példa (1)

```
Delimiter //
CREATE TRIGGER ins_termekek BEFORE INSERT ON
    termekek FOR EACH ROW
Begin
  SET NEW.fogyar = NEW.nagykerar * 1.2;
End; //
Delimiter ;
```

Beszúrás előtt kiszámítja a fogyár-at, az ezután lefutó insert beszúrhatja ezt az adatot is.

Trigger példa (2)

```
DELIMITER //
CREATE TRIGGER upd_termek BEFORE INSERT ON
    termekek FOR EACH ROW
BEGIN
   UPDATE keszlet SET darab = darab + NEW.db WHERE
    id = NEW.termek_id;
END;
DELIMITER ;
```

A készlet táblában növekszik a darab annyival, amennyi db-vel az új termék létrejön.

Trigger példa (3)

```
DELIMITER //
CREATE TRIGGER upd_check BEFORE UPDATE ON
    termekek FOR EACH ROW
BEGIN
IF NEW.bonusz < 0 THEN
   SET NEW.bonusz = 0;
ELSEIF NEW.bonusz > 100 THEN
   SET NEW.bonusz = 100;
END IF;
END; //
DELIMITER ;
```

Csak 0 – 100 % közötti bónusz adható meg!

BITMAN

Triggerek kezelése

- SHOW triggers;
- SHOW CREATE TRIGGER trigger_neve;
- DROP TRIGGER trigger_neve;

Témakörök

- Tárolt eljárások
- MySQL tárolt rutinok
- Az SPL nyelv elemei
- Bolt: Tárolt eljárás példák
- Triggerek
- Bolt: Trigger példák

Táblajáték – Napló tábla létrehozása

```
Use bolt;
Create table Naplo(
user_id VARCHAR(15),
Idopont timestamp Default Current_Timestamp,
Leiras VARCHAR(100)
);
```

```
mysql> use bolt;
Database changed
mysql> Create table Naplo(
    -> user_id VARCHAR(15),
    -> Idopont timestamp Default Current_Timestamp,
    -> Leiras VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

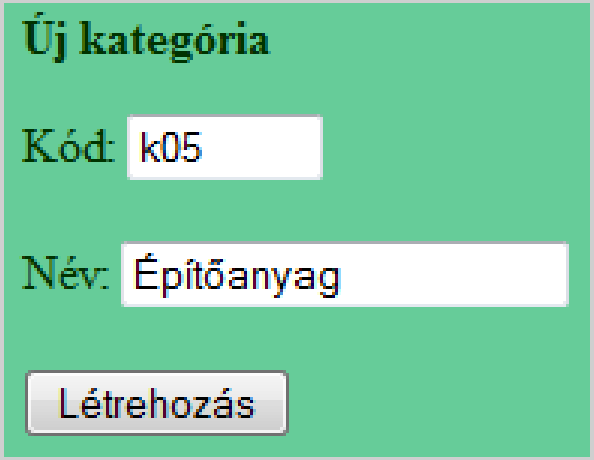
Kategória beszúrást naplózó trigger

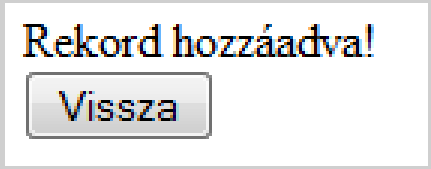
```
DELIMITER //
CREATE TRIGGER ins_kat AFTER INSERT ON kategória
FOR EACH ROW
BEGIN
Insert into Naplo (user_id, Leiras) Values (
user(), CONCAT('Insert: ',NEW.Kkód,'-',NEW.Név));
END; //
DELIMITER ;
```

```
mysql> DELIMITER //
mysql> CREATE TRIGGER ins_kat AFTER INSERT ON kategória
    -> FOR EACH ROW
    -> BEGIN
    -> Insert into Naplo (user_id, Leiras) Values (
    -> user(), CONCAT('Insert: ',NEW.Kkód,'-',NEW.Név));
    -> END; //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;_
```

A trigger kipróbálása

- Új kategória létrehozása (pl. php-ből):





- Trigger ellenőrzése: select * from naplo;

```
mysql> select * from naplo;
+----------------+---------------------+-------------------------+
| user_id        | Idopont             | Leiras                  |
+----------------+---------------------+-------------------------+
| root@localhost | 2013-04-03 14:55:20 | Insert: k05-Épít?anyag  |
+----------------+---------------------+-------------------------+
1 row in set (0.00 sec)
```


BITMAN

Kategória módosítást naplózó trigger (1)

```
DELIMITER //
CREATE TRIGGER upd_kat AFTER UPDATE ON
    kategória FOR EACH ROW
BEGIN
DECLARE s1 VARCHAR(50) DEFAULT ' ';
DECLARE s2 VARCHAR(50) DEFAULT ' ';
if NEW.Kkód != OLD.Kkód then
   SET s1=CONCAT(OLD.Kkód,'->',NEW.Kkód);
END IF;
if NEW.Név != OLD.Név then
   SET s2=CONCAT(OLD.Név,'->',NEW.Név);
END IF;
```

Kategória módosítás naplózó trigger (2)

Insert into Naplo (user_id, Leiras) Values (
user(), CONCAT('Update: ',s1,' ',s2));
END; //
DELIMITER ;

```
mysql> DELIMITER //
mysql> CREATE TRIGGER upd_kat AFTER UPDATE ON kategória
    -> FOR EACH ROW
    -> BEGIN
    -> DECLARE s1 VARCHAR(50) DEFAULT '';
    -> DECLARE s2 VARCHAR(50) DEFAULT '';
    -> if NEW.Kkód != OLD.Kkód then
    -> SET s1=CONCAT(OLD.Kkód,'->',NEW.Kkód);
    -> END IF;
    -> if NEW.Név != OLD.Név then
    -> SET s2=CONCAT(OLD.Név,'->',NEW.Név);
    -> END IF;
    -> Insert into Naplo (user_id, Leiras) Values (
    -> user(), CONCAT('Update: ',s1,' ',s2));
    -> END; //
Query OK, 0 rows affected (0.02 sec)
```


BITMAN

A trigger kipróbálása

- Update kategória set név='Elektronika' where kkód='k05';

```
mysql> update kategória set név='Elektronika' where kkód='k05';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

- Trigger ellenőrzése: select * from naplo;

```
mysql> select * from naplo;
+----------------+---------------------+-----------------------------------+
| user_id        | Idopont             | Leiras                            |
+----------------+---------------------+-----------------------------------+
| root@localhost | 2013-04-03 14:55:20 | Insert: k05-Épít?anyag            |
| root@localhost | 2013-04-03 15:15:24 | Update:  Épít?anyag->Elektronika  |
+----------------+---------------------+-----------------------------------+
2 rows in set (0.01 sec)
```


BITMAN

Adatbázis Rendszerek II.

Ellenőrző kérdések

BITMAN

Ellenőrző kérdések 1.

1. Egy bemenő paraméter nélküli tárolt rutin visszaad egy értéket. Ez a rutin:

A: Tárolt eljárás

B: Tárolt függvény

C: Lehet tárolt eljárás és tárolt függvény is.

2. Melyik kulcsszóval adjuk meg a visszatérő érték típusát?

A: retur

B: returs

C: return

D: returns

Ellenőrző kérdések 2.

3. Tárolt rutinokban a változókat hol deklaráljuk?

A: A BEGIN előtt

B: A BEGIN után

C: Bárhol, a BEGIN előtt vagy után

4. Melyik objektumhoz tartoznak a tárolt rutinok?

A: A rendszerhez

B: Az adatbázishoz

C: A táblákhoz

D: A felhasználókhoz

Ellenőrző kérdések 3.

5. Mi a DELIMITER?

A: Maximális méret paraméter

B: Parancssori sorvégjel

C: Felhasználói quota

D: Tárolt rutin kezdetét jelölő parancsszó

6. Elvileg hogyan hívhatjuk meg a Maci() tárolt rutint?

A: PLEASE Maci();

B: RUN Maci();

C: CALL Maci();

D: SELECT Maci();

E: CALL Maci;

F: SELECT Maci;

G: PUSH Maci();

BITMAN

Ellenőrző kérdések 4.

7. Szintaktikailag helyes-e az alábbi kód:

```
CREATE FUNCTION SQR (in a int) RETURNS int
BEGIN
   SET a = a*a;
   RETURN a;
END;
```

A: IGEN **B:** NEM

8. Melyik kulcsszóval adjuk meg a visszatérő értéket?

A: retur **B:** returs

C: return **D:** returns

BITMAN

Ellenőrző kérdések 5.

9. Hányszor fut le az alábbi ciklus?

```
DECLARE i, db int DEFAULT 5;
WHILE i <= db DO
   SET i = i - 1;
END WHILE;
```

A: 4-szer. **B:** 5-ször. **C:** 6-szor. **D:** Végtelen ciklus

10. Írja be a hiányzó kifejezéseket!

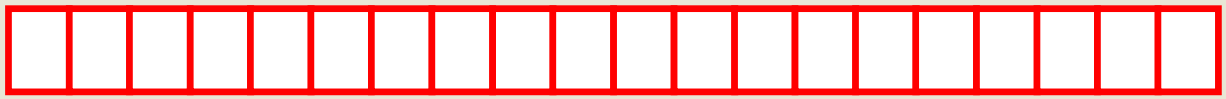
DECLARE CONTINUE [1.] [2.] 1062 SELECT 'szöveg';

BITMAN

Ellenőrző kérdések 6.

11. Mit ír ki az alábbi kódrészlet?

```
DECLARE cc char(8) DEFAULT '0';
DECLARE i INT DEFAULT 0;
REPEAT
   SET cc = CONCAT(cc, i);
   SET i = i + 1;
UNTIL i >=10 END REPEAT;
SELECT cc;
```



BITMAN

Ellenőrző kérdések 7.

12. Írja be a hiányzó kifejezéseket!

```
DECLARE C1 CURSOR [1.] SELECT … ;
BEGIN
    OPEN [2.];
    ciklus: LOOP
        [3.] C1 [4.] változók;
        további műveletek … ;
        IF vizsgálat THEN [5.] [6.]; END IF;
    END LOOP [7.] ;
    [8.] C1;
END;
```

Ellenőrző kérdések 8.

13. Mi a funkciója a következő utasításnak?

DECLARE CONTINUE HANDER FOR 1062 SELECT 'Warning';

A: A select parancs hibája esetén 1062-es figyelmeztető kóddal leállítja a tárolt rutin futását

B: A FOR ciklus 1062-es hibája esetén lekérdezi a Warning változó értékét

C: A FOR ciklus hibája esetén 1062-es kóddal figyelmeztet

D: 1062-es hiba esetén kiírja: Warning

E: Deklarál egy Warning nevű változót a 1062-es hiba lekezelésére

Ellenőrző kérdések 9.

14. A SELECT A; parancs:

A: Parancssorból kiadva helyes.

B: Tárolt eljárásban helyes.

C: Mindkét helyen helyes.

D: Egyik helyen sem helyes.

15. Mi a SHOW PROCEDURE eljárásnév parancs hatása?

A: Kiírja az összes eljárás nevét

B: Kiírja az adott eljárást létrehozó sorokat

C: Kiírja az adott eljárás jellemzőit (státuszát)

D: Hibaüzenet, a parancs ugyanis hibás

Ellenőrző kérdések 10.

16. Igaz vagy Hamis az állítás?

- [ ] Tárolt rutinokban a SELECT mindig kiírja a képernyőre a változók értékét.
- [ ] A trigger tárolt eljárás.
- [ ] A SHOW PROCEDURES STATUS parancs kiírja az eljárások nevét a képernyőre.
- [ ] A CALL @a = Negyzet(5); parancs egy függvényt hív.
- [ ] Delimiter lehet a ; (pontosvessző)
- [ ] Az OUT típusú paraméternek nem lehet a tárolt rutinban az értékét módosítani.
- [ ] A DECLARE i INT; parancsot a tárolt rutinban a Begin előtt kell kiadni.
- [ ] Cursor deklaráció után deklarálhatunk változókat.

Ellenőrző kérdések 11.

17. Melyik művelet lehet egy trigger elindítója?

A: Create
B: Insert
C: Alter
D: Delete
E: Before
F: Update
G: Drop
H: Select
I: After

18. Mely kulcsszavak használhatók INSERT típusú triggernél?

A: NEW
B: OLD
C: Mindkettő

19. Mely kulcsszavak használhatók DELETE típusú triggernél?

A: NEW
B: OLD
C: Mindkettő

BITMAN

Ellenőrző kérdések 12.

20. Írja be a hiányzó kifejezéseket!

```
CREATE TRIGGER ins_kat AFTER [1.] [2.]
Kategória FOR [3.] ROW
BEGIN
Insert [4.] Naplo Values ([5.](OLD.Ár,'-',NEW.Ár));
END;
```


BITMAN

Ellenőrző kérdések 13.

21. Igaz vagy Hamis az állítás?

- ☐ A trigger tetszőleges adatot módosíthat.
- ☐ Létrehozható két Before trigger egy táblához.
- ☐ Létrehozható két Update trigger egy táblához.
- ☐ A For Each Row típusú trigger törléskor mindig többször lefut.
- ☐ Trigger törzsében lehet kiíratni a képernyőre adatot.
- ☐ Insert típusú triggernél nem használható az OLD kulcsszó.
- ☐ Update típusú trigger törzsében nem lehet Insert parancs.
- ☐ A SHOW TRIGGER triggernév parancs helyes.
- ☐ A DELETE TRIGGER triggernév parancs helyes.

Felhasznált irodalom

- php.net – MySQL Manual
- www.w3school.com – PHP MySQL Tutorial
- www.w3school.com – HTML Reference
- www.tizag.com – MySQL Tutorial
- www.tutorialspoint.com – MySQL Tutorial

BITMAN

VĚGE