# Datové a procesní modely

## Relační databáze

### Přednáška 6 a

**Marian Kamenický**

Syntea software group a.s.
marian.kamenicky@syntea.cz

MFFUK Praha

2019/20

# SQL

- Structured Query Language

- přenositelný mezi počítačovými systémy (**?**)
- komplexní DB jazyk
- interaktivní dotazy "ad hoc"
- **neprocedurální jazyk**
- programový přístup k databázi

# SQL - neprocedurální jazyk

sql povel

Povel

žádný objekt ničeho;

sql skript

```
/* ---------------------  */
/* Založení DB Mzdy  */
/* ---------------------  */
Create Table Osoba (
      Id  Serial Prim..Key,
      Jm VarChar(30),
      Pr   VarChar(30),
            …….. );
Create Table  Odd (
            …..
            );
 ……
```

Textový soubor.

Soubor se zkratkou ____.sql

Provádí se povel po povelu.

Pokud všechny povely dobře, provede se celý skript.

Pokud nějaký povel chybný přeruší se jím skript.

Objekt opetračního systému.

i.Relační databáze je však také vybavena standardním procedurálním jazykem

i.Každý výrobce SQL má vlastní procedurální jazyk.

ii.Starndard SQL se nazývá **SQL/PSM**

# SQL/PSM - procedurální jazyk SQL databází

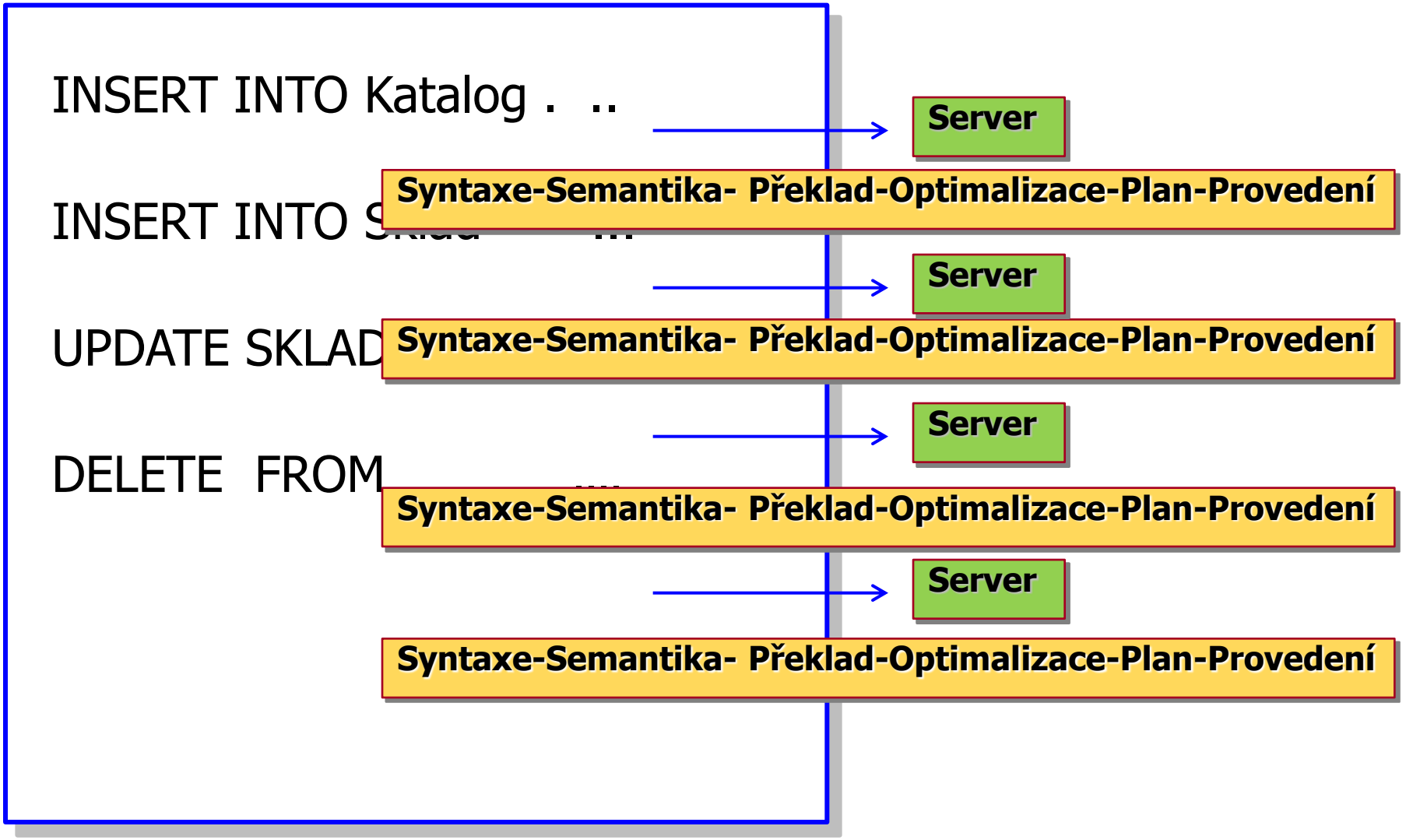
- SQL/PSM – standard SQL [ANSI-ISO]
  - IBM DB2
  - MySQL
  - PostgreSQL
- Vlastní proprietární rozšíření – deriváty normy
  - Oracle PL/SQL
  - PostgreSQL PL/SQL
  - Microsoft T-SQL
  - SyBase T-SQL
  - …….

# SQL

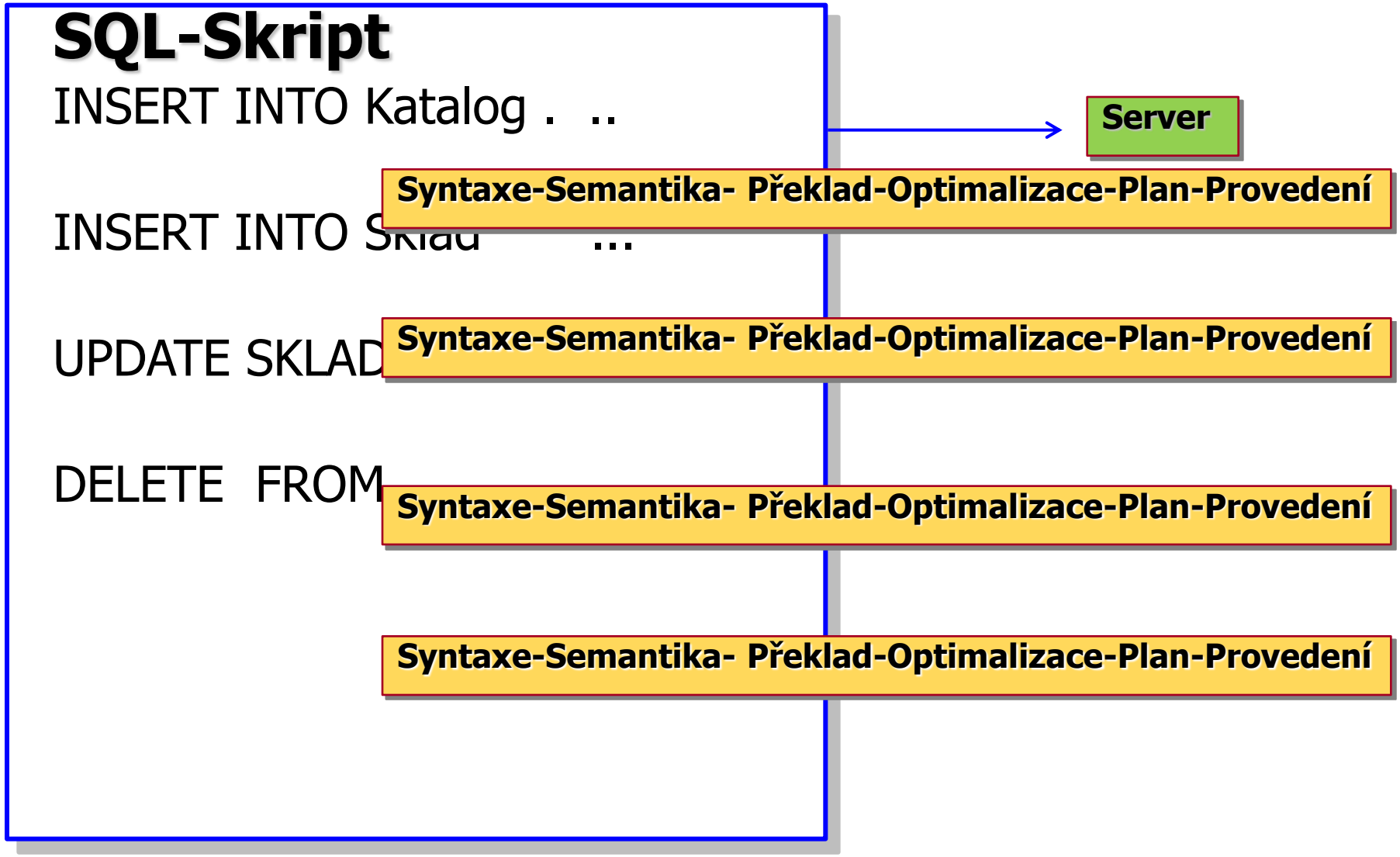
- Structured Query Language

- přenositelný mezi počítačovými systémy (**?**)
- komplexní DB jazyk
- interaktivní dotazy "ad hoc"
- **neprocedurální jazyk**
- programový přístup k databázi

# SQL – neprocedurální jazyk



# SQL – neprocedurální jazyk



**a něco uloženého**

# Uložené úlohy

```
INSERT INTO Katalog ...
INSERT INTO Sklad    ...
UPDATE SKLAD SET   ...
DELETE  FROM
....
```

```
INSERT INTO Katalog ...
INSERT INTO Sklad    ...
UPDATE SKLAD SET   ...
DELETE  FROM
....
```

```
INSERT INTO Katalog ...
INSERT INTO Sklad    ...
UPDATE SKLAD SET   ...
DELETE  FROM
....
```

- provádění stejné [podobné] sekvence povelů
- provádění ucelené jednotky práce
- provádění úlohy
- prováděnou úlohu - katalogizovat, uložit

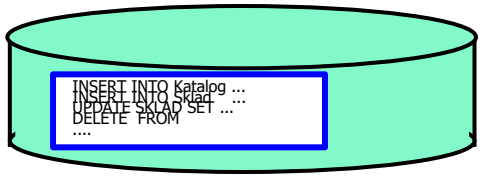
# Uložené úlohy

```
INSERT INTO Katalog ...
INSERT INTO Sklad   ...
UPDATE SKLAD SET    ...
DELETE  FROM
....
```

```
INSERT INTO Katalog ...
INSERT INTO Sklad   ...
UPDATE SKLAD SET    ...
DELETE  FROM
....
```

```
INSERT INTO Katalog ...
INSERT INTO Sklad   ...
UPDATE SKLAD SET    ...
DELETE  FROM
....
```

- provádění stejné [podobné] sekvence povelů
- provádění ucelené jednotky práce
- provádění úlohy
- prováděnou úlohu - katalogizovat, uložit



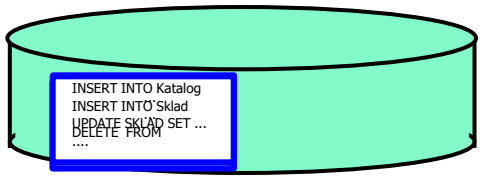
# Uložené úlohy

CALL ...

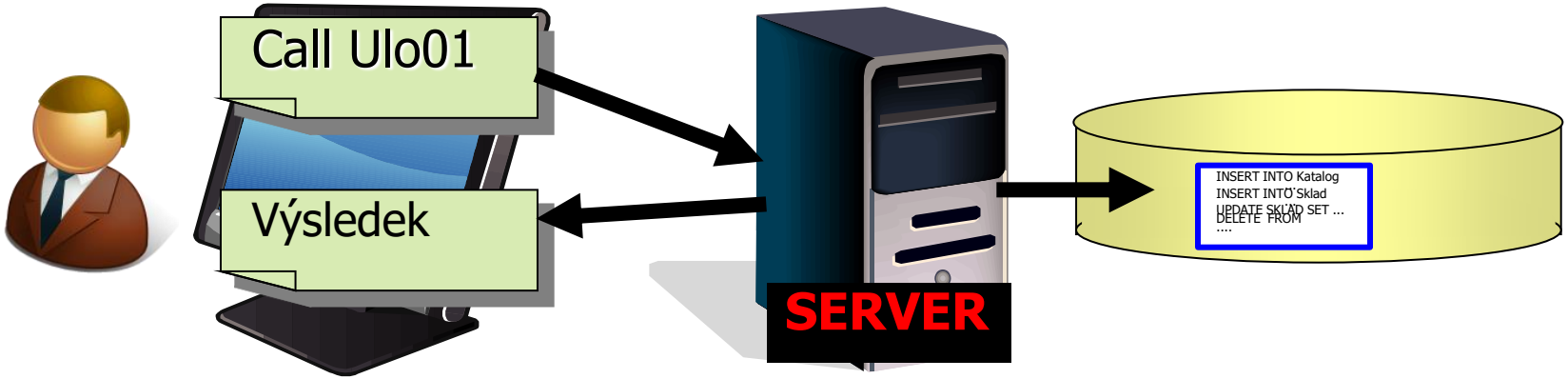
CALL ...

CALL ...

- uložená úloha se aktivuje povelem **CALL ()**



# Uložená úloha




- sada povelů SQL
- uložená jako jedna úloha na serveru
- úloha pojmenovaná
- úloha zkompilovaná [pro rychlejší použití]
- vyvolávaná a prováděná na serveru
- (metody DB serveru)
- uložená úloha je **databázový objekt**

# Potřeba jazyka pro psaní UP - UU

- deklarace proměnných
- další možnosti s nimi pracovat
- řízení toku
- smyčky, podmínky
- ošetření zvláštních stavů [chybových]

- ANSI SQL
- 90 leta - nemá prostředky pro psaní UP
- komerční DBS - vývoj - vlastní [proprietární ] jazyky
- 1992 PL/SQL - Oracle -nejpopulárnější
- 1995 T-SQL - Sybase/MS Server
- 1996 SPL - Informix
- od 1990 Jim Melton & parta [ ANSI Sql komise ]

# Jazyk v SQL/PSM - charakteristika

- jednoúčelový, zcela nový
- jednodychý procedurální s integrovaným SQL
- úzká vazba na prostředí SQL serverů
- bohatý repertoár řídících konstrukcí
- větvení [IF... ], smyček, hnízdění bloků
- konfortní model zachycení a zpracování chyb
- datové typy a funkce
- přebrány z hostitelského SQL serveru

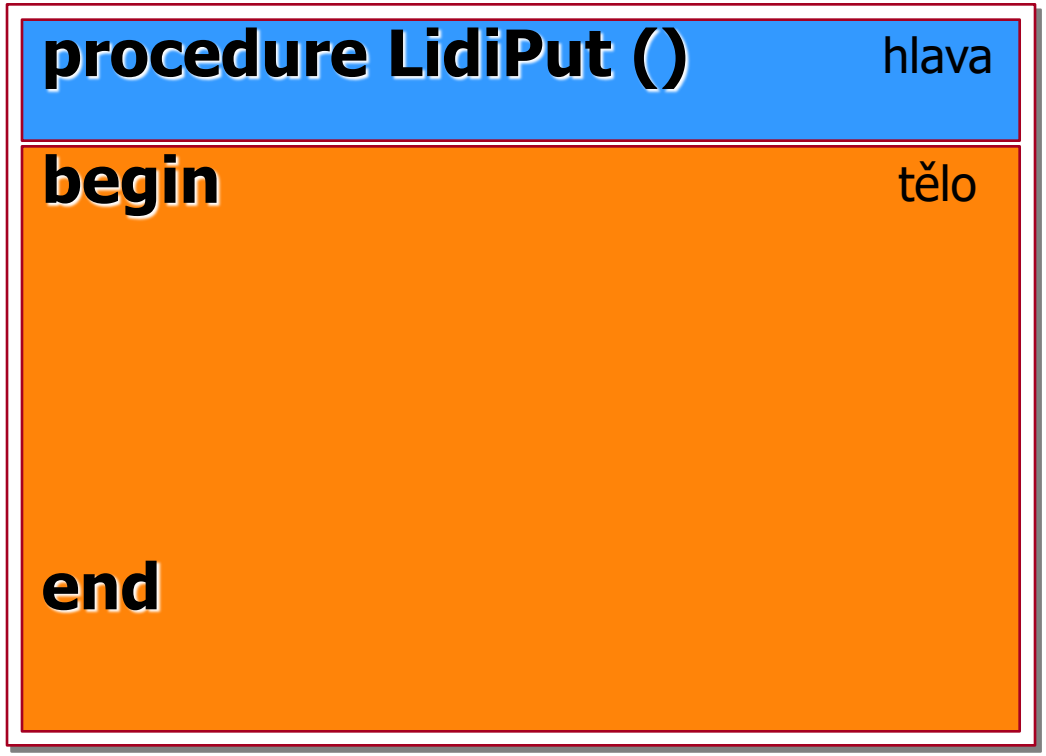
Standard SQL/PSM navíc

- popis uspořádání procedur do modulů
- popis externích procedur [C, Cobol, ...]
- nový jazyk s viditelnou inspirací
- PL/1, ADA, Modula, ...

# Uložené úlohy

Hlava
- interface úlohy

Tělo
- kódování úlohy
- mezi BEGIN ... END




```
CREATE PROCEDURE  LidiPut ()
BEGIN
    SELECT * FROM Lidi;
END
```

Error

# Vytváření db objektů

```
CREATE Table Tab    (
    -- -definice tabulky---
    id    int
);
```

konec povelu CREATE

```
CREATE INDEX ix_ -- definice indexu  --;
```

konec povelu CREATE

```
CREATE DB_objekt    Jmeno     definice_objektu ;
```

konec CREATE

```
CREATE PROCEDURE  LidiPut ()
BEGIN
        SELECT * FROM Lidi;
END
```

konec povelu CREATE

```
CREATE PROCEDURE
LidiPut ()
   BEGIN
        SELECT * FROM Lidi ;
   END ;
```

vadí nám DELIMITER

konec povelu CREATE

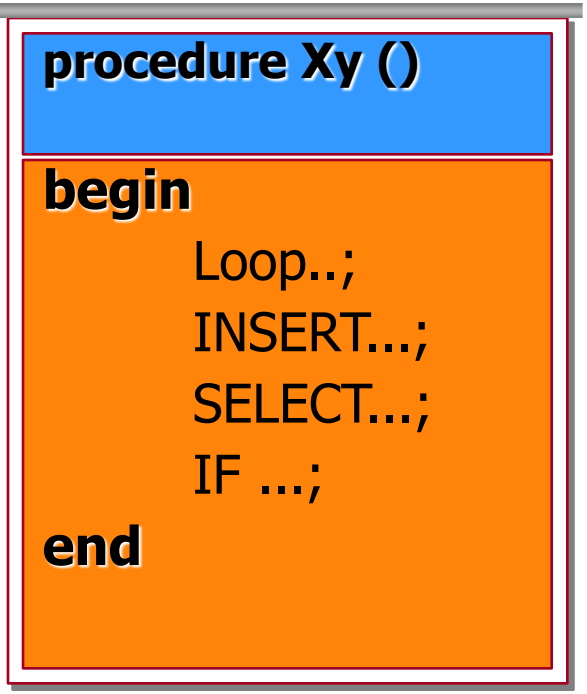
# MySQL vytvoření uložené úlohy a její vyvolání

```
DELIMITER //
CREATE PROCEDURE
LidiPut ()
   BEGIN
        SELECT * FROM Lidi ;
   END //
DELIMITER ;
```

```
CALL LidiPut ();
```

# Programovací jazyk uložených úloh

- všechny povely SQL
- povely pro řízení programu
  - každý DB stroj vlastní jazyk SP
  - SQL 2003 normalizován jazyk IBM DB2
  - MySQL užívá tuto normu SQL/PSM
- v úloze lze definovat lokální proměnné




```
CREATE PROCEDURE  Xyz ()
BEGIN
    DECLARE  i,j,k  INT;
    DECLARE   t      INT  DEFAULT 1;
    ...
    SET   i = j+k;
    SET   i = CASE ..... END;

END $$
```

# Podmíněné příkazy [ IF , CASE ]

```
CASE   CaseHodnota
       WHEN  whenHodnota  THEN  seznamPříkazů
       WHEN  whenHodnota  THEN  ...; ...; ...;
       [ELSE                    ...; ...; ...;
END    CASE;
```

```
CASE x
       WHEN 1 THEN SELECT Jmeno   From Osoba Into  Jm_;
       WHEN 2 THEN SELECT Plat    From Osoba Into   Plat _;
       ELSE            SELECT DatNar  From Osoba Into   DatN_;
END    CASE;
```

# PL/PgPSM - jazyk procedur v PostgreSQL

```
-- ================  --
-- Příklad užití povelu case   --
-- ================  --
Create  Or Replace Function foo1(a integer)
Returns void As
$$
   CASE a
      WHEN 1, 3, 5, 7, 9  THEN    PRINT a, ' je číslo liché';
      WHEN 2, 4, 6, 8, 10 THEN    PRINT a. ' je čílso sudé';
      ELSE                                PRINT a, ' není z intervalu 1..10';
   END CASE;
$$ LANGUAGE plpgpsm;
```

# Události

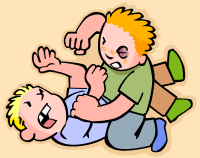
ve světě se dějí události

- o některých událostech se chcete dozvědět
- když nastanou
- že nastaly ,...
- případně na událost reagovat
- událost se obvykle děje na nějakém objektu [ ve spojení ]

# Události

ve světě se dějí události

o některých událostech se chcete dozvědět



**3 techniky**

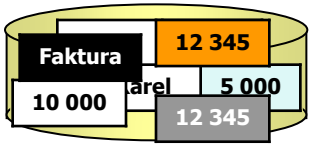
- **monitorování**
- každých 30 minut voláte do školy

- **dohoda** s objektem události - že Vám dá vědět
- **delegace** Vaší reakce na objekt události
- perou se ==> rákoska, nařežte mu
- zkouška: 4 nebo 5 ==> dát pohlavek
- 1 ==> pohladit po hlavě

# Události

- v DB se dějí události - mění se data

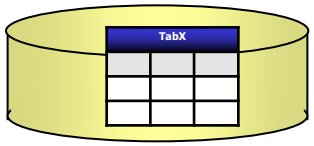
**Insert** **Update** **Delete**



# Události

- v DB se dějí události - mění se data

**Insert** **Update** **Delete**




- na jakém objektu se to děje ?
- na tabulce [na sloupci, na řádce]
- pro každou událost [ tabulky ] lze zadat akci [rutinu]
- automaticky se spůstí pro danou událost

**Spoušť** **Trigger**

- kolik událostí nastane **?**

```
Update Osoby
    Set  Plat = Plat + 1000;
```

- 1

1 povel Update na tabulce Osoby

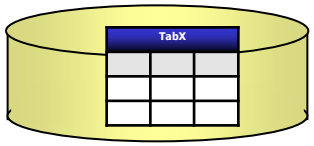
- 555

na 555 řádcích [osobách] tabulky

# Události

- v DB se dějí události - mění se data

**Insert** **Update** **Delete**




- na jakém pbjektu se to děje ?
- na tabulce
- pro každou událost [ tabulky ] lze zadat rutinu
- automaticky se spůstí pro danou událost

**Spoušť** **Trigger**

```
Update Osoby

    Set  Plat = Plat + 1000;
```

- kolik událostí nastane

Nastanou **dvě** [trochu] **různé** události [události dvou druhů]

- událost změny [ **Update** ] na tabulce **1 x**
- událost změny [ **Update** ] na řádce tabulky **555 x**

# Události

- do události chceme zasáhnout ve dvou různých fázích [průběhu]

**před tím, než nastala** — před zkouškou

- nemá špínu za nechty ? — kontrola
- nemá-li kapesník, půjčte mu ho — nasazení default hodnot
- má-li teplotu > 38 poslat domů — zamezení události

**dynamické zajištění integrity dat**

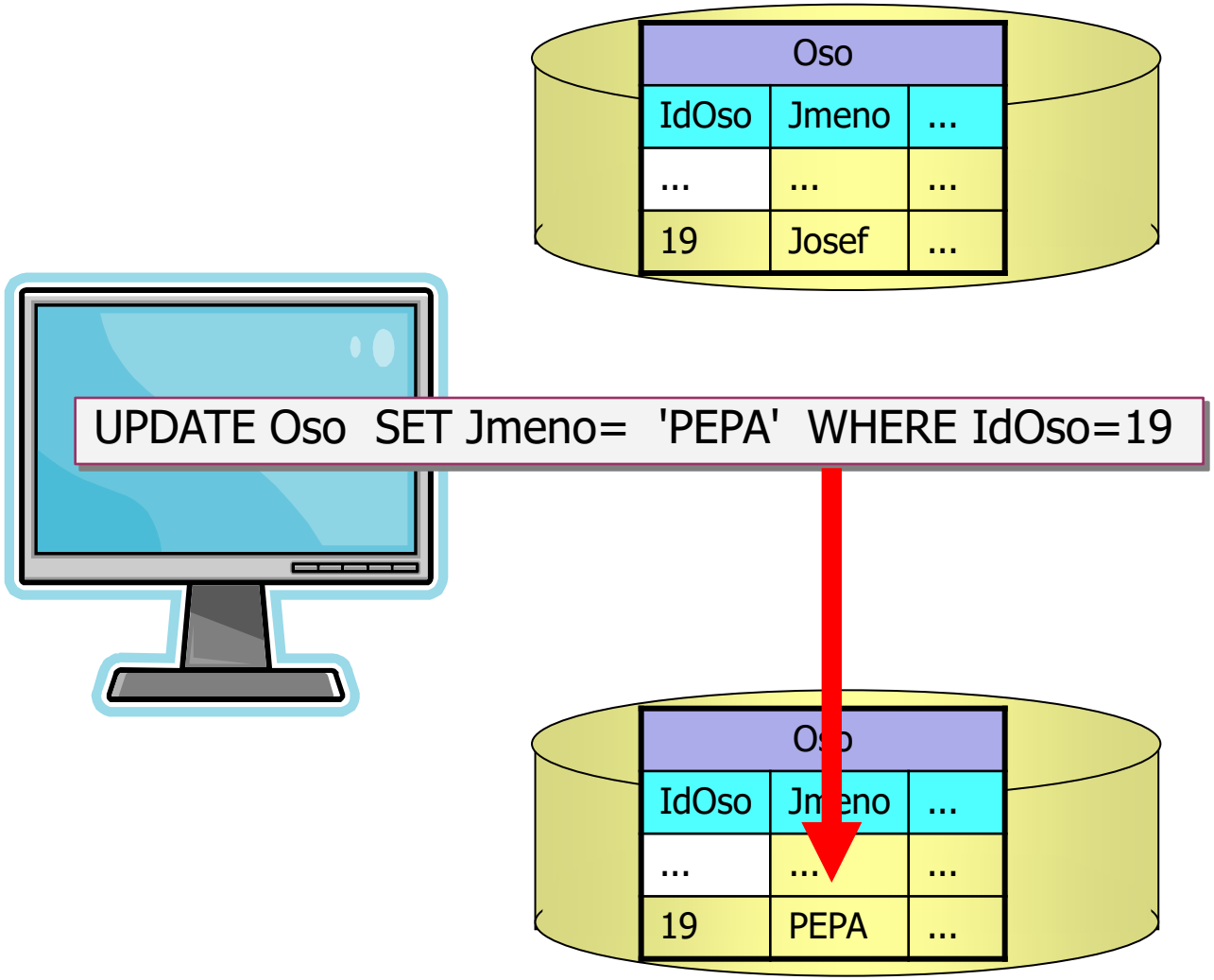
**poté**

- zapsat do knížky
- výsledek 4,5 => pohlavek — protokolování
- 1,2 => pohlazení — úprava jiných tabulek
- poslat do cukrárny — akce vně aplikace [DB]
- oznámit [rozvedené] matce

**následné akce uvnitř DB i venku**

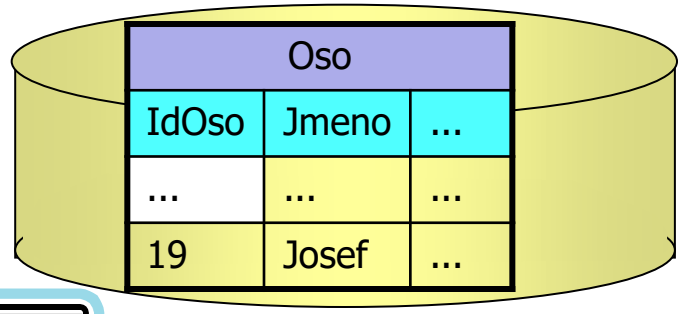
# Update tabulky bez spouště

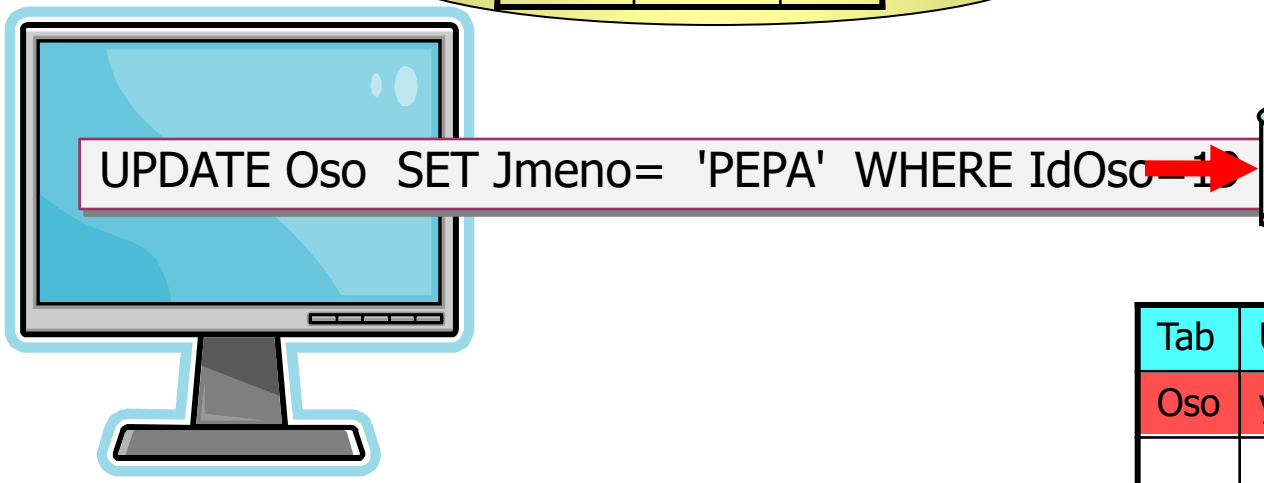
Bez spouště



# Update tabulky se spouští

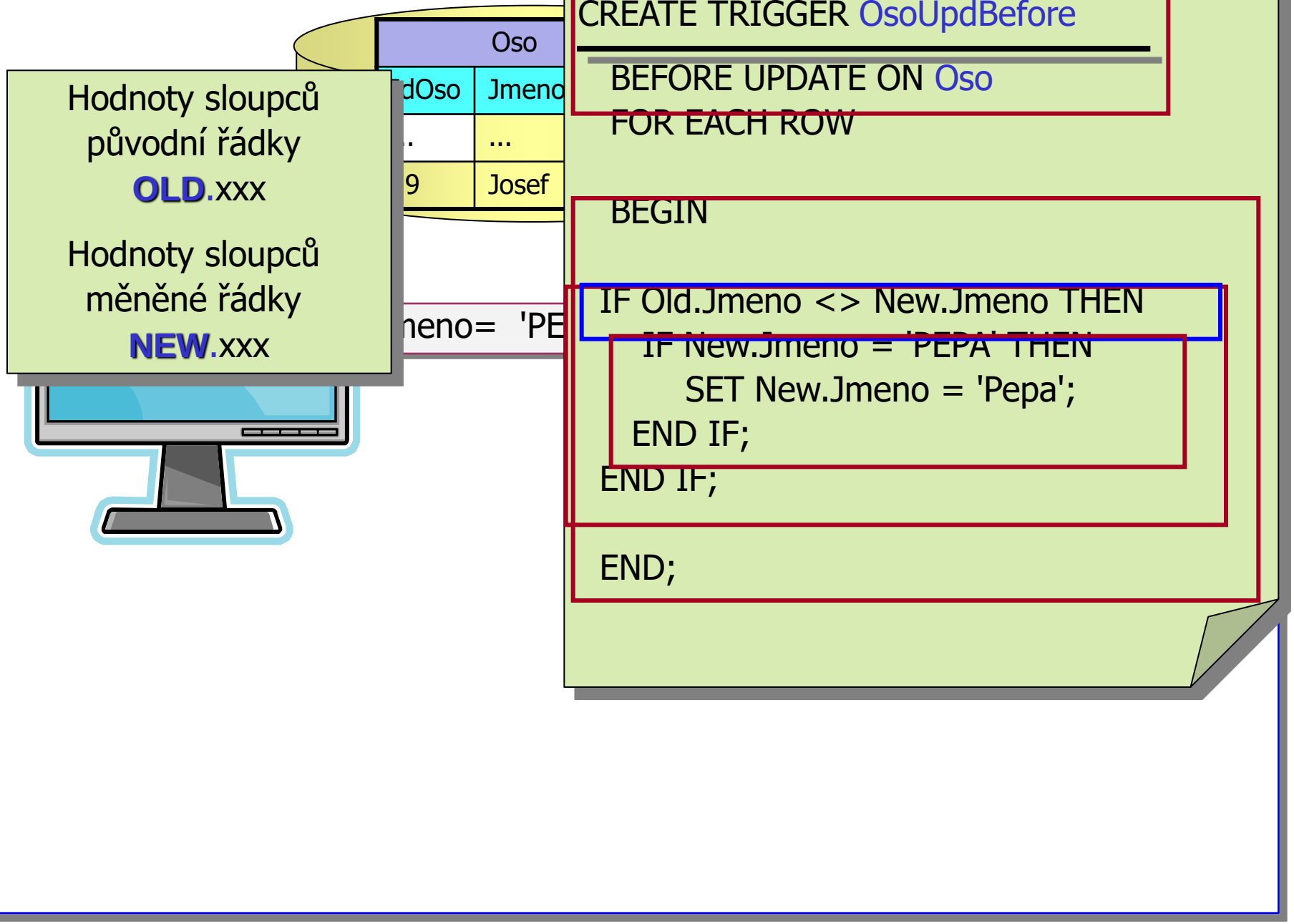
Se spouští



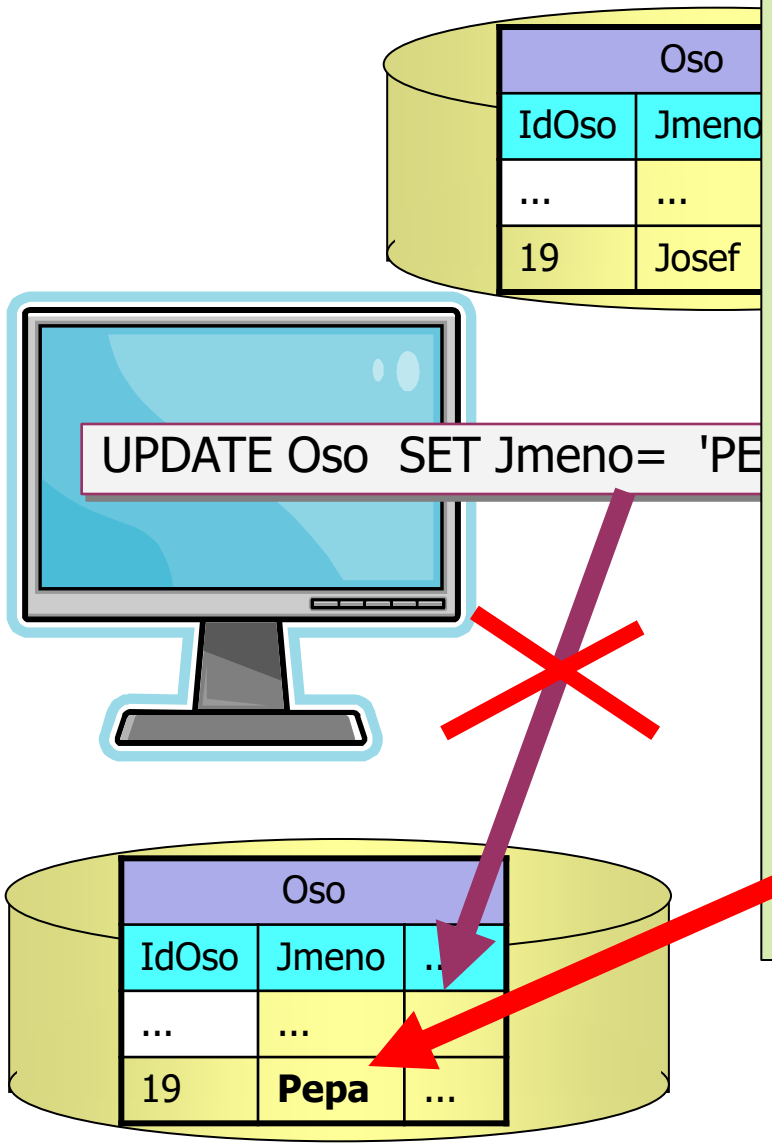




| Tab | UpdateTrigger(Before) | .. |
|---|---|---|
| Oso | yes | .. |
| | | |

# Update tabulky se spouští

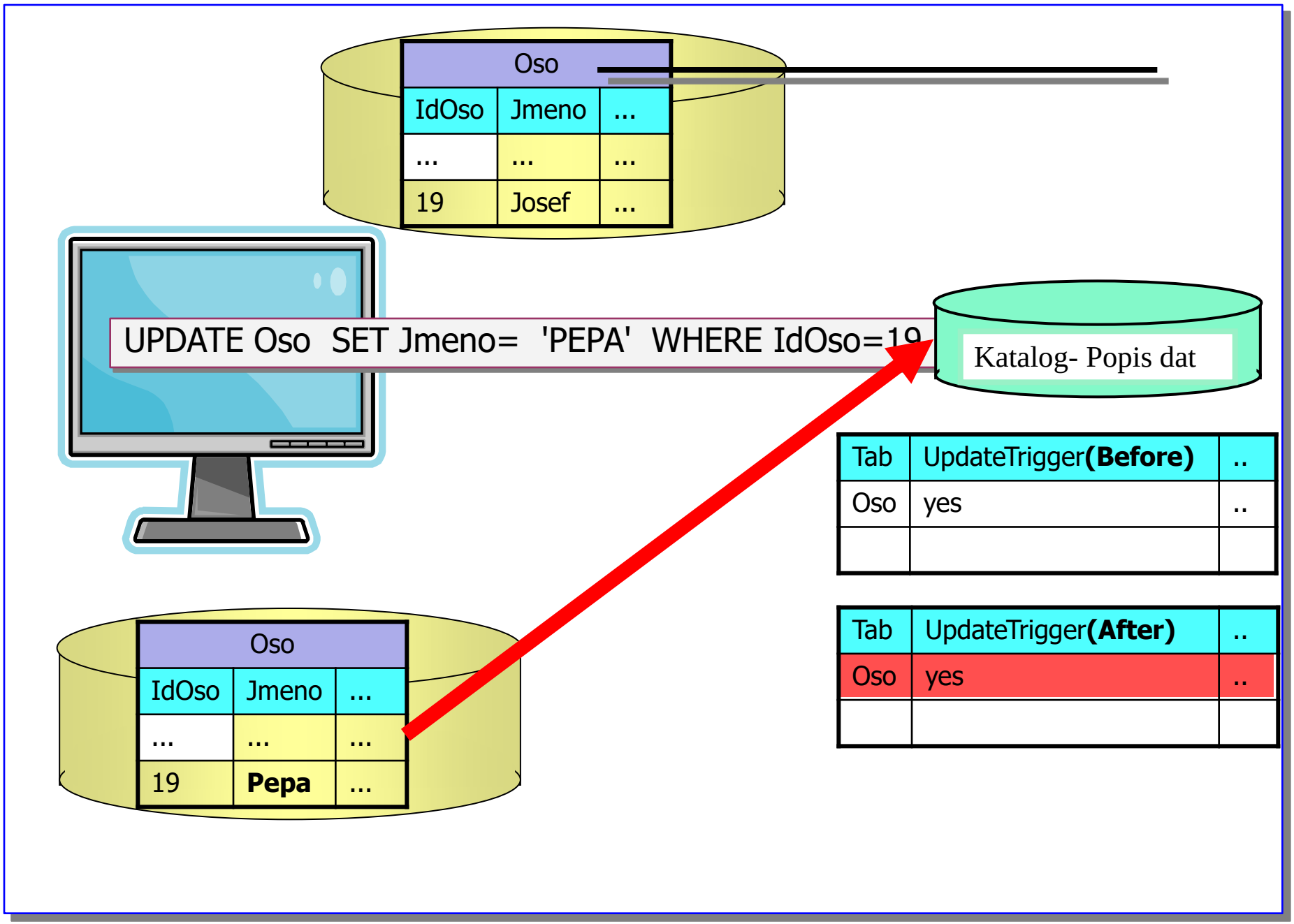


# Update tabulky se spouští

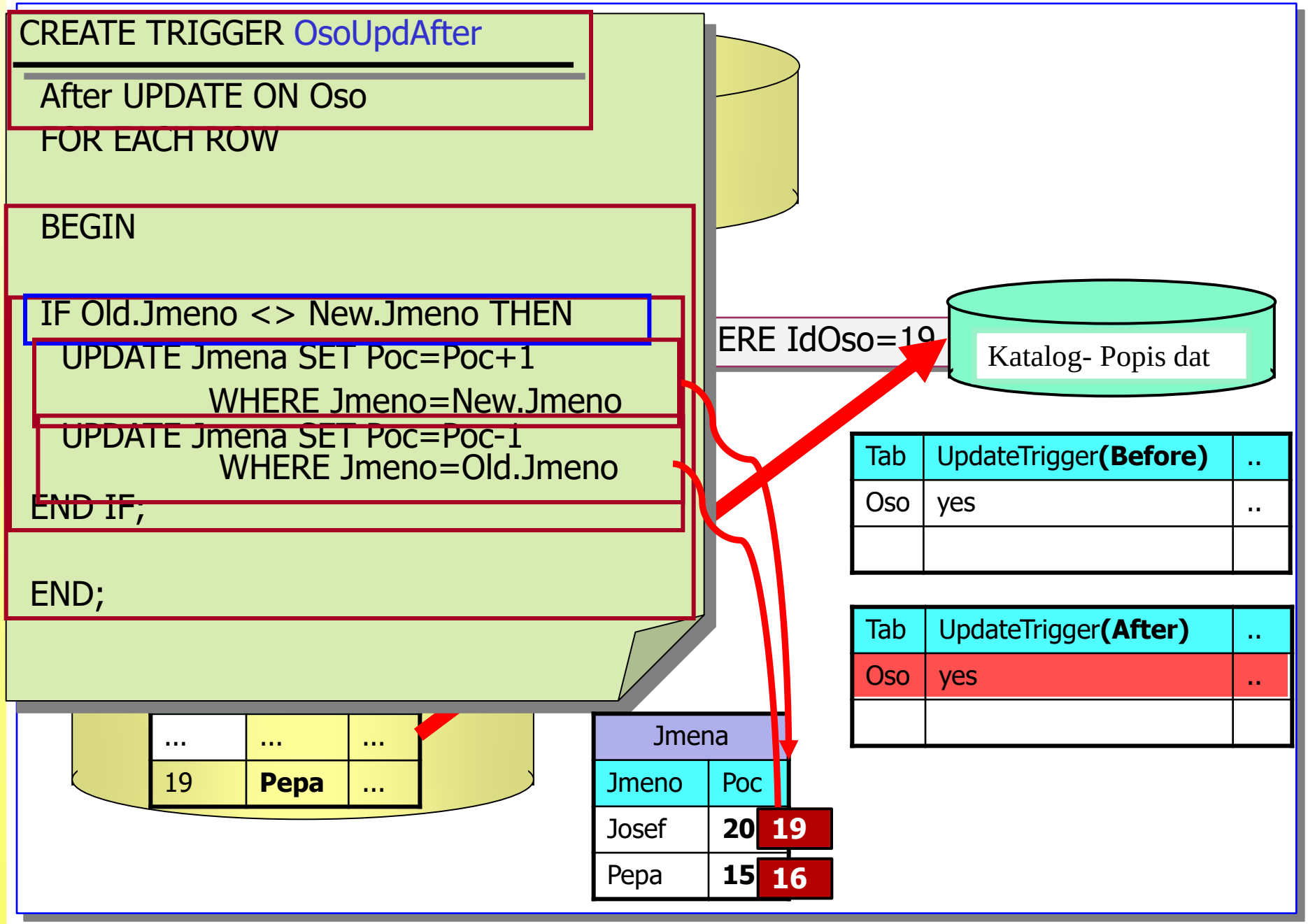



```
CREATE TRIGGER OsoUpdBefore
  BEFORE INSERT ON Oso
  FOR EACH ROW

  BEGIN

 IF Old.Jmeno <> New.Jmeno THEN
     IF New.Jmeno = 'PEPA' THEN
         SET New.Jmeno = 'Pepa';
    END IF;
 END IF;

 END;
```

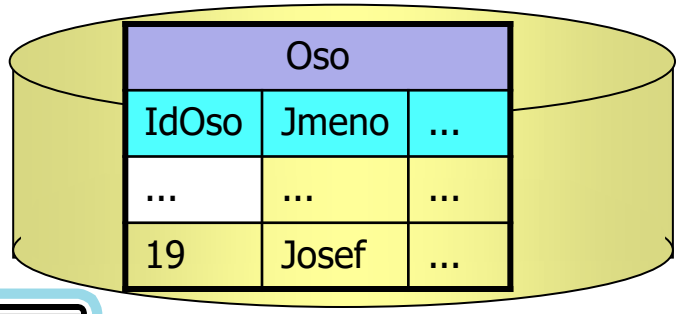
# Update tabulky se spouští

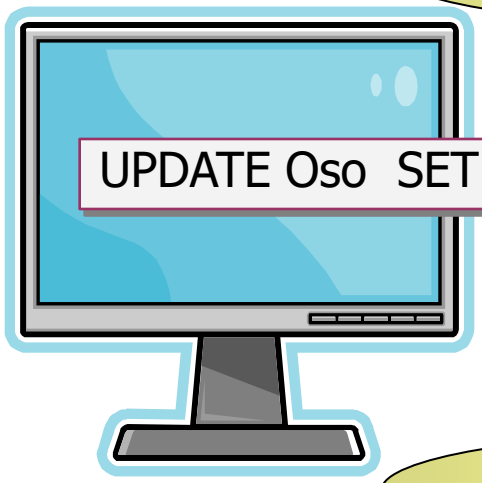


# Update tabulky se spouští

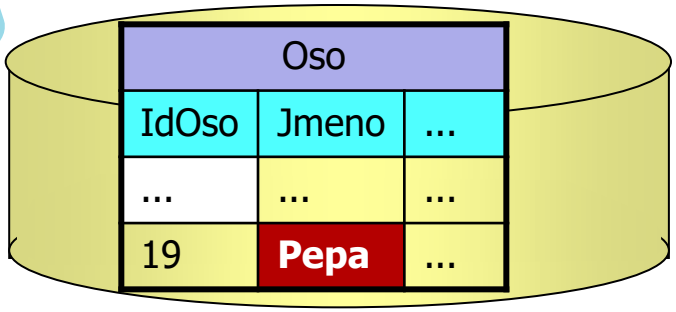


# Update tabulky se spouští



```
UPDATE Oso  SET Jmeno=  'PEPA'  WHERE IdOso=19
```

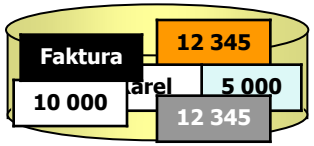



| Jmena | |
|---|---|
| Jmeno | Poc |
| Josef | **19** |
| Pepa | **16** |

Povel UPDATE ukončen

# Spouště - triggery

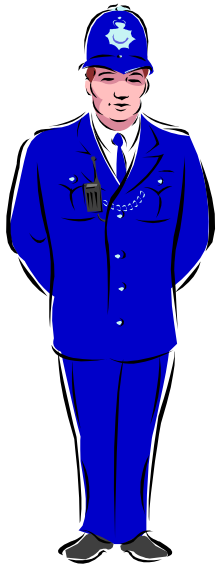
## Spoušť [ trigger ]




- databázový objekt (konstrukce)

- realizuje dynamická a
- nedeterministická obchodní pravidla

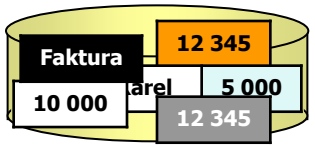
- centralizovaně kontroluje a monitoruje
- všechny změnové události v DB
- i oproti všem manipulacím programů



- **centrální ochranka !!!** a následný servis

# Spouště se spouštějí

- u změnových operaci
  [ **Insert, Update, Delete**]



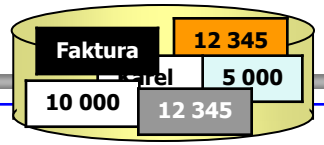

- PŘED provedením změny .....
- PO provedení změny .....
- MÍSTO provedení změny .....

**Before**
**After**
**Instead Of**

- jednou pro daný povel .....
- či není-li zadáno .....
- jednou pro každou řádku .....
- vkládanou
- měněnou
- rušenou

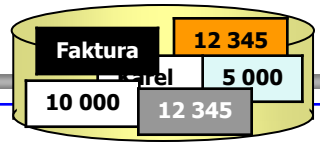
**For Each Statement**
For Each Row
**For Each Row**

# Užívání spouští - BEFORE for each row



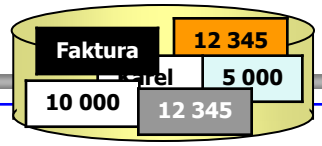

?

# Užívání spouští - BEFORE for each row




- zabránění změny hodnot určitých polí za určité situace
  - nezměnitelné hodnoty
- inicializace hodnot (zobecnění DEFAULT – i dynamické)
  - dle vyhodnocení SELECT z jiných tabulek (nebo stejné)
  - nasazení posloupnosti 1,2,3,4,5,... do určitých sloupců
  - inicializace výrazem - závislost na jiných sloupcích
    - alfa=beta+gama [sloupec alfa výrezem jiných]
  - do sloupců se nasazují údaje [ dynamicky ]
  - CURRENT_DATE
  - CURRENT_USER

# Užívání spouští - BEFORE for each row

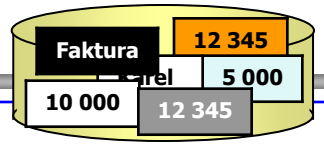



- normalizace změnových hodnot
  - normalizace textu
    - zkratka státu 1.znak vellký, další malé
    - gramatická normalizace
      - v němčině nahrazování SS ostrým ß
      - ue ->ü atd

- kontrola
- zda povel na řádce povolit [na některé **ano**, jinde **ne**]

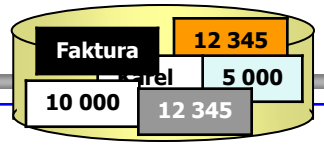
- DELETE
  - zabránění rušení některých řádek (algoritmická závislost)

# Užívání spouští - BEFORE for each row



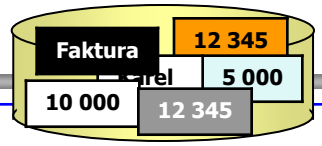

- zabránit povelu (za určitých podmínek)
  - *na určité řádce*
- iniclializovat některá pole
- normalizovat hodnoty některých polí

# Užívání spouští - AFTER for each row



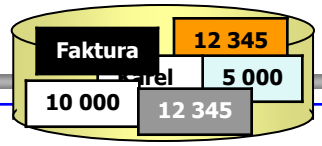

?

# Užívání spouští - AFTER for each row



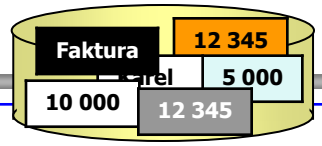

- je-li žádoucí INSERT, UPDATE, DELETE pro
  - jiné řádky téže tabulky
  - změny jiných tabulek
- pokud je třeba testovat data oproti sumárním údajům
  - zbytku tabulky
  - jiných tabulek

# Užívání spouští - AFTER for each row




- je-li třeba aktivovat operace mimo DB
  - vyslat varování mimo DB
  - potřeba změnit informace mimo DB
  - zahájit nebo pokračovat v řízení operací mimo DB
  - v závislosti na změnách v DB
- protokolování změn
  - včetně údajů o CURRENT_USER
  - CURRENT_DATE

# Užívání spouští - AFTER for each row



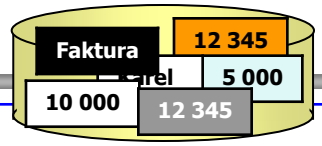

- je-li třeba aktivovat operace mimo DB
  - vyslat varování mimo DB
  - potřeba změnit informace mimo DB

## DB Banka - tabulka Ucet

- klesne-li stav účtu pod zadanou hranici
- zaslat majiteli účtu zprávu emajlem

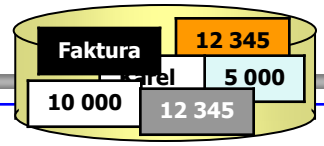
  - CURRENT_DATE

# Užívání spouští Instead Of




- nenormalizováno
- spoušť se provede místo dané akce
- zamezení nějaké změnové operace na určité tabulce
- úprava standardní chování změnové operace
  - (například DELETE)
    - nejprve záznam převést do archivní tabulky
    - pak teprve původní záznam zrušit
- realizace UPDATE neaktualizovatelných pohledů

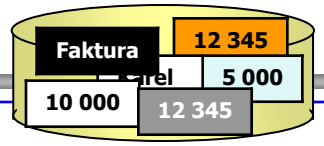
# Užívání spouští Instead Of




```
Create    TRIGGER   Bohuzel
                    INSTEAD OF DELETE
                    ON          Konto
AS
                    PRINT ' Sorry, konta nelze rušit '
END
```

- povel DELETE na tabulce Konto se neprovede
- místo toho se napíšee uvedená zpráva
- rozhodně by bylo vhodné
- takový pokus o zrušení protokolovat

# Oslovení původních a změněných dat




- původní data lze oslovit s kvalifikací **OLD**

```
IF    OLD.Pocet > 0          THEN   .....
IF    OLD.Jmeno  IS NULL THEN  .....
```

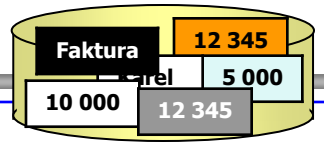
- změněná data lze oslovit s kvalifikací **NEW**

```
IF    Old.Pocet <> New.Pocet THEN   .....
IF   New.Jmeno  ="Karel"        THEN  .....
```

- kvalifikace **NEW** a **OLD** lze změnit za své zkratky
- **Referencing New AS Nova_Obj [ alias ]**
- **Referencing Old AS Stara_Obj [ alias ]**

```
IF    Stara_Obj. Pocet <> Nova_Obj. Pocet THEN   .....
```

# Přístup ke kontextovým datům - proměnným




```
NEW.Pocet   =  OLD.Pocet+1;
NEW.Celkem = NEW.Mnoz * NEW. Cena;
```

?

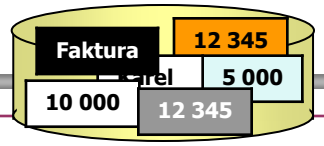
- OLD (insert, update, delete)
  - INSERT
  - UPDATE předchozí
  - DELETE aktuální

?

- NEW (insert, update, delete)
  - INSERT nová hodnota sloupce
  - UPDATE nová hodnota sloupce
  - DELETE není dostupná

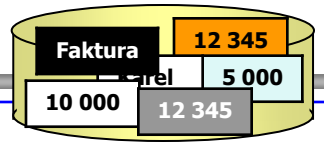
- prefixy NEW a OLD lze pro spoušť přejmenovat

# Vytvoření spouště




```
CREATE    [ OR REPLACE ]          TRIGGER   JmenoTrigeru

BEFORE | AFTER | INSTEAD OF   INSERT OR UPDATE OR DELETE
INSERT  | DELETE  | UPDATE      [OF seznam_sloupců]
ON            Tabulka
[REFERENCING   OLD  [ROW]   [AS]  StaryZaznam
               NEW [ROW]   [AS]  NovyZaznam
               OLD  TABLE   [AS]  StaraTabulka
               NEW TABLE   [AS]  NovaTabulka
]
FOR EACH  { ROW | STATEMENT }   [MODE]
[WHEN       (podmínkaTrigeru) ]
jedenSqlPovel
|
BEGIN
          ... vnitřní část trigeru - akce ...
END
```

# PostgreSQL spouště - omezení standardu




- nelze zadat alias pro prefixy **OLD** a **NEW**

```
Create Trigger ... On Tabulka Referencing Old as Stara ...
```

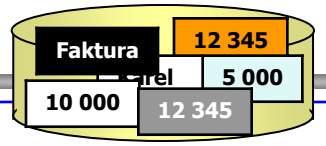
- pro UPDATE nelze definovat spoušť pro změnu sloupců

```
Create Trigger ... Update Of Sloup1, Sloup4 ON Tabulka ...;
```

- uvnitř PostgreSQL spouště nutno pouze
- volat uživatelskou fukci [která vrací Void]

```
Create Trigger Osoba_UpdBef Before Update On Osoba
        For Each Row
        Execute Procedure MojeFunkce (seznam_argumentů);
```

# Různé dialekty spouští



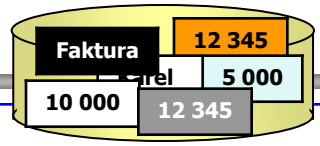

```
Create   TRIGGER    NazevTrigru
         AFTER      UPDATE
         ON         Tab
AS
         .......
END
```

Norma

```
Create   TRIGGER    NazevTrigru
         ON         Tab
         FOR        UPDATE
AS
         .......
END
```

```
Create   TRIGGER    NazevTrigru
         FOR        Tab
         AFTER      UPDATE
AS
         .......
END
```

# Spouště samose měnících tabulek




**[ Oracle, MySQL ]**

- zabrání, aby spuště měnili tabulku
- které spoušt náléží

ORA-yyy: table XYZ is mutating, trigger/function may not see it

- mnohdy nelze uvnitř spouště tabulku spouště číst (Select)

proč ne ??

**[SQL Server]**

- umožňuje rekurzivitu volání spouští
  - uvnitř spouště lze měnit tabulku spouště