Aktivní databáze, triggery

Tomáš Hejl & Jiří Kunčar

Osnova

- úvod - terminologie
- syntax a sémantika vybraných databázových systémů
  - Starburst
  - Oracle
  - DB2
  - Chimera
- shrnutí principů aktivních databází
- praktické použití aktivních pravidel

Úvod

Databáze

- kontrola dat pomocí integritních omezení
  - ne vždy dostatečná

Aktivní databáze

- rozšíření databázových systémů o *aktivní pravidla*

Aktivní pravidla = triggery

- Událost - Podmínka - Akce
- na databázové úrovni
- na jednom místě

Aktivní pravidla

- Událost - Podmínka - Akce
  (reakce na změnu dat v DB, vyhodnocení podmínky a příslušné akce)

- komplexnější pravidla než v integritních omezeních

- usnadnění programátorské práce

- pravidla implementována přímo v databázi, společná pro všechny aplikace, které ji používají

=> silnější než obyčejná "pasivní" databáze

Historie aktivních databází

- první pokusy koncem 80.let
- nestihlo se do SQL-92

- vývojáři přináší vlastní implementace, co možná nejblíže rozpracovanému návrhu standardu
  - 1.polovina 90.let - Oracle
  - 1996 - DB2 od IBM, svůj návrh prosadí do SQL

Starburst

Starburst

- vyvíjeno IBM Almaden Research Center
- postaveno na SQL
- aktivní pravidla v rozšíření SARS (Starburst Active Rules System)
- jednoduchá syntaxe i sémantika
- rozšíření jazyka umožňuje vytváření a mazání aktivních pravidel

Starburst - UPA

Událost-Podmínka-Akce (ECA, Event-Condition-Action)

Událost

- **`INSERT, DELETE, UPDATE`**

Podmínka

- boolovský predikát, vyjádřený v SQL

Akce

- Libovolné SQL příkazy
  - **`SELECT, INSERT, DELETE, UPDATE`**...
- Příkazy pro řízení transakcí (**`ROLLBACK WORK`**)

Starburst - syntaxe

```
CREATE RULE <název_pravidla>
ON <název_tabulky>
WHEN <události>
[ IF <podmínka> ]
THEN <SQL-akce>
[ PRECEDES <seznam_názvů_pravidel> ]
[ FOLLOWS <seznam_názvů_pravidel> ]


<události> = INSERTED | DELETED |
UPDATED [(<názvy_sloupců>)]
```

Starburst - příklad

```
CREATE RULE vysoke_platy
ON zamestnanci
WHEN INSERTED, DELETED, UPDATED
IF (SELECT avg(plat) FROM
zamestnanci) > 100
THEN UPDATE zamestnanci SET
 plat = 0.9 * plat
```

Starburst - Čistý efekt

- pomocné tabulky
  - **`INSERTED`** - co bylo přidáno
  - **`DELETED`** - co bylo smazáno
  - **`OLD-UPDATED`** - co se změnilo (předchozí stav)
  - **`NEW-UPDATED`** - co se změnilo (nový stav)

Čistý efekt (Net effect)

V tabulkách jsou jen čisté výsledky:

Příklady:

- několik **`UPDATE`** stejného řádku má ve výsledku efekt jako samotný poslední z nich.
- **`INSERT`** a **`DELETE`** - efekt jako **`DELETE`**

Starburst - Další syntaxe

- Sdružování pravidel do sad
  - **CREATE RULESET** <nazev_sady>
  - **ALTER RULESET** <nazev_sady>
    [**ADDRULES** <pravidlo>]
    [**DELRULES** <pravidlo>]
  - **DROP RULESET** <nazev_sady>
- manipulace s pravidly
  - **DEACTIVATE RULE** <pravidlo> **ON** <tabulka>
  - **ACTIVATE RULE** <pravidlo> **ON** <tabulka>
  - **DROP RULE** <pravidlo> **ON** <tabulka>

Starburst - chování triggerů

- *odložené spuštění* - pravidla se kontrolují po COMMITu celé transakce
- možnost spuštění ručně
- ruční spouštění pravidel
  - **`PROCESS RULES`**
  - **`PROCESS RULE`** `<pravidlo>`
  - **`PROCESS RULESET`** `<sada_pravidel>`
- jedno pravidlo může sledovat více událostí
- více pravidel může sledovat jednu událost
  - pořadí pravidel je určeno pomocí **`FOLLOWS`** a **`PRECEDES`**
  - musí být acyklické
  - zajištění, aby se triggery nezacyklily, je pouze na programátorovi

Starburst - sémantika

Pravidlo je:

Spuštěné (triggered)

- nastala událost, ke které se váže

Bráno v úvahu (considered)

- je vyhodnocena a splněna podmínka

Provedeno (executed)

- je vykonána akce pravidla

Oracle

Oracle

- triggery vyvíjeny podle připravované specifikace SQL

Dva typy aktivních pravidel:

- řádkové triggery (row-level)
  - monitorují události na jednotlivých řádcích, při změně více řádků se spouští na každém zvlášť
- příkazové triggery (statement-level)
  - monitorují celé příkazy, které mohou měnit více řádků najednou

Oracle - syntaxe

```
CREATE TRIGGER <název_triggeru>
{BEFORE | AFTER} <události>
ON <název_tabulky>
[[ REFERENCING <reference> ]
FOR EACH ROW
[ WHEN (<podmínka>) ]]
 <PL/SQL_kód>


<události> = INSERT | DELETE | UPDATE
[OF <seznam_sloupců>]


<reference> = OLD AS <stará_hodnota> | NEW AS
<nová_hodnota>
```

Oracle - UPA

Událost-Podmínka-Akce (ECA, Event-Condition-Action)

Událost

- **`INSERT, DELETE, UPDATE`**

Podmínka

- řádkový predikát, vyjádřený v SQL
- pouze pro řádkové triggery!

Akce

- Libovolný PL/SQL kód
  - **`SELECT, INSERT, DELETE, UPDATE`**...
  - bez DDL příkazů a transakčních příkazů

Oracle - další syntaxe

- predikáty **`INSERTING, DELETING, UPDATING`**
  - pro určení konkrétní události, která pravidlo spustila
- reference **`OLD, NEW`**
  - staré a nové hodnoty změněného sloupce
  - pouze u řádkových triggerů
- granularita
  - **`FOR EACH ROW`** = řádkový trigger, jinak příkazový

Oracle - spouštění triggerů

- není zpožděné, nastává okamžitě s událostí
- dvě možnosti spuštění akce
  - **`BEFORE <událost>`** - těsně před provedením události
  - **`AFTER <událost>`** - ihned po provedení události
- nelze spustit ručně (příkazem jako u Starburstu)
- trigger může spustit jiný trigger
  - maximální hloubka zanoření 32, pak výjimka
- při výjimce nebo chybě jsou vráceny všechny změny původní operace i všech triggerů

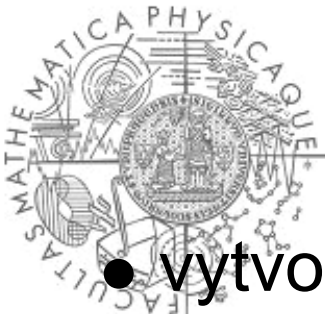
Oracle - pořadí spouštění

- příkazové "before" triggery
- Pro každý řádek v tabulce
  - řádkové "before" triggery
  - změna řádku a kontrola jeho integrity
  - řádkové "after" triggery
- kontrola integrity na úrovni příkazu
- příkazové "after" triggery

Pořadí v rámci jednotlivých úrovní

- Podle pořadí vytvoření
- Nově (verze 11.1) klauzule **`FOLLOWS`** jako u Starburstu

DB2



DB2 - triggery

- vytvořeno IBM Almaden Research Center
- snaha o jednoznačnou sémantiku
- podle zkušeností se systémem Starburst

- každý trigger monitoruje jedinou událost
  - **`UPDATE, DELETE`** nebo **`INSERT`**

- spouštění stejné jako u Oracle
  - **`BEFORE`** nebo **`AFTER`**
  - řádkové a příkazové triggery

- více triggerů pro jednu událost
  - uspořádání podle času vytvoření
  - řádkové i příkazové triggery jsou promíchány

DB2 - syntaxe

```
CREATE TRIGGER <název_triggeru>
[ BEFORE | AFTER ] <událost>
[OF <sloupce>]
ON <název_tabulky>
[ REFERENCING <reference> ]
[FOR EACH ROW | FOR EACH STATEMENT]
WHEN  <podmínka>
<SQL_kód>
```

DB2 - detaily

- Reference - rozdílné pro příkazové a řádkové triggery

```
<reference> = [OLD AS  <původní_hodnoty> ]
[ NEW AS <nové_hodnoty> ]
[ OLD_TABLE AS  <původní_tabulka> ]
[ NEW_TABLE AS <nová_tabulka> ]
```

- kód akce nesmí obsahovat
  - DDL příkazy (měnit schéma DB)
  - transakční příkazy
- kód akce smí obsahovat volání chyb (=> rollback příkazu)

DB2 - sémantika

- **`BEFORE`** triggery
  - vhodné ke kontrole dat, před vložením do DB
  - nesmí měnit obsah DB (=> nespouští další triggery)
  - mohou upravovat měněná data vkládáním do proměnných **`NEW`**
  - mohou vyvolávat chyby

- **`AFTER`** triggery
  - zastupují aplikační logiku
  - spuštěny po změně dat
  - stav před událostí musí být odvozen z přechodových tabulek (`<tabulka>` **`MINUS`** `NEW_TABLE`) **`UNION OLD_TABLE`**

DB2 - příklady

```
CREATE TRIGGER ZadanyVek
BEFORE UPDATE OF Vek ON Uzivatele
REFERENCING NEW AS UpravenyUzivatel
FOR EACH ROW
WHEN (UpravenyUzivatel.Vek IS NULL)
SIGNAL SQLSTATE '70005' ('Vek musi byt
uveden')
```

```
CREATE TRIGGER SledovaniPoctuUzivatelu
AFTER UPDATE ON Uzivatele
REFERENCING OLD_TABLE AS Tab
FOR EACH STATEMENT
INSERT INTO PoctyUzivatelu VALUES
( CURRENT_DATE,(SELECT COUNT(*) FROM Tab) )
```

Chimera

Chimera

- objektově orientovaný databázový jazyk
  - aktivní pravidla = triggery
- **třídy** (*object classes*)
  - obsaženy v definici schémat
  - popisují ovlivňování objektů událostmi
  - podporují okamžité i odložené vykonání
- **atributy třídy** určují její **stav**
  - typy konstruktorů atributů:
  - record, set, list
  - atomické typy (integer, string, ...)
  - jména definovaných tříd

Chimera - příklad třídy

define object class Zamestnanec
attributes Jmeno: **string**,
Plat: **integer**
end;

define object class Oddělení
attributes Jmeno: **string**,
Zamestnanci: **set-of(**Zamestnanec**)**
end;

- třídy jsou hierarchicky uspořádány
- obsahují mimo atributů, také definice operací, intergritních omezení a triggerů

Chimera - deklarativní výrazy

Termy

- atomické
  - konstanty (6, 0, 1, ...)
  - proměnné (X, Y, Z, ...)
- složené
  - konstruktory (**`set, list, record`**)
  - atributy, selektory, operátory

Formule

- atomické
  - predikátový symbol a seznam parametrů
  - typy: `integer(X)`
  - třídy: `Zamestnanec(Z)`
  - porovnání: `Z.jmeno='Jan'`
  - náležení: `Z` **`in`** `O.Zamestnanci`
- složené
  - konjunkce a disjunkce formulí
  - negace atomických formulí

`not``(Z` **`in`** `O.Zamestnanci)`

Chimera - procedurální výrazy

- **select(**cílový seznam **where** podmínka**)**
  - `select(Z where Zamestnanec(Z), Z.jmeno = 'Jan')`
- **create(**třída, hodnota, výstupní proměnná**)**
  - `create(Zamestnanec, ['Jan', 55000], Z)`
- **delete**(třída, proměnná)
  - `delete(Zamestanec, Z)`
- **modify**(název atributu, proměnná, term)
  - `modify(Zamestnanec.Plat, Z, Z.plat * 0.96)`

Chimera - triggery

- **transakční řádka** (*transaction line*)
  - nejmenší jednotka, která může být zachycena triggerem
  - zřetězení procedurálních výrazů oddělených čárkou a ukončené středníkem
  - platnost proměnných je definována v rámci jedné řádky
  - reference získané v predikátu select nebo create je možné používat v dalších výrazech na dané transakční řádce

Chimera - syntaxe definice trigerů

```
<definice-triggeru> ::=
   define <volba> trigger <jméno triggeru>
   [ for <jméno třídy> ]
   events <události>
   condition <formule>
   actions <proceduralni výraz>
   [ { before | after } <jména triggerů>]
   end
<volba> ::=
   [ {event-consuming | event-preserving }]
   [ {deferred | immediate }]
<události> ::=
   create | delete | modify
   [(<jméno-atributu>)]
```

Chimera - syntaxe triggerů 2

- události:
  - `display`
  - `generalize`
  - `specialize`
- **targeted** vs. **untargeted**:
  - definovaná proměnná **`Self`** odkazující na danou třídu

- pořadí
  - **`before`** a **`after`** určují částečné uspořádání
  - úplné uspořádání je určeno časem vytvoření triggeru

Chimera - sémantika triggerů

- odložené (*deffered*) triggery
  - volány příkazy commit nebo savepoint
  - okamžité triggery spuštěné akcí odloženého triggeru se zařadí do fronty s odloženými triggery
- okamžité (*immediate*) triggery
  - volány po dokončení transakční řádky
- pořadí
  - before a after určují částečné uspořádání
  - úplné uspořádání určeno systémem
    - čas vytvoření triggeru

Chimera - sémantika více triggerů

- množina spuštěných triggerů
  - stejně jako u Starburstu
- výběr z množiny

DOKUD je množina spuštěných *neprázdná* {
    vyber jeden trigger nejvyšší v pořadí
    POKUD je podmínka *splněna* {
        vykonej akci triggeru
    }
}

Základní pojmy

- události (*events*)
  - změna stavu databáze
  - hledání záznamu
  - časové události (v 17h, každý pátek)
  - definované aplikací
- podmínky (*conditions*)
  - databázový dotaz: `FALSE` pokud výsledek dotazu je prázdný, jinak `TRUE`
  - databázový predikát
- akce (*actions*)
  - libovolná manipulace s daty zahrnující:
    - transakční příkazy (`ROLLBACK`)
    - pravidla zpracování (aktivace/deaktivace pravidel)
    - externí procedury

Vyhodnocení triggeru

- pravidlo:událost = 1:1 nebo 1:n
- **okamžité** (*immediate*)
  - před (`BEFORE`), po (`AFTER`), namísto (`INSTEAD OF`)
- **odložené** (*defered*)
  - mohou nastat na konci transakce (odstartované příkazem `COMMIT WORK`)
  - po uživatelském příkazu
  - následkem uživatelského příkazu (např. `PROCESS RULES`)
- **oddělené** (*detached*)
  - v kontextu samostatné transakce vypuštěné z počáteční poté, co nastala událost, která ho vyvolala
  - možná příčinná závislost počáteční a oddělené transakce

Vykonání akce

- **okamžité** (*immediate*)
  - následuje ihned po vyhodnocení podmínky (nejčastěji používané)
- **odložené** (*deferred*)
  - akce je odsunuta na konec transakce
  - akce je vyvolána uživatelským příkazem
- **oddělené** (*detached*)
  - probíhá v kontextu samostatné transakce vypuštěné z počáteční transakce ihned po vyhodnocení podmínky
    - možné příčinné závislosti počáteční a oddělené transakce

Úroveň sledování změn

- **úroveň instancí** (*instance level*)
  - změna řádku tabulky nebo objektu v případě objektivě orientovaných databází
  - přechodové hodnoty jsou uchovávany v poměnných **`OLD`** a **`NEW`**
- **úroveň příkazů** (*statement level*)
  - událostí je příkaz manipulující s daty
  - přechodové hodnoty jsou společné změny uchovávané v tabulkách:
    - **`INSERTED`** a **`DELETED`**
    - **`OLD-UPDATE`** a **`NEW-UPDATED`**

Více pravidel ve stejnou dobu

Konfliktní množina (*conflict set*)

- aktivní pravidla, která mají být aktivována současně
- je nutné určit pořadí, v jakém budou pravidla vykonávána

Způsob výběru dalšího pravidla

- výběr dalšího pravidla ke spuštění nastává po vyhodnocení každé podmínky a případném vykonání nějakého pravidla
- alternativně se seznam všech aktivovaných pravidel provádí jedno po druhém, dokud seznam není prázdný

Výběr pravidla z konfliktní množiny

Výběr je ovlivněn **prioritami**:

- **úplné uspořádání** je svázáno s číselnou prioritou
- **částečné uspořádání** je určeno číselnou nebo relativní prioritou
  - soulad mezi úplným systémovým uspořádáním a uživatelsky definovaným částečným uspořádáním je uržován systémem nebo nedeterministickým výběrem
- bez explicitně vyjádřené priority
  - systém urdžuje vlastní úplné uspořádání
  - výběr všech pravidel je nederministický

Opakovatelnost

- mějme dvě stejné transakce na stejné databázi se shodnou množinou aktivních pravidel
- pokud jsou výsledky obou transakcí stejné říkáme, že vykonání je ***opakovatelné***

Práce s pravidly

aktivace a **deaktivace** pravidel

- deaktivace aktivních pravidel může být extrémně nebezpečná, protože ty jsou většinou navrhovány pro zachování referenční integrity
- je součástí autorizačních mechanismů
  - jedná se také o práva k vytvoření, editaci a smazání aktivních pravidel
  - tyto akce může provádět pouze *administrátor* nebo uživatel s explicitně přidanými právy (**`GRANT PRIVILEGE`**)
- možnost sdružovat pravidla do skupin pro jednodušší práci

Využití aktivních databází

Interní a externí využití

Interní

- správa odvozených dat, kontrola integrity, replikace
  - obvykle
- správa verzí, zabezpečení přístupu, logování

Externí

- *business rules*
  - upozornění bez změny obsahu databáze
- sdílení zásad všemi aplikacemi

Udržování integrity

- **integritní omezení** (*constraints*) jsou zadána predikáty označované jako **integritní pravidla**
- hlídají platnost podmínky vyjádřené predikátem
- **statické** omezení je predikát nad stavem databáze
- **dynamické** omezení je predikát nad přechodem porovnávající stavy způsobené transakcí
- **vestavěné** omezení je speciální konstrukce jazyka
  - např. v SQL92 klíče, unikátní atributy, **`NOT NULL`**, referenční integrita
- **obecné** omezení je specifikováno libovolným predikátem/dotazem
  - není podporováno ve všech DB

Generování pravidel

- pravidla mohou být generována poloautomaticky
- deklarativní část pravidla, tj. *podmínka*
- akce, které mohou tuto podmínku porušit, tj. *událost* pravidla, se dá syntakticky určit z podmínky
- nápravná akce, aby podmínka opět začala platit, tj. *akce* pravidla, může být dvojího druhu
  - v rušících pravidlech (*abort rules*) se vyvolá rollback
  - pro opravná pravidla (*repair rules*) je nutné vymyslet, jak znovu nastolit konzistentní stav

Opravné akce (SQL-92)

- opravné akce při porušení referenční integrity
  - **`CASCADE`**
  - **`RESTRICT`**
  - **`SET NULL`**
  - **`SET DEFAULT`**

Příklad - referenční integrita

Tabulky:

- Zaměstnanci
  - každý zaměstnanec patří do nějakého oddělení
  - atribut `oddeleni` odkazuje na dané oddělení
- Oddělení
  - je identifikované číslem oddělení `oddeleni_id`

Akce:

1. vlož zaměstnance
2. smaž oddělení
3. aktualizuj číslo oddělení (`oddeleni_id`)
4. převel zaměstnance na jiné oddělení

Zamestnanci:

```
EXISTS ( SELECT * FROM Oddeleni
         WHERE oddeleni_id=Zamestnanci.oddeleni)
```

Příklad - referenční integrita

```
CREATE RULE OddZam1 ON Zamestnanci
WHEN INSERTED, UPDATED (oddeleni)
IF EXISTS (
     SELECT * FROM Zamestnanci
     WHERE NOT EXISTS (
          SELECT * FROM Oddeleni
          WHERE oddeleni_id=Zamestnanci.oddeleni
     )
)THEN ROLLBACK
```

Příklad - referenční integrita

```
CREATE RULE OddZam2 ON Oddeleni
WHEN DELETED, UPDATED (oddeleni_id)
IF EXISTS (
     SELECT * FROM Zamestnanci
     WHERE NOT EXISTS (
          SELECT * FROM Oddeleni
          WHERE oddeleni_id=Zamestnanci.oddeleni
     )
)THEN ROLLBACK
```

Opravovací pravidlo

```
CREATE RULE OpravOddZam1 ON Zamestnanci
WHEN INSERTED IF EXISTS (
     SELECT * FROM INSERTED
     WHERE NOT EXISTS (
          SELECT * FROM Oddeleni
          WHERE oddeleni_id=Zamestnanci.oddeleni
     )
)THEN UPDATE Zamestnanci
     SET oddeleni=NULL
     WHERE oddeleni IN (
          SELECT oddeleni FROM INSERTED
     ) AND NOT EXISTS (
          SELECT * FROM oddeleni
          WHERE oddeleni_id=Zamestnanci.oddeleni)
```

Opravovací pravidlo 2

```
CREATE RULE OpravOddZam2 ON Zamestnanci
WHEN UPDATED(oddeleni) IF EXISTS (
     SELECT * FROM NEW-UPDATED
     WHERE NOT EXISTS (
          SELECT * FROM Oddeleni
          WHERE oddeleni_id=Zamestnanci.oddeleni
     )
)THEN UPDATE Zamestnanci
     SET oddeleni=99
     WHERE oddeleni IN (
          SELECT oddeleni FROM NEW-UPDATED
     ) AND NOT EXISTS (
          SELECT * FROM oddeleni
          WHERE oddeleni_id=Zamestnanci.oddeleni)
```

- 99 je defaultní hodnota oddelení

Opravovací pravidla

- kaskádové mazání a update ...

Odvozování dat

- *pohled* je tabulka nebo třída, jejichž obsah je odvozen od obsahu databáze jako výsledek dotazu
- *odvozený atribut* vychází z nějaké formule a lze jej transparentně využívat v aplikaci

Odvozená data

- *virtuální* : jejich obsah je spočítán, kdykoliv se přistupuje k pohledu nebo odvozenému atributu
- *materializovaná* : jsou perzistentně uložena v databázi jako každá jiná data, musejí být tedy přepočítána kdykoliv se změní data, na kterých závisejí

Materializace odvozených dat

- úplná obnova (*refresh*)
  - data se po každé změně zdrojových dat přepočítají
  - snadná automatická údržba pravidel
- inkrementální obnova (*incremental refresh*)
  - ze změny zdrojových dat se odvodí změna odvozených dat
  - obvykle na úrovni n-tic, které mají být přidány/smazány

Příklad - pohled

Zobrazte oddělení, kde pracuje alespoň jeden "bohatý" zaměstnanec.

```
DEFINE VIEW BohataOddeleni AS (
    SELECT DISTINCT Oddeleni.jmeno
    FROM Oddeleni, Zamestnanci
    WHERE
    Oddeleni.oddeleni_id=Zamestanci.oddeleni
    Zamestnanci.mzda > 50000
)
```

Příklad - odvozování dat

Které změny v tabulkách mohou vynutit přepočítání pohledu?

1. vložení zaměstnance
2. vložení oddělení
3. smazání zaměstnance
4. smazání oddělení
5. aktualizace čísla oddělení
6. aktualizace platu
7. aktualizace umístění zaměstnance

Příklad - obnovovací pravidlo 1

```
CREATE RULE ObnovBohataOddeleni1 ON Zamestnanci
WHEN INSERTED, DELETED,
      UPDATED(oddeleni), UPDATED(mzda)
THEN
     DELETE * FROM BohataOddeleni;
     INSERT INTO BohataOddeleni:
     ( SELECT DISTINCT Oddeleni.jmeno
          FROM Oddeleni, Zamestnanci
          WHERE
          Oddeleni.oddeleni_id=Zamestanci.
oddeleni                  AND
          Zamestnanci.mzda > 50000
     )
```

Příklad - obnovovací pravidlo 2

```
CREATE RULE ObnovBohataOddeleni2 ON Oddeleni
WHEN INSERTED, DELETED,
      UPDATED(oddeleni_id)
THEN
     DELETE * FROM BohataOddeleni;
     INSERT INTO BohataOddeleni:
     ( SELECT DISTINCT Oddeleni.jmeno
          FROM Oddeleni, Zamestnanci
          WHERE
          Oddeleni.oddeleni_id=Zamestanci.
oddeleni                  AND
          Zamestnanci.mzda > 50000
     )
```

Příklad - inkrementální obnovovací pravidlo

```
CREATE RULE InkrementujBohataOddeleni ON Oddeleni
WHEN INSERTED
THEN
     INSERT INTO BohataOddeleni:
     ( SELECT DISTINCT INSERTED.jmeno
          FROM INSERTED, Zamestnanci
          WHERE
          INSERTED.oddeleni_id=Zamestanci.
oddeleni                  AND
          Zamestnanci.mzda > 50000
     )
```

Replikace

Replikace případ odvozování dat, kde se udržuje několik kopií stejných dat.

- distribuované systémy
  - více serverů
- Primární a sekundární kopie
  - změny prováděny pouze na primární kopii a jsou **asynchronně** propagovány na sekundární kopie (ty jsou *read-only*)
  - zachytávací modul (*capture module*)
- Synchronní řešení
  - střídavě primární a sekundární role
  - využívá distribuované transakce
  - není vždy vyžadováno aplikací
  - dosti náročné

Replikace - zachytávací pravidla

Aktivní pravidla poskytují mechanismus pro implementaci zachytávacího modulu.

- změny jsou udržovány ve změnových tabulkách

```
CREATE RULE Capture1 ON Primary
WHEN INSERTED
THEN INSERT INTO PosDelta
     (SELECT * FROM INSERTED)
CREATE RULE Capture2 ON Primary
WHEN DELETED
THEN INSERT INTO NegDelta
     (SELECT * FROM DELETED)
```

Replikace - zachytávací pravidla

```
CREATE RULE Capture3 ON Primary
WHEN UPDATED
THEN
    INSERT INTO PosDelta
        (SELECT * FROM NEW-UPDATED)
    INSERT INTO NegDelta
        (SELECT * FROM OLD-UPDATED)
```

Změny uložené v tabulkách `PosDelta` a `NegDelta` jsou posléze aplikovány na sekundární kopie.

Dotazy ?

Pokud nejsou, děkujeme za pozornost.