

VNOŘENÉ SQL

(EMBEDDED SQL)

PL/SQL

DB2 – 1. přednáška

Literatura - zdroje

- Pokorný, Jaroslav: Databázové systémy, ISBN 978-80-01-05212-9
- Chapter 6: Embedded SQL, Pro*C/C++ Precompiler Programmer's guide, Release 9.2, Oracle Corporation
 - http://download.oracle.com/docs/cd/B10501_01/appdev.92.0/a97269/pc_06sql.htm
- Embedded SQL : Introduction to Pro*C, Ankur Jain and Jeff Ullman
 - <http://infolab.stanford.edu/~ullman/fcdb/oracle/or-proc.html>

Obsah přednášky

- Vnořené (Embedded) SQL
- Hostitelské a indikátorové proměnné
- Příkazy INSERT, UPDATE, DELETE, SELECT
- Kurzory
- Ošetření chyb

Účel použití vnořeného SQL

- Použití SQL v programovacích jazycích
- SQL vepsané ve zdrojovém kódu jiného programovacího jazyka
- Všechny příkazy interaktivního SQL lze použít ve vnořeném SQL
- Databázový systém musí mít podporu vnořeného SQL ve použitém programovacím jazyce

Příklady DBMS

IBM DB2

Oracle

Microsoft SQL Server

MySQL

Příklady programovacích jazyků

IBM DB2

C/C++, COBOL, FORTRAN, REXX

Oracle

Microsoft SQL Server

MySQL

Příklady programovacích jazyků

IBM DB2

C/C++, COBOL, FORTRAN, REXX

Oracle

Ada, C/C++, COBOL, Fortran, Pascal, PL/1

Microsoft SQL Server

MySQL

Příklady programovacích jazyků

IBM DB2

C/C++, COBOL, FORTRAN, REXX

Oracle

Ada, C/C++, COBOL, Fortran, Pascal, PL/1

Microsoft SQL Server

Od verze 2008 ukončena oficiální podpora

MySQL

Příklady programovacích jazyků

IBM DB2

C/C++, COBOL, FORTRAN, REXX

Oracle

Ada, C/C++, COBOL, Fortran, Pascal, PL/1

Microsoft SQL Server

Od verze 2008 ukončena oficiální podpora

MySQL

Nepodporováno

Příklad vnořeného SQL (Oracle a C):

```
#include <stdio.h>
#include <sqlca.h>
int main(void)
{
    EXEC SQL INSERT INTO osoby (prijmeni) VALUES
        ('Sequens');
    return 0;
}
• Vložíme hlavičkový soubor struktury SQL
Communication Area (SQLCA)
• Příkazy vnořeného SQL začínáme direktivou EXEC SQL
a ukončujeme středníkem
```

Oracle - C/C++

- Oracle prostředí pro SQL vnořené v C/C++ se nazývá Pro*C/C++
- Zdrojové soubory s příponou *.pc
- Prekompilátor Pro*C/C++ přeloží *.pc na čistý C/C++ kód
 - Nahrazení konstrukcí vnořeného SQL voláním standardní run-time knihovny
- C/C++ kód je standardně zkompilován

Hostitelské proměnné

- Předávání dat mezi SQL a C/C++
- Ve vnořeném SQL označujeme dvojtečkou
- Vstupní
 - EXEC SQL INSERT INTO osoby (prijmeni)
VALUES (:prijmeni_osoby);
- Výstupní
 - EXEC SQL SELECT prijmeni INTO
:prijmeni_osoby FROM osoby;

Indikátorové proměnné

- Motivace:** Jaký bude obsah proměnné, když SELECT vrátí NULL?
- Řešení:** Použijeme druhou, „spřátelenou“ proměnnou, která bude indikovat, jak a zda je původní proměnná naplněna
- Indikátorová proměnná se zapisuje bezprostředně za původní proměnnou, oddělená dvojtečkou

Indikátorové proměnné u výstupních proměnných

| Indikátorová proměnná | Hostitelská proměnná |
|-----------------------|--|
| -2 | Oříznutá hodnota z databáze, hodnota se do hostitelské proměnné nevejde a její velikost nemůže být určena. |
| -1 | Nedefinovaná hodnota, v databázi byla NULL. |
| 0 | Hodnota z databáze, není to NULL. |
| >0 | Oříznutá hodnota z databáze, hodnota se do hostitelské proměnné nevejde. Indikátorová proměnná obsahuje velikost hodnoty v databázi. |

Indikátorové proměnné u výstupních proměnných - příklad

```
EXEC SQL SELECT pocet_deti INTO
:pocet:ind_pocet FROM osoby WHERE prijmeni =
'Sequens';
if (ind_pocet == -1) /* NULL v databázi */
pocet = 0;
```

Indikátorové proměnné u vstupních proměnných

- Ovlivní zápis do databáze v příkazech INSERT nebo UPDATE

| Indikátorová proměnná | Zápis do databáze |
|-----------------------|--|
| -1 | Zapíše se NULL (hodnota hostitelské proměnné se bude ignorovat). |
| >= 0 | Zapíše se hodnota hostitelské proměnné. |

Příkaz INSERT

- Shodné použití jako při interaktivním SQL
- Zápis proveden až po zapsání změn příkazem COMMIT

```
EXEC SQL INSERT INTO osoby (jmeno, prijmeni)
VALUES (:jmeno_osoby, :prijmeni_osoby);
```

Příkaz UPDATE

- Shodné použití jako při interaktivním SQL
- Zápis proveden až po zapsání změn příkazem COMMIT

```
EXEC SQL UPDATE osoby
SET jmeno = :jmeno_osoby
WHERE prijmeni = :prijmeni_osoby;
```

Příkaz DELETE

- Shodné použití jako při interaktivním SQL
- Smazání provedeno až po zapsání změn příkazem COMMIT

```
EXEC SQL DELETE FROM osoby
WHERE prijmeni = :prijmeni_osoby;
```

Příkaz SELECT

- V zásadě shodné použití jako u interaktivního SQL
- Nutno vyřešit, kam a jak uložíme to, co SELECT vrátí
- Vrátí-li SELECT nejvýše jeden řádek, je situace jednoduchá

```
EXEC SQL SELECT jmeno, prijmeni
INTO :jmeno_osoby, :prijmeni_osoby
FROM osoby;
```

Příkaz SELECT

- Vrací-li SELECT více než jeden řádek, je třeba použít ke zpracování výsledku tzv. **kurzor**

Jazyk PL/SQL - procedurální jazyk

Obsah

- Jazyk PL/SQL - procedurální jazyk:
 - Základy jazyka
 - Datové typy
 - Řízení toku programu
 - Kurzory
 - Ošetření chyb – výjimky
 - Procedury a funkce

Deklarativní vs. procedurální programování

- Při deklarativním programování v podstatě říkáme, co se má udělat, ale už nespecifikujeme jak se to má udělat.
 - Plus: Jeden příkaz.
 - Minus: Žádná kontrola nad procesem získávání dat.
- Při procedurálním (imperativním) programování říkáme, jakým způsobem data získat sekvenci příkazů.
 - Plus: Kontrola nad prováděním úlohy.
 - Minus: Více příkazů.

Přehled SQL

- Hlavním omezení jazyka SQL - jedná o neprocedurální jazyk
 - SQL- ideální příklad deklarativního jazyka.
 - Vychází to z potřeb, pro které byly vytvořeny.
 - Zajišťuje bezproblémovou manipulaci s daty.
- Příkazy jazyka SQL se provádějí sekvenčně, bez možnosti klasických programátorských konstrukcí (cykly, podmínky, procedury, funkce, případně objektové programování).
 - Jazyk nebyl navržen pro návrh aplikační logiky.
- Proto většina databázových platform nabízí rozšíření umožňující naprogramovat složitější algoritmy pro práci s daty.
- **Řešení:** PL/SQL

Motivační příklad

- Business pravidlo, které požaduje, aby aplikace umožňovala tvorbu nových uživatelů. Typicky by mělo být vyřešeno:
 - Má uživatel, který uživatele přidává vůbec právo přidávat uživatele?
 - Ověřit, zda zadávaný uživatel již v DB neexistuje.
 - Ověřit, zda heslo i opakované heslo souhlasí a zda heslo vyhovuje stanoveným pravidlům kvality hesel.

Motivační příklad

- Nebude-li některý z kroků splněn -> nutno ukončit zadávání uživatele a o porušení pravidel informovat uživatele.
 - Alternativně přidat záznam do systémového logu.
- Je-li vše v pořádku, vložit nového uživatele do DB a informovat vkládajícího uživatele o úspěšném přidání.
 - Dále mohou proběhnout dodatečné operace jako přiřazení výchozích práv, odeslání mailu novému uživateli apod.

PL/SQL

- **PL/SQL** - Procedural Language/**SQL**
- **PL/SQL** rozšiřuje **SQL** o konstrukce s procedurálního programování.
- Doplňuje SQL, nikoliv nahrazuje.
 - PL/SQL kombinuje možnosti deklarativního a imperativního programování.

Jazyk PL/SQL - možnosti

- Umožňuje deklarovat konstanty, proměnné, kurzory.
- Vyhodnocení IF/THEN rozhodovacích struktur.
- Nabízí podporu dynamických deklarácí.
- Podpora transakčního zpracování.
- Chybové stavy procesu je možné ošetřit pomocí výjimek.
- Podpora modularity (vkládání modulů i do sebe).
- Podporuje dědičnost.
- Jazyk PL/SQL je case insenzitivní (nerozlišuje velikost písmen u klíčových slov a názvů proměnných).
 - Pozor na řetězce, ty jsou stále case-sensitivní.

Vlastnosti jazyka PL/SQL

- Úzká integrace s SQL
 - V PL/SQL lze využívat SQL DML příkazy, transakční příkazy, SQL funkce, operátory.
 - Plná podpora SQL datových typů.
- Vysoký výkon
 - Do databáze může být odeslán jeden PL/SQL blok obsahující více jednotlivých SQL příkazů.
 - Předkompilování uložených bloků PL/SQL kódu -> efektivnější volání příkazů.
- Bezpečnost
 - Přesun kódu z aplikace do DB. Možnost skrýt interní detaily.

Možnosti použití PL/SQL

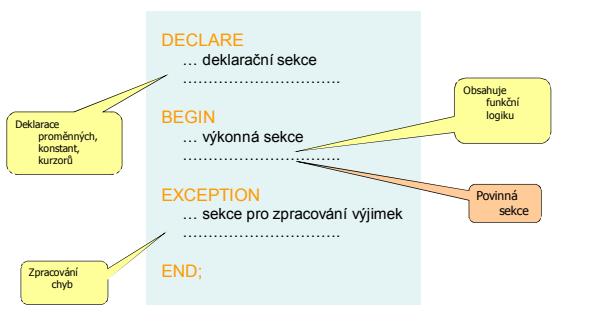
- Možnost tvorby vlastních funkcí, procedur, triggerů, datových typů ...
- Přesun aplikáční logiky ze strany klienta na stranu serveru.
 - Zapouzdření bussines pravidel a jiné komplikované logiky do DB.
 - Modularita a abstrakce.
- Zajištění komplexních integritních omezení.
- Transakce.
- Podpora výjimek.
 - Předdefinované i vlastní výjimky.

Možnosti použití PL/SQL

- Zajištění zabezpečení dat.
 - Audit
 - Uživatelská práva
- Kontrola syntaxe a platnosti objektů.
- Platformě nezávislý.
 - Oracle je implementován v rámci mnoha hardwarových platform - PL/SQL je ale stejný na všech těchto platformách.

Struktura jazyka PL/SQL

- Základní jednotkou PL/SQL programu je blok, spojující související deklarace a výrazy



Výpis textu na konsoli v PL/SQL

1. Nutno aktivovat příkazem SET SERVEROUT ON.
 - V SQL Developeru: View -> DBMS Output -> Enable DBMS output...
 2. Použít balíku DBMS_OUTPUT, procedury PUT_LINE

Příklad

```
BEGIN  
    DBMS_OUTPUT.PUT_LINE('Hello World!');  
END;  
/
```

Struktura jazyka PL/SQL

• Identifikátory

- V PL/SQL slouží k pojmenování konstant, proměnných, výjimek, kurzorů ...
 - Minimální délka je jeden znak; maximální délka je 30 znaků.
 - První znak identifikátoru musí být písmeno. Následovat může opět písmeno, číslice a znaky \$, #

• Literály

- Pro oddělení desetinné části se používá tečka: **2.6, .5 (0.5)**
 - Pro zápis velmi velkých i velmi malých čísel lze využít vědeckou notaci: **2E14, 1.0E-14**

Proměnné v PL/SQL

1. Je nutné před prvním použitím vždy deklarovat.
 2. PL/SQL je typový jazyk.
 3. Během deklarace je možné proměnnou inicializovat, případně omezit, že nesmí nabývat hodnoty NULL
 4. Lokálně definované proměnné mají rozsah platnosti jen v rámci bloku kde jsou definovaná (a v blocích uvnitř definovaných). Mimo blok neexistují.

```
DECLARE
    v_promenna1 NUMBER(3);
    v_promenna2 NUMBER NOT NULL DEFAULT 88;
    v_promenna3 NUMBER := 77;
    v_konstantaPI CONSTANT NUMBER := 3.14; -- Definice konstanty

BEGIN
    v_promenna1 := 33;
    DBMS_OUTPUT.PUT_LINE(v_promenna1);
    DBMS_OUTPUT.PUT_LINE('Hodnota proměnné 2 je' || v_promenna2);

END;
```

Datové typy - přehled



Datové typy

• Zvláštnosti:

- Maximální velikost VARCHAR2 je v PL/SQL rovna 32 767 bajtům, přičemž při použití v tabulce je to jen 4 000 bajtů.
- PL/SQL definuje datový typ BOOLEAN, přičemž ale tento typ nelze využít při definici v tabulce.

Příkaz přiřazení :=

- *název_proměnné* := *hodnota*,
- Pozor na záměnu s =
– To se používá k porovnání dvou hodnot, typicky v IF-ELSE.

Použití SQL dotazů v PL/SQL

- Standardně lze využívat jen DML operace.
 - DDL lze využít jen ve formě tzv. dynamických dotazů.
- Obvykle v těchto dotazech využíváme poměrných definovaných v PL/SQL bloku.
 - Jenak je můžeme použít proměnné v místech, kde se může objevit výraz.
 - Jenak lze proměnné využít pro získání konkrétních hodnot z provedeného dotazu.

Práce s daty v tabulkách v PL/SQL

Dotaz pro získávání dat

Počet specifikovaných proměnných se musí rovnat počtu sloupců. Nebo se musí jednat o záznam s počtem políím

```
SELECT [* | seznam_položek]
INTO [seznam_položek nebo proměnná typu záznam]
FROM název_tabulky
WHERE podmínky_výběru

DECLARE
    i_pocet NUMBER;
BEGIN
    SELECT COUNT(*) INTO i_pocet FROM A_HR.zamestnanci;
    DBMS_OUTPUT.PUT_LINE('Počet zam.: ' || i_pocet);
END;
```

Podmínkou úspěšnosti takto zadанého dotazu je, aby byl vrácen vždy jen jeden řádek. Jinak musí být použit cursor.

Pozor na datové typy.

Práce s daty v tabulkách v PL/SQL

- Definice datového typu proměnné pomocí atributu **%TYPE**.
 - %TYPE umožňuje odvodit datový typ proměnné z datového typu sloupce tabulky či datového typu jiné proměnné.
 - V případě databázového sloupce se odvozuje pouze datový typ, nikoliv omezení (NOT NULL, CHECK). V případě odvození s proměnné se odvozují i proměnné.

```
DECLARE
    v_jmeno uctitel.jmeno%TYPE;
    v_Id uctitel.Id%TYPE;
    v_id2 v_Id%TYPE
BEGIN
    SELECT jmeno, Id, TRUNC(Id,1) INTO v_jmeno, v_Id, v_id2
    FROM uctitel WHERE Id=2;
    -- výpis hodnot proměnných
    DBMS_OUTPUT.PUT_LINE('Jméno ' || v_jmeno);
    DBMS_OUTPUT.PUT_LINE(Id' || v_Id);
END;
```

Samozřejmě možno s daty i manipulovat příkazy **INSERT, UPDATE, DELETE**

%TYPE a výchozí hodnota

```

DECLARE
    credit  PLS_INTEGER RANGE 1000..25000;
    debit   credit%TYPE; -- inherits data type

    name      VARCHAR2(20) := 'John Smith';
    upper_name name%TYPE; -- inherits data type and default value
    lower_name name%TYPE; -- inherits data type and default value
    init_name name%TYPE; -- inherits data type and default value

BEGIN
    DBMS_OUTPUT.PUT_LINE ('name: ' || name);
    DBMS_OUTPUT.PUT_LINE ('upper_name: ' || UPPER(name));
    DBMS_OUTPUT.PUT_LINE ('lower_name: ' || LOWER(name));
    DBMS_OUTPUT.PUT_LINE ('init_name: ' || INITCAP(name));
END;
/

```

name: John Smith
 upper_name: JOHN SMITH
 lower_name: john smith
 init_name: John Smith

- Výchozí hodnota je „zděděna“ jen v případě, že odkazovaný objekt není sloupcem tabulky a nemá nastaveno omezení NOT NULL.

%TYPE a NOT NULL

- V případě, že odkazovaný objekt má nastaveno omezení NOT NULL, musí proměnná, která se na něj odkazuje definovat novou hodnotu.

```

DECLARE
    name      VARCHAR2(20) NOT NULL := 'John Smith';
    same_name name%TYPE;

BEGIN
    NULL;
END;
/

```

same_name name%TYPE;

*

ERROR at line 3:
 ORA-06550: Line 3, column 15:
 PL/SQL: variable declared NOT NULL must have an initialization assignment

Práce s daty v tabulkách v PL/SQL

- **Pozor** na to, aby názvy proměnných v bloku nekolidovaly s názvy sloupců v tabulkách, k nimž blok přistupuje.
 - Při vyhodnocování v PL/SQL mají názvy sloupců tabulek přednost před názvy proměnných. To může mít nepříjemné konsekvence.

```

DECLARE
    zamstnanec_id number(3) := 101;
BEGIN
    DELETE FROM zamstnanci WHERE zamstnanec_id = zamstnanec_id;
    DBMS_OUTPUT.PUT_LINE('Smazáno ' || SQL%ROWCOUNT || ' záznamů.');
END;

```

Smazáno 107 záznamů.

Konverze datových typů

- Explicitní konverze**

- Využívá vestavěných funkcí databázového systému: **TO_DATE**, **TO_CHAR** a **TO_NUMBER**.

- Implicitní konverze**

- PL/SQL může některé datové typy konvertovat mezi sebou automaticky.
- Implicitní konverze je možná mezi tzv. **kompatibilními typy**.
 - Tabulka kompatibilních typů je na následujícím snímku.

Konverze datových typů

- Možné implicitní konverze:

| From: | To: | BLOB | CHAR | CLOB | DATE | LONG | NUMBER | PLS_INTEGER | RAW | UROWID | VARCHAR2 |
|--------------------|-----|------|------|------|------|------|--------|-------------|-----|--------|----------|
| BLOB | | | | | | | | | Yes | | |
| CHAR | | | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | |
| CLOB | | | | | Yes | | | | | | Yes |
| DATE | | Yes | | | Yes | | | | | | Yes |
| LONG | | Yes | | | | Yes | | | | | Yes |
| NUMBER | | Yes | | | | Yes | | | | | Yes |
| PLS_INTEGER | | Yes | | | | Yes | | | | | Yes |
| RAW | | Yes | Yes | | Yes | | | | | | Yes |
| UROWID | | Yes | | | | | | | | | Yes |
| VARCHAR2 | | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | |

Vnořené bloky PL/SQL



Komentáře v PL/SQL

Příklad

```
DECLARE
    /* jednořádkový komentář
    v_promenna1 NUMBER(3);
    v_promenna2 NUMBER NOT NULL DEFAULT 88;
    v_promenna3 NUMBER := 77;

BEGIN
    /* víceřádkový
    komentář
    */
    v_promenna1 := 33;
    DBMS_OUTPUT.PUT_LINE(v_promenna1);
    DBMS_OUTPUT.PUT_LINE('Hodnota proměnné 2 je ' || v_promenna2);

END;
```

Příklad ošetření výjimek v PL/SQL

Příklad

```

DECLARE
    v_jmeno ucitel.jmeno%TYPE;
    v_Id ucitel.id%TYPE;

BEGIN
    SELECT jmeno, Id INTO v_jmeno, v_Id
    FROM ucitel WHERE Id=2;

    DBMS_OUTPUT.PUT_LINE('Jméno'||v_jmeno);
    DBMS_OUTPUT.PUT_LINE('Id'||v_Id);

EXCEPTION
    -- ošetření výjimky při nenašení dat
    WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('Data nenašena');

    -- ošetření výjimky při nalezení více řádků splňujících podmínku
    WHEN TOO_MANY_ROWS THEN DBMS_OUTPUT.PUT_LINE('Mnoho řádků');
END;

```

Řízení toku programu - podmínky

Zápis podmínky

```

IF podminka
THEN
    posloupnost_příkazů
END IF;

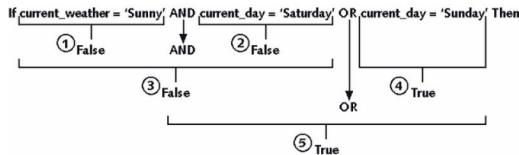
nebo

IF podminka
THEN
    posloupnost_příkazů1
ELSIF podminka2
THEN
    posloupnost_příkazů2
ELSE
    posloupnost_příkazů3
END IF;

```

Řízení toku - komplexní podmínky

- **AND** je vyhodnoceno před **OR**.
 - Dobrou praxí je používat závorky.



Řízení toku programu

Příkaz CASE pro vícenásobné větvení programu

```
CASE
    WHEN podminka1 THEN posloupnost_příkazů1;
    WHEN podminka2 THEN posloupnost_příkazů2;
    ..
    WHEN podminkaN THEN posloupnost_příkazůN;
    [ELSE posloupnost_příkazůN+1;]
END CASE;
```

Podmínka může být i například v_promenna BETWEEN 1 AND 5

CASE

```

DECLARE
    grade CHAR(1);
BEGIN
    grade := 'B';

CASE grade
    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
    WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
    WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
    WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor');
    ELSE DBMS_OUTPUT.PUT_LINE('No such grade');
END CASE;
END;
/

```

- V případě splnění podmínky příkazu CASE dojde k provedení dané větve až počemž následuje skok na další příkaz v bloku.
 - Další podmínky se již netestují.

Řízení toku programu - cykly

Jednoduchý cyklus LOOP

```
LOOP
    posloupnost_příkazů
    IF podmínka THEN
        .. ukončuje se příkazem EXIT
    END IF;
END LOOP;

nebo

LOOP
    posloupnost_příkazů
    EXIT WHEN podmínka;
END LOOP;
```

Příklad

```
DECLARE
    V_pocet NUMBER := 0;
BEGIN
    LOOP
        v_pocet:=v_pocet+1;
        IF v_pocet >=100 THEN
            EXIT;
        END IF;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('v_pocet' || v_pocet );
END;
```

Řízení toku programu - cykly

Cyklus FOR s čítačem

```
FOR počítadlo IN [REVERSE] Nejnižší_hodnota.. Nejvyšší_hodnota
LOOP
    posloupnost_příkazů
END LOOP;
```

Příklad

```
BEGIN
    FOR v_citac IN 1..3
    LOOP
        DBMS_OUTPUT.PUT_LINE('v_citac' || v_citac );
    END LOOP;
END;
```

Řízení toku programu - cykly

Cyklus WHILE s podmínkou na začátku

```
WHILE podmínka
LOOP
    posloupnost_příkazů
END LOOP;
```

Příklad

```
DECLARE
    V_pocet NUMBER := 0;
BEGIN
    WHILE v_pocet < 100
    LOOP
        v_pocet:=v_pocet+1;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('v_pocet' || v_pocet );
END;
```

Řízení toku programu - cykly

- Pro opuštění smyčky definuje PL/SQL následující příkazy:
 - **EXIT** a **EXIT WHEN**
- Pro přeskovení příslušné iterace cyklu:
 - **CONTINUE** a **CONTINUE WHEN**

```
DECLARE          DECLARE
    i number;      i number;
BEGIN            BEGIN
    FOR i IN 1..4   FOR i IN 1..4
    LOOP           LOOP
        DBMS_OUTPUT.PUT_LINE(i);
        CONTINUE WHEN i=2;
        EXIT WHEN i=2;
    END LOOP;
END;             END;
```

PL/SQL

Kurzory, procedury, funkce, trigger- úvod, příklad

DB2 – 2. přednáška

Kurzory

- Prvátní pracovní oblasti, které jsou databázovým serverem vytvořeny pro každý příkaz SQL.
 - Kurzory jsou v podstatě ukazatele do paměti, ne přímo do dat. Výsledné záznamy dotazu jsou do paměti vloženy ve chvíli otevření kurzoru.
 - V podstatě se jedná o mechanismus, kdy SELECT příkazu přiřadíme jméno, přes které pak můžeme manipulovat s daty získanými příkazem.
 - **Implicitní** kurzory jsou vytvářeny automaticky databázovým serverem, není nutné je otevřít, zavírat, deklarovat nebo z něj načítat data,
 - **Explicitní** – deklarování programátorem
- Základní kroky pro práci s explicitními kurzory
- Deklarace kurzoru
 - Otevření kurzoru
 - Výběr dat prostřednictvím kurzoru
 - Uzavření kurzoru

Explicitní kurzory - syntaxe

- Deklarace kurzoru
`CURSOR <název kurzoru> IS <příkaz SELECT>;`
- Otevření kurzoru
`OPEN <název kurzoru>;`
- Výběr dat prostřednictvím kurzoru (opakovat v cyklu)
`FETCH <název kurzoru> INTO <seznam proměnných>;`
- Uzavření kurzoru
`CLOSE <název kurzoru>;`

Explicitní kurzory – testování stavu

Pro testování stavu kurzu jsou k dispozici atributy

%ROWCOUNT
Zjistíte pořadového čísla aktuálního záznamu
(pokud nebyl vybrán žádný, je hodnota 0)

%FOUND
Pokud poslední příkaz FETCH načel nějaký záznam, má atribut hodnotu TRUE
Používá se pro zjištění konce cyklu

%NOTFOUND
Používá se pro zjištění konce cyklu

%ISOPEN
Pokud je cursor otevřen, má hodnotu TRUE

Použití: <název kurzu>%ROWCOUNT

Práce s explicitními kurzory

Příklad s využitím explicitního kurzu

```
DECLARE
    v_jmeno uctitel.jmeno%TYPE;
    v_Id uctitel.id%TYPE;

    CURSOR k1 IS
        SELECT jmeno, id FROM uctitel;

    BEGIN
        OPEN k1;
        LOOP
            FETCH k1 INTO v_jmeno, v_Id;
            EXIT WHEN k1%NOTFOUND;
            DBMS_OUTPUT.PUT_LINE(jméno || v_jmeno || ', ' || v_Id);
        END LOOP;
    CLOSE k1;
END;
```

Explicitní kurzory

DECLARE

-- Deklarujeme dva totičné kurzy:

CURSOR c1 IS SELECT * FROM A_HR.zamestnanci

WHERE zamestnanci.id > 150;

CURSOR c2 IS SELECT * FROM A_HR.zamestnanci

WHERE zamestnanci.id > 150;

BEGIN

OPEN c1;

DELETE FROM A_HR.zamestnanci WHERE id_zamestnanci > 150;

OPEN c2;

Zpracování obou kurzorů ve smyčce, zobrazení c1 a c2 na DBMS_OUTPUT

CLOSE c1;

CLOSE c2;

END;

Po provedení bloku kód zjistíme, že ažkoliv samotný výpis byl proveden až po příkazu DELETE, tak cursor c1 zaměstnance s id > 150 vypisuje. Důležitý je totiž okamžik otevření kurzu. Kurzor po otevření udržuje obraz dat takový, jaký byl, a nefunguje jako dynamicky ukazatel na živá data.

Snímek 6

k2 kleckova; 15.3.2017

Explicitní kurzory - CLOSE

- Pokud je cursor deklarován v balíčku (tj. ne ve funkci či proceduře uvnitř balíčku) a cursor je otevřen, zůstane otevřen doby, než bude **EXPLICITNĚ uzavřen**, nebo je ukončena session, která jej otevřela.
- Pokud je cursor deklarován v deklarační sekci (v bloku PL/SQL), databáze sama uzavře cursor ve chvíli, kdy je blok dokončen.
 - Nicméně se důrazně doporučuje cursor explicitně uzavřít!

Záznamy

Struktura typu záznamu zapouzdřuje více polížek i rozdílných datových typů.

Deklarace záznamu

```
DECLARE
  TYPE <název proměnné typu záznam> IS RECORD
    (
      <název atributu> <datový typ>
      [,<název atributu> <datový typ>...]
    );
  
```

Příklad

```
DECLARE
  TYPE rec_učitel IS RECORD
    (
      jméno       učitel.jméno%TYPE,
      id          učitel.id%TYPE
    );
  
```

Neboli po zadání:

```
DECLARE
  rec_učitel učitel%ROWTYPE;
```

Práce s kurzory a záznamy

```
DECLARE
  rec_trpaslik a_snehurstka.trpaslici%ROWTYPE;

  CURSOR k1 IS SELECT jméno, id FROM a_snehurstka.trpaslici;

  BEGIN
    OPEN k1;

    LOOP
      FETCH k1 INTO rec_trpaslik.jméno,rec_trpaslik.id;
      EXIT WHEN k1%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('jméno ' || rec_trpaslik.jméno || ' id ' || rec_trpaslik.id);
    END LOOP;

    CLOSE k1;
  
```

Práce s kurzory a záznamy

S využitím záznamů můžeme s kurzory pracovat mnohem efektivněji

Cyklus FOR s explicitním kurzorem
(kurzor v tomto případě nemusíme ani otevírat ani zavírat, dokonce ani cyklicky vybírat data pomocí příkazu `FETCH`, všechny tyto úkony za nás provede server standardně)

Příklad

```

DECLARE
    rec_uditel uditel%ROWTYPE;

CURSOR k1 IS
    SELECT jmeno, id FROM uditel;

BEGIN
    FOR rec_uditel IN k1
    LOOP
        DBMS_OUTPUT.PUT_LINE('jméno || '|| rec_uditel.jmeno || ', '|| rec_uditel.id);
    END LOOP;
END;

```

Práce s implicitními kurzory

- a) Příkaz **SELECT ... INTO ... FROM ...** musí vrátit alespoň jeden a nejvýše jeden řádek, počet sloupců musí odpovídat počtu proměnných uvedených za klauzulí **INTO** včetně použitelnosti datových typů.

b) Následující příklad ukazuje využití implicitního kurzoru pro sady výsledků s omezeným počtem řádků (řekněme méně než 100)

```
For x in (select ... from ... where ...)  
Loop  
    Process ...  
End loop;
```

```
BEGIN
    FOR x IN (SELECT jmeno,Id FROM trpaslici)
    loop
        DBMS_OUTPUT.PUT_LINE('Jméno ' || x.jmeno || ',Id ' || x.Id);
    END LOOP;
END;
```

Kurzory s parametry

Kurzor můžeme rozšířit o parametry, které budou dosazeny do dotazu až během otevření kurzu.

Deklarace explicitního kurzoru s parametry

CURSOR <název kurzoru> [(<název parametru> <datový typ>, ...)] IS <příkaz SELECT>;

Příklad

```

DECLARE
    rec_uictel      uictel%ROWTYPE;
    CURSOR k1 (jmeno VARCHAR2) IS
        SELECT jmeno, id FROM uictel WHERE jmeno LIKE (jmeno || '%');

BEGIN
    FOR rec_uictel IN k1 ('Zá')
    LOOP
        DBMS_OUTPUT.PUT_LINE('Jméno ' || rec_uictel.jmeno || ', id ' || rec_uictel.id);
        EXIT;
    END LOOP;

    FOR rec_uictel IN k1 ('Smr')
    LOOP
        DBMS_OUTPUT.PUT_LINE('Jméno ' || rec_uictel.jmeno || ', id ' || rec_uictel.id);
        EXIT;
    END LOOP;
END;

```

Explicitní vs. implicitní kurzory

- Implicitní kurzory jsou obecně výkonnější.
- Implicitní kurzory jsou z pohledu kódu úspornější a automatizují operace **OPEN**, **CLOSE**, **FETCH** a **EXIT WHEN .. %NOTFOUND** explicitního kurzu.
- Nehrozí opomnětí **EXIT WHEN**, což má za následek uváznutí v nekonečné smyčce.
- Explicitní kurzory lze předávat parametrem mezi jednotlivými procedurami/funkcemi.
 - Využívá se při tom datový typ **SYS_REFCURSOR**.
 - Větší flexibilita než implicitní kurzory.

Ošetření chyb

V zásadě se mohou v PL/SQL vyskytnout 2 druhy chyb:
Syntaktické – projeví se ještě v procesu komilace (upozorní nás na ně překladač)
Run-time – projeví se až za běhu programu

Nejčastěji se vyskytují následující výjimky:

| | |
|-------------------------|--|
| DUP_VAL_ON_INDEX | výskyt duplicitní hodnoty ve sloupci, který přípustí jen jedinečné hodnoty |
| INVALID_NUMBER | neplatné číslo nebo data nemohou být převedena na číslo |
| NO_DATA_FOUND | nebyly nalezeny žádné záznamy |
| TOO_MANY_ROWS | dotaz vrátil více než jeden záznam |
| VALUE_ERROR | problém s matematickou funkcí |
| ZERO_DIVIDE | dělení nulou |

Ošetření chyb

Všeobecná syntaxe pro zpracování výjimek:

```
EXCEPTION
  WHEN <název výjimky> THEN <příkazy>
    | WHEN <název výjimky> THEN <příkazy> ...
    WHEN OTHERS THEN <příkazy>;
  END;
```

Výjimku můžeme navodit nebo simuloval příkazem

```
RAISE <název výjimky>;
```

například

```
RAISE NO_DATA_FOUND;
```

Ošetření chyb

- Pro zachycení výjimky je nutné definovat její obsluhu (**exception handler**).
- Každý handler obsahuje **WHEN** klauzuli, ve které specifikuje typ výjimky, kterou „zachytává“. Po té následuje řada příkazů spojených s obsluhou dané výjimky.
- Speciální typem handleru je **WHEN OTHERS THEN**
 - Tímto handlerek jsou zachyceny všechny výjimky, které nebyly zachyceny handlery před WHEN OTHERS THEN.
 - Použití WHEN OTHERS THEN garantuje, že žádná z výjimek nezůstane neobslužena.
 - Ošetření OTHERS musí být uvedeno jako poslední ošetření výjimek v bloku.

Ošetření chyb

- Pozor na chyby v deklarační (DECLARE) části.
 - Chyby v deklarační části jsou ihned propagovány do nadřazených bloků.

```
DECLARE
  i number(1) := 20;
BEGIN
  NULL;
EXCEPTION
  WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Chyba');
  -- Tato obsluha nebude využita
END;
```

- Obdobně nelze zachytit výjimku vyvolanou v EXCEPTION bloku stejným EXCEPTION blokem.

Ošetření chyb

Aktuální kód chyby vráci systémová funkce **SQLCODE**
a její textový popis systémová funkce **SQLERRM**,
takže při zpracování výjimky máme k dispozici tyto údaje.

Příklad

```
DECLARE
  v_vysledek NUMBER(9,2);
BEGIN
  v_vysledek := 5/0;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(' Chyba ');
    DBMS_OUTPUT.PUT_LINE('Kód chyby: ' || SQLCODE);
    DBMS_OUTPUT.PUT_LINE('Popis chyby: ' || SQLERRM);
END;
```

Definování vlastních výjimek

Máme možnost definovat i vlastní výjimky.
Pro vlastní výjimky je SQLCODE rovno 1 a SQLERRM vraci Text User-Defined Exception

Syntaxe

```
DECLARE
    <název výjimky> EXCEPTION;

BEGIN
    <příkazy>;
    RAISE <název výjimky>;
EXCEPTION
    WHEN <název výjimky> THEN <příkazy>;
END;
```

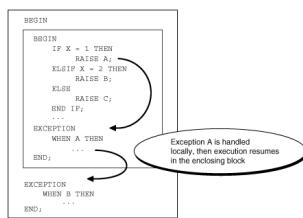
Definování vlastních výjimek

Příklad definice vlastní výjimky pro kontrolu počtu trpaslíků.

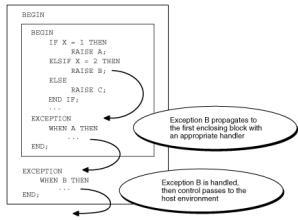
```
DECLARE
    PRILIS_MNOHO_TRPASLIKU EXCEPTION;
    v_pocet_trpasliku NUMBER;

BEGIN
    select count(*) INTO v_pocet_trpasliku FROM trpasliki;
    IF v_pocet_trpasliku > 7 THEN
        RAISE PRILIS_MNOHO_TRPASLIKU;
    END IF;
EXCEPTION
    WHEN PRILIS_MNOHO_TRPASLIKU
    THEN DBMS_OUTPUT.PUT_LINE('Trpaslíků může být maximálně sedm');
    END;
```

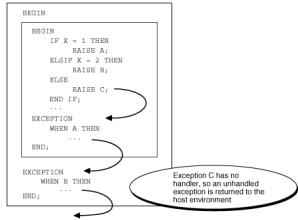
Propagace výjimek



Propagace výjimek



Propagace výjimek



Nezachycené výjimky

- Jsou „vypropagovány“ až do hostitelského prostředí databáze.
 - To pak rozhodne, jak naloží z výsledkem běhu PL/SQL bloku.
- V případě nezachycené vlastní výjimky skončí běh s chybovým hlášením:
 - ORA-06510: PL/SQL: unhandled user-defined exception.**
 - Při propagaci vlastní výjimky mimo blok, kde je definována, způsobí ztrátu informace o jméně vlastní výjimky.

Procedure a funkce

Bloky příkazů jazyka PL/SQL lze pojmenovat a uložit ve spustitelné formě do databáze. Tento blokům říkáme procedure, resp. funkce.

Vlastnosti procedur a funkcí:

- ▶ Jsou uloženy ve zkompilovaném tvaru v databázi.
 - ▶ Mohou volat další procedury či funkce, či samy sebe.
 - ▶ Lze je volat ze všech prostředí klienta.
- Funkce, na rozdíl od procedury, vrací jedinou hodnotu (procedura může vracet hodnot více, resp. žádnou).

Procedure

```
CREATE PROCEDURE jméno_procedure
[[formální_parametry]] AS
[[lokální deklarace]]
BEGIN
[[výkonné příkazy]]
[[EXCEPTION
    osetření nestandardních stavů]]
END;
```

Příklad

```
CREATE PROCEDURE pln_cislo AS
p_cislo number;          /* lokální deklarace bez kl. slova DECLARE */
BEGIN
FOR i IN 1..20 LOOP
INSERT INTO Nic(cislo) VALUES (i); /* tabulka Nic musí být
                                    předem vytvořena */
COMMIT;
END LOOP;
END;
```

Tato procedura zapíše do tabulky Nic čísla od 1 do 20.

Příklad

```

PROCEDURE prevod AS
p_jmenoVARCHAR2(15);
CURSOR k1 IS
SELECT upper(jmeno_p) FROM Pacient;
BEGIN
OPEN k1;
LOOP
FETCH k1 INTO p_jmeno;
dbms_output.put_line(p_jmeno);
EXIT WHEN k1%notfound;
END LOOP;
CLOSE k1;
END;

```

Poznámka:
Tato procedura vytiskne na konzolu jména všech pacientů z tabulky Pacient a převede všechna písmena na velká

Procedure

Formální parametry procedury
Jméno_parametru [IN OUT IN OUT] typ_parametru [:= hodnota]
Specifikace jednotlivých parametrů v seznamu jsou odděleny čárkou.
Parametry mohou být vstupní, výstupní a vstupně-výstupní.
Pouze vstupní parametry mohou být inicializovány. K inicializaci můžeme použít buď klauzuli DEFAULT nebo přiřadit hodnotu (:=)

Příklad s parametry

```

CREATE PROCEDURE deleni
(delenec IN number,delitele IN number) parametry
AS
BEGIN
dbms_output.put_line(delenec/delitele);
END;

```

Procedura vytiskne na konzolu podíl hodnot zadaných jako skutečné parametry.

Kompilace a spuštění procedury

1. Zápis ukončíme znakem . (tečka) na novém řádku
2. Příkazem RUN přeložíme proceduru
3. Příkazem EXECUTE[[seznam skutečných parametrů]] proceduru vykonáme

Poznámka:

Pro předchozí příklad nezapomeneme zadat příkaz SET SERVEROUT ON pro přesměrování výstupu na konzolu

Zápis, komplikace a spuštění v SQL*Plus

```
CREATE PROCEDURE deleni
(delenec IN number,delitec IN number) AS
BEGIN
dbms_output.put_line(delenec/delitec);
END;
.
RUN
SET SERVEROUT ON
EXECUTE deleni(10,2);
```

Ošetření chyby při dělení nulou

```
CREATE PROCEDURE deleni
(delenec IN number,delitec IN number) AS
BEGIN
dbms_output.put_line(delenec/delitec);
EXCEPTION
WHEN zero_divide THEN
dbms_output.put_line('Chyba při dělení nulou');
END;
```

Funkce

```
CREATE FUNCTION jméno_funkce [(formální_parametry)]
    RETURN typ_návratové_proměnné AS [lokální deklarace]
BEGIN
    [vykonné příkazy]
    [EXCEPTION
        ošetření nestandardních stavů]
END;
```

Příklad funkce

```
CREATE FUNCTION f_deleni
(delenec IN number, delitel IN number) RETURN number
AS
Vysledek number;
BEGIN
vysledek := delenec/delitel;
RETURN vysledek;
EXCEPTION
WHEN zero_divide THEN
dbms_output.put_line('Chyba při dělení nulou');
END;
```

Vyvolání funkce

```
SET SERVEROUT ON
begin
dbms_output.put_line (f_deleni(12,4));
end;
```

Zrušení procedury či funkce

DROP PROCEDURE jméno_procedury

DROP FUNCTION jméno_funkce

PL/SQL

Aktivní databáze - trigger

DB2 – 2. přednáška

Úvod

- databáze
 - obraz skutečnosti
 - reálná data a na ně kladené požadavky
- omezení dat
 - integritní omezení
 - jednoduché, rychlé
 - ne vždy postačuje

Aktivní pravidla

- ▶ *aktivní pravidla (active rules)*
 - pro vyhodnocení složitých podmínek kladených na data (tzv. business rules)
 - kontrola na databázové úrovni
 - usnadnění práce – auditovatelnost, bezpečnost
 - ▶ *triggery (triggers)*
 - v překladu „spoušt“, „kohoutek“
 - jiný název pro aktivní pravidla
 - v praxi je dávána přednost názvu trigger
- AKTIVNÍ PRAVIDLA = TRIGGERY

40

Úvod – historický vývoj

Jinak řečeno

- Trigger („spoušt“) je „procedura“, která se spustí při výskytu nějaké sledované události.
- V relačních databázích **trigger = aktivní pravidlo**
- Konec 80. let
 - první snahy o formální definici
- SQL92
 - triggers neobsahuje
 - nedostatky ve standardizačních dokumentech
- SQL1999
 - triggers již obsahuje

Starburst

- IBM, Almaden Research Center
- Starburst Active Rule System
- Získalo popularitu
 - Jednoduchá syntaxe a sémantika
 - Množinově orientovaná
- Pravidla založena na ECA-paradigmatu (*Event-Condition-Action*)

ECA-paradigma

- **Událost** (Event)
 - SQL-příkazy pro manipulaci s daty (*INSERT, DELETE, UPDATE*)
- **Podmínka** (Condition)
 - booleovský predikát nad stavem databáze, vyjádřen pomocí SQL
- **Akce** (Action)
 - provádí libovolné SQL dotazy (například *SELECT, INSERT, DELETE, UPDATE*)
 - navíc mohou obsahovat příkazy pro manipulaci s aktivními pravidly a transakční instrukci *ROLLBACK WORK*

Sémantika aktivních pravidel

- Jednoduchá a intuitivní
- Když nastane **Událost**, pokud je splněna **Podmínka**, proved' **Akci**.
- Říkáme, že pravidlo je:
 - **spuštěno** (triggered) – pokud nastane příslušná **Událost**
 - **vyhodnoceno** (considered) – po výhodnocení dané Podmínky
 - **vykonáno** (executed) – po provedení jeho **Akce**

Vlastnosti aktivních pravidel (triggerů)

- Jsou přidané do schématu databáze a jsou sdílené všemi aplikacemi.
- Mohou být dynamicky aktivovány a deaktivovány každou transakcí.
- Mohou tvořit skupiny.
- Každé pravidlo ve Starburstu má jedinečné jméno a je spojeno s jednou určitou tabulkou, zvanou *rule's target*.
- Každé aktivní pravidlo může sledovat více *Událostí*, tzv. *rule's triggering operations*.
- Jeden SQL příkaz může být sledován více pravidly.
 - Pořadí pravidel je určeno na základě jejich částečného uspořádání.

TRIGGER

Základní pojmy
Problémy
Použití

Základní pojmy

- ▶ syntax
 - zápis triggeru v daném DB systému
- ▶ sémantika
 - kdy se pustí
 - jak proběhne
 - jak se nazýjeme volají
 - nekonečné cykly
 - ...
- ▶ vybrané modely aktivních pravidel
 - historicky významné nebo prakticky používané
 - zajímavé implementované

Problémy s triggery

- standardizace
 - není
 - snaha by byla
 - od 80. let
 - v normě SQL-92 nejsou standardně uvedeny
- SQL 1999 – základní podmínky realizace, ale ...

49

Problémy s triggery

- proprietární řešení výrobců DB systémů
 - rozdíly v syntaxi i sémantice
 - vazba aplikace na konkrétního výrobce
- technické problémy
 - nekonečné vzájemné volání triggerů (retriggering)
 - několik možných řešení
 - používají se všechna

50

Spouště (triggery)

- Jedná se o PL/SQL objekty spouštěné vyvoláním příslušné události v DB
- Vyvolání může způsobit DML událost, DDL operace nebo speciální DB událost
- Vyvolat trigger můžeme před nebo po provedení operace
- Existuje také možnost vyvolání místo příslušné operace
- Je možné omezit vyvolání podmínkou

Existující typy DML triggerů

- DML aktivované triggery:
 - při rušení řádků (`DELETE`)
 - při vkládání řádků (`INSERT`)
 - při modifikaci určitých sloupců (`UPDATE OF`)
- Způsob vyvolání triggeru:
 - jednou při celé operaci
 - pro každý řádek (`FOR EACH ROW`)
vstupující do zpracování operace
- Možnost kombinací operací (slučování `OR`)

Zápis DML triggeru:

```
CREATE OR REPLACE TRIGGER jméno
BEFORE | AFTER | INSTEAD OF
DELETE | INSERT | UPDATE OF cols
ON tabulka
[ způsob odkazování ]
[ FOR EACH ROW ]
[ WHEN ( podmínka ) ]
AS pl/sql kód
```

Způsob odkazování

- Definuje, jak budou přístupné původní a nové záznamy (vstupující do DML operací)
- Implicitně :*new*, :*old*, :*parent*
- Existuje klauzule


```
REFERENCING
      [ OLD AS jméno ]
      [ NEW AS jméno ]
      [ PARENT AS jméno ]
```

Způsob vyvolávání triggerů

- Pro BEFORE a AFTER je trigger chápán jako tzv. statement trigger a vyvolán je pouze jedenkrát (není-li klauzulí FOR EACH ROW explicitně stanoveno jinak)
- V případě INSTEAD OF triggeru je trigger implicitně chápán jako rádkový trigger, protože zde statement trigger nemá prakticky žádný význam

DDL triggery

- Jsou vyvolány po provedení DDL příkazu
- Mohou být BEFORE, AFTER
- Mohou být omezeny podmínkou (WHEN)
- Definují se dvěma způsoby:
 - jméno události ON DATABASE
 - jméno události ON jméno schématu
- Existuje řada definovaných událostí, např. CREATE, ALTER, DROP, RENAME, GRANT, COMMENT, AUDIT, DDL

Triggery databázových událostí

- Pracují stejně jako DDL triggery, pouze je jiná množina povolených událostí
- Typicky se jedná o události zásadních událostí v celé databázové instanci, např. STARTUP, SHUTDOWN, LOGON, LOGOFF, SERVERERROR, SUSPEND apod.
- Uvnitř DDL a databázových triggerů nelze provádět jiné DDL operace
- Velmi specifické použití

Omezení triggerů

- BEFORE a AFTER triggery nelze specifikovat nad pohledy
- V BEFORE triggerech není možné zapisovat do :old záznamů
- V AFTER triggerech nelze zapisovat ani do :old, ani do :new záznamů
- INSTEAD OF triggery pracují jen s pohledy, mohou číst :old i :new, ale nemohou zapisovat ani do jednoho
- Nelze kombinovat INSTEAD OF a UPDATE
- Nelze definovat trigger nad LOB atributem

Dvě zásadní omezení

- Nelze použít transakce, pokud je zpracovávána jiná transakce (tedy prakticky nelze použít transakce vůbec)
- Není možné sledovat (ani modifikovat) data v tabulce, která způsobila vyvolání DML triggeru – toto omezení je často velice nepříjemné
- Jediné známé řešení: zrcadlení tabulek

Rozlišení operace v triggeru

- Trigger může být volán různými operacemi (např. INSERT OR DELETE)
- V průběhu triggeru je třeba rozlišit, která operace se provádí
- Existují logické proměnné INSERTING, DELETING a UPDATING použitelné v rozhodování
- Pozn. Je třeba užívat řízení výjimek, protože chyba v triggeru ukončí obvykle celou nadřazenou transakci

Emulace AUTOINCREMENT

- Některé SQL databázové systémy používají modifikátor typu AUTOINCREMENT pro definici číslování primárních klíčů
- Je možné toto chování emulovat umístěním BEFORE INSERT triggeru, který výčte novou hodnotu ze sekvence a modifikuje :new.id na tuto hodnotu

Starburst

- Starburst Active Rule System
 - DDL rozšíření projektu Starburst
 - jednoduchá syntaxe i sémantika
- *událost – podmínka – akce (Event-Condition-Action, ECA)*
 - princip fungování triggerů
 - „Když nastane událost a je splněna podmínka, vykonej akci“
 - zni to jednoduše
 - má to spousty háčků

62

událost – podmínka – akce

- událost
 - **INSERT, DELETE, UPDATE**
 - manipulační primitiva
- podmínka
 - libovolná SQL podmínka
- akce
 - libovolný SQL příkaz
 - SELECT, INSERT
 - DELETE, UPDATE
 - příkaz řízení transakce
 - ROLLBACK WORK

63

Syntaxe aktivních pravidel Starburst

```
CREATE RULE <jméno pravidla> ON <jméno tabulky>
WHEN <události>
[ IF <SQL podmínka> ]
THEN <SQL příkazy>
[ PRECEDES <seznam jmen pravidel> ]
[ FOLLOWES <seznam jmen pravidel> ]
```

64

Příklad vytvoření aktivního pravidla

```
CREATE RULE platy2 ON zamestnanci
WHEN INSERTED, DELETED, UPDATED
IF (SELECT avg(plat) FROM zamestnanci) > 100
THEN UPDATE zamestnanci SET plat = 0.9 * plat
FOLLOWES platy1
```

65

Příklad I. – pokračování

```
CREATE RULE RegulacePlatu ON
Zamestnanci
WHEN INSERTED, DELETED, UPDATED
(Plat)
IF (SELECT AVG(Plat) FROM
Zamestnanci) > 100
THEN UPDATE Zamestnanci
SET Plat = 0.9 * Plat
```

| Zaměstnanec | Plat |
|-------------|------|
| Božena | 90 |
| Jan | 90 |
| Josef | 110 |

- ▶ Průměrný plat zaměstnance je 97.
 - ▶ Uvažujme transakci, která přidá záznamy
- | Bořivoj | 150 |
|---------|-----|
| Oldřich | 120 |
- ▶ Vložení spustí pravidlo **Regulace platu**.
 - ▶ Nový průměrný plat je 112
⇒ podmínka je splněna
⇒ provede se akce

66

Příklad I. – pokračování

- ▶ Nový stav databáze po provedení pravidla:

| Zaměstnanec | Plat |
|-------------|------|
| Božena | 81 |
| Jan | 81 |
| Josef | 99 |
| Bořivoj | 135 |
| Oldřich | 108 |

- Průměrný plat je nyní 101
⇒ podmínka je splněna
⇒ provede se akce
- Nový stav databáze:

| Zaměstnanec | Plat |
|-------------|------|
| Božena | 73 |
| Jan | 73 |
| Josef | 89 |
| Bořivoj | 121 |
| Oldřich | 97 |

- ▶ Operace UPDATE v akci pravidla způsobí, že se pravidlo spustí znovu.

Příklad I. – dokončení

- Pravidlo je opět spuštěno díky operaci UPDATE.
- Pravidlo je vyhodnoceno, ale již se neprovede
 - Průměrný plat je ted' 91.
- Algoritmus provádění aktivních pravidel končí.
- Nebezpečí „zacyklení“ v případě špatně definovaných pravidel.

Příklad II.

- Uvažujme databázi jako na začátku Příkladu I.
- K databázi přidáme nové aktivní pravidlo *VysocePlacení*,
 - Pravidlo vkládá do pohledu *VysocePlaceníZaměstnanci* (VP2) ty nově přidané zaměstnance, kteří mají plat vyšší než 100.

```
CREATE RULE VysocePlacení ON Zaměstnanci
WHEN INSERTED
IF EXISTS (SELECT * FROM INSERTED WHERE Plat > 100)
THEN INSERT INTO VP2 (SELECT * FROM INSERTED
WHERE Plat > 100)
FOLLOWS RegulacePlatu
```

Příklad II. - pokračování

- Uvažujme nyní znovu přidání Bořivoje a Oldřicha do databáze.

| Zaměstnanc | Plat |
|------------|------|
| Božena | 90 |
| Jan | 90 |
| Josef | 110 |
| Bořivoj | 150 |
| Oldřich | 120 |

70

- Operace INSERT spustí obě pravidla.
- Algoritmus zpracovávání pravidel vybere nejprve pravidlo **RegulacePlatu**.
- Pravidlo **RegulacePlatu** se provede díky rekurzi dvakrát.

Příklad II. – dokončení

- Tabulka Zaměstnanci se dostane do stavu jako v Příkladě I.

| Zaměstnanec | Plat |
|-------------|------|
| Božena | 73 |
| Jan | 73 |
| Josef | 89 |
| Bořivoj | 121 |
| Oldřich | 97 |

- Nyní je pravidlo **RegulacePlatu** „nespuštěno“ a pravidlo **VysocePlacení** je „spuštěno“.
 - Pravidlo pokládá za vloženou tuto dočasnou tabulkou:
- | | |
|---------|-----|
| Bořivoj | 121 |
| Oldřich | 97 |
- 71
- Pouze řádek (Bořivoj,121) je vložen do VPZ.

Příklad II. – dokončení

- Tabulka Zaměstnanci se dostane do stavu jako v Příkladě I.

| Zaměstnanec | Plat |
|-------------|------|
| Božena | 73 |
| Jan | 73 |
| Josef | 89 |
| Bořivoj | 121 |
| Oldřich | 97 |

- Nyní je pravidlo **RegulacePlatu** „nespuštěno“ a pravidlo **VysocePlacení** je „spuštěno“.
 - Pravidlo pokládá za vloženou tuto dočasnou tabulkou:
- | | |
|---------|-----|
| Bořivoj | 121 |
| Oldřich | 97 |
- 72
- Pouze řádek (Bořivoj,121) je vložen do VPZ.

PL/SQL

Trigger – aktivní databáze - použití

DB2 – 3. přednáška

1

Doplňení syntaxe

- ▶ unikátní jméno pravidla
 - ▶ asociováno se specifickou tabulkou – cílem pravidla
- ▶ použití
 - ▶ pravidlo sleduje více událostí
 - ▶ stejná událost více pravidly
- ▶ příkazy PRECEDES a FOLLOW
 - ▶ použít lze pouze v době vzniku pravidla
 - ▶ určují částečné uspořádání (partial order)
 - ▶ vztah uspořádání musí být acylický
- ▶ sdružování pravidel do skupin
- ▶ aktivace a deaktivace pravidel

2

2

Sémantika – terminologie

Pravidlo je:

- *spuštěné (triggered)*
 - pokud nastane jím sledovaná událost
 - ostatní pravidla nespúštěná
 - spuštěné neznamená „výkonává se akce“, „výhodnocuje se podmínka“

3

3

Sémantika – terminologie

- *bráno v úvahu (considered)*
 - podmínka pravidla je vyhodnocena
- *provedeno (executed)*
 - příslušná akce je vykonána
 - vykonání je odložené
 - ve chvíli provedení příkazu COMMIT WORK
 - explicitně voláním PROCESS RULE

4 4

Sémantika – spuštění pravidel

- pravidlo je spuštěno
 - poté co nastane událost
 - pokud událost sleduje více pravidel, pak tvoří *konfliktní množinu (conflict set)*

5 5

Sémantika – spuštění pravidel

- algoritmus vyhodnocení pravidel

DOKUD je množina spuštěných pravidel M neprázdná {
vyber pravidlo R s nejvyšší prioritou z M a označ jako nespuštěné
POKUD je podmínka R splněna {
proved akci R
}
}
}
- jednoznačně opakovatelné
 - díky úplnému uspořádání

6 6

Sémantika - cykly

- mohou nastat
 - trigger T1 způsobí akci která znova spustí trigger T1...
- konečný stav (*quiescent state*)
 - je určen prázdnou konfliktní množinou
- zajistit konečnost konečnost vyvolávání triggerů je na autorovi pravidel ☺

7 7

Sémantika – detailly

- stavové přechody (*state transitions*)
 - transformace jednoho stavu databáze do druhého
 - výkonání SQL příkazů transakcemi
- vloženo, vymazáno, změněno (*inserted, deleted, updated*)
 - množiny popisující přechody
 - plní se n-ticemi změněnými SQL příkazy
 - představují všechny změny, které povedou ze stavu S1 do stavu S2

8 8

Sémantika – detailly

- čistý efekt (*net effect*)
 - znamená, že se každá n-tice změněných dat objeví právě jedně z množin vloženo, vymazáno, změněno
 - např. vložení a následné vymazání n-tice má nulový efekt
 - insert a následný update má čistý efekt insert nové hodnoty

9 9

Sémantika - detaily

- ▶ probíhá algoritmus vyhodnocování pravidel
- ▶ pravidlo je spuštěno
 - pokud je množina operací pravidlem sledovaných neprázdná
 - vztáženo k aktuálnímu přechodovému stavu
 - přechodový stav se mění s vykonáváním pravidel
 - akce spuštěného pravidel vyústí ve změnu množin
 - znova se utvoří konfliktní množina
- ▶ končí se prázdnou konfliktní množinou
 - konečným stavem

10 10

Úroveň sledování změn

- **úroveň instancí** (*instance level*)
 - změna řádku tabulky nebo objektu v případě objektově orientovaných databází
 - přechodové hodnoty jsou uchovávány 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**

11

Více pravidel ve stejnou dobu

- **Konfliktní množina** (*conflict set*)
 - aktívni 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

12

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 určován systémem nebo nedeterministickým výběrem
- bez explicitně vyjádřené priority
 - systém udržuje vlastní úplné uspořádání
 - výběr všech pravidel je nederministický

13

Opakovatelnost

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

14

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

15

Využití aktivních databází

16

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

17

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

18

Generování pravidel

- pravidla mohou být generována poloautomaticky
- deklarativní část pravidla**, tj. podmínka akce, které mohou tuto podmínuku 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 znova nastolit konzistentní stav

19

Opravné akce (SQL-92)

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

20

Příklad - referenční integrita

Tabulky:

Zaměstnanci

- každý zaměstnanec patří do nějakého oddělení
- atribut `oddeleni_id` 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)
```

21

Příklad - referenční integrita

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

22

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
```

23

Opravovací pravidlo

```

CREATE RULE OpravaOddeleni1 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)

```

25

99 je defaultní hodnota oddelení

Opravovací pravidla

- kaskádové mazání a update ...

28

Odvazování 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 přeponována kdykoliv se změní data, na kterých závisí

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

28

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=Zamestnanci.oddeleni) AND
        (Zamestnanci.mzda > 50000)
```

29

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

30

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=Zamestnaci.oddeleni) AND
                    (Zamestnaci.mzda > 50000)
            )
    
```

32

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=Zamestnanci.oddeleni) AND
        (Zamestnanci.mzda > 50000))
```

3-

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=Zamestnaci.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 (*sou pouze read-only*)
 - zachytávací modul (*capture module*)
- **Synchronní řešení**
 - střídavě primární a sekundární role
 - využívá distribuované transakce
 - není vždy vyžadováno aplikaci
 - dosti náročné

34

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)
```

35

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.

36

Temporální databáze

DB2 – 4. přednáška

1

Dotazy nad temporální databází

Dotazovací jazyk – TSQL2

- TSQL2 byl navržen skupinou asi 18 badatelů, kteří už dříve samostatně navrhli mnoho různých temporálních dotazovacích jazyků.
- Cílem TSQL2 bylo sjednotit přístupy k temporálním datovým modelům a dotazovacím jazykům založeným na kalkulu a získat sjednocené rozšíření SQL-92 a datový model, nad kterým by mohl být založen další výzkum.
- TSQL2 bylo začleněno do vyvíjejícího se standardu SQL3.

Pojetí času

- Časová osa TSQL2 je na obou koncích omezena, ale dostatečně daleko (18 miliard let)
- U časových údajů jsou možné různé granularity
- Časové typy
 - DATE, TIME, TIMESTAMP, INTERVAL, PERIOD
- Poznámka:
18 miliard let je zvoleno podle doby, před jakou mělo dojít k Velkému třesku.
Po sobě jdoucí chronony mohou být seskupovány do granulí, různým seskupováním dosahujeme různých granularit.
Všechny uvedené časové typy kromě PERIOD byly už v SQL-92.

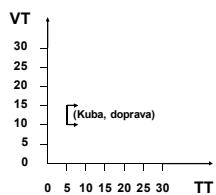
Datový model

- Je použit BCDM (Bitemporal Conceptual Data Model)
- Řádek relace je orazítkován množinou bitemporálních chrononů
- Bitemporální chronon je dvojice (chronon transakčního času, chronon času platnosti)
- Poznámka:
Množina bitemporálních chrononů se také nazývá bitemporální element.
Každý bitemporální chronon odpovídá malému obdélníku ve dvourozměrném prostoru (transakční čas x čas platnosti).
Protože nejsou povoleny duplicitní řádky, tak celá historie jednoho faktu je na jednom řádku. Rikáme, že BCDM je slívený datový model.

Příklad bitemporální relace (1)

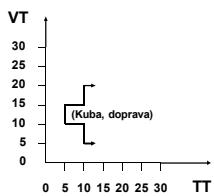
- Relace Zaměstnanec – umístění lidí v odděleních určitého podniku
- Schéma (Jméno, Oddělení) + časové razítko
- Předpokládejme granularitu 1 den u času platnosti i u transakčního času

Příklad bitemporální relace (2)



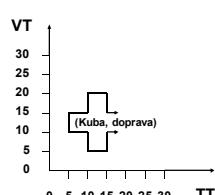
Časový úsek našeho zájmu je daný měsíc v daném roce, tedy 15 známená např. 15. 6. 2015.
Zaměstnanec Kuba byl najmut jako dočasná výpomoc v oddělení dopravy na dobu od 10. do 15.
a tento fakt se objevil v databázi 5.
Sípky směřující doprava značí, že rádek nebyl logicky smazán; jeho platnost pokračuje až do
času *until changed* (U.C.).

Příklad bitemporální relace (3)



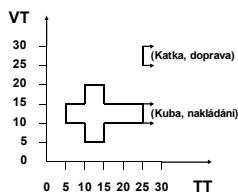
Personální oddělení zjistilo, že Kuba byl ve skutečnosti přijat na dobu od 5. do
20. a databáze je opravena 10.

Příklad bitemporální relace (4)



Později bylo personální oddělení informováno, že samotná oprava byla
nesprávná a že Kuba skutečně byl skutečně přijat na dobu od 10. do 15. a
tato oprava se objeví v databázi 15.

Příklad bitemporální relace (5)



Personální oddělení zjistí, že Kuba sice pracoval od 5. do 10., ale ne v oddělení dopravy, nýbrž v oddělení nakládání.
Navíc je najmota Katka pro práci v oddělení dopravy na dobu od 25. do 30.
Všechny zmíněné změny se objeví v databázi 25.

Příklad bitemporální relace (6)

| Jméno | Oddělení | Časové razítka |
|-------|-----------|---|
| Kuba | Doprava | $\{(5, 10), \dots, (5, 15), \dots, (9, 10), \dots, (9, 15), (10, 5), \dots, (10, 20), \dots, (14, 5), \dots, (14, 20), (15, 10), \dots, (15, 15), \dots, (24, 10), \dots, (24, 15)\}$ |
| Kuba | Nakládání | $\{(U.C., 10), \dots, (U.C., 15)\}$ |
| Katka | Doprava | $\{(U.C., 25), \dots, (U.C., 30)\}$ |

„Negrafická“ reprezentace obrázku
U.C.(until changed) znamená, že řádek nebyl z databáze logicky smazán (neboli stále si myslíme, že platí)

TSQL2

- Temporal Structured Query Language
- cílem sjednotit přístup k temporálním datovým modelům a dotazovacím jazykům
- rozšíření SQL92
- částečně obsažené v SQL3

Časová ontologie

- Lineární časová struktura, omezená z obou stran (+-18 miliard let)
- diskrétní reprezentace reálného času, která může být považována za diskrétní, hustou nebo průběžnou
- granule – seskupení po sobě jdoucích chrononů – různá granularita
- nevyžaduje výběr mezi diskrétní, hustou nebo průběžnou ontologií

Časová ontologie

- Dokonce nedovoluje otázky, které by nutili rozhodnout mezi modely
- Nelze se ptát, zda okamžik A předchází okamžik B – pouze v rámci zvolené granularity (vteřiny, dny,...)
- Přidává datový typ **PERIOD**

Datový model

- jednoduchý
- zachovává obecnost a jednoduchost relačního modelu
- separátní modely pro prezentaci dat, ukládání dat a využití dotazů
- Více koordinovaných datových modelů zvládne co by jeden nedokázal

Stavba jazyka

- striktní nadmnožina SQL92
 - pro příklad temporálních relací budeme používat databázi pacientů
- ```
CREATE TABLE Předpis(Jmeno CHAR(30), Lekar CHAR(30), Lek CHAR(30), Dávka
CHAR(30), Frekvence INTERVAL MINUTE)
AS VALID STATE DAY AND TRANSACTION
```
- frekvence je počet minut mezi dávkami
  - valid time – na kdy je lék předepsán
  - transaction time – příchod záznamu do databáze

---



---



---



---



---



---



---

## Druhy relací

- snímková – žádná temporální podpora
- valid-time state **AS VALID STATE**
- valid-time event **AS VALID EVENT**
- transaction-time **AS TRANSACTION**
- bitemporal state **AS VALID STATE AND TRANSACTION**
- bitemporal event **AS VALID EVENT AND TRANSACTION**
- typ relace se může změnit **ALTER TABLE**

---



---



---



---



---



---



---

## Definice schématu

- Příklad
- ```
CREATE TABLE Předpis (Jméno CHAR(30),
Lékař CHAR (30), Lék CHAR (30), Dávka CHAR (30),
Frekvence INTERVAL MINUTE)
AS VALID STATE DAY AND TRANSACTION
```
- **Čas platnosti má granularitu 1 den** (Čas platnosti specifikuje úsek, kdy měl pacient lék předepsaný).
 - **Transakční čas má granularitu 1 ms nebo menší** (Transakční čas určuje, kdy se tato informace nalézala v databázi. Řádky, které nebyly updatovány nebo smazány, budou mít transakční čas obsahující nynější okamžik).

SELECT

- Novým klíčovým slovem **SNAPSHOT** získáme snímek z temporální relace
- Kdo někdy měl předepsané léky?
SELECT SNAPSHOT Jmeno
FROM Predpis
- Kdo někdy měl předepsaný aspirin?
SELECT SNAPSHOT Jmeno
FROM Predpis
WHERE Lek = 'Aspirin'

SELECT

- Kdo měl předepsané léky a kdy?
SELECT Jmeno
FROM Predpis
- Defaultní chování vrací historii
- TSQL2 automaticky provádí koalescenci
- Výsledkem je množina řádků, každý s periodou, kdy pacient bral jeden či více léků

Přeorganizování (Restructuring)

- Jeden z nejsilnějších prostředků
- Koalescence se automaticky provádí na výsledek dotazu – toto umožňuje provést ji na řádky v klauzuli **FROM**
- Kdo bral lék celkem déle než 6 měsíců?

Přeorganizování

- ```
SELECT Jmeno, Lek
FROM Predpis(Jmeno, Lek) AS P
WHERE CAST(VALID(P) AS INTERVAL MONTH)
 > INTERVAL '6' MONTH
```
- Přeorganizování na Jmene a Leku, výsledkem je maximální doba kdy byl lék předepsán
  - **VALID(P)** vrací valid-time prvky z P
  - operátor **CAST** konvertuje co vyjde z valid

---



---



---



---



---



---



---

## Přeorganizování

- Kdo užíval Aspirin?
- ```
SELECT SNAPSHOT P1.Jmeno
FROM Predpis(Jmeno) AS P1, P1(Lek) AS P2
WHERE P2.Lek = 'Aspirin' AND VALID(P2) = VALID(P1)
```
- Spárování
 - Jak přeorganizování, tak spárování je „syntaktická hříčka,” dá se přepsat pomocí vnořených selectů

Štěpení (Partitioning)

- Často chceme zkoumat maximální periody timestamp
 - klíčové slovo **PERIOD**
 - Kdo bral stejný lék déle než 6 měsíců v kuse?
- ```
SELECT SNAPSHOT Jmeno, Lek, VALID(P)
FROM Predpis(Jmeno, Lek) (PERIOD AS P
WHERE CAST(VALID(P) AS INTERVAL MONTH)
 > INTERVAL '6' MONTH
```

---



---



---



---



---



---



---

## Štěpení

- Alternativa

```
SELECT Jmeno, Lek
FROM Predpis(Jmeno, Lek)(PERIOD AS P
WHERE CAST(VALID(P) AS INTERVAL MONTH)
 > INTERVAL '6' MONTH
```

- Pro každý pár lék-jméno pouze jeden výsledek s maximální délkou užívání.
- štěpení není „syntaktické cukrátko“

---



---



---



---



---



---



---

## VALID

- Jaké léky měla Michaela předepsány v roce 1996?

```
SELECT Lek
VALID INTERSECT(VALID(Predpis), PERIOD [1996] DAY)
FROM Predpis
WHERE Name = 'Michaela'
```

- Výsledkem je seznam léků společně s časem, kdy byl předepsán.

---



---



---



---



---



---



---



---

## Aktualizace dat

```
INSERT INTO Predpis
VALUES('Michaela', 'Dr. Sova', 'Aspirin', '100mg',
 INTERVAL '8:00' MINUTE)
```

- Nespecifikovali jsme timestamp, default:

```
VALID PERIOD(CURRENT_TIMESTAMP,
NOBND(CURRENT_TIMESTAMP))
```

- Otevřený konec (konec je aktuální čas)

---



---



---



---



---



---



---



---

## Aktualizace dat

- ```
INSER INTO Predpis
VALUES('Michaela', 'Dr. Sova', 'Aspirin', '100mg',
INTERVAL '8:00' MINUTE)
VALID PERIOD '[1996-01-01 – 1996-06-30]'
```
- automatická koalescence
 - transaction time je roven **CURRENT_TIMESTAMP**
 - VALID** je takto použitelné i v **DELETE** a **UPDATE**

Aktualizace dat

- DELETE** může změnit více záznamů kvůli překrývání timestampů – režii řeší TSQL2


```
UPDATE Predpis
SET Davka TO '50mg'
WHERE Name = 'Melanie' AND Lek = 'Aspirin'
```
- Dojde ke změně všech současných a budoucích! dávek

Události (Event Relations)

- Doteď jsme se zabývali jen stavem, který je po nějaký čas pravdivý
- Eventy zaznamenávají okamžité události


```
CREATE TABLE LabTest (Jmeno CHAR(30), Lekar CHAR(30), TestID INTEGER)
AS VALID EVENT HOUR AND TRANSACTION
```
- Události se také dají Štěpit a Přeorganizovávat

Události (Event Relations)

- Byl nějaký pacient jediný, kdo šel na testy od konkrétního lékaře?

```
SELECT L1.Jmeno, L2.Lekar
FROM LabTest(Jmeno) AS L1, L1(Lekar) AS L2,
      LabTest(Lekar) AS L3
WHERE VALID(L1) = VALID(L2) AND L2.Lekar = L3.Lekar
      AND VALID(L1) = VALID(L3)
```

Podpora Transaction time

- Doted' jsme neřešili, že tabulka Predpis podporuje transaction time
- Jaké předpisy Michaela měla?


```
SELECT Lek
      FROM Predpis
      WHERE Jmeno = 'Michaela'
```
- Vrací historii jak je nejlépe známá, včetně oprav

Podpora Transaction time

- Můžeme udělat v databázi rollback
- Kdyby bylo 1.6.1996, jaké předpisy by Michaela měla?


```
SELECT Lek
      FROM Predpis AS P
      WHERE Jmeno = 'Michaela'
          AND TRANSACTION(P) OVERLAPS DATE '1996-06-01'
```
- Default je **TRANSACTION(P) OVERLAPS CURRENT_TIMESTAMP**

Podpora Transaction time

- Kdy byla Michaelina data, validní k 1.6.1996 naposledy opravována?

```
SELECT SNAPSHOT BEGIN(TRANSACTION(P2))
FROM Predpis AS P1P2
WHERE P1.Jmeno = 'Michaela' AND P2.Jmeno = 'Michaela'
AND VALID(P1) OVERLAPS DATE '1996-06-01'
AND VALID(P2) OVERLAPS DATE '1996-06-01'
AND TRANSACTION(P1) MEETS TRANSACTION(P2)
```

Agregační funkce

- ```
SELECT COUNT *
FROM Predpis
WHERE Jmeno = 'Michaela'
```
- vrací valid-time state relaci
  - jak se měnil počet předpisů v libovolném bodě v čase

---



---



---



---



---



---



---

## Agregační funkce

- TSQL2 přidává funkci **RISING**
  - nejdelší období, kdy atribut monotónně rostl
- ```
SELECT SNAPSHOT RISING (Davka)
FROM Predpis
WHERE Jmeno = 'Michaela' AND Lek = 'Aspirin'
```
- dotaz vrátí množinu období, kdy atribut rostl

Vývoj a verzování schématu

- SQL dovoluje schéma měnit pomocí **ALTER** - vývoj.
- Pokud má relace podporu transaction-time tak se schéma pro tuto relaci verzuje
- V praxi se celé schéma stane množinou relací transaction-time
- Když chci jinou verzi: **SET SCHEMA DATE '1996-08-19'**

Některé další konstrukce

- *surrogate* – unikátní hodnota, vhodná k porovnání na shodu; TSQL2 přidává sloupec **SURROGATE** a unární funkci **NEW**
- *vacuuming* – odstranění zastaralých dat
 - s podporou transaction-time data nemizí z databáze, ale přidá se timestamp o smazání

Shrnutí (TSQL2)

- přidává práci s prvky které se mění časem
- lze používat i konvenční relace
- periody jsou nový typ s daným trváním v čase

Temporální databáze

DB2 – 4. přednáška

1

Literatura

- Carlo Zaniolo: Advanced database systems
 - kapitoly 5 a 6
- ISO / IEC 9075 (1-4,9-11,13,14): (2008)
 - ISO/IEC 9075-1:2008 Framework (SQL/Framework)
 - ISO/IEC 9075-2:2008 Foundation (SQL/Foundation)
 - ISO/IEC 9075-3:2008 Call-Level Interface (SQL/CLI)
 - ISO/IEC 9075-4:2008 Persistent Stored Modules (SQL/PSM)
 - ISO/IEC 9075-9:2008 Management of External Data (SQL/MED)
 - ISO/IEC 9075-10:2008 Object Language Bindings (SQL/OLB)
 - ISO/IEC 9075-11:2008 Information and Definition Schemas (SQL/Schemata)
 - ISO/IEC 9075-13:2008 SQL Routines and Types Using the Java TM Programming Language (SQL/JRT)
 - ISO/IEC 9075-14:2008 XML-Related Specifications (SQL/XML)

www.wikipedia.org

Temporální databáze

- Příklady: SIS, analýzy reklamy
- Konvenční databáze reprezentují stav
- Změny stav upravují
- Temporální databáze podporují prvek času
- Dotazy přes časové období

Případová studie

- Databáze zaměstnanců
Zamestnanec(Jmeno, Plat, Titul)
- Jaký je Davidův plat?
SELECT Plat
FROM Zamestnanec
WHERE Jmeno = 'David'

Případová studie

- Přidáme do záznamu datum narození
Zamestnanec(Jmeno, Plat, Titul, DatumNarozeni **DATE**)
- Kdy se David narodil?
SELECT DatumNarozeni
FROM Zamestnanec
WHERE Jmeno = 'David'
- Tedy v SQL je (omezená) temporální podpora

Případová studie

- Nyní chceme přidat záznam o vývoji v zaměstnání
Zamestnanec(Jmeno, Plat, Titul, DatumNarozeni, Start **DATE**, Stop **DATE**)
- Datový typ je stejný jako Datum Narození, ale dopad mnohem větší

Případová studie

- Temporální projekce
- Jaký je Davidův současný plat?

```
SELECT Plat
FROM Zamestnanec
WHERE Jmeno = 'David' AND Start <= CURRENT_DATE
AND CURRENT_DATE <= Stop
```

- Díky novým sloupcům je dotaz složitější

Případová studie

- Potřebujeme zjistit **historii** platů zaměstnanců
- Potřebujeme zjistit **maximální dobu** kdy měli stejný plat
- V SQL velmi obtížné
- Koalescence

Případová studie

| Jméno | Plat | Titul | Datum Narození | Start | Stop |
|-------|-------|----------|----------------|------------|------------|
| David | 60000 | Magistr | 1945-04-09 | 1995-01-01 | 1995-06-01 |
| David | 70000 | Magistr | 1945-04-09 | 1995-06-01 | 1995-10-01 |
| David | 70000 | Doktor | 1945-04-09 | 1995-10-01 | 1996-02-01 |
| David | 70000 | Profesor | 1945-04-09 | 1996-02-01 | 1997-01-01 |

| Jméno | Plat | Start | Stop |
|-------|-------|------------|------------|
| David | 60000 | 1995-01-01 | 1995-06-01 |
| David | 70000 | 1995-06-01 | 1997-01-01 |

```

CREATE TABLE Temp(Plat, Start, Stop)
AS SELECT Plat, Start, Stop
FROM Zamestnanec
WHERE Jmeno = 'David'
repeat
    UPDATE Temp T1
    SET (T1.Stop) = (SELECT MAX(T2.Stop)
                     FROM Temp AS T2
                     WHERE T1.Salary = T2.Salary AND T1.Start < T2.Start
                     AND T1.Stop >= T2.Start AND T1.Stop < T2.Stop)
    WHERE EXISTS (SELECT *
                  FROM Temp AS T2
                  WHERE T1.Salary = T2.Salary AND T1.Start < T2.Start
                  AND T1.Stop >= T2.Start AND T1.Stop < T2.Stop)
until no updates
DELETE FROM Temp T1
WHERE EXISTS (SELECT *
              FROM Temp AS T2
              WHERE T1.Salary = T2.Salary
              AND ((T1.Start > T2.Start AND T1.Stop <= T2.Start)
              OR (T1.Start >= T2.Start AND T1.Stop < T2.Stop)))

```

Případová studie

- Problém s tímto řešením – repeat-until
- SQL řešení existuje – pomocí zahnízděných **NOT EXISTS**
- SQL nemá prostředky pro práci s „timestampy“
- Řešení v TSQL2 (Davidova historie platů)

```

SELECT Plat
FROM Zamestnanec
WHERE Jmeno = 'David'

```

Případová studie

- Temporální join
- Reorganizujeme schéma, čímž se vyhneme problémům v SQL

Zamestnanec1(Jmeno, Plat, Start **DATE**, Stop **DATE**)
Zamestnanec2(Jmeno, Titul, Start **DATE**, Stop **DATE**)

- Jaká je Davidova historie platů?

```

SELECT Plat, Start, Stop
FROM Zamestnanec1
WHERE Jmeno = 'David'

```

Případová studie

- Co se stane, chceme-li následně vztah mezi obdobími platu a titulu?
- SQL dotaz musí zjistit, jak se překrývá řádek z tabulky Zamestnanec1 s řádky ze Zamestnanec2

Případová studie

- Najdi historii platů a titulů pro všechny zaměstnance

```
SELECT Zamestnanec1.Jmeno, Plat, Titul, Zamestnanec1.Start,
       Zamestnanec1.Stop
FROM Zamestnanec1, Zamestnanec2
WHERE Zamestnanec1.Jmeno = Zamestnanec2.Jmeno
       AND Zamestnanec2.Start <= Zamestnanec1.Start
       AND Zamestnanec1.Stop <= Zamestnanec2.Stop
UNION
SELECT Zamestnanec1.Jmeno, Plat, Titul, Zamestnanec1.Start,
       Zamestnanec1.Stop
...
```

Případová studie

- 4 případy jak se záznamy překrývají
- Najdi historii platů a titulů pro všechny zaměstnance (TSQL2)

```
SELECT Zamestnanec1.Jmeno, Plat, Titul,
FROM Zamestnanec1, Zamestnanec2
WHERE Zamestnanec1.Jmeno = Zamestnanec2.Jmeno
```

Shrnutí

- Data závislá na čase jsou běžně využívána
- Netemporální databáze nemají dostatečnou podporu pro práci s těmito údaji
- Temporální databáze umožňuje jednodušší dotazy

Časová doména

- Modelování a reprezentace času
- Čas v temporální logice = libovolná množina okamžiků s částečným uspořádáním
- Další axiomy model upřesňují
 - Lineární – uspořádání celé množiny
 - Větvení – lineární do teď, pak možné budoucnosti
 - Cyklický – rekurentní proces (týden)

Časová doména

- Axiomy mohou charakterizovat *hustotu*
- ***Diskrétní modely*** – isomorfni přirozeným číslům
 - každý bod v čase má jednoho předchůdce
- ***Husté modely*** – isomorfni racionálním nebo reálným číslům
 - mezi každými dvěma momenty existuje další
- ***Průběžné modely*** – isomorfni reálným číslům

Časová doména

- V průběžném modelu každé reálné číslo odpovídá bodu v čase
- V diskrétním modelu každé přirozené číslo odpovídá nedělitelné jednotce času s libovolným trváním – *chronon*
- *Chronon* není bod, ale úsečka

Časová doména

- Obvykle se používá *diskrétní model*
 - nepřesnost měření
 - přirozenost v jazyce
 - přirozená modelace událostí, které trvají

Časová doména

Další axiomy:

- *Omezení*
- Koncept *vzdálenosti*
- *Relativní a Absolutní čas*

Datové typy času

- **okamžik** – specifický chronon
- **událost** – něco, co se stalo v daném okamžiku
- **doba události** – okamžik, kdy se událost stala ve skutečnosti
- **SQL92** – DATE, TIME, TIMESTAMP
- **časová perioda** – čas mezi dvěma okamžiky
 - neplést s typem INTERVAL

Datové typy času

- **časový interval** – známý časový úsek, ale nemá specifické hraniční okamžiky
- **množina okamžiků**
- **temporální elementy** – konečné sjednocení period

Asociování faktů s časem

- **Valid time** – čas, po který fakt **byl/je/bude** pravdivý.
- **Transaction time** – období, po které je fakt uložený v databázě
- **snímek** – nepodporuje ani jeden model, okamžik v databázi
- **bitemporální** – podporuje oba modely

Snímek

| | | |
|--|--|--|
| | | |
| | | |
| | | |

- Zachycuje, co je aktuálně pravda

Transaction-time relace

- Snímek se neupravuje
- Dojde ke změně aktuálního snímku, který se poté přidá do relace
- Využitelné pro dotazy na stav databáze v minulosti

Valid-time relace

- Uchovává platnost dat
- Kterakoliv část se dá upravit
- Nedá se určit předchozí stav databáze

Bitemporální relace

- „append only“ jako transaction-time
- Platnost dat jako valid-time
- S rozvojem databáze transaction-time roste monotonně, ale valid-time může mít velký rozptyl

Shrnutí datového modelu

Temporální datový model by měl splňovat mnoho požadavků“

- jasná sémantika aplikace
- konzistentní, minimální rozšíření existujícího modelu
- souvislé chování faktů v čase
- snadná implementace, vysoký výkon

Tyto požadavky se zdají protichůdné

Shrnutí datového modelu

- Simultánní zaměření na prezentaci dat, uložení dat a efektivní vyhodnocování dotazů komplikuje zachycení časově proměnných dat
- Mnoho nekompatibilních datových modelů s mnoha dotazovacími jazyky

Objektově orientované databáze Objektově relační databáze

Obsah

Objektově orientované databáze (ODMG 93)
Objektově relační databáze

Rozšířitelnost, uživatelsky definované typy a funkce
Opravdové ORSŘBD (SQL:1999, 2003, SQL4)

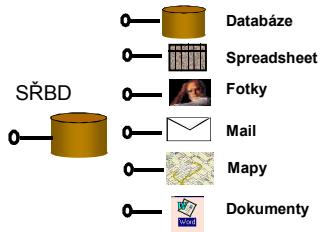
Proč více db technologií

Požadavky nových aplikací:

- ▶ nové typy objektů a funkcí
- ▶ OO analýza a návrh vs. relační db

"Relační databáze je podobná garáži, která vás nutí rozmontovat vaše auto a uložit díly do malých zásuvek..."

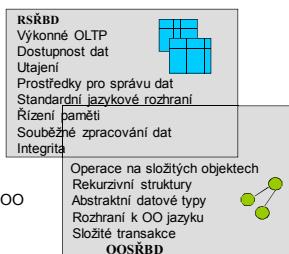
Cíl: integrace a správa dat v jednom systému



Objektově orientované databáze

objektový datový model:

- je v souladu s viděním světa (entita \Rightarrow objekt)
- definice složitých objektů a jejich manipulace



Objektově orientované databáze

1993: konsorcium vůdčích výrobců OOSRBD \Rightarrow návrh standardu ODMG-93.

- nadmnožina obecnějšího modelu Common Object Model (COM) vytvořené skupinou OMG. Převzat byl jeho definici jazyk IDL.
- dotazovací část Object Query Language (OQL), která souvisí s koncepcí dotazovací části standardu SQL92.
- rozhraní k OO PJ C++, Smalltalk (k Java: nahrazeno Java Data Objects (JDO))

2001: skupina rozpuštěna (verze ODMG 3.0)

OOPJ + SŘBD = OOSRBD

Základní koncepty ODMG-93

- ▶ **třída** (nebo *typ*), *instance* (nebo *objekt*), *atribut*, *metoda* a *integritní omezení*
 - třída - šablona pro instance (objekty), které mohou sdílet atributy a metody.
 - doména atributů: primativní typ dat, abstraktní typ dat (ADT), nebo odkaz na třídu.
 - metoda je funkce (její implementace je skryta) aplikovatelná na instance třídy (výpočet založený na hodnotách atributů).
- ▶ **identifikátor objektu (OID)**
 - každý objekt má jednoznačný identifikátor, prostřednictvím kterého lze z databáze získat odpovídající objekt.

Základní koncepty ODMG-93

- **zapouzdření**

- data jsou "zabalená" spolu s metodami. Jednotkou zapouzdření je objekt. Metody jsou platné pouze na příslušných objektech, se kterými jsou zapouzdřeny.

- **hierarchie tříd, dědění**

- podtřída ⇒ hierarchie
- dědění je proces znamenající pro podtřídu osvojení všech atributů a metod z nadtřídy.
- vícenásobné dědění (⇒ problémy např. řešení konfliktů stejných jmen zděděných atributů a metod).

Nástup OO db technologie

Zdroje: OO programování, OO analýza a návrh, relační SŘBD

- Objektově relační mapování (**ORM**)

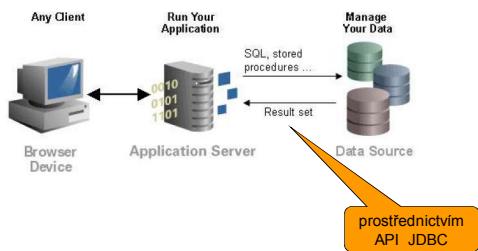
Př.: Hibernate (rozhraní: Session, Transaction, Query)

TopLink (ORM aplikace vlastněná Oracle, Inc.)

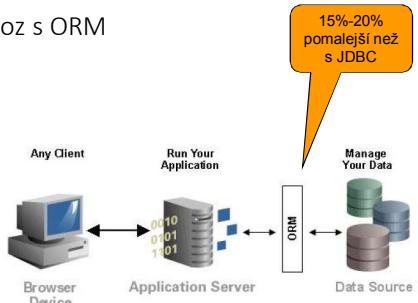
Přístup k objektům: např. Hibernate Query Language → SQL; programovací jazyk + metody realizující vstup do SQL databáze

- problém: méně sémantiky v relacích, impedance mismatch v přístupu k objektům

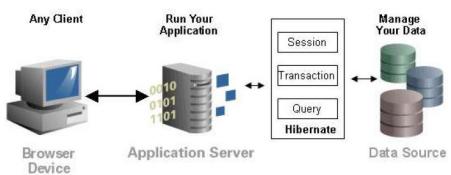
Provoz bez ORM



Provoz s ORM



Příklad: Hibernate



Nástup OO db technologie

- 2. pol. 80. let - OO databáze
Př.: O2 (→ Unidata → Informix → IBM)
ObjectStore (dnes Execelon)
- Versant Object Database, Objectivity/DB, GemStone, GemStone/J, DB4Objects, Jasmine,
Dotazování: OQL – neúplný, jinak: např. Objectivity/SQL++
- Dnes: Caché, ObjectStore, Versant Object Database
- částečný neúspěch: nenabídly pružnost a výkonnost relačních SŘBD

Shrnutí - relační model

- Relační model je jednoduchý a elegantní, ale je naprosto rozdílný od objektového modelu.
- Relační databáze nejsou navrhovány pro ukládání objektů a naprogramování rozhraní pro ukládání objektů v databázi je velmi složité.
- Relační databázové systémy jsou dobré pro řízení velkého množství dat, vyhledávání dat, ale poskytují nízkou podporu pro manipulaci s nimi.
- Jsou založeny na dvouzměrných tabulkách a vztahy mezi daty jsou vyjadřovány porovnáváním hodnot v nich uložených.
- Jazyky jako SQL umožňují tabulky propojit za běhu - aktuální vyjádření vztahu mezi daty.

13

Shrnutí - objektově orientovaný model

- Objektově orientovaný model je založen na objektech – strukturách kombinujících daný kód a data.
- Objektové databázové systémy umožňují využití hostitelského objektového jazyka jako je třeba C++, Java, nebo Smalltalk přímo na objekty "v databázi", – lze odstranit nepříjemné přeskakování mezi jazykem aplikace (např. C) a dotazovacím jazykem (např. SQL)
- Programátor jednoduše používá objektový jazyk k vytváření a přístupu k metodám.
- ODBMS jsou výborné pro manipulaci s daty.
- Opomeneme - li programátorskou stránku - některé typy dotazů jsou efektivnější než v RDBMS díky dědičnosti a referencím.

14

Objektově-relační databáze

"Rozšířená relační" a "objektově-relační" jsou synonyma pro databázové systémy, které se snaží sjednotit rysy jak relačních, tak objektových databází.
ORDBMS je specifikována v rozšíření SQL standardu — SQL3.
Příklady: Informix, IBM, Oracle, Unisys,

15

Datový model

- ORDBMS využívají datový model tak, že "přidávají objektovost do tabulek".
- Všechny trvalé informace jsou stále v tabulkách, ale některé položky mohou mít bohatší datovou strukturu, nazývanou abstraktní datové typy (ADT).
- Podpora ADT je atraktivní - operace a funkce asociované s novými datovými typy mohou být použity k indexování, ukládání a získávání záznamů na základě obsahu nového datového typu.
- ORDBMS jsou nadmnožinou RDBMS a pokud nevyužijeme žádné objektové rozšíření jsou ekvivalentní SQL2.
- ORDBMS má omezenou podporu dědičnosti, polymorfismu, referencí a integrace s programovacím jazykem.

16

Nástup OR db technologie

Důvody:

- obdržet maximum z rozsáhlých investic do relační technologie (data, nabité zkušenosti),
- využít výhody v pružnosti, produktivitě a provozních přínosů OO modelování,
- integrovat databázové služby do systémů výroby a dalších aplikací.

Nástup OR db technologie

- 90. léta - OR přístup - ORSŘBD
 - kombinace OO a relačních SŘBD
 - Př.: 1992: UniSQL/X, dále: HP - OpenODB (později Oadapter)
 - 1993: Montage Systems (později Illustra) - komerční verze Postgres
- dnes: DB/2, INFORMIX, ORACLE, Sybase Adaptive Server+Java, OSMOS, Unidata

Nástup OR db technologie

- počátky: univerzální servery
 - rozšiřitelnost relačního přístupu
 - Příklady:
 - Informix (Universal Server),
 - ORACLE 7.3
 - DB2 Common Server (1995), později DB2 Universal Database,

Princip: **z dolu nahoru**

Použití: pro již existující data v relační DB

Objektově relační databáze

Dva přístupy:

- *univerzální paměť*, kdy všechny druhy dat jsou řízeny SŘBD, jde o integraci (různými způsoby!)
 - ⇒ univerzální servery
- *univerzální přístup*, kdy všechna data jsou ve svých původních (autonomních) zdrojích

Technika: middleware

- brány (min. dva nezávislé servery)
- zobrazení schémat, transformace dotazů
- objektové obálky: Persistence Software, Ontologic, HP, Next, ... (problémy: výkon)
- DB založené na Web

Rozšiřitelnost, uživatelsky definované typy a funkce

Možnosti ADT:
black box
white box

Požadavek: manipulace BLOB (v RSŘBD atomicky)

Rozšiřitelnost: možnost přidávání nových datových typů + programů (funkce), „zabalených“ do speciálního modulu

⇒ UDT (uživatelsky definované typy)

UDF (uživatelsky definované funkce)

Problém: zapojení do relačního SŘBD (včetně SQL !)

DB/2: relační extender

Informix: DataBlades

ORACLE: cartridges

Sybase: Component Integration Layer.

univerzální servery

Rozšiřitelnost, uživatelsky definované typy a funkce

Př.: DB/2 v r. 2006:

- MapInfo
- NetOwl (přirozený jazyk v business intelligence)
- EcoWin (časové řady, makroekonomická časové řady, ...)
- GIS a prostorové objekty
- SQL expander (matematické, finanční, konverzní, ... funkce)
- VideoCharger (audio a video objekty v reálném čase)
- text, XML, audio, video, obrázky
- FormidaFire (integrace heterogenních dat)
- ...

Př.: Informix v r. 2006:

- C-ISAM, Excalibur Text Search, Geodetic, Image Foundations, Spatial, TimeSeries, Video Foundation, Web

Rozšiřitelnost, uživatelsky definované typy a funkce

Standardizace: SQL/MM

(Př.: Full-Text - řeší ADT + odpovídající funkce)

Implementace: technologie „plug in“ pomocí různých technik:

Př.: DataBlades - přímý přístup k databázovému jádru
ORACLE 7.3 - architektura více serverů a API

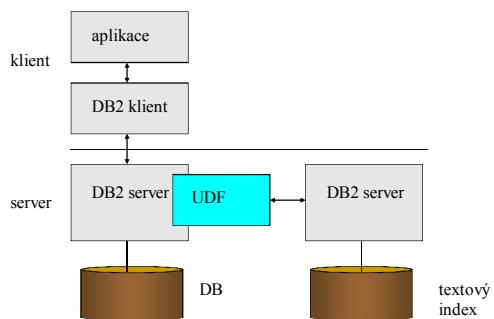
Příklad - textový extender

```
SELECT časopis, datum, titul
FROM ČLÁNKY
WHERE CONTAINS(text_článku, ('"databáze" AND
      ("SQL" | "SQL92") AND NOT "dBASE")') = 1;
Další funkce: NO_OF_MATCHES (kolikrát se zadaný vzorek vyskytoval
v textu), RANK (hodnota pořadí v odpovědi na základě nějaké míry).
SELECT časopis, titul
FROM ČLÁNKY
WHERE NO_OF_MATCHES (text_článku, 'databáze') > 10;
SELECT časopis, datum, titul RANK(text_článku, ('"databáze" AND
      ("SQL" | "SQL92")')) AS relevantní
FROM ČLÁNKY
ORDER BY relevantní DESC;
```

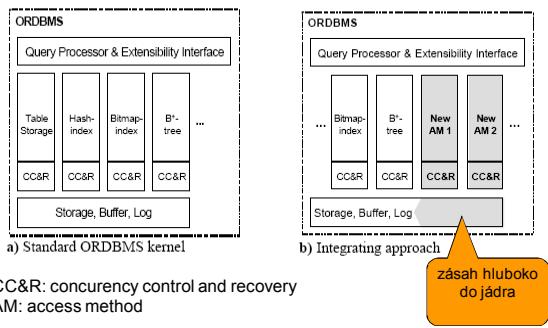
Architektura známých produktů

- přidání zvláštního aplikačního rozhraní (API) a speciálních serverů (také ORACLE 7.3 – viz např. CONTEXT, Media Server, OLAP),
- simulace OR na úrovni middleware (také ORACLE 7.3 - viz např. část Spatial Data Option),
- úplné přepracování databázového stroje (např. Illustra Information Technology),
- přidání OO vrstvy k relačnímu stroji (např. INFORMIX Universal Server, IBM D2/6000 Common Server, Sybase Adaptive Server + Java).

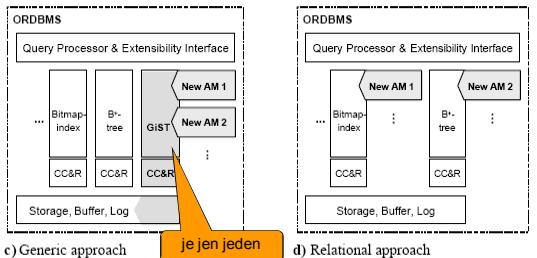
Interakce DB a textového extenderu v DB2



Architektury rozšířitelnosti (1)



Architektury rozšiřitelnosti (2)



GiST: Generalized Search Tree

AM na vrcholu relačního SŘBD

„Opravdové“ ORSŘBD

Won Kim: „rozšiřitelnost je pouze druhotný, i když užitečný rys, jde o důsledek OO přístupu“
Stonebraker: „rozšiřitelnost typu “plug-in” (např. ORACLE) jako sice vhodnou pro konektivitu aplikace-aplikace, nicméně nemá nic do činění s databázovou “plug-in”. Jde o pouhý middleware, který nezakládá OR technologii“.

Požadavky:

- datový model s hlavními rysy ODMG-93
- odpovídající objektový jazyk vyšší úrovně

Řešení: OO rozšíření SQL naplňuje (přibl.) tyto požadavky

Dnes: standard SQL:1999, SQL:2003 + další vývoj

Objektově relační modelování

- Rozšíření relačního modelu o objekty a konstrukty pro manipulaci nových datových typů,
- atributy n-tic jsou složité typy, včetně hnizděných relací,
- zachovány jsou relační základy včetně deklarativního přístupu k datům,
- kompatibilita s existujícími relačními jazyky (tvoří podmnožinu).

Příklad: hnízděná vs. normalizovaná relace

| časopis | titul | autoři | klíčová_slova | datum | | |
|---------|----------|--------------|-------------------|-------|--------|------|
| | | | | den | měsíc | rok |
| CW | OLAP | {Kusý, Klas} | {hvězda, dimenze} | 23 | duben | 1998 |
| SN | Databáze | {Novák, Fic} | {RDM, schéma} | 15 | květen | 1998 |

| časopis | titul | autor | klíčové_slovo | den | měsíc | rok |
|---------|----------|-------|---------------|-----|--------|------|
| CW | OLAP | Kusý | hvězda | 23 | duben | 1998 |
| CW | OLAP | Kusý | dimenze | 23 | duben | 1998 |
| CW | OLAP | Klas | hvězda | 23 | duben | 1998 |
| CW | OLAP | Klas | dimenze | 23 | duben | 1998 |
| SN | Databáze | Novák | RDM | 15 | květen | 1998 |
| SN | Databáze | Novák | schéma | 15 | květen | 1998 |
| SN | Databáze | Fic | RDM | 15 | květen | 1998 |
| SN | Databáze | Fic | schéma | 15 | květen | 1998 |

Normalizace do 4NF

| časopis | titul | titul | autor | titul | klíčové slovo |
|---------|----------|----------|-------|----------|---------------|
| CW | OLAP | OLAP | Kusý | OLAP | hvězda |
| SN | Databáze | OLAP | Klas | OLAP | dimenze |
| | | Databáze | Novák | Databáze | schéma |
| | | Databáze | Fic | Databáze | RDM |

| titul | den | měsíc | rok |
|----------|-----|--------|------|
| OLAP | 23 | duben | 1998 |
| Databáze | 15 | květen | 1998 |

Nevýhody 4NF:

- spojení v dotazech

Nevýhody pouhé 1NF

- ztráta vztahu řádek = 1 objekt

SQL:1999

Pět částí:

- SQL/Framework
 - SQL/Foundations
 - SQL/CLI (Call Level Interface*)
 - SQL/PSM (Persistent Store Modules**)
 - SQL/Bindings
- 75 str.
1100 str.
400 str.
160 str.
250 str.
(SQL Embedded, Dynamic SQL, Direct invocation)

* alternativa k volání SQL z aplikačních programů (implementace: ODBC)
** procedurální jazyk pro psaní transakcí
*** dynamický, vnořený, přímý SQL

SQL:1999

- podpora objektů
- uložené procedury
- triggers
- rekurzivní dotazy
- rozšíření pro OLAP
- procedurální konstrukty
- výrazy za ORDER BY
- savepoints
- update prostřednictvím sjednocení a spojení



Objekty: od SQL3 k SQL:1999

- SQL3 pro podporu objektů používá:
 - uživatelem definované typy (**ADT**, pojmenované typy řádků a odlišující typy),
 - konstruktory typů pro typy řádků a typy odkazů,
 - konstruktory typů pro typy kolekcí (množiny, seznamy a multimnožiny),
 - uživatelem definované funkce (UDF) a procedury (UDP),
 - velké objekty (Large Objects neboť LOB).
- Standard SQL:1999 - podmnožina celkové koncepce

UDT - User-Defined Types

- **Objektově relační DB systémy** nejsou plně objektové (nemodelují objekty a vztahy mezi nimi přímo).
- V nejnižší vrstvě jsou relační a objekty nad nimi jsou simulovány pomocí SRBD.
- Čistě objektové systémy zatím nebyly standardizovány, neměly standardní jazyk.

UDT - User-Defined Types

Přínosy

- Standardizace (SQL 1999) a podpora výrobců relačních systémů
- Uživatelské datové typy
- Data reprezentována objekty (které mají OO vlastnosti)
- Jednoznačná identifikace objektu — OID
- OID umožnuje tvorbu trvalých vztahů a zjednodušuje indexaci
- Vztahy (1:1, 1:N, N:N) jsou součástí definice dat a ne v algoritmu
- Vnořené struktury (např. adresa) a tabulky
- Kolekce (řetězec, posloupnost, seznam, soubor) a kurzor

37

UDT - User-Defined Types

SQL 1999 zavedlo podporu pro objektově relační databáze

- Objektové rysy
 - Uživatelské datové typy (UDT) — definice struktur a jednoduchá dědičnost, vzácné typy (vycházejí z vestavěných) — objekt, kolekce
 - Typované tabulky — tabulka je typu UDT (tedy je to třída a řádky jsou objekty)
- Nové relační rysy
 - Nové datové typy — LOB (BLOB, CLOB), BOOLEAN, ARRAY, ROW (složený sloupec)
 - Regulární výrazy — WHERE x SIMILAR TO regexp
 - Databázové triggery — podpora aktivních prvků databáze
 - OID — objekty lze reprezentovat jako řádky tabulek (tříd) a mít reference typu REF (identifikuje řádek v typované tabulce)
 - Přístup k hodnotám struktur přes operátor .
 - dědičnost, polymorfismus

38

ADT – abstraktní datový typ

Abstraktní datový typ (ADT) je výraz pro typy dat, které jsou nezávislé na vlastní implementaci.

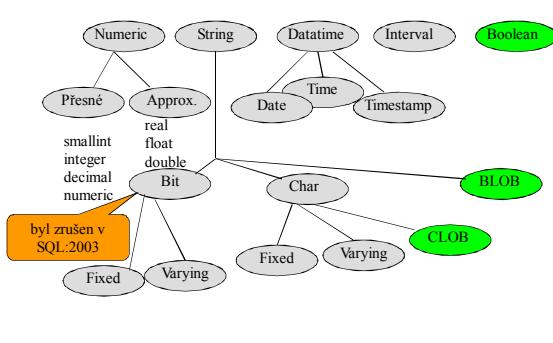
Abstraktní datový typ je implementačně nezávislá specifikace struktury dat s operacemi povolenými na této struktuře.

Základní ADT jsou například:

asociativní pole
zášobník
seznam
fronta
množina
zobrazení
textový řetězec
strom
halda - prioritní fronta
hasovací tabulka

39

Předdefinované typy v SQL:1999



Typ Boolean

```
SELECT č_odd, EVERY(plat > 20000) AS všichni_bohatí,  
      SOME(plat > 20000) AS někteří_bohatí  
  FROM zam  
 GROUP BY č_odd;
```

Výsledek:

| č odd | všichni bohatí | někteří bohatí |
|-------|----------------|----------------|
| A35 | FALSE | FALSE |
| J48 | TRUE | TRUE |
| Z52 | FALSE | TRUE |

Další typy v SQL:1999

Konstruované atomické typy:

- reference

Konstruované kompozitní typy:

- array /* podtyp collection */
 uspořádaný seznam dané maximální délky
 nejsou povoleny žádná pole polí nebo vícedimensionální
 pole
- row

Pz.: v SQL:2003 je více podtypů kolekcí (v implementacích rovněž)

Pz.: k typům existují nové funkce (BIT_LENGTH,
POSITION, SUBSTRING, ...)

Typ pole

```
CREATE TABLE zprávy(
    ID      INTEGER
    autoři VARCHAR(15) ARRAY[20]
    titul  VARCHAR(100)
    abstrakt FULLTEXT
    • přístup ke složkám pole pozičním (pořadovým číslem), např. autoři[3],
    • funkce CARDINALITY, porovnání =, <, >, zřetězení ||, CAST
    • UNNEST (odhnižení),
    • možnost WITH ORDINALITY (lze generovat sloupec offset odpovídající pořadovému číslu prvku v poli)
SELECT z.ID, a.jméno
FROM zprávy AS z, UNNEST(z.autoři) as a(jméno)
```

Další typy v SQL:1999

UDT:

- *odlišující typy* (jsou zatím budované pouze na předdefinovaných typech)
 - *struktuované typy* (mohou být definované s více atributy, které jsou předdefinovaných typů, typu ARRAY, nebo dalšího strukturovaného typu)
 - chování je realizováno pomocí funkcí, procedur a metod
 - mohou být organizovány do hierarchií s děděním
- Poznámka: jde o abstraktní datový typ (ADT)

Odlišující typy

Princip: - přejmenování (rozšíření) předdefinovaných typů + odlišné chování

```
CREATE TYPE TYP_MÍSTNOSTI
AS CHAR(10) FINAL;
CREATE TYPE METRY
AS INTEGER FINAL;
CREATE TYPE KV_METRY
AS INTEGER FINAL;
CREATE TABLE místnosti(
    m_id      TYP_MÍSTNOSTI
    m_délka   METRY
    m_šířka   METRY
    m_obvod   METRY
    m_plocha  KV_METRY);
```

hlásí chybu

OK

```
UPDATE místnosti
SET m_plocha = m_délka
UPDATE místnosti
SET m_šířka = m_délka
```

Pozor: srovnej s pojmem DOMAIN!

Typ řádku - nepojmenovaný

```
CREATE TABLE osoby (
    jméno VARCHAR(20),
    adresa ROW(ulice     CHAR(30),
               č_domu CHAR(6),
               město CHAR(20),
               PSČ   CHAR(5)),
    datum_narození DATE);

INSERT INTO osoby
VALUES('J.Pokorný', ('Svojetická', '2401/2', Praha 10, 10000),
      '1948-04-23');

SELECT o.adresa.město
FROM osoby o
```

Typ řádku – pojmenovaný ADT

- datová struktura (+ metody)
- vhodné pro modelování entit a jejich chování

v podstatě
definice třídy

Př.: osoba, student, oddělení, ...
CREATE TYPE zaměstnanec_t AS(
č_zam INTEGER
jméno VARCHAR(20));

| film | role | herc |
|-------|-------|-------------|
| Evita | sluha | (23, Kepka) |
| ... | ... | ... |

Použití

jako typ sloupců

id připomíná
OID v OO

| id | č_zam | jméno |
|-------|-------|-------|
| 23712 | 23 | Kepka |
| ... | ... | ... |

Typ řádku – pojmenovaný ADT

```
CREATE TABLE zaměstnanci OF zaměstnanec_t
(PRIMARY KEY č_zam);
```

Co je vlastně potom výslednou tabulkou?

- unární relace, jež n-tice jsou objekty se dvěma komponentami.

Uživatelsky definované procedure a funkce

programy vyvolatelné v SQL: *procedure a funkce*

- procedury mají parametry typu IN, OUT, INOUT
- funkce mají parametry jen typu IN, vracejí hodnotu

konstrukce programů:

- hlava i tělo v SQL (buď 1 SQL příkaz nebo BEGIN...END)
- hlava v SQL, tělo externě definované

v SQL/PSM

volání programů:

- procedura: CALL jméno_procedury(p1,p2,...,pn)
- funkce: funkcionálně f(x,y)
- uložená procedura (*stored procedure*): CALL statement z klientského program, který se volá pod řízením databázového managera.

v ADT přibudou *metody*

Uživatelsky definované procedure a funkce

Př.: DB2 UDB/OSF White Box ADT

```
CREATE TYPE bod AS (
    x DOUBLE,
    y DOUBLE,
);
CREATE FUNCTION distance(p1 BOD, p2 BOD) RETURNS INTEGER
LANGUAGE SQL INLINE NOT VARIANT
RETURN sqrt((p2.y-p1.y)*(p2.y-p1.y) + (p2.x-p1.x)*(p2.x-p1.x));
SELECT Z.jméno
FROM zam Z, město M
WHERE M.název = 'Ostrava'
    AND distance(Z bydliště, M střed) < 25;
```

(Uživatelsky definované) metody

SQL:1999 přidává metody

Rozdíly metod a funkcí:

- metody jsou vždy svázány s typem, funkce nikoliv,
- daný datový typ je vždy typem prvního (nedeklarovaného) argumentu metody,
- metody jsou uloženy vždy ve stejném schématu, ve kterém je uložen typ, ke kterému mají nejblíže. Funkce nejsou omezeny na specifické schéma.
- funkce i metody mohou být polymorffické, liší se v mechanismu volby konkrétní metody v run time,
- signatura a tělo metod jsou specifikovány odděleně,
- volání metod (tečková notace + argumenty v závorkách).

ADT - plány v SQL3

```
CREATE TYPE zaměstnanec_t AS
(PUBLIC
    č_zam      INTEGER
    jméno     VARCHAR(20),
    adresa    adresa_t,
    vedoucí   zaměstnanec_t,
    datum_nástupu DATE,
PRIVATE
    základní_plat  DECIMAL(7,2),
    příplatek    DECIMAL(7,2),
PUBLIC
    FUNCTION odpr_léta(p zaměstnanec_t) RETURNS INTEGER
        <zdrojový kód pro výpočet počtu odpracovaných let>,
PUBLIC
    FUNCTION mzda(p zaměstnanec_t) RETURNS DECIMAL
        <zdrojový kód pro výpočet mzdy>);
```

ADT - skutečnost v SQL:1999

```
CREATE TYPE zaměstnanec_t AS(
    č_zam      INTEGER
    jméno     CHAR(20),
    adresa    adresa_t,
    vedoucí   zaměstnanec_t,
    datum_nástupu DATE,
    základní_plat  DECIMAL(7,2),
    příplatek    DECIMAL(7,2))
INSTANTIABLE
NOT FINAL
REF č_zam
METHOD odpr_léta() RETURNS INTEGER
METHOD mzda() RETURNS DECIMAL;
```

```
CREATE METHOD odpr_léta
FOR zaměstnanec_t
BEGIN ... END;
CREATE METHOD mzda
FOR zaměstnanec_t
BEGIN ... END;
```

ADT - skutečnost v SQL:1999

Pz.: NOT FINAL ... může mít další podtyp
REF umožňuje chápat data (řádky) v tabulkách daného typu
jako objekty. Zde: odkaz pomocí č_zam

Př.: nechť v ADT je REF zadán
REF IS SYSTEM GENERATED
V definici tabulky lze tento „identifikační“ atribut pojmenovat
REF IS PID SYSTEM GENERATED

Podtypy

```
CREATE TYPE osoba_t AS(
    jméno          CHAR(20),
    adresa         adresa_t,
NOT FINAL
CREATE TYPE zaměstnanec_t UNDER osoba_t(
    č_zam          INTEGER
    vedoucí        DATE,           zaměstnanec_t, /*zaměstnanec_t je datum_nástupu
    základní_plat  DECIMAL(7,2),
    příplatek      DECIMAL(7,2))
NOT FINAL
REF č_zam
METHOD odpr_jéta() RETURNS INTEGER
METHOD mzda() RETURNS DECIMAL;
```

Podtypy

```
CREATE TYPE úředník_t UNDER zaměstnanec_t ... podtypy
CREATE TYPE dělník_t UNDER zaměstnanec_t ...
• strukturované typy mohou být podtypem dalšího UDT
• UDT dědí strukturu (atributy) a chování (metody) ze svých
nadtypů
    • povolena je jednoduchá dědičnost (vícenásobná odložena,
    nevyškytuje se ani v SQL:2003)
• v SQL:1999 strukturované typy musí být NOT FINAL a
odlišující typy musí být FINAL (v SQL:2003 uvolněno)
• substituovatelnost: na místě daného typu může být
hodnota podtypu
```

Podtabulky

- aparát závislý na aparátu typů *zaměstnanec_t*
musí být podtypem
CREATE TABLE osoby OF osoby_t
CREATE TABLE zaměstnanci OF zaměstnanec_t
UNDER osoby;
- dědí sloupce, IO, triggery, ... dané nadtabulky

Podtabulky

*zaměstnanec_t
musí být podtypem
osoby_t*

- Požadavky konsistence pro podtabulky a nadtabulky
 - každá n-tice v nadtabulce (např. osoby) může korespondovat nejvýše k jedné n-tici v podtabulkách (např. zaměstnanci a OON)
 - tj. každá entita musí být nespecifitčejší typ
- Výběr omezený na tabulku X pomocí
FROM ONLY (X)
– jinak i z podtabulek X.

Přístup k hodnotám atributů

Každý atribut má (automaticky) metody pro výběr hodnot a aktualizace

- výběr hodnot
SELECT z.jméno()
FROM zaměstnanci z
- aktualizace ve 3 krocích (*generátor* a *mutátor*)
SET novýZam = zaměstnanec_t()
novýZam.č Zam('7897890')
novýZam.jméno('Jarda')
INSERT INTO zaměstnanci(novýZam)

Reference a dereference

A co typ REF?

jeho hodnoty lze „vidět“

- generované uživatelem (REF USING <předdefinovaný typ>)
- generované systémem (REF IS SYSTEM GENERATED) – implicitně (viz OID v OO systémech)
- odvozené (REF<seznam atributů>)

```
CREATE TYPE účet_t AS  
    č účtu INT,  
    klient REF(zákazník_t),  
    typ CHAR(1),  
    otevřen DATE,  
    úrok DOUBLE PRECISION,  
    zůstatek DOUBLE PRECISION,  
    );  
FINAL REF IS SYSTEM GENERATED;  
CREATE TABLE účty OF účet_t  
(PRIMARY KEY č účtu);
```

reference

tabulka účty má zvláštní
atribut podobný oid
tzv. samoodkazující sloupec

Reference a dereference

Co se děje, je-li odstraněn odkazovaný objekt:

- ▶ Nic – implicitně REFERENCES ARE NOT CHECKED
- ▶ Možnost akce, je-li REFERENCES ARE CHECKED ON DELETE
(pak SET DEFAULT, SET NULL, CASCADE, NO ACTION, RESTRICT)

Dereference

- ▶ lze dělat pouze tehdy, je-li definováno umístění objektů typu REF
to jedna tabulka) (v SQL:1999 je

```
CREATE TABLE zákazníci OF zákazník_t;
CREATE TABLE účty OF účet_t
(PRIMARY KEY číslo_účtu,
 klient WITH OPTIONS SCOPE zákazníci
);
```

alokace

dereference,
cesta

Pz.: připomíná referenční integritu

```
SELECT u.klient->jméno
FROM účty u
WHERE u.klient->adresa.město = "Suchdol" AND u.zůstatek > 100000;
```

Reference a dereference

- dereference cestou a/nebo funkcí DEREF

srovnej

```
SELECT u.otevřen, u.klient
FROM účty u;
```

a

```
SELECT u.otevřen, DEREF(u.klient)
FROM účty u;
```

DEREF vraci
n-tici

Reference a dereference

```
INSERT INTO účty
VALUES('376291', (SELECT REF(z) FROM ZÁKAZNÍCI z WHERE JMÉNO =
'Josef Novák'), 'S', ...);
```

Reference a dereference

- výhody používání REF:
 - sdílení objektů
 - nejsou zbytečně kopírována data
 - změna se provádí na jednom místě
- odkaz na metodu:

```
SELECT u.klient() -> jméno
FROM účty u
WHERE u.klient() ->mzda() > 10000;
Pz.: metody bez parametrů nevyžadují ()
```

Za SQL:1999, 2003

```
CREATE TABLE zaměstnanci
(id INTEGER PRIMARY KEY,
jméno VARCHAR(30),
adresa ROW(
    uliceCHAR(30),
    číslo_d CHAR(6),
    město CHAR(20),
    PSČ CHAR(5)),
projekty SET (INTEGER),
děti LIST(osoba),
odměny MULTISET (MONEY) kolekce
```

je v SQL:2003

Kolekce v SQL:2003

- ARRAY a MULTISET

Omezení: typ prvků v kolekci nemůže být REF nebo ADT obsahující REF.

- Operace a predikáty

| | |
|---|----------------|
| $nt1$ MULTISET EXCEPT [DISTINCT] $nt2$ | $nt1 - nt2$ |
| $nt1$ MULTISET INTERSECT [DISTINCT] $nt2$ | $nt1 \cap nt2$ |
| $nt1$ MULTISET UNION [DISTINCT] $nt2$ | $nt1 \cup nt2$ |
| CARDINALITY(nt) | $ nt $ |
| nt IS [NOT] EMPTY | |
| nt IS [NOT] A SET | |

Kolekce v SQL:2003

| | |
|--|--|
| $SET(nt)$ | odstraň duplicity z nt |
| $nt1 = nt2$ | rovnost multimnožin |
| $nt1 \text{ IN } (nt2, nt3, \dots)$ | být v seznamu multimnožin |
| $nt1 \text{ [NOT] SUBMULTISET OF } nt2$ | porovnání multimnožin |
| $r \text{ [NOT] MEMBER OF } nt$ | $r \in nt?$ |
| $\text{CAST}(\text{COLLECT}(col))$ | hnízděná tabulka založená na col |
| $\text{POWERMULTISET}(nt)$ | množina všech neprázdných podmnožin nt |
| $\text{POWERMULTISET_BY_CARDINALITY}(nt, c)$ | množina všech neprázdných pomnožin nt s kardinalitou c |

O-R v komerčních produktech

- INFORMIX: kolekce set, multiset, list (bez omezení délky)
- Oracle od verze 8i (r. 1999):
 - místo ADT -- typ objektů
 - notace: `CREATE TYPE ... AS OBJECT(...);`
 - kolekce
 - **VARRAY** (ekvivalentní **ARRAY** z SQL:1999), avšak neumožňuje **DELETE** pro daný prvek pole
 - **NESTED TABLE** (neuspořádaná neohraničená kolekce prvků)
Pz.: hnizdění do 1 úrovni, obecněji v Oracle 9i

```
CREATE TYPE Kde_všude AS VARRAY(4) OF Adresa
```

O-R v komerčních produktech

- viditelnost id

```
SELECT REF(o) INTO refToOsoba
FROM osoby AS o
WHERE o.jmeno = 'Novák, J.'
```

O-R v komerčních produktech

```
CREATE TYPE Auta AS TABLE OF Auto_t
CREATE TABLE FIRMY (
    vozový_park Auta
    ...)
NESTED TABLE vozový_park STORE AS vozy;
Pz.: lze zadat, kde mají být „podtabulky“ Auta uloženy
SELECT *
FROM FIRMY AS f, f.vozový_park AS vp
WHERE 'Buick' IN (SELECT vp.značka FROM vp);
```

O-R v komerčních produktech

Dotazy na hnizděnou tabulku:

- pomocí THE
- lze s ní zacházet jako s jinými relacemi

```
SELECT vp.SPZ
FROM THE (
    SELECT vozový_park FROM FIRMY
    WHERE jméno_f= 'Komix')
) vp
WHERE vp.ZNAČKA='Buick';

D.: Najdi SPZ všech Buicků firmy Komix.
```

O-R v komerčních produktech

Metody v ORACLE

- Specifikace v CREATE TYPE pomocí MEMBER FUNCTION, MEMBER PROCEDURE
- Tělo v příkazu CREATE TYPE BODY
- Přístup

```
SELECT u.klient.jméno
FROM účty u
WHERE u.klient.mzda > 10000;
```

Typ řádku – pojmenovaný

- na rozdíl od ADT není zapouzdřený.

```
CREATE ROW TYPE účet_t
    č_účtu INT,
    klient REF(zákazník_t),
    typ CHAR(1),
    otevřen DATE,
    úrok DOUBLE PRECISION,
    zůstatek DOUBLE PRECISION,
);
CREATE TABLE účty OF účet_t
    (PRIMARY KEY č_účtu);
• příkaz není součástí standardu SQL! Lze ho nalézt např. v DB/2.
```

Problémy s OO v SQL

- tabulky jsou jediné pojmenované entity
- typ REF aplikovatelný pouze na objekty dané řádkem
- UDT je prvním krokem k OO
 - umožnit perzistence, musí být objekt v tabulce
 - individuální instanci nelze přiřadit jméno
 - nelze použít dotazy na všechny instance ADT

Návrh OR DB: Transformace E-R → OR

- 1. Fáze: typy
 - typy entit → strukturované typy
 - složené atributy → pojmenované typy řádků, nepojmenované typy řádků v strukturovaném typu, také strukturovaný typ je možný
 - vícehodnotové atributy → pole typovaných hodnot (odhad max je důležitý)
 - odvozené atributy → přidej metodu do definice strukturovaného typu

odstraněno v SQL:2003 pomocí SET

Návrh OR DB: Transformace E-R → OR

- typy vztahů – dvousměrné nebo jednosměrné
 - N:1 → pomocí REF + pole obsahující hodnoty primárního klíče (jestliže dvousměrné)
 - M:N → pomocí dvou polí obsahujících hodnoty primárních klíčů (jestliže dvousměrné).
 - ISA hierarchie → hierarchie typů
- 2. fáze: typované tabulky

| Kritérium | RDBMS | ORDBMS | OODMS |
|--|---|---|---|
| Definovaný standard | SQL (ANSI X3H2) | SQL/3 (in process) | OODM-V2.0 |
| Podpora pro objektově orientované programování | Spatřit; programátoři stráví 25% času kódování mapováním objektového programu do databáze | Omezená hraniv na nové datové typy První a rozšíření | |
| Jednoduchost používání | Struktury tabulek je jednoduché použití mnoha dostupných nástrojů pro koncové uživatele | Totéž co RDBMS, navíc s nějakými matoucími rozdíly | OK pro programátory; nějaký SQL přístup pro koncové uživatele |
| Jednoduchost vývoje | Poskytuje nezávislost dat z aplikace, dobrá pro jednoduché vztahy | Poskytuje nezávislost dat z aplikace, dobrá pro jednoduché vztahy | Objekty jsou přesnou cestou k modelu; může využít širokým rozsahem typů a vztahů |
| Rozšířitelnost a obsah | Žádná | Omezená hraniv na nové datové typy | Může pracovat s libovolnou složitostí; lze využít různé funkce, může mít metody a jakékoli struktury |
| Složité datové vztahy | Pro model obtížné | Pro model obtížné | Může pracovat s libovolnou složitostí; uživatel může přist metody a jakékoli struktury |
| Výkon versus spolupracovatelnost | Úroveň bezpečnosti se mění s dodavatelem, je třeba vzájemně porovnat; doslnutí oběho významu rozsáhlé testování | Úroveň bezpečnosti se mění s dodavatelem, je třeba vzájemně porovnat; doslnutí oběho významu rozsáhlé testování | Úroveň bezpečnosti se mění s dodavatelem; většina OODMS dodávají programátorům rozšíření funkcií DBMS definovaném nových tříd |
| Distribuce, replikace, a spojené databáze | Rozšířila | Rozšířila | Poda dodavatele; pár jich poskytuje rozšířilou podporu |
| Výslednost produktu | Velmi vysoké | Nerazí; rozšíření jsou nová, stále se definiuj a jsou relativně neprozehodnitelné | Relativně vysoké |
| Podpora pro lidi a univerzálnost SQL | Sírká podpora nástrojů a trénovaných vývojářů | Může využívat výhod nástrojů RDBMS a vývojářů | Vybaveno SQL, ale určeno pro objektově orientované programování |
| Softwareové ekosystémy | Poskytováno hlavními RDBMS společnostmi | Poskytováno hlavními RDBMS společnostmi | ODBMS-výrobci založili emulovat RDBMS-výrobce, aby žádny nenabízí velký obchod jiným ISV |
| Zivotaschopnost výrobce | Očekávaná pro hlavní zábehnuté RDBMS výrobce | Očekávaná pro hlavní RDBMS výrobce; UniSQL busuje | Menší než se čekalo; stále se očekává zmenšování |

Závěr

- současně implementace ORSŘBD - určité (snad oprávněné) námitky
 - paralela:
 - výtky OOSŘBD - málo databázové
 - výtky ORSŘBD - málo objektové
- Pomalu přibývají výkonné vývojové prostředky, nové metodologie
- největší problém a současně výhoda: univerzálnost
- trh?

Distribuované databázové systémy

Literatura

- Tanenbaum, van Steen: Distributed Systems - Principles and Paradigms, *2nd ed*
- Chow, Johnson: Distributed Operating Systems & Algorithms
- Kshemkalyani, Singhal: Distributed Computing - Principles, Algorithms and Systems
- Coulouris, Dollimore: Distributed Systems - Concepts and Design, *4th ed*
- Singhal, Shivaratri: Advanced Concepts in Operating Systems
- Sinha: Distributed Operating Systems - Concepts and Design
- Santoro: Design and Analysis of Distributed Algorithms
- Mullender: Distributed Systems, *2nd ed*

* slajdy nejsou u skripta

"Definice"

Distribuovaný systém je takový systém propojení množiny nezávislých uzlů, který poskytuje uživateli dojem jednotného systému

- **Jednotný obraz systému, virtuální uniprocessor**
 - Hardwarová nezávislost
 - ▶ jsou tvorenými nezávislými "počítacími" vlastním procesorem a pamětí
 - ▶ jedinou možností komunikace přes komunikační (síťové) rozhraní
 - Dojem jednotného systému
 - ▶ uživatel (člověk i software) komunikuje se systémem jako s celkem
 - ▶ nemusí se starat o počet uzlů, topologii, komunikaci apod.
- **distribuovaný systém** - celá škála různých řešení
 - ▶ multiprocesory na jedné základové desce
 - ▶ celosvětový systém pro miliony uživatelů
- Uvažované terminy v dalším výkladu: uzel = ('běžný') počítač, softwarový systém
 - ▶ principy, algoritmy

Distribuované databázové systémy

= databázové systémy + počítačové síť

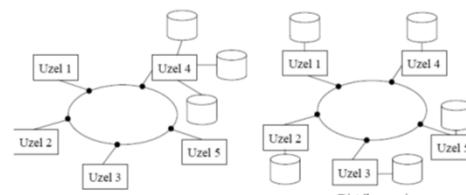
DBS = integrace
PS = decentralizace

DDBS= integrace dat bez centralizace (je variantou DVS)

Požadavky na moderní databáze

- **cloud, distribuované databáze** (decentralizace úložiště dat, úmyslná redundance pro odolnost proti vypadámkům a rychlosť, velké objemy dat a velké množství operací /big data/, atd.)
- **problematické datové typy** (údaje klíč-hodnota, objekty, nestrukturované dokumenty, RDF grafy, atd.)
- **iterativní vývoj** (časté změny schématu databáze nebo dokonce žádné schéma, různé/nejasné způsoby použití databáze, atd.)
- **vysoké požadavky na škálovatelnost** (mobilní zařízení jako klienti i úložiště/poskytovatele dat, nerovnoměrné rozložení zátěže prostorově i časově, specifické požadavky na dostupnost, předem neznámé dotazy - nelze optimalizovat indexy, atd.)

Koncepce distribuovaného DBS



- množina uzlů propojených komunikační sítí
 1. každý uzel je sám o sobě DBS
 2. z každého uzlu lze transparentně zpřístupnit data kdekoli v síti

Teorém CAP u sdílených/distribuovaných systémů

- **Consistency**
 - každý uzel/klient vidí ve stejný čas stejná data, (data konzistentní nezávisle na běžících operacích či jejich umístění)
- **Availability**
 - každý požadavek obslužen, úspěšně nebo neúspěšně, (nepřetržitý provoz, vždy možnost zapsat a číst data)
- **Partition Tolerance**
 - funkční navzdory chybám sítě nebo výpadkům uzlů. (možnost výpadku části infrastruktury, např. odstávka pro údržbu)

Teorém:

U sdílených systémů je možné uspokojit maximálně 2 ze 3 požadavků.

Eric Brewer (+ N. Lynch), 2000

CA / CP / AP

Consistence + Availability

= 2fázový commit, protokoly pro (in)validaci cache
(např. Cluster databases, LDAP, xFS file system)

Consistency + Partition tolerance

= agresivní zamykání, ustojí malé výpadky
(např. distribuované databáze a zamykání, protokol pro většinovou shodu)

Availability + Partition tolerance

= střídání uzlů, řešení konfliktů, optimistická strategie
(Coda, Web caching[!], DNS)

Brewer, Eric A.: Towards Robust Distributed Systems. Portland, Oregon, July 2000.
Keynote at the ACM Symposium on Principles of Distributed Computing (PODC) on 2000-07-19.

Požadavky na „ideální“ DDBS

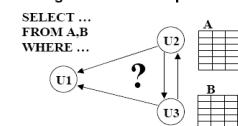
- **uživateli se jeví distribuovaný systém přesně jako nedistribuovaný**
- lokální autonomie
- nespolehlání se na centrální uzel
- nepřetržitá činnost
- nezávislost na umístění dat (transparentnost)
- nezávislost na fragmentaci
- nezávislost na replikaci dat
- distribuované zpracování dotazu
- distribuovaná správa transakcí
- nezávislost na HW, OS, SŘBD, komunikační síti

Některé problémy

- přenos v síti je pomalý \Rightarrow minimalizace počtu a objemu zpráv

- Zpracování dotazů

- globální/lokální optimalizace

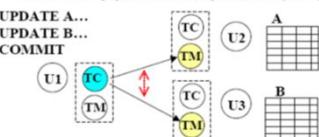


- správa systémového katalogu
- šíření aktualizace při replikaci

- schéma s primární kopí, ...

Některé problémy

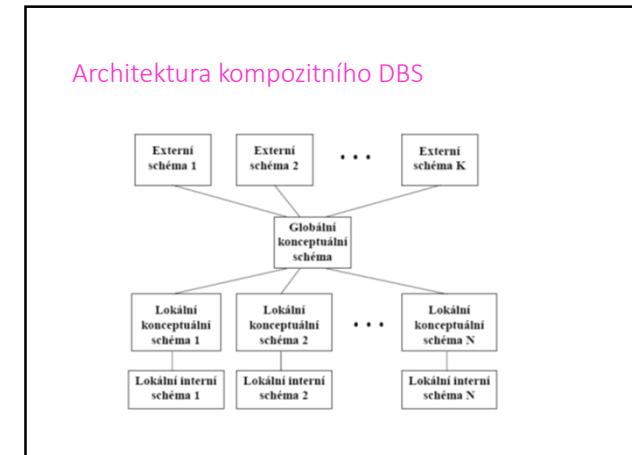
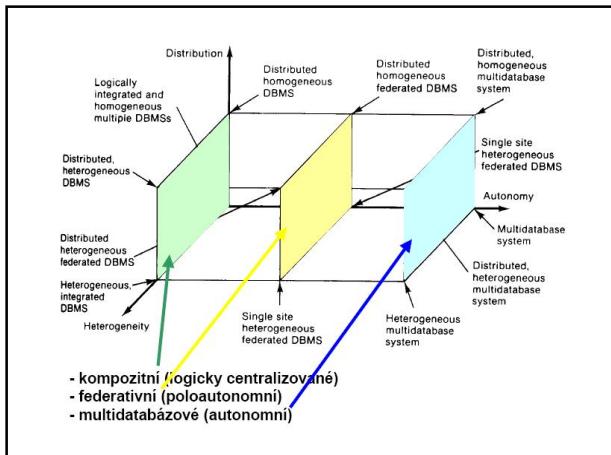
- zotavení po chybách a poruchách
 - dvoufázový potvrzovací protokol (2PC)



- souběžnost
 - schéma s primární kopí
 - nebezpečí globálního uváznutí

Taxonomie

- autonomie: těsná integrace, poloautonomie, úplná izolace
- distribuce: jeden uzel, distribuovaný
- heterogenita: homogenní, heterogenní



Distribuované databáze - Distribuovaný výpočetní systém

Distribuovaný výpočetní systém má vlastnosti:

- Je rozložen do uzelů sítě spojených komunikačními kanály
- V uzlech je možné autonomně ukládat a zpracovávat data
- Prostředky v uzlech mohou být nehomogenní
- Uživatel nemusí vědět o existenci ostatních uzelů (tzv. transparentnost)

Distribuované databáze - důvody proč

- lokální autonomie (odpovídají struktuře decentralizovaných organizací. Data uložena v místě nejčastějšího využití a zpracování - zlevnění provozu). V centralizované DB je nutné připojovat se ke vzdálené databázi = přidavná režie, cena komunikace, zatížení sítě
- zvýšení výkonu (inherentní paralelismus rozdelením zátěže na více počítačů)

Distribuované databáze - důvody proč

- spolehlivost (replikace dat, degradace služeb při výpadku uzlu, přesunutí výpočtu na jiný uzel)
- lepší rozšiřitelnost konfigurace (přidání procesorů, uzelů)
- větší schopnost sdílet informace integrací podnikových zdrojů
- uzel mohou zachovat autonomní zpracování a současně virtuálně zabezpečovat globální zpracování
- agregace informací (z více bází dat lze získat informace nového typu)

Distribuované databáze - důvody proti

- složitost (distribuce databáze, distrib. zpracování dotazu a jeho optimalizace, složité globální transakční zpracování, distribuce katalogu, paralelismus a uvíznutí, případná integrace heterogenních dat do odpovídajících schémát, složité zotavování z chyb)
- cena (komunikace je navíc)
- bezpečnost

Distribuované databáze - důvody proti

- obtížný přechod (neexistence spolehlivého automatického konverzního prostředku z centralizovaných DB na DDB)

Výhody – nevýhody distribuovaných systémů

Shrnutí

- **Výhody**
 - Efektivita zpracování (data mohou být uložena blízko místa nejčastějšího používání)
 - Zvýšená dostupnost
 - Vyšší výkonnost
 - Rozšířitelnost
- **Nevýhody**
 - Složitost
 - Distribuce řízení
 - Bezpečnost

Požadavky na DDBS

- Transparentnost distribuce (míra viditelnosti distribuce dat pro uživatele)
- Autonomie (distribuce řízení)
- Heterogennost
- Výkon (vysoká průchodnost krátká odezva)

Požadavky DSŘBD

Dle formulace (autor Date):

- lokální autonomie – každé místo má lokální SŘBD
- vše je distribuované – ve všech službách se nespoléhá na žádné centrální místo
- kontinuálnost – akce v jednom místě (např. odstranění tabulky) by neměla příliš narušovat provoz DDBS jako celku
- nezávislost na místě – uživatel nemusí vědět, kde jsou uložena potřebná data
- nezávislost na fragmentaci – nemusí vědět, kde jsou fragmenty
- nezávislost na replikaci

Požadavky DSŘBD

- možnost distribuovaného zpracování dotazu – nemělo by být nutné přesouvat data ke zpracování dotazu do jednoho místa
- možnost distribuovaného zpracování transakcí – může dojít ke konfliktu s 1. Pro zajištění korektnosti se používá 2 fázový potvrzovací protokol
- nezávislost na hardware
- nezávislost na OS
- nezávislost na síti
- nezávislost na DBMS

Implementace realizují jen část ideí Date. (Ingres Star, IBM DRDA, Oracle 7 a vyšší)

Formy transparentnosti

- **Datová nezávislost** = imunita uživatelské aplikace ke změnám v definici a organizaci dat. Je požadavkem i pro centralizované DB
 - **Logická nezávislost** (logická struktura databáze)
 - **Fyzická nezávislost** (konkrétní způsob uložení dat)
- **Síťová transparentnost** = ukrytí síťových detailů = uživatel neví o síti

Příklad: Oracle – firemní síťové rozhraní Net

Definice spojení z jedné databáze na druhou (citací domén):
 CREATE DATABASE LINK prodeje.cz.europe

Po vytvoření linku mají uživatelé lokální databáze přístup ke vzdálené databázi

Příklad transakce:
 INSERT INTO zamest@prodeje.cz.europe;
 SELECT FROM zamest@prodeje.cz.europe;
 COMMIT;

Příklad: Oracle – firemní síťové rozhraní Net

Lokální transparentnost je zajišťována pomocí

CREATE SYNONYM jméno_syn FOR jméno

Např.

CREATE SYNONYM z FOR
 zamest@prodeje.cz.europe

Formy transparentnosti

- **Replikační transparentnost** = neobtěžovat uživatele skutečnosti, že pracuje s daty existujícími ve více kopíech = uživatel neví o replikách
- **Fragmentační transparentnost** = uživatelův dotaz je specifikován na celou relaci, ale musí být vykonán na jejím fragmentu = uživatel neví o fragmentech

Rozšíření ANSI/SPARC referenčního modelu

(Architektura schémat v DDBS)

1. **Externí schéma** (CREATE VIEW)
2. **Globální konceptuální schéma**
 (log.struktura ve všech uzlech = „model podniku“ = globální katalog)
3. **Lokální konceptuální schéma**
 (CREATE TABLE, zpracovává replikace a fragmenty)
4. **Lokální interní schéma**
 (fyzická organizace dat, např. jazyk C)

Taxonomie DDBS

Důsledky ANSI/SPARC architektury - hlediska (vzájemně ortogonální):

- autonomie lokálních systémů
- distribuce dat
- heterogenity systémů

Taxonomie DDBS

ad 1.

- **těsně integrované** (uživatel vidí data centralizovaná v jediné databázi). DDB je vybudována nad lokálnimi DB. Každé místo má úplnou znalost o datech v celém DDBS a může zpracovávat požadavky používající data z různých míst.
- **semiautonomní** systémy (lokální DBMS pracují nezávisle a sdílejí svoje lokální data v celé federaci). Jen část jejich dat je sdílena.
- **zcela autonomní** = izolované. Lokální DBMS pracují nezávisle a neví o ostatních DBMS. Pro vzájemnou komunikaci potřebují softwarovou vrstvu pracující nad jednotlivými DBMS

Taxonomie DDBS

- Aut Distr Heter
-
- T N N Logicky integrované a homogenní vícenásobné DBMS
- T A N Distribuované homogenní DBMS
- T N A Heterogení integrované DBMS
- T A A Distribuované heterogení DBMS
- P N N Homogenní federativní DBMS v jednom místě
- P A N Distribuované homogenní federativní DBMS
- P N A Heterogení federativní DBMS v jednom místě
- P A A Distribuované heterogení federativní DBMS
- Ú N N Multidatabázové systémy
- Ú A N Distribuované homogenní multidatabázové DBMS
- Ú N A Heterogení multidatabázové systémy
- Ú A A Distribuované heterogení multidatabázové DBMS

Distribuované zpracování transakcí

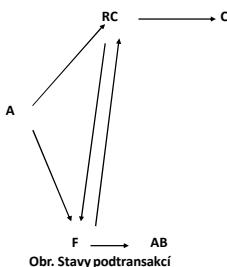
Transakce = posloupnost operací, které se provedou buď všechny nebo žádná –zajišťuje je transakční monitor (Atomicity, Konsistence, Isolation, Durability)

Transakce v DDBS rozdeleny na podtransakce (provedeny na různých místech).

Transakční zpracování v jednotlivých uzlech nestačí.

Distribuované zpracování transakcí

- ▶ Podtransakce má stavy:
- ▶ A (Active)
- ▶ C (Committed)
- ▶ AB (Aborted)
- ▶ RC (Ready to Commit)
- ▶ F (Failed)
- ▶ Transakční provoz zajišťují zprávy:
- ▶ PREPARE to COMMIT
- ▶ READY to COMMIT
- ▶ COMMIT
- ▶ COMMIT SUCCESFULL
- ▶ ABORT
- ▶ ABORT COMPLETED
- ▶ Provedení READ/WRITE na datech



Dvoufázový potvrzovací protokol

- F1. Koordinátor zašle všem místům (kde se provádí podtransakce dané transakce) zprávu s požadavkem na připravenost (PREPARE to COMMIT). Hláší-li některé místo nepřipravenost, provede koordinátor ROLLBACK transakce ve svém místě a pošle zprávu všem místům na ABORT podtransakcí.
- F2. Ohláší-li všechna místa úspěšnost (READY to COMMIT), tak koordinátor potvrdí transakci a její lokální části ve svém místě. Pak odešle zprávy s požadavkem na potvrzení (COMMIT) všem participantům. Tyto zprávy budou dříve nebo později doručeny.

Jiný způsob

- Pravidlo o potvrzení lze formulovat také:
- Koordinátor zruší transakci právě když alespoň jeden participant hlásí zrušení transakce
 - Koordinátor potvrdí transakci právě když všichni participanti hlásí, že jsou připraveni ji potvrdit
- Poznámka: **Nevrací-li se odpověď, je po uplynutí time-out transakce zrušena.**

Postup návrhu DDBS

- Top down:**
- Návrh GCS
 - Návrh fragmentace
 - Návrh alokace
 - Návrh fyzické struktury

Postup návrhu DDBS

Bottom up:

- Výběr společného DB modelu pro popis globálního schéma
- Překlad každého lokálního schéma do společného DB modelu
- Integrace lokálních schémat do společného globálního schéma

Distribuce globálních relací

- s replikami
- ve fragmentech
- s replikami i ve fragmentech

Fragmentace

1. Horizontální

$$F_i = \text{Selection}_{P_i} R$$

$$R = \bigcup F_i$$

2. Odvozená horizontální

Založena na horizontální fragmentaci jiné relace
 Používá polospojení = $R[t_1 \theta t_2] S = (R[t_1 \theta t_2] S) [Atr(R)]$

3. Vertikální

$$F_i = R(A_i)$$

$$R = F_1 \text{ join } F_2 \dots \text{ join } F_n$$

4. Smíšená

Příklad

| Př. relace KNIHY - horizontální fragmentace | | | | |
|---|-------|-------|-----------|-----------|
| F2000 | ISBN | INV_C | CENAR_VYD | VYDAVATEL |
| ----- | ----- | ----- | ----- | ----- |
| 1590 | 120 | 590 | 2000 | KOP |
| 1910 | 121 | 720 | 2000 | Springer |
| F2002 ISBN INV_C CENAR_VYD VYDAVATEL | | | | |
| ----- | ----- | ----- | ----- | ----- |
| 0235 | 122 | 900 | 2002 | AP |
| 0235 | 123 | 900 | 2002 | AP |
| 1488 | 124 | 899 | 2002 | Alfa |

Příklad odvozené horizontální fragmentace

Př. - odvozená horizontální fragmentace relace KNIHY
 relace VYDAVATELÉ(VYDAVATEL, ZEMĚ, OBRAT\$)

VYDAVATELÉ1 = Selection (OBRAT\$ > 100mil) VYDAVATELÉ
 VYDAVATELÉ2 = Selection (OBRAT\$ <= 100mil)VYDAVATELÉ

K1 =

KNIHY Semijoin (VYDAVATEL=VYDAVATEL) VYDAVATELÉ1

K2 =

KNIHY Semijoin (VYDAVATEL=VYDAVATEL) VYDAVATELÉ2

Odvozená horizontální fragmentace

např.

VYDAVATELÉ:

| VYDAVATEL | ZEMĚ | OBRAT\$ |
|-----------|------|---------|
| AP | USA | 200 |
| Alfa | SK | 20 |
| BC | NL | 120 |
| KOP | CZ | 15 |
| Springer | D | 500 |

Odvozená horizontální fragmentace

VYDAVATELÉ1:

| | | |
|----------|-----|-----|
| AP | USA | 200 |
| BC | NL | 120 |
| Springer | D | 500 |

VYDAVATELÉ2:

| | | |
|------|----|----|
| Alfa | SK | 20 |
| KOP | CZ | 15 |

Odvozená horizontální fragmentace

| K1: | ISBN | INV_C | CENAR_VYD | VYDAVATEL |
|-----|------|-------|-----------|-----------|
| | 1910 | 121 | 720 | 2000 |
| | 0235 | 122 | 900 | 2002 |
| | 0235 | 123 | 900 | 2002 |

| K2: | ISBN | INV_C | CENA_R_VYD | VYDAVATEL |
|-----|------|-------|------------|-----------|
| | 1590 | 120 | 590 | 2000 |
| | 1488 | 124 | 899 | 2002 |

Příklad vertikální fragmentace

Př. - vertikální fragmentace relace KNIHY
předpokládané funkční závislosti

INV_C → { ISBN, CENA }
ISBN → { R_VYD, VYDAVATEL }

Bezztrátová vertikální fragmentace

| FR1 (ISBN, R_VYD, VYDAVATEL) FR2 (INV_C, ISBN, CENA) | | | | | |
|--|-------|-----------|-------|------|------|
| ISBN | R_VYD | VYDAVATEL | INV_C | ISBN | CENA |
| 1590 | 2000 | KOP | 120 | 1590 | 590 |
| 1910 | 2000 | Springer | 121 | 1910 | 720 |
| 0235 | 2002 | AP | 122 | 0235 | 900 |
| 1488 | 2002 | Alfa | 123 | 0235 | 900 |
| | | | 124 | 1488 | 899 |

Smíšená fragmentace

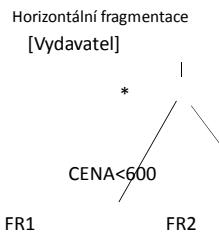
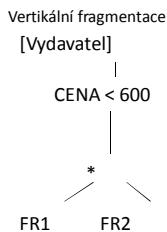
| Př. - smíšená fragmentace | | FR1_2000 | | FR1_2002 | |
|---------------------------|-------|-----------|------|----------|--------|
| ISBN | R_VYD | VYDAVATEL | ISBN | R_VYD | VYDAVA |
| 1590 | 2000 | KOP | 0235 | 2002 | AP |
| 1910 | 2000 | Springer | 1488 | 2002 | Alfa |

+ FR2

Zpracování dotazu v distribuovaném prostředí

- Zjistit jména vydavatelů knih levnějších než 600
- v centralizované DB:
KNIHY (CENA < 600) [Vydavatel]
 - Projection Vydavatel (Selection CENA<600 KNIHY)
 - Select Vydavatel from KNIHY where CENA<600
 - v horiz. fragm. DB: (F2000 ∪ F2002) (CENA<600) [Vydavatel]
 - ve vertik. fragm. DB: (FR1 * FR2) (CENA<600) [Vydavatel]

Zpracování dotazu v distribuovaném prostředí – strom dotazu



Strategie zpracování přenosu dat

Problém přenosu dat mezi uzly lze redukovat na problém řešení spojení relaci umístěných v různých uzlech sítě

- **Jednoduché zpracování spojení**
-spojení se zpracovává v jednom uzlu sítě
- **Paralelní zpracování spojení**
-ve více uzlech se současně zpracovávají podvýrazy dotazu obsahující spojení
- **Zpracování pomocí polospojení**
-mezi uzly se přenáší pouze ty n-tice, které budou operací spojení skutečně spojeny

Př.DB s relacemi

| | | |
|---|----------|----------|
| KNIHA | EXEMPLÁŘ | VÝPUJČKA |
| v uzlech | | |
| S1 | S2 | S3 |
| KNIHA(ISBN, AUTOR, TITUL) | | |
| EXEMPLÁŘ(ISBN, INV_C, R_VYD, CENA, VYDAVATEL) | | |
| VÝPUJČKA(INV_C, C_CT, D_VRAC) | | |

Strategie 1:

předp. v dotazu potřebu spojení KNIHA * EXEMPLAR * VÝPUJČKA v S3.
Nechť je
 $|EXEMPLAR| \geq |KNIHA| \geq |VÝPUJČKA|$
S2: $T1 \leftarrow VÝPUJČKA$
S2: $T2 := T1 * EXEMPLAR$
S1: $T3 \leftarrow T2$
S1: $T4 := T3 * KNIHA$
S3: $T5 \leftarrow T4$

Strategie 2:

```

select TITUL from KNIHA
where ISBN in (select ISBN from EXEMPLAR
  where INV_C in (select INV_C
    from VÝPUJČKA
    where
    D_VRAC<1.6.))
  
```

Strategie 2:

```

select TITUL from KNIHA
where ISBN in (select ISBN from EXEMPLAR
  where INV_C in (select INV_C
    from VÝPUJČKA
    where
    D_VRAC<1.6.))
  
```

Strategie 2:

```
(VYPUJCKA(D_VRAC<1.6.)([INV_C])
*
(EXEMPLAR*KNIHA([INV_C,TITUL])
[TITUL]

projectionT T I U L
(projection INV.C (selection D.VRAC <1.6. VYPUJCKA)
join
(projection T I T U L , INV.C (EXEMPLAR join KNIHA))
}
```

Strategie 3:

| | |
|----------------------------|--|
| polospojení | $(R [A = B] S) [R]$ |
| | $(R < A = B] S)$ |
| | R Semijoin F S = Projection |
| Atrib (R) (R join F S) | |
| platí | $(R < A = B] S) \neq (S < B = A] R)$ |

Strategie 3:

Chceme vyčíslet EXEMPLAR * KNIHA výsledek
 S2 S1 S2
 Postup: S2: T1 := EXEMPLAR [ISBN]
 S1: T2 \leftarrow T1
 S1: T3 := KNIHA * T2
 S2: T4 \leftarrow T3
 S2: T5 := EXEMPLAR * T4
 v S1 vyhodnoceno KNIHA < ISBN=ISBN > EXEMPLAR
 převedení EXEMPLAR * KNIHA
 na (KNIHA < ISBN=ISBN > EXEMPLAR) * EXEMPLAR

Konkrétní příklad

| KNIHA: | AUTOR | TITUL |
|--------|---------------|---------------------------|
| ISBN | | |
| 212 | Yannakoudatis | Architectural Logic of DB |
| 621 | Ullman | Principles of DB |
| 145 | Elmasri | Fundamentals of DBS |
| 420 | Date | An Introduction to DBS |

| EXEMPLAR: | ISBN | INV_C | R_VYD | CENA | VYDAVATEL |
|-----------|------|-------|-------|----------|-----------|
| 212 | 150 | 2000 | 1500 | Springer | |
| 212 | 151 | 2000 | 1500 | Springer | |
| 145 | 415 | 2002 | 2500 | BC | |
| 145 | 451 | 2002 | 2500 | BC | |

Konkrétní příklad

| | | | |
|---------------------------------|--------|-------|-----------|
| T1 = Projection ISBN EXEMPLAR = | ISBN | | |
| 212 | | | |
| 145 | | | |
| T2 ← T1 | | | |
| T3 = KNIHA * T2 = | ISBN | AUTOR | TITUL |
| | Yannak | | Architect |
| | Elmasr | i | Fundamen |
| T4 ← T3 | | | |

Konkrétní příklad

T5 = EXEMPLAR * T4 =

| ISBN | INV_C | ROK_V | CENA | VYDAVATEL | AUTOR | TITUL |
|------|-------|-------|------|-----------|---------|---------------|
| 212 | 150 | 2000 | 1500 | Springer | Yannak | Architectural |
| 212 | 151 | 2000 | 1500 | Springer | Yannak | Architectural |
| 145 | 415 | 2002 | 2500 | BC | Elmasri | Fundamentals |
| 145 | 451 | 2002 | 2500 | BC | Elmasri | Fundamentals |

Postup při návrhu fragmentů

Respektovat hlediska:

1. rozdělit relace do lokálních serverů tak, aby aplikace zatěžovaly servery stejněméně. (Musí být známa informace o předpokládaných přistupech k relacím)
2. lokality zpracování (maximalizovat lokální)

Postup při návrhu fragmentů

Respektovat hlediska:

3. přístupnosti a spolehlivosti (např. replikací zlepšíme spolehlivost a read-only dostupnost),
4. maximalizovat stupeň paralelismu zpracování dotazu
5. dostupnosti a ceny paměti v jednotlivých uzlech

Problémy

Nelze vyhovět všem (složité optimalizace)

Nutno vyřešit dva problémy:

- **co má fragment obsahovat**
- **kam fragment umístit**

Návrh fragmentů při primární horizontální fragmentaci

Úkol: Rozhodnout, které řádky tabulky umístit do fragmentu

Použitá informace: četnost selekčních predikátů aplikovaných na relaci jednotlivými aplikacemi

Selekční predikáty jsou tvořeny z jednoduchých predikátů

Pravidlo MC (minimality a úplnosti množiny jednoduchých predikátů)

- **Každý fragment tabulky musí být přístupný jedinečným způsobem alespoň jedné aplikaci.**
- **Vlastnost minimality množiny jednoduchých predikátů znamená, že vypuštěním kteréhokoliv z nich se poruší úplnost množiny jednoduchých predikátů.**
- **Naopak přidání dalšího predikátu způsobí zavedení dalšího (nenutného) fragmentu se stejnými statistickými vlastnostmi**

Pravidlo MC (minimality a úplnosti množiny jednoduchých predikátů)

- **Všechny řádky libovolného fragmentu tabulky musí být přístupné se stejnou pravděpodobností každému procesu definovanému pro fragment (Completeness).**
- **Tzn. chceme, aby všechny řádky fragmentu měly stejné statistické vlastnosti, pak fragment lze zpracovávat při optimalizaci dotazu jako jeden celek.**

Příklad

Fragmenty jsou určeny kombinací mintermových predikátů z množiny jednoduchých predikátů, která splňuje MC podmínky.

Příklad:

ZEMĚ = "ČÍNA"
PROFIT > 1000 000.00

Návrh fragmentů při odvozené horizontální fragmentaci

- Odvozená horizontální fragmentace se používá k usnadnění spojení mezi fragmenty
- Distribuované spojení relací R a S =
 - = spojení mezi horizontálně fragmentovanými relacemi. Provádí se porovnání všech fragmentů Ri se všemi fragmenty Sj
- Graf distribuovaného spojení - hrany spojují fragmenty, jejichž spojení nejsou prázdná

Návrh fragmentů při odvozené horizontální fragmentaci

! Navrhnut fragmenty tak, aby vzniknul:

- Jednoduchý graf spojení R join S má jednoduchý graf spojení, platí-li $R_i = R \text{ Semijoin } S_j$
- Nesouvislý graf spojení

Návrh fragmentů při vertikální fragmentaci (které atributy do kterého fragmentu)

Pro bezzávratnost musí každý fragment obsahovat primární klíč. Fragmentace ostatních atributů závisí na frekvenci současného přístupu k nim z aplikace (užívá se statistické shlukování atributů).

- Minimalizovat počet fragmentů potřebných ke splnění požadavku
- Minimalizovat zátěž sítě

Alokace fragmentů a replik

- Plně replikovaná DB
- Parciálně replikovaná DB
(problém
 - stupeň replikace,
 - odkud čist
)

Užitek z umístění kopie musí být větší než cena za umístění !
Možnost navrhnut nereplikované řešení a postupně zavádět repliky

Jak měřit cenu za umístění a užitek replik?

Označme:
 i index fragmentu
 j " uzlu
 k " požadavku
 fkj frekvenci požadavku k na uzel j
 rki počet vyhledávacích přístupů požadavku k na i-tý fragment
 uki " změnových " " " "
 nki = rki + uki
 R globální relaci
 Ri fragment

Možnosti rozmístění replik horizontálních fragmentů

Bez uvažování replik umístíme R_i do toho uzlu, kde počet přístupů na R_i je největší. Tj. je-li počet lokálních přístupů k R_i v uzlu j

$$B_{ij} = \sum_k (f_{kj} * n_{ki})$$

umístíme R_i do uzlu j^* , kde je B_{jj^*} maximální.

Možnosti rozmístění replik horizontálních fragmentů

Replikované umístíme R_i do všech uzlů j , kde cena čtecích přístupů je větší než cena změnových přístupů na R_i od požadavků z ostatních uzlů.

$$B_{ij} = \sum_k (f_{kj} * r_{ki}) - C * \sum_{k' \neq j} (f_{kj'} * u_{ki})$$

C je konstanta, zohledňuje dražší změnový přístup oproti čtecímu

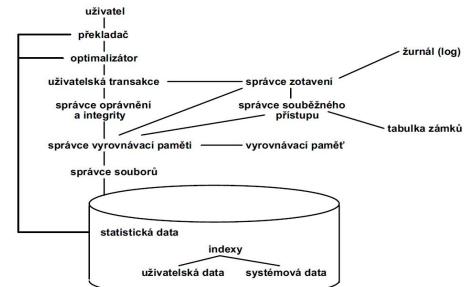
R_i je umístěn do všech uzlů j^* , kde je $B_{jj^*} > 0$

Oceňování výhodnosti vertikální fragmentace

? Důsledek rozdělení fragmentu R_i u uzlu r na dva fragmenty R_s a R_t umístěné v uzlech s a t .

1. Požadavky A_s a A_t vydávaných v uzlech s a t , které pracují pouze s atributy z R_s a z R_t se stanou lokální v uzlech s a t . (Usetří vzdálený odkaz)
2. Požadavky A_1 původně lokální v uzlu r , používající pouze atributy z R_s nebo z R_t se stane nelokální. (Odkaz navíc pro každý požadavek)
3. Požadavky A_2 původně lokální v uzlu r , používající atributy jak z R_s tak z R_t se stane nelokální. (Dva odkazy navíc pro každý požadavek)
4. Požadavky A_3 z uzlů jiných než r, s, t , používající atributy jak z R_s tak z R_t zůstanou nelokální (každý požadavek ale potřebuje jeden odkaz navíc)

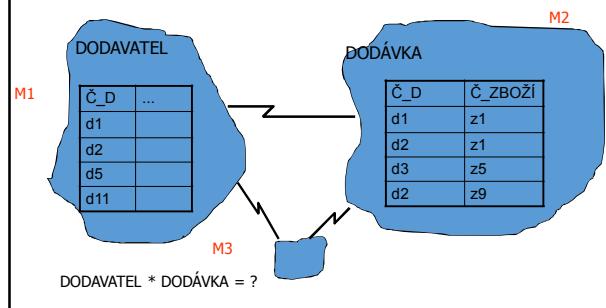
Struktura databázového systému



Distribuované zpracování dotazu

- problémy (přidavné k centralizovanému případu)
 - cena přenosů
 - cpu, disk, #přenášených Byte, #přenášených zpráv
 - paralelismus / překryvání prodlív
 - Jak minimalizovat uběhnutý čas?
 - Nebo jak minimalizovat spotřebu zdrojů?

Optimalizace distribuovaného dotazu – polospojení



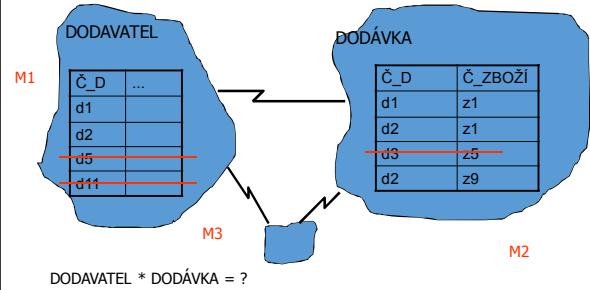
Polospojení

volba plánů:

- P1: přesuň DODÁVKA-> M1; *; přesuň výsledek -> M3
- P2: přesuň DODÁVKA->M3; přesuň DODAVATEL->M3; *
- ...
- další?

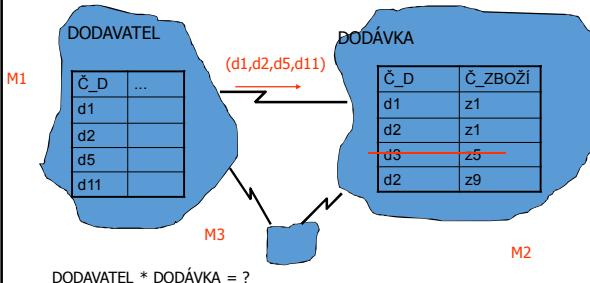
Polospojení

Idea: před přesuny redukovat tabulky



Polospojení

Idea: před přesuny redukovat tabulky, např DODÁVKA



Polospojení

Formálně:

$$\text{DODÁVKA}' = \text{DODÁVKA} <^* \text{DODAVATEL}$$

Pz: vyjádření polospojení pomocí RA:

$$R <^* S \equiv (R * S)[R]$$

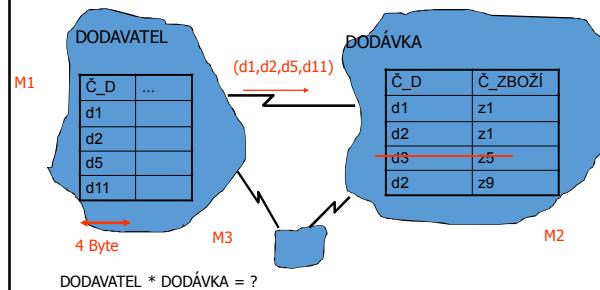
Polospojení – příklad

Velikost hodnoty každého atributu je 4 Byte

Q: cena přenosu (#Byte) pro polospojení
 $\text{DODÁVKA}' = \text{DODÁVKA} <^* \text{DODAVATEL}$

Polospojení

Idea: před přesuny redukovat tabulky



Polospojení – příklad

Předpoklad: hodnota každého atributu 4 Byte

Q: cena přenosu (#Byte) pro polospojení
DODÁVKA' = DODÁVKA <* DODAVATEL

Zde: Q = 4*4 Byte

Strategie

krok1: navrhni plán s polospojením(i)
krok2: odhadni jeho cenu (# přenesených Byte)

Polospojení – příklad

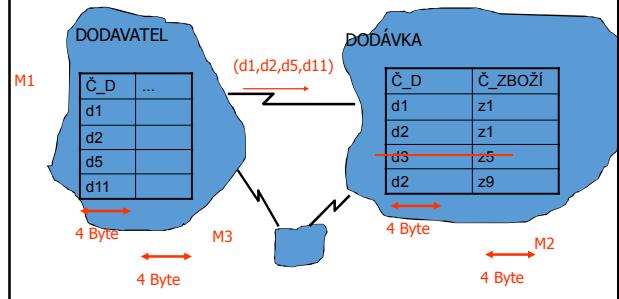
P3:

- redukuji DODÁVKA na DODÁVKA'
- DODÁVKA' -> M3
- DODAVATEL -> M3
- proved * v M3

Cena?

Předpoklad: hodnota každého atributu 4 Byte

Polospojení



Polospojení – příklad

• Q pro P3:

- 4*4 Byte - redukuji DODÁVKA na DODÁVKA'
- 3*8 Byte - DODÁVKA' -> M3
- 4*8 Byte - DODAVATEL -> M3

72 Byte celkem

Další plány?

P4:

- redukuji DODÁVKA na DODÁVKA'
- redukuji DODAVATEL na DODAVATEL'
- DODÁVKA' -> M3
- DODAVATEL' -> M3

Další plány?

P5:

- redukuj DODAVATEL na DODAVATEL'
- DODAVATEL' -> M2
- proved* v M2
- přesuň výsledek -> M3

Replikace dat

- fragment relace je **replikovaný**, je-li uložen současně na více místech.
- **úplná replikace** relace: případ, kdy je relace umístěna ve všech místech
- **plně redundantní databáze**: případ, kdy je databáze umístěna ve všech místech

Replikace dat

- **výhody**
 - dostupnost: chyba v místě s relací R nevede k nedostupnosti R,
 - paralelismus: dotazy nad R mohou být zpracovávány paralelně na více místech,
 - redukce přenosů dat.
- **nevýhody**
 - složitější aktualizace: každá replika R musí být aktualizována,
 - zvýšená složitost řízení souběžného zpracování: možnost vzniku nekonzistence dat.

Strategie aktualizací

- v závislosti - KDY:
 - synchronní (rychlejší)
 - asynchronní (pomalejší)
- v závislosti - KDE:
 - primární kopie (master)
 - aktualizace kdekoliv



Strategie aktualizací

| Synchronní | Aktualizace kdekoliv |
|--|---|
| + žádné nekonzistence (identické kopie) | + jakékoli místo může zpracovat transakci (změnu) |
| + čtení lokální kopie poskytuje aktuální hodnotu | + zátižení je rovnoměrně distribuováno |
| + změny jsou atomické | - kopie je třeba synchronizovat |
| - transakce musí aktualizovat všechna místa | |
| - delší čas transakce | |
| - nižší dostupnost | |
| Asynchronní | Promarní kopie |
| + transakce je vždy lokální (dobrý čas odpovědi) | + žádná synchronizace mezi místy není nutná (je v primární kopii) |
| - nekonzistence dat | + existuje vždy jedno místo, které má všechny aktualizace |
| - lokální čtení ne vždy vrací aktuální hodnotu | - v primární kopii může být velké zátižení |
| - změny do všech kopií nejsou garantovány | - čtení lokální kopie nemusí poskytovat aktuální hodnoty |
| - replikace není transparentní | |

Replikační strategie

| | master | kdekoliv |
|--------|--|--|
| synch | <ul style="list-style-type: none"> + změny nemusí být koordinovány + žádné nekonzistence - dlouhá odezva - v hodině pro málo změn - lokální kopie mohou být pouze čteny | <ul style="list-style-type: none"> + žádné nekonzistence + elegantní (symetrické) řešení - dlouhá odezva - změny musí být koordinovány |
| asynch | <ul style="list-style-type: none"> + koordinace není nutná + krátký čas odezvy - lokální kopie nejsou up-to-date - nekonzistence | <ul style="list-style-type: none"> + žádná centrální koordinace + nejkratší čas odezvy - změny mohou být ztraceny (urovnávání rozdílů) - nekonzistence |

Distribuovaný monitor globálního volání

- řešení protokolů:
- primární kopie
- většinový protokol
 - konsensus kvórem
- ...
- slabá konsistence

Protokol primární kopie

- zvol repliku prvku dat jako jeho **primární kopii**.
 - její místo je **primární místo** pro daný prvek dat
 - různé prvky dat mohou mít různá primární místa
- požaduje-li transakce zamknout prvek P, požaduje to na primárním místě pro P.
 - odтud se provádějí propagace změn na kopie
- **výhody**
 - souběžné zpracování jako na nereplikovaných datech – jednoduchá implementace.
- **nevýhody**
 - v případě chyby v primárném místě, je P nedostupný, dokonce i když jiná místa s replikou P jsou dostupná

Slabě konsistentní replikace

- negarantuje uspořádatelnost
- Př.: **replikace master-slave** – aktualizace jsou prováděny v jednom “master” místě a jsou propagovány do “slave” míst.
 - Propagace není součástí transakce !:
 - Může nastat bezprostředně po potvrzení transakce,
 - Může být periodická
 - Data v podřízených místech mohou být pouze čtena, nikoliv aktualizována
 - Netřeba používat zámky na vzdálených místech
 - Využitelné při distribuci informací (např. centrální úřad a jeho pobočky)

Mobilní databáze

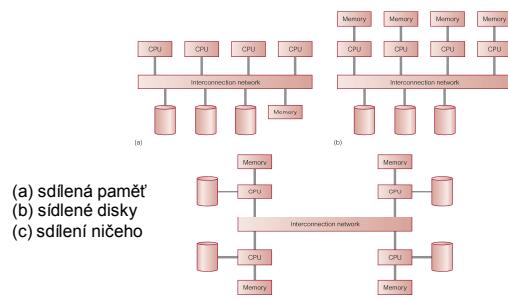
- **mobilní databáze** je rozšířením distribuovaného DBS.
- mobilní databáze může obsahovat databáze spojené sítí pevných linek a databáze zabudované na mobilní stanice.
- charakteristiky:
 - bezdrátová síť má omezenou šířku pásmá,
 - energie používané v mobilních stanicích mobilních stanic má omezenou dobu života,
 - kvůli energetickým omezením, mobilní stanice nejsou vždy dostupné,
 - mobilní stanice se pohybují různou rychlosťí a v různých oblastech.

Mobilní databáze

- Uvedené charakteristiky jsou důvodem, proč CC (**Computational complexity theory**) problém v mobilních databázích je těžší než v distribuovaných databázích.
- Odpojení stanice je dlouhé, takže uzamykací protokoly a časová razítka nejsou vhodné. 2PC (**two-phase commit protocol**) je nevhodný také, protože dostupnost je redukována.
- Pro prostředí s mobilními databázemi byly navrženy různé transakční modely založené na uvolnění vlastnosti ACID a uvolnění uspořádatelnosti.
- Je mnohem těžší navrhovat mobilní databáze systemy pracující v reálném čase kvůli nepredikovatelnosti prostředí.
 - Zdroj: Philip A. Bernstein, Eric Newcomer (2009): *Principles of Transaction Processing, 2nd Edition*, Chapter 8, Morgan Kaufmann (Elsevier), ISBN 978-1-55860-623-4
 - Arora & Barak 2009, Chapter 1: The computational model and why it doesn't matter
 - Sipser 2006, Chapter 7: Time complexity

Paralelní ŠRBD

- Nejde o DSŘBD!



Závěry

- Distribuované SŘBD nabývají na významu, především jejich varianty s replikacemi, nověji mobilní databáze, NoSQL databáze
- nová téma:
 - polospojení
 - distribuované zotavení z chyb a souběžné zpracování
 - horizontální distribuce, uvolnění ACID
- aplikace:
 - paralelní db /klastry
 - aktivní disky
 - gridy
 - replikované db (servery pro e-komerči)

ZPRACOVÁNÍ VELKÉHO OBJEMU DAT

Hadoop a další možnosti

2017

Big Data - definice Gartner

„soubory dat, jejichž velikost je mimo schopnosti zachycovat, spravovat a zpracovávat data běžně používanými softwarovými nástroji v rozumném čase.“

2017

Big Data 3V

- **Objem (volume)** – množství dat vznikajících v rámci provozu firem roste exponenciálně každý rok,
- **Typ (variety)** – různorodost typů dat vznikajících, například nestrukturované textové soubory, semi-strukturovaná data (XML), data o geografické poloze, data z logů,
- **Rychlosť (velocity)** – rychlosť s jakou data vznikají a potřeba jejich analýzy v reálném čase vznikají díky pokračující digitalizaci většiny transakcí, mobilním zařízením a vznikajícímu počtu internetových uživatelů.

2017

Big Data 3V Variety: Structured → Unstructured

- Volume: Terabytes → Zettabytes Velocity: batch → streaming data

2017

Real-time database and analytics

These are typically in-memory, scale-out engines that provide low-latency, cross-data center access to data, and enable distributed processing and event-generation capabilities.

- **Interactive analytics:** Includes distributed MPP (massively parallel processing) data warehouses with embedded analytics, which enable business users to do interactive querying and visualization of big data.
- **Batch processing:** Hadoop as a distributed processing engine that can analyze very large amounts of data and apply algorithms that range from the simple (e.g. aggregation) to the complex (e.g. machine learning).

2017

Řešení big data

- **Hardware** – konsolidovaná integrovaná řešení s důrazem na výkonnost (storage, server, ...)
- **Distribuce dat** – např. Hadoop
- **Data Management** – např. NoSQL
- **Analýza a vizualizace** – trend: in-memory

2017

Hadoop

Apache Hadoop je programové prostředí, které umožňuje paralelní běh big data aplikací v rámci výpočetního clusteru. Zahrnuje sadu nástrojů pro distribuované pořizování, ukládání a zpracování velkých dat.

Je to open-source systém volně dostupný i pro komerční použití, podobně jako třeba webový server Apache. Existují ale také komerční distribuce, součástí jejichž licence je i provozní podpora (např. Hortonworks, Cloudera, MapR).

2017

Hadoop Apache Foundation

- Framework – sada open-source komponent určených pro zpracování velkého množství nestrukturovaných a distribuovaných dat HDFS (Hadoop Distributed File Systém)
- Verze 2012 – až 4000 uzlů
- Programový model: map-reduce

2017

KDY POUŽÍT HADOOP?

- Ke zpracování je potřeba vysoký výpočetní výkon ... data mining, statistické metody
- Data přibývají velmi rychle (desetitisíce zpráv za sekundu) ... analýzy dat v reálném čase
- Celkový objem dat k uložení je velký (desítky TB) ... spolehlivé, dostupné úložiště

2017

Hadoop vs. databáze

Tabulkové vs. soubory

Data v Hadoopu mohou být uložena ve zdrojové struktuře i formátu. Systém HDFS zajistí bezpečné a efektivní uložení souborů libovolné velikosti (limitem je pouze kapacita celého clusteru).

Pevné schéma vs. volná struktura

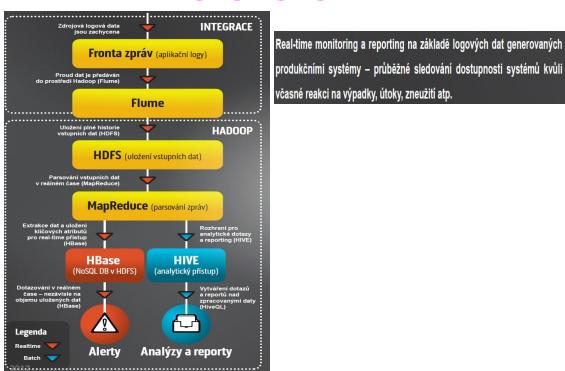
V Hadoopu se data ukládají jako soubory a struktura se definuje až při jejich použití.

Transakčnost vs. dostupnost a škálovatelnost

Hadoop upřednostňuje rychlosť odpovědi před úplností (např. při výpadku části clusteru). Spolehlivost je zajištěna redundancí – každý uzel clusteru je kdykoli nahraditelný.

2017

CASE STUDY



Komponenty Hadoop systému

MapReduce

prostředí pro vývoj JAVA aplikací pro řešení distribuovaných úloh metodou lokálního zpracování dat (Map) a skládání výsledků (Reduce)

Distribuované - rovnéměrné rozložení úlohy do jednotlivých výpočetních uzlů s ohledem na jejich výkon

Škálovatelné - snadné doplnění dalších uzlů do clusteru

Robustní - automatická obnova úlohy při lokálním výpadku

2017

Komponenty Hadoop systému

HDFS
distribuovaný souborový systém, který umožňuje bezpečně a efektivně ulákat a zpracovávat velké soubory v rámci Hadoop clusteru

Flume
zpracování objemných proudů dat v reálném čase

Sqoop
dávkové přenosy dat z relačních databází

2017

Komponenty Hadoop systému

Mahout
knihovna pro data mining a strojové učení

MapReduce

Pig
programové prostředí a skriptovací jazyk pro snazší programování MapReduce (vyvinut Yahoo!)

2017

Komponenty Hadoop systému

HBASE
nerelační databáze v HDFS
Umocňuje asynchronní učení řídkých dat (po vzoru Bigtable od Google) a velmi rychlé distribuované prohledávání, čtení i zápis
Rychlosť odezvy je dležitější než časová konzistence a úplnost

Impala
in-memory databáze pro přístup do HDFS a HBASE v reálném čase (vyvíjen Cloudera)

Hive
prostředí datového skladu nad HDFS (vyvinut Facebook)
Umocňuje výkonné tabulky, schýmlata a reporty přímo nad daty v HDFS
Podporuje přístup ke strukturovaným datům pomocí SQL, dotazů (obdobu SQL.)

2017

Komponenty Hadoop systému

2017

Největší uživatelé Hadoop technologií

Yahoo – primární data z prohléďávání webu, vkládání reklam; přes 100,000 CPU (~50 PB)
Facebook – kopie hlavních dimenzií a faktů pro analýzy; přes 11,000 CPU (~15 PB)
Ebay – optimalizace vyhledávání, analytické zpracování; přes 4,000 CPU (~5 PB)
Last.Fm – vyhledávání vzorů v hudebních skladbách, statistiky používání, analýzy oblibenosti; přes 800 CPU (~800 TB)

YAHOO! facebook ebay last.fm

2017

Příklad: Instant Messenger 2006

- 150 Gb / day - log file
- 1 měsíc (2006) - 4,5 TB
- June 2006 – 245 mil uživatelů zalogovaných, 180 mil konverzujících
- 255 miliard zpráv

2017

2011 12 TB tweet na Twitteru 30 miliard RFID tagů

- 4,6 miliardy mobilních zařízení
- Ročně se prodá 100 milionů GPS zařízení
- Přes 2 miliardy lidí na Internetu

2017

On-line learning Sociální sítě – sentiment analysis

- **Internetová bezpečnost** – spam filtering. Credit card transaction detection, intrusion detection
- **Finance** – finanční rozhodování – online portfolio selection, sequential onvestment...

2017

Learning

- **Batch - (off-line) learning**
 - Observe a batch of training data
 - Learn a model from them
 - Predict new samples accurately
- **On-line learning Observe a sequence of data**
 - Learn a model incrementally as instances come
 - Make the sequence of online predictions accurately
 - Vysoká adaptabilita, paralelizace...

2017

SHRNUTÍ

2017

Apache Hadoop

- Open-source softwarový rámec odvozený z Google MapReduce a Google File System za účelem efektivního zpracování velkého objemu dat.
- Hlavní podstata spočívá v rozdělení zadaných úloh na drobnější úkoly, které se v několika počítacovém clusteru zpracovávají paralelně a na závěr vybranou funkcí agregují požadovaný výstup.
- Jedná se o vylepšení již zaběhnuté technologie grid computingu uváděné pod názvem cloud computing.
- Příklad:
 - společnost s třiceti miliony zákazníků doluje údaje z datových skladů, reportů a záznamů call center pro vylepšení svých služeb.
 - Operace, která trvá týden, zabere použitím nové techniky jen pár hodin. (V Hadoop rámci jsou zabudované další softwarové komponenty pro práci s velkým objemem dat.)

2017

DALŠÍ MOŽNOSTI

2017

High-performance analytic appliance (HANA)

- Technologie **high-performance analytic appliance (HANA)** (od společnosti SAP) je platforma rozložitelná v cloudu.
- Je vhodná pro tvorbu real-time analýz a vývoj real-time aplikací nad velkým objemem operačních a transakčních dat.
- Replikační a synchronizační server zajišťuje paralelní in-memory zpracování (data jsou zavedena přímo v operační paměti) - mnohonásobně rychlejší přístup.
- HANA podporuje různé technologické standardy, mezi mezi jinými i structured query language (SQL).

2017

Apache Cassandra

- Apache Cassandra je masivně škálovatelná NoSQL databáze.
- Je navržena pro práci s velkým objemem real-time dat na několika datových centrech s žádným kritickým bodem pro selhání celého systému.
- Poskytuje vysoký databázový výkon a neustálou dostupnost.
- Cassandra na rozdíl od Hadoop nepracuje na bázi key – value (klíč – hodnota) úložiště, nýbrž jako column – family (sloupec – rodina) úložiště, kde se ukládaný objekt skládá z trojice: název, klíč, časový otisk.
- V praxi to znamená upřednostnění využití projektu Cassandra na úkor Hadoop při zpracování dat z naměřených časově podložených údajů nebo bezpečnostních logů, kde čas hraje významnou roli.

2017

Apache Cassandra + Hadoop

- Není vyloučena spolupráce obou projektů v rámci jednoho clusteru.
- Časově založené (real-time) aplikace spouští Cassandra a dávkově založené analýzy a dotazy zpracuje Hadoop.
- Zdroje:
 - <http://www.datastax.com/technologies/cassandra>
 - HOWARD, Philip. Cassandra and Hadoop. Dostupné z: <http://www.blooresearch.com/blog/IMBlog/2012/1/cassandra-hadoop.html>

2017

MongoDB

- Mezi účinné způsoby zpracování velkých objemů dat lze zařadit i dokumentově orientovanou NoSQL databáze MongoDB.
- MongoDB je v určitých aspektech koncepčně blízká relačním databázím, tzn. poskytuje vysoký výkon, zaručuje dostupnost a snadnou škálovatelnost.
- Celé rozestavení hostuje mnoho databází, databáze drží řadu kolekcí, kolekce pak několik dokumentů a dokumenty jsou množiny dvojic klíč – hodnota.
- MongoDB používá vlastní dotazovací jazyk podobný SQL.
- Pro hromadné operace nad daty má k dispozici implementaci návrhového vzoru MapReduce.

2017

MongoDB

- MongoDB pro provozování velkých nebo zatížených aplikací využívá vlastnosti sharding a replication.
- Sharding distribuuje jeden logický databázový systém přes více strojů v clusteru.
- Dokumenty jsou rozdělovány podle specifického „shard“ klíče.
- Replikace databáze zajišťuje redundanci, zálohování a automatické překonávání chyb.
- Při úvahách nad architekturou nové aplikace by tak měl být vždy vzat v úvahu NoSQL princip, zvláště při ukládání velkého objemu **nestrukturovaných** dat.

2017

Závěr

- Existuje mnoho metod pro zpracování velkého objemu dat.
- Jen omezené množství se po implementaci v praxi skutečně osvědčí.
- Rozhodnutí, zda zvolit NoSQL databázi nebo distribuovaný souborový systém, využít technologie cloud computingu či nikoliv, závisí pouze na prioritách v dané situaci a samotných společnostech.

2017

Hype křivka

- Analytická společnost Gartner zavedla do informační teorie graf, který hodnotí vyspělost konkrétních IT uvedených na informačním trhu, popř. jejich připravenost na implementaci.
- Jednotlivé body zralosti spojuje **hype křivka** („křivka humbuku“).
- **Hype křivka** ukazuje grafickou reprezentaci stavu zralosti, přijetí technologie a její potenciál řešit konkrétní úlohy.

2017

Hype křivka

- Grafické vyjádření **hype křivky** rozděluje technologii do pěti fází vývoje:
 - 1. Počáteční zájem** – média nadšeně popularizují novou technologii a vkládají do ní občas až přehnaně velký potenciál, přítom technologie zatím většinou nemá žádné praktické využití.
 - 2. Vrchol očekávání** – časná publicita technologie a opěvování očekávaných pozitivních výsledků je doprovázena kritickými ohlasy; naděje a očekávání začínají klesat.
 - 3. Deziluze** – zájmu ubývá, mnoho experimentů a pokusů o implementaci nové technologie se na první pokus nezdáří.
 - 4. Pozitivní sklon osvícení** – pochopení technologie více do hloubky vede k nalezení jejích výhod. Objevují se další generace produktů od poskytovatelů technologických služeb.
 - 5. Plošina produktivity** – technologie je plně přijata a využívána.

2017

Hype křivka

- Sledované technologie v různých fázích svého vývoje vykazují určitou míru „**viditelnosti**“, a to nejen ve smyslu viditelnosti v médiích, ale především ve smyslu vkládaných očekávání a nadějí.
- Tato závislost „**viditelnosti**“ na „**stavu zralosti**“ je prakticky nezávislá na tom, o jakou konkrétní technologii jde.

2017

Hype křivka technologie cloud computing



Dostupné z:
<http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp>

Hype křivka technologie cloud computing

- Na konferenci Business & Information Fórum 2011, věnované informačním a komunikačním technologiím jako nástroji pro úspěšné podnikání, bylo několikrát zmíněno, že po prvotním zájmu se nyní **cloud computing** nachází na hype křivce v nejnižším bodu, v údolí **Deziluze**.
- Byl predikován pozvolný vzestup - technologie **cloud computingu** se stane užitečnou a běžně užívanou součástí společností využívajících IT služby.

2017

SQL/MM

SQL Multimedia and Application Packages

3013

4

- Mezinárodní norma ISO/IEC 13249-1 byla připravena technickou komisí ISO/IEC JTC 1 *Informační technologie*, subkomisi SC 32, *Počítačové aplikace*.

ISO/IEC 13249

- ISO/IEC 13249 sestává z následujících částí pod obecným názvem *Informační technologie - Databázové jazyky - Knihovny SQL pro multimédia a další použití*:
 - – Část 1: Základní rámec
 - – Část 2: Plnotextová data
 - – Část 3: Prostorová data
 - – Část 5: Statický obraz

2017

143

- ISO/IEC 13249

- Ostatní části této normy specifikují požadavky a všechny jsou závislé na různých částech ISO/IEC 9075 a rovněž na této části ISO/IEC 13249.

4

- klasické (relační, objektové) databáze
 - pevně daná struktura i sémantika (schéma databáze, tj. typované atributy, tabulky, integritní omezení, funkční závislosti, dědičnost, atd.)
 - „umělá“ povaha dat (člověkem vytvářené atributy a jednoznačně interpretovatelné atributy)
 - víme co hledáme = stačí dotazy na úplnou shodu
 - multimedialní databáze
 - kolekce obrázků, audia, video, časových řad, textů, XML, atd.
 - obecné kolekce nestrukturovaných dat (**dokument**)
 - vnitřní struktura i sémantika je skrytá a nejednoznačná - závislá na aplikaci, datech, i subjektivitě uživatele
 - „analogová“ povaha dat (digitalizace signálů/u sýrových dat)
 - nevíme pořádně co hledáme ani jak se ptát = nestáčí dotazy na úplnou shodu

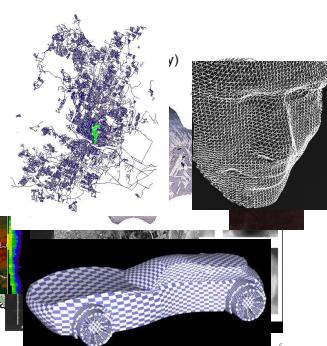
2017

14

Příklady multimedialních dat (1)

obnovou databáze

- obrazové databáze**



201

Typy MDB systémů

• text-based retrieval systémy

- vyhledávání pouze podle textové anotace (meta-informace)
 - automatické anotování (např. images.google.com využívá textu na stránce, kde je na obrázek odkaz, případně název souboru obrázku)
 - ruční anotace – většinou kvalitnější, anotuje expert, který ví, jak anotovat
- dotazy podobně jako u fulltextových vyhledávačů, tj. množina klíčových slov
- výhoda – využití stávající implementace fulltextových vyhledávačů
- nevýhody
 - nelze aplikovat na neanotované kolekce, ruční anotování je drahé
 - anotace je vždy nějak nepřesná (subjektivní, neúplná, zavádějící, atd.)
 - získané dokumenty mohou být úplně irrelevantní
 - nezískali jsme dokumenty, které jsou relevantní - „netrefili“ jsme se do anotace

2017

13

Typy MDB systémů

• content-based retrieval systémy

- vyhledávání pouze podle obsahu
- různé metody popisu obsahu
- výhody
 - vyhledávání podle skutečného obsahu
 - nezávislost na anotaci,
- nevýhody – mnoho různých metod modelování struktury a sémantiky obsahu, kterou vybrat?

• hybridní systémy

- kombinují výše zmíněné dva

2017

14

Modality vyhledávání

• dotazování (querying)

- dotaz v kontextu dokumentu
 - dokument chápán jako databáze, kde hledáme dílčí fragment
 - rozpoznávání/analýza obrazu, vyhledávání v DNA sekvencích, řetězcích, apod.



• dotaz v kontextu kolekce

- celý dokument představuje sémantickou strukturu

2017

15

Dotazování podle podobnosti

• query-by-example typy dotazů

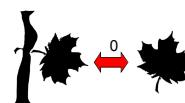
- ptáme se přímo nějakým dokumentem (ať dokumentem z databáze ve které hledáme, nebo z jiného)
- navíc specifikujeme rozsah dotazu nebo výsledku
 - bodový dotaz
 - rozsahový dotaz – práh r
 - k nejbližším sousedů - k
 - reverzních k nejbližším sousedů - k a další...

2017

16

Kritika metrických vlastností

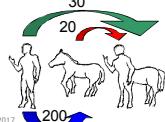
ad reflexivita: objekt nemusí být sám sobě podobný



ad pozitivita: objekt je maximálně podobný (totožný) jinému objektu

ad symetri: objekt 1 je podobný objektu 2 jinak, než je tomu naopak (záleží na směru porovnávání)

ad trojúhelníková nerovnost: obecně nepřipadá tranzitivita

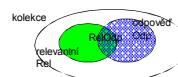


17

Kvalita vyhledávání vs. efektivita vyhledávání

• kvalita vyhledávání (retrieval effectiveness) je úspěšnost vyhledání dokumentu vzhledem k očekávání uživatele

- vždy subjektivní, nelze dosáhnout dokonalosti
- měření na základě subjektivně ohodnocené kolekce
- nejčastěji přesnost $P = |\text{RelOdp}| / |\text{Odp}|$ a úplnost $R = |\text{RelOdp}| / |\text{Rel}|$



• rychlosť vyhledávání (retrieval efficiency) ovlivňuje reálnou použitelnost a škálovatelnost

- I/O operace, množství výpočtů podobnosti/vzdáleností, ostatní CPU náklady
- potřeba speciálních přístupových metod, resp. indexování, sekvenční průchod je u velkých databází nereálný

2017

18

Vývoj

- Motivace
 - SFQL (1991-1992)
 - Structured Full-text Query Language
 - Rozšíření SQL pro práci s full-textovými daty
 - Vytvořen „Full-textovou“ komunitou
 - Konflikt klíčových slov (CONTAINS)
 - Full-Text vs. Prostorová data
 - Možný konflikt klíčových slov je třeba řešit

2017

19

Řešení

- SQL/MM (1999-2002)
 - Jde o řadu UDT a UDF dle SQL:1999
 - Okruhy působnosti
 - Full-textová data
 - Prostorová data
 - Obrázky (statické i video)

2017

20

Obsah

- Úvod
- **Část 1: Framework**
- Část 2: Full-Text
- Část 3: Spatial
- Část 4: General Purpose Facilities
- Část 5: Still Image
- Část 6: Data Mining

2017

21

Framework

- Jednotlivé části SQL/MM jsou na sobě poměrně nezávislé
- Framework je část společná a závazná „zbytku světa“
- Poskytuje:
 - definici společného konceptu užitého v dalších částech SQL/MM
 - hlavní rysy přístupu k definici těchto částí

2017

22

Obsah

- Úvod
- Část 1: Framework
- **Část 2: Full-Text**
- Část 3: Spatial
- Část 4: General Purpose Facilities
- Část 5: Still Image
- Část 6: Data Mining

2017

23

Full-Text > Úvod

- „Full-Text“ ≈ textová data lišící se od běžných znakových řetězců
 - Obvykle delší záznamy
 - Specifické operace
 - Způsob indexování
 - Index slov v dokumentu
 - Index vzájemných vzdáleností slova a frázi
 - ...

2017

24

Full-Text > Typ "FullText"

- Konstruktory
 - Řetězec znaků
 - Řetězec znaků + zadání jazyka
- Konverzi do běžných SQL znak. řetězců.
 - FullText_to_Character
- Vyhledávací metody
 - ano/ne (CONTAINS)
 - Rank (RANK)

2017

25

Full-Text > vyhledávání

- Vzorek (+ wildcards)
- Odvozená slova (STEMMED)
- Slova s podobným nebo stejným významem (THESAURUS, SYNONYM)
- Stejně znějící slova (SOUNDS LIKE)
- Dle pozice v textu (NEAR, ...)
- Dle konceptu textu (IS ABOUT)

2017

26

Full-Text > Podpora jazyků

- SQL/MM počítá především s podporou jazyků, kde je snadné výpočetně roznechat jednotlivé tokeny.

2017

27

Full-Text > Příklad > tabulka

```
CREATE TABLE informace (
    číslo_dokumentu INTEGER,
    dokument          FULLTEXT
) ;
```

2017

28

Full-Text > Příklad > dotaz

```
SELECT číslo_dokumentu
  FROM informace
 WHERE dokument .CONTAINS
   (
     'STEMMED FROM OF "standard"
      IN SAME PARAGRAPH AS
      SOUNDS LIKE "sequel"'
   ) = 1;
```

2017

29

Obsah

- Úvod
- Část 1: Framework
- Část 2: Full-Text
- **Část 3: Spatial**
- Část 4: General Purpose Facilities
- Část 5: Still Image
- Část 6: Data Mining

2017

30

Obsah

- Úvod
- Geometrický model
- Příklady
- Datový katalog
- Závěr

2017

31

Úvod

2017

32

Prostorové databáze

- Databáze, ve kterých jsou kombinována formátovaná data a prostorová data

2017

33

Využití prostorových databází

- Geografické informační systémy (GIS)
 - Plánování výstavby měst
 - Řízení dopravy
 - Mapování nalezišť nerostných surovin
 - Distribuční síť
 - Bankovnictví
 - Pojišťovnictví

2017

34

Využití prostorových databází

- CAD/CAM (Computer-aided design and manufacturing)
 - Návrh integrovaných obvodů
 - Návrh mostu
 - Návrh motoru

2017

35

Co potřebujeme?

- Vhodný dotazovací jazyk
- Použití standardních dotazovacích jazyků je nevhodné
 - Neposkytuje žádnou podporu pro dotazování nad prostorovými daty

2017

36

Řešení

- Nevyvíjet zcela nový dotazovací jazyk, ale využít jazyka SQL
- Proč?
 - SQL je databázový standard
 - V praxi se často vyskytují prostorová data společně s lexikálními daty

2017

37

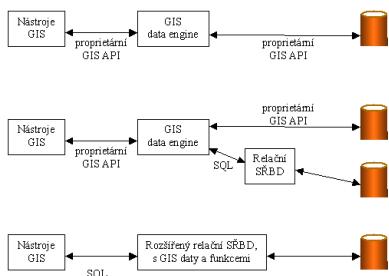
Co musí jazyk splňovat?

- Musí umožnit formulování dotazů následujících typů
 - Dotazy výhradně na prostorové vlastnosti (Najdi všechna města rozdělená řekou)
 - Dotazy pouze na neprostorové vlastnosti (Kolik lidí žije v Ostravě?)
 - Kombinace předechozích typů (Vypiš jména všech sousedů parcely číslo 15 v Soukenické ulici)

2017

38

Historický vývoj GIS databází



2017

39

Historie ukládání prostorových dat v relačních databázích

- Prostорová data jsou uložena v tabulkách v 1NF
- Prostорová data jsou uložena jako nestrukturované rozsáhlé binární objekty (BLOB) či jako "opaque types"
- Prostорová data jsou uložena v relační databázi rozšířené o typy prostorových dat
- Prostорová data jsou implementovaná pomocí uživatelsky definovaných typů

2017

40

Vznik SQL/MM (1999)

- SQL/MM:
 - Part 1: Framework
 - Part 2: Full-text
 - **Part 3: Spatial**
 - Part 5: Still images
 - Part 6: Data mining

2017

41

Vlivy na utváření SQL-MM Spatial

- OpenGIS konsorcium: OpenGIS Simple Features Specification (geometrický model)
- Standard SQL:1999 (objektová orientace)

2017

42

Co SQL/MM Spatial definuje?

- Ukládání, výběr, dotazování a aktualizaci jednoduchých prostorových objektů
- Reprezentaci prostorových objektů pomocí prostorových datových typů (Spatial types)
- Funkce pro práci s prostorovými objekty

2017

43

Dělení SQL/MM Spatial

- Standard SQL/MM Spatial je rozdělen do několika klauzulí
- Jednotlivé klauzule se zabývají:
 - Prostоровыми типами и их методами (Spatial types)
 - Каталогом (Information Schema)
 - ...

2017

44

Datové typy pro reprezentaci prostorových objektů

2017

45

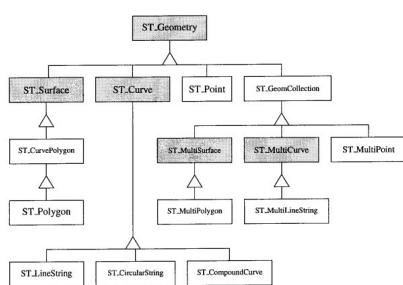
Prostorové objekty

- Body:
 - pouliční lampa, lavička...
- Křivky:
 - koryto řeky, hranice pobřeží...
- Polygony:
 - půdorys budov, území států, pozemky...

2017

46

SQL geometrický model



2017

47

SQL geometrický model

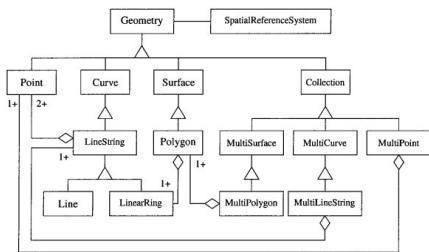
- Hierarchický model, zachycuje vztahy mezi prostorovými datovými typy (dědičnost)
- Prostоровé datové typy reprezentují 0, 1 a 2 dimenzionální geometrické útvary
 - Prostоровé datové typy jsou založeny na 2D geometrii s lineární interpolací mezi vrcholy
- Obsahuje abstraktní typy, od kterých nelze vytvářet instance (ST_Surface...)

2017

48

SQL geometrický model

- Vychází z geometrického modelu OGC



2017

49

0-dimenzionální objekty

- Bod:
 - ST_Point
 - X a Y souřadnice vzhledem k nějakému systému souřadnic
 - ST_MultiPoint
 - Kolekce bodů

2017

50

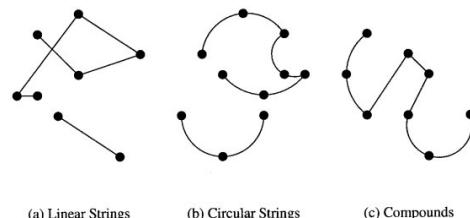
1-dimenzionální objekty

- Křivky:
 - ST_LinearString
 - ST_CircularString
 - ST_CompoundCurve
 - Kombinace ST_LinearString a ST_CircularString
 - ST_MultiLineString
 - Neexistují ST_MultiCircularString a ST_MultiCompoundString

2017

51

1-dimenzionální objekty



2017

52

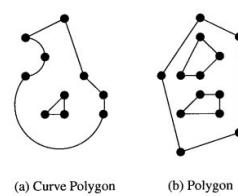
2-dimenzionální objekty

- Povrchy
 - ST_CurvePolygon
 - ST_Polygon (potomek ST_CurvePolygon)
 - ST_MultiPolygon
- Hranice (boundary) je reprezentována uzavřenou křivkou (resp. křivkami, obsahuje-li povrch díry)

2017

53

2-dimenzionální objekty



2017

54

Rozdíly SQL oproti OGC modelu

- SQL model vypouští typy Line a LinearRing, místo nich zaveden nový typ ST_LineString
- Zavedeny nové typy pro reprezentaci křivek a povrchů tvořených kruhovými oblouky (ST_CircularString...)

2017

55

Rozdíly SQL oproti OGC modelu

- U agregovaných typů (kolekcí) není z SQL modelu zřejmé, ze kterých „jednoduchých“ typů jsou složeny
 - Není například zřejmé, zda se typ ST_MultiPoint skládá z objektů typu ST_Point

2017

56

Prostorové systémy souřadnic

- Každý prostorový (geometrický) objekt je asociován s nějakým prostorovým systémem souřadnic (Spatial Reference System)
- Mnoho druhů souřadnicových systémů
 - U některých je třeba brát v úvahu zakřivení zemského povrchu

2017

57

Prostorové systémy souřadnic

- Reprezentují se typem ST_SpatialRefSys
- Všechny hodnoty sloupců reprezentujících prostorová data (prostorové atributy) v SQL dotazu musí být definovány ve stejném souřadnicovém systému

2017

58

Prázdné prostorové objekty

- SQL geometrický model nezahrnuje samostatný typ pro reprezentaci prázdného prostorového objektu
- Instance každého typu může obsahovat prázdný prostorový objekt

2017

59

Prázdné prostorové objekty

- Pokud je návratovou hodnotou nějaké metody prázdný prostorový objekt a návratový typ není přesně definován, je vrácen jako prázdný prostorový objekt bod
- Mezi prázdnými prostorovými objekty lze provádět implicitní i explicitní přetypování

2017

60

Operace na prostorových objektech

- Průnik
- Sjednocení
- Relace „sousedí“
 - Objekt A sousedí s objektem B
- Relace „obsahovat“
 - Objekt A obsahuje objekt B
- Vzdálenost dvou objektů
- ...

2017

61

Typické dotazy

- Najdi všechna města vzdálená od Londýna nejvíce 50 km
- Najdi 5 nejbližších měst k Londýnu
- Najdi všechny dvojice měst, která jsou do sebe vzdálena nejvýše 200 km
- Najdi mi všechny budovy, které sousedí s vlakovým nádražím

2017

62

Metody definované na prostorových datových typech

- Rozdeleny do čtyř základních kategorií:
 - Konverze prostorových objektů do/z externích datových formátů
 - Práce s atributy prostorových objektů
 - Porovnávání prostorových objektů
 - Generování nových prostorových objektů

2017

63

Konverze do/z externích datových formátů

- SQL/MM standard definuje tři implementačně nezávislé externí datové formáty pro reprezentaci prostorových objektů:
 - Textová reprezentace
 - Well-known text representation (WKT)
 - Binární reprezentace
 - Well-known binary representation (WKB)
 - Geography markup language (GML)

2017

64

Externí datové formáty

- Textová reprezentace
 - point (10 10)
 - multipolygon (((1 1, 2 2, 1 2, 1 1),((10 10, 10 20, 20 20, 20 10, 10 10)))
- Binární reprezentace
 - 010100000000001010

2017

65

Externí datové formáty

- Geography markup language

```
<gml:Polygon srsName="EPSG:4326">
  <gml:outerBoundaryIs>
    <gml:LinearRing>
      <gml:coordinates>
        <gml:ctuple>0,0</gml:ctuple>
        <gml:ctuple>50,0</gml:ctuple>
        <gml:ctuple>50,40</gml:ctuple>
        <gml:ctuple>0,0</gml:ctuple>
      </gml:coordinates>
    </gml:LinearRing>
  </gml:outerBoundaryIs>
```

2017

66

Konverze z externích datových formátů

- Pomocí konstruktorů prostorových datových typů
 - Pouze z WKT a WKB reprezentace
- Z GML pomocí speciálních funkcí
 - ST_LineFromGML, ST_MPointFromGML...
 - Pro zpětnou kompatibilitu OGC specifikací
 - ST_PointFromText...

2017

67

Konverze do externích datových formátů

- Tři metody
 - ST_AsText
 - ST_AsBinary
 - ST_AsGML

2017

68

Práce s atributy prostorových objektů

- Všechny prostorové datové typy mají nějaké společné atributy (př. dimenze)
- Každý podtyp v SQL geometrickém modelu si přidává další, jemu specifické, atributy
 - Př. Polygon – plocha, kterou zabírá

2017

69

Příklady metod

- ST_Boundary
 - Vrací hranici prostorového objektu
- ST_IsEmpty
- ST_X
 - Vrací X-ovou souřadnici bodu
- ST_Length
 - Vrací délku křivky

2017

70

Porovnávání prostorových objektů

- ST_Equals
 - Test na prostorovou rovnost dvou objektů
- ST_Intersect, ST_Crosses, ST_Overlaps
 - Test na prostorový průnik dvou objektů
- ST_Touches
 - Test, jestli se objekty dotýkají hranicemi, ale neprotínají se
- ST_Within, ST_Contains
 - Test, jestli jeden objekt obsahuje jiný

2017

71

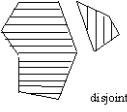
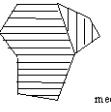
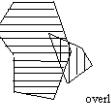
Porovnávání prostorových objektů

- Metody pro porovnávání objektů jsou založeny na binárních prostorových predikátech
- Všechny výše uvedené metody vracejí 1 pokud je příslušná relace splněna (TRUE), jinak vrací 0
- Další metody
 - ST_Distance

2017

72

Binární prostorové predikáty

-  disjoint
-  meet
-  overlap
-  covers/
covered_by
-  inside/
contains
-  equal

2017

73

Rozdíl mezi predikáty contains a covers

- A covers B nabývá TRUE, když $\text{interior}(B) \subset \text{interior}(A)$ a současně $\text{boundary}(A) \cap \text{boundary}(B) \neq \emptyset$
- A contains B nabývá TRUE, když $\text{interior}(B) \cup \text{boundary}(B) \subset A$

2017

74

Generování nových prostorových objektů

- Nové objekty mohou být výsledkem provedení nějaké operace na množině stávajících objektů
 - ST_Intersection
 - ST_Union
 - ST_Difference

2017

75

Generování nových prostorových objektů

- Mohou být získány pomocí nějakého algoritmu provedeného na jednom objektu
 - ST_Buffer („nárazníková“ zóna)
 - ST_ConvexHull (konvexní obal)
 - ST_Envelope
 - Vrací minimální ohrazení obdélník (MOO) k danému objektu

2017

76

Přehled metod

| | | | |
|--------------------|------------------|--------------------|---------------|
| ST_Geometry | ST_Dimension | ST_Length | ST_Distance |
| ST_Point | ST_CoordDim | ST_StartPoint | ST_Equals |
| ST_LineString | ST_GeometryType | ST_EndPoint | ST_Relate |
| ST_CircularString | ST_SRID | ST_IsClosed | ST_Disjoint |
| ST_CompoundCurve | ST_IsEmpty | ST_IsRing | ST_Intersects |
| e | ST_IsSimple | ST_NumPoints | ST_Touches |
| ST_CurvePolygon | ST_IsValid | ST_PointN | ST_Crosses |
| ST_Polygon | | ST_NumCurves | ST_Within |
| ST_GeomCollection | ST_Boundary | ST_CurveN | ST_Contains |
| ST_MultiPoint | ST_Envelope | | ST_Overlaps |
| ST_MultiLineString | ST_ConvexHull | ST_Area | |
| ST_MultiPolygon | ST_Buffer | ST_Perimeter | |
| ST_AsText | ST_Intersection | ST_Centroid | |
| ST_AsBinary | ST_Union | ST_PointOnSurface | |
| ST_AsGML | ST_Difference | ST_ExteriorRing | |
| | ST_SymDifference | ST_InteriorRings | |
| ST_Transform | | ST_NumInteriorRing | |
| | ST_X | ST_InteriorRingN | |
| | ST_Y | | |

2017

77

Rozšiřování SQL geometrického modelu

- Při rozšiřování geometrického modelu o nový datový typ je třeba vytvořit ještě další typ reprezentující množinu objektů takového typu
 - Viz např. ST_Polygon a ST_MultiPolygon

2017

78

Nedostatky SQL geometrického modelu

- Mějme následující dotaz:

```
SELECT ST_Intersection(ST_Polygon(  
    'polygon((10 10, 10 20, 20 20, 20  
    10, 10 10))', 1),  
    mnozina_bodu)  
FROM tabulka  
WHERE ...
```

- Sloupec mnozina_body obsahuje hodnoty typu ST_MultiPoint

2017

79

Nedostatky SQL geometrického modelu

- Výsledek dotazu může být

- Prázdný objekt
- Jeden bod (ST_Point)
- Množina bodů (ST_MultiPoint)

- Jeho získání ale není triviální, neboť ST_Point a ST_MultiPoint se nachází v různých větvích geometrického modelu a tudíž můžeme použít pouze obecných metod definovaných v ST_Geometry

2017

80

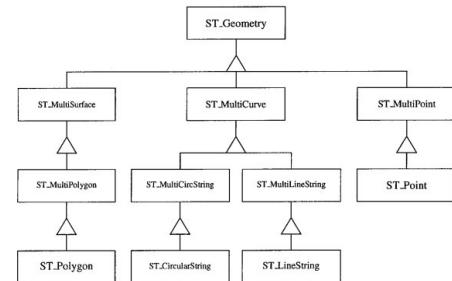
Nedostatky SQL geometrického modelu

- Řešení problému
 - Modifikace geometrického modelu

2017

81

Modifikovaný geometrický model



2017

82

Diskuse k metodám

- Chybí konstruktory využívající GML
 - Řešení: Využití WKT konstruktorů
- Chybí podpora dalších důležitých externích formátů
 - Př. „shape format“

2017

83

Diskuse k metodám

- Některé metody poskytují téměř stejnou funkcionality
 - ST_Intersects
 - ST_Crosses
 - ST_Overlaps

2017

84

Příklady

2017

85

Příklad 1: Pojišťovna

2017

86

Pojišťovna

- Pojišťování budov v záplavových oblastech

2017

87

Pojišťovna

```
CREATE TABLE reky {
    nazev          VARCHAR(30) PRIMARY KEY,
    mnozstvi_vody  DOUBLE PRECISION,
    koryto         ST_LineString,
    zaplavova_oblast ST_MultiPolygon
}

CREATE TABLE budovy {
    zakaznik      VARCHAR(50) PRIMARY KEY,
    ulice         VARCHAR(50),
    mesto          VARCHAR(20),
    zip            VARCHAR(10),
    pozemek        ST_Polygon
}
```

2017

88

Pojišťovna

- Rozšíření záplavové oblasti řeky FLOOD o 2 km ve všech směrech

```
UPDATE reky
SET zaplavova_oblast =
    zaplavova_oblast.ST_Buffer(2,
    'KILOMETER')
WHERE nazev='FLOOD'
```

2017

89

Pojišťovna

- Najdi všechny zákazníky, jejichž budovy se nachází v záplavové oblasti řeky FLOOD

```
SELECT zakaznik, ulice, mesto, zip
FROM budovy AS b, reky AS r
WHERE b.pozemek.ST_Within(
    r.zaplavova_oblast) = 1
```

2017

90

Příklad 2: Banka

2017

91

Banka

- Síť poboček
- Každý zákazník může mít založen jeden, nebo více účtů
- Všechny účty jednoho zákazníka jsou spravovány právě jednou pobočkou

2017

92

Banka

```
CREATE TABLE zakaznici {  
    id      INTEGER PRIMARY KEY,  
    jmeno   VARCHAR(20),  
    ulice   VARCHAR(25),  
    mesto   VARCHAR(10),  
    stat    VARCHAR(2),  
    zip     VARCHAR(5),  
    typ     VARCHAR(10),  
    lokace  ST_Point  
)
```

2017

93

Banka

```
CREATE TABLE poboxy {  
    id      INTEGER PRIMARY KEY,  
    nazev   VARCHAR(12),  
    manager  VARCHAR(20),  
    ulice   VARCHAR(20),  
    mesto   VARCHAR(10),  
    stat    VARCHAR(2)  
    zip     VARCHAR(5),  
    lokace  ST_Point,  
    oblast  ST_Polygon  
)
```

2017

94

Banka

```
CREATE TABLE ucty {  
    id      INTEGER PRIMARY KEY,  
    id_vlastnika INTEGER NOT NULL,  
    id_poboxy  INTEGER NOT NULL,  
    typ     VARCHAR(10) NOT NULL,  
    zustatek  DECIMAL(14, 2) NOT NULL,  
    CONSTRAINT fk_zakaznici FOREIGN  
        KEY(id_vlastnika) REFERENCES zakaznici(id),  
    CONSTRAINT fk_poboxy FOREIGN KEY(id_poboxy)  
        REFERENCES poboxy(id)  
)
```

2017

95

Banka

- Najdi všechny zákazníky se zůstatkem na účtu větším než \$10 000, a kteří bydlí dále jak 20 mil od příslušné pobočky, která jim účet spravuje

```
SELECT DISTINCT z.id, z.jmeno  
FROM zakaznici AS z JOIN ucty AS u ON  
    (z.id=u.id_vlastnika)  
WHERE u.zustatek > 10000 AND  
    z.lokace.ST_Distance((SELECT p.lokace  
    FROM poboxy AS p WHERE p.id=u.id_poboxy),  
    'MILES') > 20
```

2017

96

Banka

- Najdi všechny dvojice poboček, jejichž oblasti působnosti se překrývají

```
SELECT p1.id, p2.id,
       p1.oblast.ST_Overlaps(p2.oblast) .
       ST_AsText()
  FROM poboicky AS p1 JOIN poboicky AS p2 ON
       (p1.id < p2.id)
 WHERE p1.oblast.ST_Overlaps(p2.oblast) .
       ST_IsEmpty() = 0
```

2017

97

Banka

- Najdi všechny zákazníky, kteří bydlí v okruhu 10 mil od nějaké pobočky, která nespravuje jejich účty

```
SELECT z.jmeno, p.id
  FROM poboicky AS p, zakaznici AS z
 WHERE p.lokace.ST_Buffer(10,
       'MILES') .ST_Contains(z.lokace)
   = 1 AND NOT EXISTS (SELECT 1 FROM ucty AS u
      WHERE u.id_vlastnika = z.id AND
            p.id = u.id_poboicky)
```

2017

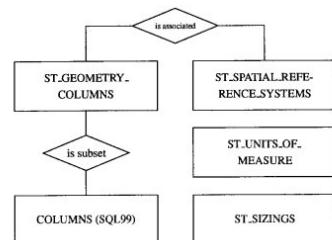
98

Datový katalog

2017

99

Pohledy (VIEWS)



2017

100

ST_GEOGRAPHY_COLUMNS

- Záznamy o všech sloupcích (atributech), které jsou deklarovány jako prostorový datový typ
- Ke každému sloupci může být přidružen patřičný souřadnicový systém

2017

101

ST_GEOGRAPHY_COLUMNS

- Jeden záznam pohledu se skládá z
 - Identifikátoru sloupce (attribut)
 - Katalog
 - Schéma
 - Tabulka
 - Název sloupce
 - Identifikátoru souřadnicového systému
 - Název souřadnicového systému
 - Numerický identifikátor

2017

102

ST_SPATIAL_REFERENCE_SYSTEMS

- Informace o definovaných souřadnicových systémech

2017

103

ST_UNITS_OF_MEASURE

- Informace o matematických jednotkách využívaných při výpočtech vzdálenosti dvou objektů, délky křivky...
- Kádá jednotka má
 - Název (KILOMETER, RADIAN..)
 - Typ (úhlová, délková)
 - Konverzní faktor vůči základní jednotce daného typu

2017

104

ST_SIZINGS

- Různé meta-proměnné a jejich hodnoty
 - Př. ST_MaxGeometryAsText
 - Maximální délka textové reprezentace (WKT) prostorového objektu

2017

105

Diskuse k datovému katalogu

- ST_SPATIAL_REFERENCE_SYSTEMS
 - Celkem primitivní
 - Práce na propracovanější organizaci souřadnicových systémů (EPSG - The European Petrol Survey Group)
- ST_SIZINGS
 - Podobný pohledu SIZING definovanému v SQL99

2017

106

Závěr

2017

107

Produkty

- IBM DB2 Spatial Extender
- IDS Spatial DataBlade
- Oracle 9i Spatial

2017

108

Možnosti implementace

- Jako příklady implementace alternativ (1) a (3) lze uvést prostorové moduly ORACLE 8 (pomocí SQL92) a od verze ORACLE 8i (pomocí objektově relačního přístupu (Pozn.: Další pionýrskou implementací OGC specifikace byl Spatial Database Engine (SDE) vytvořený pracovištěm ESRI (Environmental Systems Research Institute) a využity pro Spatial Extender v SŘBD DB2.).)

2017

109

Shrnutí

- SQL-MM Spatial
 - Standardizuje ukládání, výběr, dotazování a aktualizaci prostorových objektů
 - Definuje množinu typů a metod pro reprezentaci 0, 1 a 2 dimenzionálních prostorových objektů
- Připravuje se druhá verze standardu

2017

110

Literatura

- K. Stolze: SQL/MM Spatial:The Standard to Manage Spatial Data in Relational Database Systems
<http://www.btw2003.de/proceedings/paper/68.pdf>
- J.Pokorný: Prostоровé objekty a SQL
http://gis.vsb.cz/Publikace/Sborniky/GIS_Ova/gis_ova_2001/sbornik/Referaty/pokorny.htm

2017

111

Obsah

- Úvod
- Část 1: Framework
- Část 2: Full-Text
- Část 3: Spatial
- **Část 4: General Purpose Facilities**
- Část 5: Still Image
- Část 6: Data Mining

2017

112

Vysvětlení návaznosti ve vývoji SQL

2017

113

ISO/IEC 9075

- Multi-part Standard
 - Part1: SQL/Framework
 - Part2: SQL/Foundation
 - Part3: SQL/CLI
 - Part4: SQL/PSM
 - Part5: SQL/SQLJ
 - Part10: SQL/OB
 - Part11: SQL/Schemata
 - Part13: SQL/JRT
 - Part14: SQL/XML

2017

114

SQL:1999

- Pět částí:
- SQL/Framework
 - SQL/Foundations
 - SQL/CLI (Call Level Interface) 
 - SQL/PSM (Persistent Store Modules **)
 - SQL/Bindings***
- * alternativa k volání SQL z aplikačních programů
** procedurální jazyk pro psaní transakcí (viz Microsoft ODBC)
*** dynamický, vnořený, přímý SQL

2017

115

SQL:1999

- podpora objektů
- uložené procedury
- triggers
- rekurzivní dotazy
- rozšíření pro OLAP (např. operátor CUBE)
- procedurální konstrukty
- výrazy za ORDER BY
- savepoints
- update prostřednictvím sjednocení a spojení

2017

116

Co nového po r. 1999

- číslo 8 neodpovídá žádné části.
 - část 9 – SQL/MED (Management of External Data)
 - část 10 - SQL/OLB (Object Language Bindings)
 - 2002 - dokončeno pět částí SQL/MM
 - Základy (Framework),
 - Úplné texty (Full Text),
 - Prostorové objekty (Spatial),
 - General Purpose Facilities (materiál společný ostatním částem, např. datové typy).
 - Nepohyblivé obrazy (Still Image).
- Zpoždění: část 7 – SQL/Temporal, část 11 - SQL/Schemata a část 14 SQL/XML.

2017

117

General Purpose Facilities

- Pokus o množinu tříd pro hlavní matematické operace
- Tato část ze standardu prozatím vypadla
 - Vysoké náklady
 - Málo uživatelů

2017

118

Obsah

- Úvod
- Část 1: Framework
- Část 2: Full-Text
- Část 3: Spatial
- Část 4: General Purpose Facilities
- **Část 5: Still Image**
- Část 6: Data Mining

2017

119

Still Image > Úvod

- Obrázky = hodnotná data
- Ukládání obrázků
- Úprava obrázků
- Vyhledávání obrázků (dle vizuálních vlastností)

2017

120

Still Image > Typy

- **SI_StillImage**
 - Pro obrazová data
- **SI_Feature**
 - Nadtyp pro různé vlastnosti obrázku (dále)
- **SI_FeatureList**
 - Seznam pro (všechny) vlastnosti obrázku

2017

121

Still Image > SI_StillImage (1)

- Konstruktory
 - BLOB
 - BLOB + Formát (JPEG, TIFF, GIF, ...)
- Metody pro přeformátování (SI_changeFormat)
- Vytvoření miniatury („Thumbnails“)
- Změna velikosti
- Ořezávání
- Rotace
- ...

2017

122

Still Image > SI_StillImage (2)

```
create type SI_StillImage as (  
  SI_content binary large  
    object(SI_MaxContLength),  
  SI_contentLength integer,  
  SI_format character varying(8),  
  SI_height integer,  
  SI_width integer,  
  ...  
)
```

2017

123

Still Image > SI_Feature (1)

- Užitečné pro vyhledávání
- Typ SI_Feature má tyto podtypy
 - SI_AverageColor
 - SI_ColorHistogram
 - SI_PositionalColor
 - SI_Texture
- Všechny vlastnosti mají metodu SI_Score, která spočítá podobnost vlastností obrázků a vrátí reálnou hodnotu mezi 0 a 1

2017

124

Still Image > SI_Feature (2)

```
create type SI_AverageColor under SI_Feature  
  (SI_AverageColorSpec SI_Color)  
  
method SI_AverageColor (  
  RedValue integer,  
  GreenValue integer,  
  BlueValue integer)  
  returns SI_AverageColor  
  
create function SI_AverageColor (image  
  SI_StillImage)  
  returns SI_AverageColor
```

2017

125

Still Image > Příklad > dotaz

```
SELECT *  
  FROM Registrovaná_logo  
 WHERE  
   SI_findTexture(Naše_logo).  
   SI_Score(Logo) > 0.9;
```

2017

126

Obsah

- Úvod
- Část 1: Framework
- Část 2: Full-Text
- Část 3: Spatial
- Část 4: General Purpose Facilities
- Část 5: Still Image
- **Část 6: Data Mining**

2017

127

Data Mining > Úvod

- Aplikační balíček
- **Data Mining**
 - Analýza (často rozsáhlých) observačních dat s cílem nalézt netušené vztahy a sumarizovat data novými způsoby tak, že jsou srozumitelná a užitečná pro jejich majitele.

2017

128

Data Mining > Techniky

- **RULE MODEL**
 - Hledá pravidla a vztahy v datech.
- **CLUSTERING MODEL**
 - Slučuje data do skupin s podobnými charakteristikami.
- **REGRESSION MODEL**
 - Předvírá klasifikaci nově pořizovaných dat.
- **CLASSIFICATION MODEL**
 - Předvírá klasifikaci (způsob shlukování), která bude nejlépe odpovídat novým datům.

2017

129

Data Mining > Stupně DM

- Vytvoření (train) DM modelu
- Testování modelu (u regression a classification technik)
- Aplikace na primární data

2017

130

Data Mining > Typy

- **DM_*Model** (Definice modelu)
 - * ≈ 'Rule', 'Clus', 'Clas' nebo 'Reg'
- **DM_*Setting** (Nastavení modelu)
- **DM_MiningData** (Testování modelu)
- **DM_*TestResult** (Výsledek testování modelu)
- **DM_*Result** (Výsledek aplikace modelu)
- **DM_*Task** (Řízení běžícího modelu)

2017

131

Závěr

- Budoucnost SQL/MM
 - SQL/MM General Purpose Facilities
 - SQL/MM Moving Images

2017

132

• **Unstructured Data and Content Management**

ORACLE11g Release 2 (11.2)

2017

133

Oracle Secure Files and Large Objects

- Oracle Secure Files, providing enhanced large object (LOB) storage functionality with file system performance, provides a foundation for a wide range of content management applications.

2017

134

XML

- Oracle XML DB and XML Developer's Kit enable you to develop performant applications that process XML content and manage XML stored natively in the database.

2017

135

Oracle Text and Ultra Search

- Oracle Text brings search engine-like full text search capabilities to the Oracle Database. Ultra Search provides a ready-to-use application, while Oracle Text provides a foundation for building your own search applications. Search regular columns, text inside various kinds of binary-format documents, and text, tags, and attributes inside XML documents.

2017

136

Oracle Spatial and Location Information

- Use features described in these manuals to implement applications that manage data with spatial organization.

2017

137

Oracle Multimedia

- Oracle Multimedia (formerly known as Oracle interMedia) lets you write applications to manage images, audio, video, and other heterogeneous media data in Oracle databases.

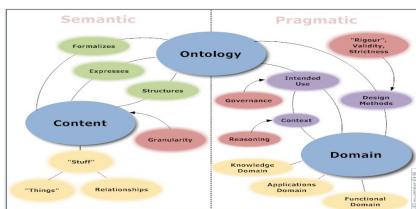
2017

138

Zdroje

- SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems
 - <http://www.btw2003.de/proceedings/paper/68.pdf>
- SQL Multimedia and Application Packages (SQL/MM)
 - <http://dbs.uni-leipzig.de/en/lehre/sql-multimedia-2001-records.pdf>
- Oracle® Spatial User's Guide and Reference: SQL Statements for Indexing Spatial Data
 - https://cwisdb.cc.kuleuven.ac.be/ora10doc/appdev.101/b10826/sdo_objindex.htm
- <http://wwwdv.informatik.uni-kl.de/courses/IMDB/WS2003/Vorlesungsunterlagen/SQL-MM.half.pdf>
(nemecky)

Sémantický web, ontologie. Sociální sítě.



1

Sémantický web

- Metody a techniky pro přiřazení významu (sémantiky) informacím na webu
- Web rozšířený o metadata
- Metadata** = data o datech
- Postaven na formátu **RDF**

2

Cíle sémantického webu

- Integrovat data** z různých zdrojů
- Umožnit **výměnu dat** mezi aplikacemi napříč celým webem
- Umožnit **kvalitnější strojové vyhledávání** informací na webu
- Umožnit **popsat vztahy** mezi daty a objekty v reálném světě
- Přiřadit** informacím na webu přesný **význam**

3

Metadata v HTML

- Pomocí **<meta>** tagů:
`<meta name="keywords" content="HTML, CSS, XML" />`
- Cíl: umožnit kvalitnější vyhledávání, než obyčejný full-text search
- Zneužíváno ve velké míře spammery
- Neumožňuje definovat vztahy a hierarchie objektů
- Dnes vyhledávače dávají přednost jiným metodám, než prohledávání **<meta>** tagů

4

RDF

- RDF** = Resource Description Framework
- Framework pro popis zdrojů na webu
- Navržen tak, aby byl strojově čitelný a pochopitelný
- Doporučení W3C
- Různé způsoby serializace (uložení do souboru), př. **RDF/XML**

5

Princip RDF

- Každému zdroji na webu přiřadí trojici:
 - Subject (subjekt, podmět)
 - Predicate (predikát, vlastnost)
 - Object (objekt, předmět)
- Při definici subjektů a predikátů je typicky potřeba definovat **URI** (Unique Resource Identifier) pro jednoznačné přiřazení významu.
- RDF dokumenty lze ukládat do **triples store** databází (databáze optimalizované pro RDF trojice) nebo serializovat pomocí XML (formát **RDF/XML**)

6

RDF/XML

- Příklad: „Obloha má modrou barvu.“
 - Podmět: „obloha“
 - Vlastnost: „mít barvu“
 - Předmět: „modrá“ („blue“)
- Serializace ve formátu RDF/XML:

```

1: <?xml version="1.0"?>
2:
3: <rdf:RDF
4:   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5:   xmlns:sky="http://fi.muni.cz/rdf/sky/">
6:   <rdf:Description rdf:about="http://fi.muni.cz/rdf/sky">
7:     <sky:color>blue</sky:color>
8:   </rdf:Description>
9: </rdf:RDF>
```

7

Triplestores

- Databáze optimalizované pro ukládání RDF trojic (subjekt, predikát, objekt)
- Mnoho implementací v různých jazycích (C, C#, PHP, Java, Perl)
- Postaveny buď nad existujícím relačním databázovým strojem (MySQL, PostgreSQL, MS SQL, Oracle), nebo vyvinuty kompletně od začátku přesně pro svůj účel (vyšší efektivita)

8

Ontologie

- Model pro popis světa složeného z typů, vlastností a vztahů
- Využití v sémantickém webu pro přiřazení významu datům (tj. pro tvorbu metadatového modelu)
- Při tvorbě ontologií je snaha o co nejpřesnější podobnost mezi objekty reálného světa a vlastnostmi modelu

9

Kategorie ontologií

- **Individua** (instance a objekty)
- **Třídy** (množiny, kolekce, pojmy, typy, druhy)
- **Atributy** (aspekty, stavy, vlastnosti, charakteristiky a parametry, kterých mohou objekty/třídy nabývat)
- **Relace** (způsoby, jakými k sobě mohou třídy a individua navzájem patřit)
- **Funkční výrazy** (komplexní struktury nad relacemi)

10

Kategorie ontologií

- **Restrikce** (formální popis platného vstupu)
- **Pravidla** (Příkazy ve formě if-then (příčin-následek) popisující logické inference, které mohou být odvozeny z výroků v dané formě)
- **Axiomy** (výroky (vč. pravidel) v logické formě, které dohromady skládají kompletní teorii, kterou ontologie popisuje. Nemusí obsahovat pouze apriorní znalosti, ale také odvozené teorie z jiných axiomů.)
- **Události** (změny atributů a relací)

11

Inference znalostí

- Pojem **inference**
 - 1) dobře navržená logická heuristika pro odvozování nových znalostí
 - 2) odvozená znalost
- **Inference znalostí** - odvozování nových znalostí na základě existujících (známých) znalostí (inferencí)
- Využití v sémantickém webu při **strojovém vyhledávání** nových znalostí

12

Inferenční enginy

- Počítačové programy, které zkouší odvodit odpověď z **báze znalostí** (knowledge base, množina axiomů/výroků/faktů/znalostí/popř. inferencí)
- Data v bázi znalostí musí být uložena takovým způsobem, aby stroj/engine dokázal odvodit a porozumět jejich významu, tj. musí být explicitně vyjádřena jejich **sémantika** (samotná data musí být doplněna o **metadata**)

13

SPARQL [„spa:kl“]

- Jazyk / protokol pro inferenci znalostí z RDF dokumentů
- Umožňuje provádět dotazy nad RDF trojicemi (triplestore databázemi)
- Podobná syntax jako SQL
- Výhoda SPARQL: dotazy jsou díky přítomnosti URI v RDF formátu globálně jednoznačné

14