Vnitřní“ programovací konstrukce (Embedded SQL)- procedurální prostředky v rámci jazyka SQL, jazyk PL/SQL

Z Na státnice zvesela!

Obsah

Embedded SQL

Přímý přístup programovacího jazyka do databázových struktur.

- SQL dotazy jsou psány přímo ve zdrojovém kódu. Syntaxe SQL je přizpůsobena syntaxi daného programovacího jazyka.
- Před kompilací programu se musí zdrojový kód zpracovat preprocesorem Embedded SQL, kdy SQL dotazy jsou nahrazeny odpovídajícím kódem programovacího jazyka.
- Nejčastěji v kombinaci s jazyky C/C++.

Podpora v databázích

Podporují

- IBM DB2 (C/C++, Java, Cobol)
- Oracle (Cobol, *Pro*C* - embedded SQL Oraclu pro C/C++)
- PostgreSQL (C/C++)

Nepodporují

- MySQL
- Microsoft SQL Server (starší verze podporovali C)

Příklad syntaxe

Oracle Embedded SQL v jazyce C:

```
{
    int a;
    /* ... */
    EXEC SQL SELECT salary INTO :a
             FROM Employee
             WHERE SSN=876543210;
    /* ... */
    printf("The salary is %d\n", a);
    /* ... */
}
```

Procedurální prostředky SQL (procedurální rozšíření SQL)

- SQL bylo původně navrženo pro získávání dat z relačních databází. SQL je deklarativní jazyk a ne imperativní, jako například C nebo Java.
- Dodavatelům DB systémů možnosti SQL nedostačovali a tak si začali implementovat vlastní procedurální rozšíření SQL.
- Standard **SQL:1999** přináší procedurální podporu pro SQL.

Procedurální rozšíření v SQL:1999

SQL:1999 standardizuje procedurální rozšíření. Rozšíření se jmenuje *SQL/PSM - SQL/Persisted Stored Modules*

- funkce a procedury - lze zapsat v SQL i v hostitelském programovacím jazyce
- řídící konstrukce - cykly, větvení, přiřazení

Příklad:

```
CREATE OR REPLACE FUNCTION hello(uid integer)
RETURNS varchar AS
$$
  BEGIN
    DECLARE real_name varchar;
    -- Get real name
    SET real_name = (SELECT name || ' ' || surname
                       FROM Users
                      WHERE Users.uid = hello.uid);
    RETURN 'Hello, ' || real_name;
  END;
$$ LANGUAGE plpgpsm;
```

```
SELECT hello(123);
```

Podpora v databázích

SQL:1999 (SQL/PSM)

Standardizované procedurální rozšíření SQL/PSM je v současné době implementováno v několika DB systémech.

- IBM DB2
- MySQL
- PostgreSQL

Vlastní (proprietární) rozšíření

- Oracle (PL/SQL)
- PostgreSQL (PL/pgSQL)
- Microsoft (T-SQL)
- Sybase (T-SQL)

Motivace

- **Šetrení komunikacního kanálu**
  - Menší množství odesílaných povelů
  - v jednom povelu je vetší množství příkazů
- **Podstatne menší objem prenesených dat** - Data se zpracují na serveru bez přenosu na klienta
- **Odlehčení klienta** - Možnost ukládat a vykonávat kód na serveru

Na co si dát při návrhu pozor

- **Hlídat na úrovni databáze všechny manipulace s daty, které ohlídat jdou**
  - Cokoli jde zadat uživatelem špatne, bude zadáno špatne
- **Integritní omezení, triggery**
  - Pozdeji je cištení nekonzistentních dat namáhavé a opravy casto nemožné
  - Lépe ohlídat vše

Jazyk PL/SQL

PL/SQL (Procedural Language/Structured Query Language) je procedurální nadstavba jazyka SQL od firmy Oracle založená na programovacím jazyku Ada.

Tato nadstavba se rozšířila a její deriváty převzaly i jiné relační databáze. Sybase a Microsoft SQL Server mají Transact-SQL, PostgreSQL má PL/pgSQL a IBM DB2 má SQL PL. Existuje též projekt Fyracle, jehož cílem je umožnit spouštění PL/SQL v relační databázi Firebird.

PL/SQL přidává k jazyku SQL konstrukce procedurálního programování. Výsledkem je strukturální jazyk, mocnější než samotné SQL. Základním stavebním kamenem v PL/SQL je blok. Program v PL/SQL se skládá z bloků, které mohou být vnořeny jeden do druhého. Obyčejně každý blok spouští jednu logickou akci v programu. Blok má následující strukturu:

Základní konstrukce

```
DECLARE
 deklarace
BEGIN
 výkonná cást
EXCEPTION
 ošetrení výjimek
END;
```

Pouze výkonná sekce je povinná, ostatní jsou doporučené. Jediné příkazy jazyka SQL, které jsou ve výkonné sekci povolené, jsou SELECT, INSERT, UPDATE, DELETE a několik dalších pro manipulaci s daty a pro kontrolu transakcí. Definiční příkazy jazyka SQL jako CREATE, DROP nebo ALTER nejsou povoleny. PL/SQL není citlivé na velikost písmen a mohou být použity komentáře ve stylu jazyka C.

Datový slovník

- **USER_OBJECTS** – Informace o externích procedurách a funkcích
  - OBJECT_TYPE je ’PROCEDURE’ / ’FUNCTION’
  - STATUS je ’VALID’ (zkompilován) / ’INVALID’
- **USER_SOURCE** – Zdrojový kód procedur, funkcí, …
- **USER_ERRORS**– Informace o chybách v procedurách, funkcích, …

--Kubas 19:53, 17. 5. 2010 (UTC)

Citováno z „http://wiki.zvesela.cz/index.php/Vnit%C5%99n%C3%AD%E2%80%9C_programovac%C3%AD_konstrukce_%28Embedded_SQL%29-_procedur%C3%A1ln%C3%AD_prost%C5%99edky_v_r%C3%A1mci_jazyka_SQL%2C_jazyk_PL/SQL“

- Stránka byla naposledy editována 31. 5. 2011 v 17:07.

Vnitřní“ programovací konstrukce (Embedded SQL)- procedurální p... http://wiki.zvesela.cz/index.php/Vnitřní“_programovací_konstrukce...

Kurzory – definice, klasifikace, použití kurzorů.

Z Na státnice zvesela!

- Kurzor je abstraktní datový typ umožňující procházet záznamy vybrané dotazem, který je s kurzorem spojen.
- prostředek pro získání informace z databáze a předání do programu v jazyce PL/SQL
- Mechanismus, kdy je možné přiřadit jméno příkazu SELECT, a manipulovat s informacemi uvnitř tohoto příkazu. (is a mechanism by which you can assign a name to a "select statement" and manipulate the information within that SQL statement.)

Obsah

Typy

- explicitní kurzor
  - nutno deklarovat, otevřít, načíst data a uzavřít
- implicitní kurzor
  - je deklarován a prováděn přímo v těle programu
  - v tomto typu kurzoru jsou povoleny pouze příkazy SQL, které vrací jednotlivé řádky nebo nevrací žádné řádky,
  - příkazy SELECT, UPDATE, INSERT a DELETE obsahují implicitní kurzory

Atributy kurzoru

- poskytují informace o stavu kurzoru
- <jméno_kurzoru>%FOUND
  - vrací TRUE, pokud při posledním FETCH načetl řádek, jinak FALSE
- <jméno_kurzoru>%NOTFOUND
  - opak FOUND
- <jméno_kurzoru>%ISOPEN
  - hodnota TRUE, pokud je kurzor otevřen, hodnota FALSE před otevřením nebo po uzavření kurzoru
- <jméno_kurzoru>%ROWCOUNT
  - atribut sdělí počet již načtených řádek, před načtením nějakého řádku má hodnotu 0. Vpříkaze UPDATE má hodnotu počtu aktualizovaných řádek.

Explicitní kurzor

- deklarace kurzoru:

  DECLARE CURSOR <jméno kurzoru>IS <dotaz>;

- otevření kurzoru:

  OPEN <jméno kurzoru>;

- načtení dat, pouze jeden řádek

  FETCH <jméno kurzoru>INTO <jméno proměnné1>, <jméno proměnné2>, …;

- zavření kurzoru

  CLOSE <jméno kurzoru>;

- Př.:

  DECLARE CURSOR kursor IS
  SELECT firma, kontakt_jmeno, adresa, mesto
  FROM zakaznici;

Implicitní kurzor

- musí se shodovat datové typy sloupců a proměnných
- platí i pro příkazy INSERT, UPDATE, DELETE
- implicitní kurzor SELECT musí vracet pouze jeden řádek

SELECT <jméno sloupce 1>, <jméno sloupce 2>INTO <jméno proměnné 1>, <jméno proměnné 2>FROM … ;

Kurzor pro cyklus LOOP

FOR rowname IN cursorname LOOP

Loop block

END LOOP;

- Př.:

```
SET SERVEROUT ON
  DECLARE
    v_firma  zakaznici.firma%TYPE;
    v_konatel  zakaznici.kontakt_jmeno%TYPE;
  CURSOR k1 IS
  SELECT firma, kontakt_jmeno  FROM zakaznici;
  BEGIN
    OPEN k1;
    LOOP
      FETCH k1 INTO v_firma, v_konatel;
      dbms_output.put_line(v_firma||' - konatel: '||v_konatel);
      EXIT WHEN k1%NOTFOUND;
    END LOOP;
    CLOSE k1;
  END;
```

Další příklad:

```
DECLARE
  tmp   osoby%ROWTYPE;
  CURSOR  plist IS
    SELECT * FROM osoby;

BEGIN
  OPEN plist;

  LOOP
    FETCH plist INTO tmp;
    EXIT WHEN plist%NOTFOUND;
    dbms_output.put_line(plist%ROWCOUNT||'. '||tmp.jmeno||' '||tmp.prijmeni);
  END LOOP;

  CLOSE plist;
END;
```

Použití kurzorů

- v triggerech
- v uložených procedurách

Citováno z „http://wiki.zvesela.cz/index.php/Kurzory_%E2%80%93_definice%2C_klasifikace%2C_pou%C5%BEit%C3%AD_kurzor%C5%AF.“

- Stránka byla naposledy editována 1. 6. 2010 v 12:16.

Uložené procedury, funkce a balíky procedur a funkcí, kompilace, spouštění.

Z Na státnice zvesela!

Obsah

Podprogram

- pojmenovaný blok, může být opakovaně volán a přebírat aktuální parametry
- typy podprogramů – procedury a funkce
- procedury a funkce lze sdružovat do logických celků – balíků (package)

Procedura (uložená)

```
CREATE  OR REPLACE PROCEDURE zrusknihu (pom_isbn CHAR) AS
BEGIN
  IF NAJDI_ZAZNAM(pom_isbn) THEN
    INSERT INTO hist_knihy (isbn) VALUES (pom_isbn);
    DELETE FROM kniha WHERE isbn=pom_isbn;
    DBMS_OUTPUT.PUT_LINE('V pořádku');
  ELSE
    INSERT INTO hist_knihy (isbn,NAZEV_K) VALUES (pom_isbn,'NENALEZEN');
    DBMS_OUTPUT.PUT_LINE('Kniha nebyla nalezena');
  END IF;
END zrusknihu;
```

Funkce (uložená)

```
CREATE FUNCTION najdi_zaznam (pom_knihy CHAR)
 RETURN BOOLEAN AS
  pocet NUMBER;
  BEGIN
     SELECT COUNT(*) INTO pocet FROM kniha WHERE isbn=pom_knihy;
     RETURN pocet=1;
  END najdi_zaznam;
```

Uložené samostatné podprogramy

- podprogramy mohou být trvale uloženy do databáze
- mohou být použity jakoukoliv aplikací, která s databázovým strojem komunikuje
- jednou přeložený a uložený podprogram patří mezi databázové objekty
- může být referován libovolným počtem aplikací
- uložené podprogramy mohou být samostatné nebo součástí balíku
- podprogramy lze volat z:
  - databázových triggerů
  - jiných uložených podprogramů
  - aplikačních programů zapsaných ve vyšším programovacím jazyce, pro nějž existuje předkompilátor (Oracle*PRO)
  - interaktivně (SQL*Plus) : EXECUTE jméno_procedury(parametry);

PACKAGE – uložené balíky

- databázový objekt, který sdružuje do skupiny logicky příbuzné typy, objekty a podprogramy
- balík dělíme obvykle na části: specifikace balíku (definuje rozhraní pro volání balíku aplikacemi (deklarace typů, proměnných, konstant, podmínek definujících nestandardní stavy, kurzorů, podprogramů dostupných „zvenčí“)) a tělo balíku (implementuje specifikaci)

```
CREATE OR REPLACE PACKAGE akce AS
  PROCEDURE vlozeni (p_isbn CHAR,p_nazev VARCHAR2,
    p_autor VARCHAR2,p_zeme CHAR);
  PROCEDURE zruseni (p_isbn CHAR);
  PROCEDURE zmena (P_ISBN CHAR,cislo NUMBER,obsah CHAR);
END akce;
```

- Ukázka vytvoření **specifikace balíku**:

```
CREATE PACKAGE jméno AS
  deklarace verejných typu a objektu
  specifikace podprogramu
END[ jméno];
```

- Ukázka vytvoření **těla balíku**:

```
CREATE PACKAGE BODY jméno AS
  deklarace lokálních typu a objektu
  tela podprogramu
[BEGIN
  inicializacní príkazy ]
END[ jméno];
```

Volání podprogramů

- při volání používáme tečkovou notaci(balík.funkce()) pro kvalifikaci jejich jména
- zabalený podprogram lze volat z databázového triggeru, jiného uloženého podprogramu, aplikace napsané pro některý z předkompilátorů, standardních klientských nástrojů (SQL*Plus)
- standardní balík (STANDARD) = definuje prostředí PL/SQL

Kompilace

- Pokud je volána procedura / funkce ve stavu INVALID, kompiluje se automaticky
  - Pokud se kompilace nezdaří, dojde k výjimce

- Ruční kompilace
  - ALTER PROCEDURE[FUNCTION] jmproc COMPILE;
  - Pokud se kompilace nezdaří, dojde k výjimce
- SQL*Plus - Získání informace o chybách
  - Pomocí príkazu SHOW ERR[OR] -- pro poslední kompilovaný objekt
  - Pomocí príkazu SHOW ERR typ jméno
    - napr. SHOW ERR FUNCTION F -- pro uvedený objekt
  - Pomocí dotazu na tabulku USER_ERRORS

- Stránka byla naposledy editována 4. 6. 2009 v 16:50.

Aktivní databáze – Oracle triggery, klasifikace a spouštění triggerů.

Z Na státnice zvesela!

- databázový trigger je procedura PL/SQL spojená s tabulkou
- trigger se automaticky provede při provádění některého příkazu SQL, je-li splněna podmínka triggeru.

Obsah

Odlišnosti triggerů od uložených podprogramů

- jsou implicitně spouštěny při modifikaci tabulky
- definují se pouze pro databázové tabulky
- nepřijímají argumenty
- lze je spustit pouze při DML příkazech UPDATE, INSERT, DELETE

Výhody triggerů

- nepovolí neplatné datové transakce
- zajišťují komplexní bezpečnost
- zajišťují referenční integritu přes všechny uzly v integrované databázi
- vytvářejí strategická a komplexní aplikační pravidla
- zajišťují audit (sledování)
- spravují synchronizaci tabulek
- zaznamenávají statistiku často modifikovaných tabulek

Příklad triggeru (zálohování dat o smazaných knihách při jejich smazání):

```
CREATE OR REPLACE TRIGGER zrusit
BEFORE DELETE ON kniha
FOR EACH ROW
BEGIN
   INSERT  INTO  zrus_knihy(isbn,autor,nazev,zrusil,datum)
                 VALUES(:OLD.isbn,:OLD.autor,:OLD.nazev,USER,SYSDATE);
END;
```

Vlastnosti triggerů

Aktivní databáze – Oracle triggery, klasifikace a spouštění triggerů. ... http://wiki.zvesela.cz/index.php/Aktivní_databáze_–_Oracle_trigger...

- CREATE OR REPLACE – může být nahrazen bez použití DROP – jedinečné jméno triggeru
- triggery jsou plně zkompilované po spuštění příkazem CREATE TRIGGER a po uložení procedurálního kódu v systémovém katalogu

Možnosti triggerů

- načasování triggeru:
  - BEFORE|AFTER|INSTEAD OF - UPDATE|INSERT|DELETE - ON <tabulka>
- volba INSTEAD OF – od Oracle 8 nemůže obsahovat množinové operátory, skupinové funkce, spojení, klauzuli DISTINCT
- volba FOR EACH ROW – určení, zda se jedná o řádkový či příkazový trigger
  - je-li klauzule uvedena, pak se trigger spouští jednou pro každý řádek
  - není-li uvedena, spustí se trigger pouze jednou bez ohledu na množství řádek
  - v těle řádkového triggeru je k dispozici jak OLD, tak NEW hodnota aktuálního řádku
- volba WHEN – podmínka určující, zda se má trigger spustit (hodnota booleovského příkazu musí být TRUE)

- Stránka byla naposledy editována 4. 6. 2008 v 20:43.

Transakce, dvoufázový uzamykací protokol, detekce uváznutí

Z Na státnice zvesela!

Databázová transakce je skupina příkazů, které převedou databázi z jednoho konzistentního stavu do druhého.

Obsah

Vlastnosti transakcí

Databázové transakce musí splňovat tzv. vlastnosti *Šablona:Cizojazyčně*:<ref>http://www.postgres.cz/index.php/Slovn%C3%ADk#ACID</ref>

- *A* - *Atomicity* - atomicita
- *C* - *Consistency* - konzistence
- *I* - *Isolation* - izolovanost
- *D* - *Durability* - trvalost

Atomicita

Databázová transakce je jako operace dále nedělitelná (atomární). Provede se buď jako celek, nebo se neprovede vůbec (a daný databázový systém to dá uživateli na vědomí, např. chybovou hláškou).

Konzistence

Při a po provedení transakce není porušeno žádné integritní omezení.

Izolovanost

Operace uvnitř transakce jsou skryty před vnějšími operacemi. Vrácením transakce (ROLLBACK) není zasažena jiná transakce, jinak i tato musí být vrácena. V důsledku tohoto chování může dojít k tzv. řetězovému vrácení (cascading rollback).

Trvalost

Změny, které se provedou jako výsledek úspěšných transakcí, jsou skutečně uloženy v databázi a již nemohou být ztraceny.

Globální vs. lokální

- **Lokální** transakce probíhá pouze na jediném uzlu.
- **Globální** (distribuovaná) transakce přesahuje rozsah jednoho uzlu.

Optimistické vs. pesimistické

Transakce se mohou lišit podle způsobu zpracování na pesimistické a optimistické.

- U **pesimistického** zpracování se v jeho průběhu změny zaznamenávají do dočasných objektů (například a nejčastěji: do řádků tabulek s příznakem dočasných dat, platných jen po dobu transakce) a teprve po přesunu/změně dat se odznačí příznak dočasnosti a data se stanou platnými. (Tento způsob se dá přibližně připodobnit přepisu souboru, při kterém se nejdříve nová verze souboru nakopíruje pod dočasným jménem a teprve poté se tento soubor přejmenuje za starý a tím ho nahradí.)
- U **optimistického** zpracování se (optimisticky) předpokládá, že při transakci nenastane chyba a nebude třeba ji vrátit zpět (přestože tato možnost je zachována). Měněné záznamy v tabulkách jsou při optimistickém zpracování transakce zapisovány „natvrdo“, současně s tím se však vytváří tzv. rollback log coby seznam SQL příkazů, které dokáží prováděné změny vrátit zpět. V případě, že při transkaci dojde k nějaké nezotavitelné chybě, tento log se provede a transakce (aby dodržela pravidlo atomicity) skončí ve výchozím stavu s chybou. Naopak, na konci transakce, při které k žádné takové chybě nedošlo, se *rollback log* maže.

Žurnály

Jsou záznamy, které uchovávají informace o průběhu transakcí a slouží k zotavení po vzniklé chybě. Žurnály musí být v každém uzlu a obsahují záznamy o historii každé transakce.

Stavy transakce

- Aktivní - od počátku provádění transakce
- Částečně potvrzený - stav po provedení poslední operace transakce
- Chybný - nelze pokračovat v normálním průběhu transakce
- Zrušený - nastane po skončení operace ROLLBACK
- Potvrzený - po úspěšném vykonání COMMIT

Prováděné operace

Pro práci s transakcemi je nutné zavést následující operace:

- *BEGIN* - začátek transakce

- *COMMIT* - ukončení transakce a uložení dosažených výsledků do databáze
- *ROLLBACK* - odvolání změn - není-li definován savepoint, (místo, po které lze provedené změny vrátit zpět) tak návrat do stavu před započetím vykonávání transakce

Dvoufázový protokol (2PL)

Dvoufázová transakce v první fázi zamyká vše co je potřeba a od prvního odemknutí (druhá fáze) již jen odemyká co měla zamčeno (již žádná operace LOCK). Tedy transakce musí mít všechny objekty uzamčeny předtím, než nějaký objekt odemkne. Dá se dokázat, že pokud jsou všechny transakce v dané množině transakcí dobře formované a dvoufázové, pak každý jejich legální rozvrh je uspořádatelný. Dvoufázový protokol zajišťuje uspořádatelnost, ale ne zotavitelnost ani bezpečnost proti kaskádovému rušení transakcí nebo uváznutí.

Striktní dvoufázový protokol (S2PL)

Problémy 2PL jsou nezotavitelnost a kaskádové rušení transakcí. Tyto nedostatky lze odstranit pomocí striktních dvoufázových protokolů, které uvolňují zámky až po skončení transakce (COMMIT). Zřejmá nevýhoda je omezení paralelismu. 2PL navíc stále nevylučuje možnost deadlocku.

Konzervativní dvoufázový protokol (C2PL)

Rozdíl oproti 2PL je ten, že transakce žádá o všechny své zámky, ještě než se začne vykonávat. To sice vede občas k zbytečnému zamykání (nevíme co přesně budeme potřebovat, tak radši zamkneme víc), ale stačí to již k prevenci uváznutí (deadlocku).

Detekce uváznutí a zotavení

- **Uváznutí se detekuje pomocí čekacího grafu**
  - Vrcholy jsou transakce T_i
  - Orientovaná hrana $T_i \rightarrow T_j$ značí, že T_i čeká, až T_j odemkne datovou položku
  - Je-li v čekacím grafu cyklus, došlo k uváznutí
- **Když se zjistí uváznutí**
  - Je nutno nalézt obětní transakci a vnutit jí **abort** (a tím i obnovu dat). Obětuje se obvykle nejmladší transakce, tj. ta, která ještě neudělala mnoho změn
  - Transakce mohou stárnout, bude-li za obět vybírána vždy nejmladší transakce. Proto je vhodné do kritéria výběru obětí zahrnout i počet transakcí provedených návratů.
  - Která data se ale mají obnovovat?
    - Totální obnova transakci úplně zruší, data se vrátí do počátečního stavu, a transakce se restartuje. To může být velmi nákladné
    - Efektivnější je, když se transakce "vrací postupně" do stavu, kdy uváznutí zmizí. Tento postup je ale náročný na evidenci kroků a změn transakcí provedených
    - Používá se např. tzv. metoda kontrolních bodů (checkpointing)

--Dejw 10:18, 31. 5. 2011 (UTC)Dejw

- Stránka byla naposledy editována 1. 6. 2011 v 14:10.

Optimalizace dotazu – Rule Based optimalizace (RBO), Cost Based optimalizace (CBO), výhody a nevýhody, výběr typu optimalizace.

Z Na státnice zvesela!

- SQL velmi flexibilní – dvěma i více různými dotazy je možné obdržet stejná data, ovšem rychlost dotazů nemusí být stejná
- důvodem optimalizace je minimalizace nákladů na:
  - zdrojový čas
  - kapacitu paměti či prostoru
  - programátorskou práci
- u malých databází optimalizace nepatrná, projeví se až u objemných x u často navštěvovaných webů může při špatně formulovaných dotazech vzrůst trafic v obou směrech

Oracle – zpracování dotazů

- zpracování příkazů se skládá z následujících komponent:
  1. parser
  2. optimalizátor
  3. generátor řádkových zdrojů
  4. vlastní provádění SQL

Optimalizátor

- jádro celého zpracování
- analyzuje sémantiku dotazu
- hledá optimální způsob jeho provádění
- rozeznáváme:
  - **rule-based optimizer** (RBO)
    - vyhodnocuje jednotlivé přístupové cesty pomocí předem daného systému pravidel, starší optimalizátor, jehož nevýhodou je možnost špatného výběru použitého indexu
    - Cena přístupu k podmnožině řádek v tabulce v klesajícím pořadí
      - Full-scan
        - Prochází se celá tabulka, u každé řádky se ověří podmínka
        - Může být vhodné, pokud procento vyhovujících řádek je dost velké
      - Index-Range-Scan
        - Vyhledání intervalu v indexu, ověření ostatních podmínek v odkazovaných řádcích
      - Unique-Index-Scan
        - Vyhledání jediné možné vyhovující řádky podle unikátního indexu
      - ROWID-Scan
        - Vyhledání řádky na základě známé hodnoty jejího fyzickéhoidentifikátoru v databázi
  - **cost-based optimizer** (CBO)
    - hledá plán s nejmenšími náklady pomocí statistik
      - tvoří je celá řada údajů o databázových objektech (tabulkách, indexech)

      - některé jsou přístupné i pro uživatele databáze pomocí pohledů či tabulek
      - aktualizují se výpočtem nebo odhadem
      - typy:
        - údaje o tabulkách – počet řádků, bloků, délka záznamu, …
        - údaje o sloupcích – počet unikátních hodnot a NULL, histogram, …
        - údaje o indexech – počet listových bloků, úrovní, clustering, …
    - jeho použití firmou Oracle doporučováno
- možnosti ladění:
  - OPTIMIZER_MODE
    - dosažení max. propustnosti nebo nejmenší odezvy
  - SORT_AREA_SIZE
    - určuje velikost paměti pro třídění
  - CURSOR_SHARING
    - vyhodnocení dotazu tak, jak byl zadán, či provázání literály
  - HASH_AREA_SIZE
    - velikost paměti při využití hashování

Dobrý návrh databáze a aplikace má daleko větší vliv na výkon, než sebelepší nastavení parametrů instance.

Obecná pravidla pro psaní SQL dotazů (do otázky ovšem nepatří)

- vyjmenování konkrétních sloupců
  - v SELECT dotazech nepoužívat v seznamu sloupců hvězdičku, protože ve většině případů nepracujeme se všemi
  - i když používáme v dotazu všechny sloupce, je lepší je vyjmenovat, aby databáze nemusela zjišťovat seznamy sloupců pro konkrétní tabulky
- používat co nejméně klauzuli LIKE
  - ve velkých textových polích se doporučuje nepoužívat, mohou obsahovat až několik GB dat
  - zamyslet se, zda nejde vyhledávání provést jinou metodou
- používat co nejméně klauzuli IN, NOT IN
  - vhodnější použití příkazů WHERE a WHERE NOT EXISTS
- používat klauzule typu LIMIT
- obecnější (nejdůležitější) podmínky umisťovat na začátek dotazu
  - snažíme se, aby systém vyřadil na začátku co nejvíce řádků, které se při následující podmínce již nezkoumají
- výběr vhodného pořadí spojení
  - vyhnout se plnému prohledávání tabulky
  - vybrat takové pořadí spojení ze všech možných pořadí, aby bylo spojeno co nejméně položek
- používat hinty
  - hint = podnět, kterým optimalizátoru určíme, jaký má použít plán vykonávání dotazu
  - hinty se aplikují na blok dotazu, ve kterém se vyskytují
  - ex. hinty pro určení cíle optimalizace, výběr přístupové metody (index scan, rowid scan, full scan), pořadí spojování tabulek, …
  - pomocí hintu si lze vynutit rychlejší přístup, tzv. index scan
  - př.:

```
SELECT jmeno, prijmeni, plat FROM ucitel WHERE pohlavi='M';
lepší:
SELECT /*+ INDEX(ucitel pohlavi_index) */ jmeno, prijmeni, plat FROM ucitele
WHERE pohlavi='M';
```

- nastavit indexy
  - procházení tabulky pomocí indexu trvá podstatně kratší dobu než bez jeho použití (je-li nutné projít celou tabulkou, pak může index naopak brzdit -- zvýšené I/O nároky)
  - změna indexů má větší sílu než změna SQL dotazů X nelze jej brát jako univerzální řešení problému
- další rady:
  - efektivně vybírat takové indexy, které načtou z tabulky co nejméně záznamů
  - používat UNION ALL namísto UNION
  - spojovat tabulky s využitím indexů
  - vytvářet indexy pro atributy, podle nichž se třídí v klauzuli ORDER BY

Citováno z „http://wiki.zvesela.cz/index.php/Optimalizace_dotazu_%E2%80%93_Rule_Based_optimalizace_%28RBO%29%2C_Cost_Based_optimalizace_%28CBO%29%2C_v%C3%BDhody_a_nev%C3%BDhody%2C_v%C3%BDb%C4%9Br_typu_optimalizace.“

- Stránka byla naposledy editována 14. 8. 2010 v 15:23.

Postrelační databáze – výhody a nevýhody, mapování, RDB, ORDB, OODB

Z Na státnice zvesela!

RDB - relational database

ORDB - object-relational database

OODB - object oriented database

Jedná se o všechny současné databáze - jde o relační databáze doplněné o nějakou "funkci" navíc, např. aktivní databáze (triggery) už jsou postrelační databáze.

Obsah

Relační databáze

Technologie relačních databází byla původně navržena E.F.Coddem a později ji implementovala IBM a jiní. Standard je popsán ANSI a ISO normou, časteji se na ni ovšem odvoláváme jako na SQL + číslo verze. Poslední je tedy SQL2. Novější verze SQL3 obsahuje navíc některá objektová rozšíření.

Datový model

RDB uchovává data v databázi skládající se z řádků a sloupců. Řádek odpovídá záznamu (record, tuple); sloupce odpovídají atributům (polím v záznamu). Každý sloupec má určen datový typ. Datových typů je omezené množství, typicky 6 nebo víc (např. znak, řetězec, datum, číslo...). Každý atribut (pole) záznamu může uchovávat jedinou hodnotu. Vztahy nejsou explicitní, ale spíše plynou z hodnot ve speciálních polích, tzv. cizí klíče (foreign keys) v jedné tabulce, který se rovná hodnotám v jiné tabulce.

Dotazovací jazyk

Pohled (view) je podmnožina databáze, která je výsledkem vyhodnocení dotazu. V RDB je pohled tabulka. RDB využívá SQL pro definici dat, řízení dat a přístupu a získávání dat. Data jsou získávána na základě hodnoty v určitém poli záznamu.

Výpočetní model

Veškeré zpracováváni je založeno na hodnotách polí záznamů. Záznamy nemají jednotné identifikátory,

které jsou neměnné po dobu existence záznamu. Neexistují žádné odkazy z jednoho záznamu na jiný. Vytvoření výsledku je prováděno pod kontrolou kurzoru, který umožňuje uživateli sekvenčně procházet výsledek po jednotlivých záznamech. Totéž platí pro update.

Objektová databáze

Pro objektové databáze neexistuje žádný oficiální standard. Standardem je de facto kniha Morgana Kaufmana The Object Database Standard: ODMG-V2.0. Důraz ODB je na přímou korespondenci mezi následujícími:

```
* Objekty a objektové vztahy v aplikaci napsané v OO jazycích
* jejich uchovávání v databázi.
```

Datový model

Objektové databáze využívají datového modelu, který má objektově orientované aspekty jako třídy s atributy a metodami a integritními omezeními; poskytují objektové identifikátory (OID) pro každou trvalou instanci třídy; podporují zapouzdření (encapsulation); násobnou dědičnost (multiple inheritance) a podporují abstraktní datové typy.

Objektové databáze kombinují prvky objektově orientovaného programování s databázovými schopnostmi. Rozšiřují funkčnost objektových programovacích jazyků (C++, Smalltalk, Java) a poskytují plnou schopnost programování databáze. Datový model aplikace a datový model databáze se ve výsledku hodně shodují a výsledný kód se dá mnohem efektivněji udržovat.

Dotazovací jazyk

Objektově orientovaný jazyk (C++, Java, Smalltalk) je jazykem jak pro aplikaci, tak i pro databázi. Poskytuje těsný vztah mezi objektem aplikace a uloženým objektem. Názorně je to vidět v definici a manipulaci s daty a v dotazech.

Výpočetní model

V RDB rozumíme dotazovacím jazykem vytváření, přístup a aktualizaci objektů, ale v ODB, ačkoliv je to stále možné, je toto prováděno přímo pomocí objektového jazyka (C++, Java, Smalltalk) využitím jeho vlastní syntaxe. Navíc každý objekt v systému automaticky obdrží identifikátor (OID), který je jednoznačný a neměnný během existence objektu. Objekt může mít buď vlastni OID, nebo může ukazovat na jiný objekt.

Objektově-relační databáze

"Rozšířená relační" a "objektově-relační" jsou synonyma pro databázové systémy, které se snaží sjednotit rysy jak relačních, tak objektových databází. ORDB je specifikována v rozšíření SQL standardu — SQL3. Do této kategorie patří např. Informix, IBM, Oracle a Unisys.

Datový model

ORDB 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). ADT je datový typ, který vznikne zkombinováním základních datových typů. Podpora ADT je atraktivní, protože operace a funkce asociované s novými datovými typy mohou být

použity k indexování, ukládání a získávání záznamů na základě obsahu nového datového typu. ORDB jsou nadmnožinou RDB a pokud nevyužijeme žádné objektové rozšíření jsou ekvivalentní SQL2. Proto má omezenou podporu dědičnosti, polymorfismu, referencí a integrace s programovacím jazykem.

Dotazovací jazyk

ORDB podporuje rozšířenou verzi SQL — SQL3. Důvodem je podpora objektů (tj. dotazy obsahující atributy objektů). Typická rozšíření zahrnují dotazy obsahující vnořené objekty, atributy, abstraktní datové typy a použití metod. ORDB je stále relační, protože data jsou uložena v řádcích a sloupcích tabulek a SQL, včetně zmíněných rozšíření, pracuje právě s nimi.

Výpočetní model

Jazyk SQL s rozšířením pro přístup k ADT je stále hlavním rozhraním pro práci s databází. Přímá podpora objektových jazyků stále chybí, což nutí programátory k překladu mezi objekty a tabulkami.

Shrnutí

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 dvourozměrných tabulkách a vztahy mezi daty jsou vyjadřovány porovnáváním hodnot v nich uložených. Jazyky jako SQL umožňují tabulky propojit za běhu, aby vyjádřily vztah mezi daty.

Naproti tomu **objektově orientovaný model** je založen na objektech, což jsou struktury, které kombinují daný kód a data. Objektové databázové systémy umožňují využití hostitelského objektového jazyka jako je třeba C++, Java, nebo Smalltalk přímo na objekty "v databázi"; tj. místo věčného přeskakování mezi jazykem aplikace (např. C) a dotazovacím jazykem (např. SQL) může programátor jednoduše používat objektový jazyk k vytváření a přístupu k metodám. Krátce řečeno, ODB jsou výborné pro manipulaci s daty. Pokud navíc opomeneme programátorskou stránku, dá se říct, že některé typy dotazů jsou efektivnější než v RDB díky dědičnosti a referencím.

citováno z Relační vs. objektově-relační vs. objektové databáze (http://www.fi.muni.cz/~xbatko/oracle/compare.html)

Citováno z „http://wiki.zvesela.cz/index.php/Postrela%C4%8Dn%C3%AD_datab%C3%A1ze_%E2%80%93_v%C3%BDhody_a_nev%C3%BDhody%2C_mapov%C3%A1n%C3%AD%2C_RDB%2C_ORDB%2C_OODB“

- Stránka byla naposledy editována 1. 6. 2011 v 13:04.

ANSI/ISO normy SQL – objektové vlastnosti jazyka SQL99

Z Na státnice zvesela!

Obsah

ANSI/ISO normy SQL

SQL-86 (SQL 87)

První standard formalizovaný ANSI

SQL-92 (SQL2)

Standard je rozdělen na tři úrovně: **entry**, **intermediate** a **full**. Někdy je také uváděn mezistupeň mezi entry a intermediate jako **transitional**. Úrovně slouží k tomu, aby mohlo být u implementací standardu (jednotlivých databází) uvedeno do jaké míry splňují daný standard.

- Entry

Jen formální změny oproti SQL-86

- Transitional
  - Podpora různých druhů spojení jako NATURAL JOIN, INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN
  - Podpora nových datových typů DATE, TIME, TIMESTAMP and INTERVAL, including various datetime and interval features (excluding time zones)
- Intermediate
  - Podpora dlouhých identifikátorů (do 128 znaků)
  - Podpora kurzorů a směru získávání dat příkazem FETCH
  - Podpora definice a používání znakových sad
- Full
  - Podpora dočasných tabulek
  - Možnost výběru přesnosti u datových typů TIME a TIMESTAMP
  - Možnost testování pravdivostních hodnot pomocí TRUE, FALSE or UNKNOWN
  - Vnořené tabulky ve FROM
  - Podpora UNION JOIN a CROSS JOIN
  - Podpora pro collations znakových sad
  - Vylepšené udělování práv

[Zdroj: http://developer.mimer.com/documentation/Mimer_SQL_Reference_Manual /Intro_SQL_Stds3.html]

SQL:1999 (SQL3)

Jedná se o standard pro relačně-objektový dotazovací jazyk (narozdíl od předchozích verzí, které byly pouze relační)

- Regulární výrazy
- Rekurzivní dotazy
- Triggery
- Procedurální rozšíření (Příkazy řízení běhu - LOOP, IF...)
- Objektové rozšíření
- nové typy STRING, BOOLEAN, REF, ARRAY, typy pro full-text, obrázky, prostorová data

Existují i novější standardy

- SQL:2003 (Představeny XML-vázané funkce, window funkce, standardizované sekvence a sloupce s automaticky generovanými hodnotami)
- SQL:2006 (SQL může být použito ve spojení s XML - možnost importu a skladování XML dat v SQL databázi, manipulaci s nimi a publikace dat v XML formě. Možnost využití XQuery)
- SQL:2008 (ORDER BY mimo definici kurzoru, INSTEAD OF triggerery, přidán TRUNCATE příkaz)

- Jednotlivé databázové servery ne vždy dodržují ANSI normu – obvykle pouze SQL-92 Entry
- Čím vice se při vývoji aplikace využijí rysy vyšší než SQL-92 Entry, tím je menší šance, že aplikace bude provozuschopná i na jiné databázi

většina z těchto rozšíření lze najít v jiných otázkách, proto zde nebudou více rozepsána.

- Stránka byla naposledy editována 1. 6. 2011 v 14:09.

Objektově relační databáze – objektové vlastnosti jazyka SQL99.

Z Na státnice zvesela!

Obsah

Objektové vlastnosti SQL:1999 (SQL3)

- kompatibilita s existujícími jazyky
- **podporuje**:
  - reference = OID + REF(T)
  - pole proměnné délky (VARRAY) – mohou obsahovat více položek stejného datového typu (pokud je tedy pole možné považovat za objektovou vlastnost).
    - při aktualizaci se s nimi pracuje jako s atomickými hodnotami – nelze jednoduše přidat nebo odebrat prvky (nevýhoda).
  - hnízděné tabulky = nested table (Pozn.: Myslím, že to nepatří do SQL:1999, ale že hnízděné tabulky jsou pouze v Oracle PL/SQL)
  - uživatelem definované typy (UDT - User Defined Types)
    - odlišující typy (Distinct Types) (musí být FINAL) založeno na vestavěných typech jako např. INTEGER - tedy odvození od těchto vestavěných typů
    - strukturované typy (ADT)
      - jednoduchá dědičnost (pomocí klíčového slova UNDER je možné vytvořit přímý podtyp/podtabulku)
      - funkce, procedury a metody (zapouzdření (PUBLIC, PRIVATE, PROTECTED metody a atributy))
    - řádkové typy
  - typované tabulky
    - tabulka je typu UDT (tedy je to třída a řádky jsou objekty (každá řádka je tedy instancí))
- **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

Hnízděné tabulky

Používají se k ukládání více hodnot do jednoho atributu tak, že atribut obsahuje v sobě celou tabulku

- Deklarace hnízděné tabulky:

```
CREATE TYPE address_t AS OBJECT (
   street   VARCHAR2(30),
   city     VARCHAR2(20),
   state    CHAR(2),
   zip      CHAR(5) );
CREATE TYPE address_tab IS TABLE OF address_t;
CREATE TABLE customers (
   custid  NUMBER,
   address address_tab )
NESTED TABLE address STORE AS customer_addresses;
```

- Vložení dat do tabulky:

```
INSERT INTO customers VALUES (1,
   address_tab(
      address_t('101 First', 'Redwood Shores', 'CA', '94065'),
      address_t('123 Maple', 'Mill Valley',    'CA', '90952')
   )
);
```

- Výběr dat z tabulky:

```
SELECT * FROM customers;
    NUMBER ADDRESS
---------- ------------------------
         1 ADDRESS_TAB(ADDRESS_T('101 First', 'Redwood Shores', 'CA', '94065'), ADDRESS_T('123 Mapl
```

```
SELECT t1.number, t2.* FROM customers t1, TABLE(t1.address) t2;
    NUMBER STREET      CITY             STATE ZIP
---------- ----------- ---------------- ----- -------
         1 '101 First' 'Redwood Shores' 'CA'  '94065'
         1 '123 Maple' 'Mill Valley'    'CA'  '90952'
```

[Zdroj: http://www.orafaq.com/wiki/NESTED_TABLE a http://psoug.org/reference/nested_tab.html]

Odlišující typy

- slouží k odlišení typů, které by jinak odlišit nešly (= pojmenujeme si jednoduchý datový typ)

```
CREATE DISTINCT TYPE us_dollar AS DECIMAL(9,2)
CREATE DISTINCT TYPE canadian_dollar AS DECIMAL(9,2)
```

Řádkové typy

- vytvoření pojmenovaných řádkových typů:

```
CREATE ROW TYPE jméno (deklarace komponent)
CREATE ROW TYPE typadresa (ulice CHAR VARYING(50), mesto CHAR VARYING(20));
CREATE ROW TYPE typherec ( jméno CHAR VARYING (30), adresa typadresa);
```

- příklad tvorby tabulky s pojmenovaným řádkovými typem:

```
CREATE TABLE FilmovyHerec OF typherec;
```

- vytvoření nepojmenovaných řádkových typů v tabulkách:

```
jméno ROW (deklarace komponent)
```

- příklad tvorby tabulky s řádkovými typem v tabulce:

```
CREATE TABLE FilmovyHerec (
        jmeno CHAR VARYING(30),
        adresa ROW ( ulice CHAR VARYING (50), mesto CHAR VARYING(20) )
);
```

Tvorba dotazů

```
SELECT FilmovyHerec.jmeno, FilmovyHerec.adresa.ulice FROM FilmovyHerec WHERE FilmovyHerec.adresa.me
```

Abstraktní datové typy

- umožňují zapouzdření atributů a operací (na rozdíl od řádkových typů)
- hodnoty jejich typů mohou být umístěny do sloupců tabulek.

```
CREATE TYPE typZamestnanec AS (
   c_zam        INTEGER
   jmeno        CHAR(20)
   adresa typAdresa,
   INSTANTIABLE NOT FINAL,
   METHOD mzda() RETURNS DECIMAL
);
```

```
CREATE METHOD mzda FOR typZamestnanec
   BEGIN … END;
```

Instance ADT vznikají

1. konstruktorem jmenoTypu()
2. operátorem NEW jmeno hodnota, např. WHERE vedouci = NEW typZam(1012,'Pavel Novák',…)
3. příkazem INSERT, např. INSERT INTO osoby VALUES(1012,'Pavel Novák',…);

Pro každý atribut jsou k dispozici funkce

- implicitně či explicitně zavedené porovnání
- zjištění hodnoty atributu z objektu jmeno_atributu(jmeno_objektu) – stejné i pro aplikaci metod,

možná i tečková notace jmeno_objektu.jmeno_atributu

Funkce, procedury a metody

- vyjádřeny v SQL/PSM (Persistant Stored Module) nebo C/C++, Java, ADA, …
- metody jsou svázány s ADT
- uživatelem definovaný typ je vždy prvním (!nedeklarovaným!) argumentem metody
- metody jsou uloženy ve schématu typu definovaném uživatelem
- metody se dědí
- metody i funkce mohou být polymorfní (liší se způsobem výběru)

```
CREATE PROCEDURE zjisti_cenu (IN cislo INTEGER, OUT c DOUBLE PRECISION)
   SELECT cena INTO c FROM knihy WHERE inv_cislo=cislo;
```

```
CALL zjisti_cenu(12134,z);
```

Podtypy a podtabulky

```
CREATE TYPE typSekretarka UNDER typZamestnanec AS (
   další atributy a metody
)
```

```
CREATE TABLE zamestnanec UNDER osoba (
   Mzda FLOAT
);
```

Reference

- použití pomocí REF(T)
- umožňuje odkazovat na jiný typ T
- obsahuje OID nějakého záznamu

```
CREATE TYPE TypHerec AS (
   jmeno CHAR(30),
   nejlepsiFilm REF (FilmTyp)
)
```

Dereference

```
SELECT Film->titul FROM hrajev WHERE herec->jmeno='Chaplin';
```

OID hodnota

- zpřístupnění klauzulemi REF IS SYSTEM GENERATED a REF IS USER GENERATED

- Stránka byla naposledy editována 2. 6. 2011 v 14:32.

Vlastnosti objektově orientovaného datového modelu

Z Na státnice zvesela!

HDM, SDM a RDM = záznamově orientované modely. V 90. letech se začaly objevovat první objektově orientované SŘBD (OOSŘBD), které umožňují pracovat s datovou abstrakcí na úrovní objektů.

- Výhody
  - snadnější aktualizace dat
  - přímé vyjádření složitých objektů modelované reality v databázi (odpadají mezikroky převodu objektů do normalizovaných tabulek relační databáze. Stejně tak je zjednodušen i opačný krok načítání objektů z databáze do aplikace)
  - součástí uložených objektů je také jejich chování (metody atd.)
- Nevýhoda
  - Vývoj a návrh objektově orientovaných modelů v jejich univerzálnosti a komplexnosti je velmi *složitý proces* ⇒ menší uplatnění tohoto typu modelu v reálných aplikacích.

Objektově orientované programování (OOP)

Koncepce objektově orientovaných databází vychází z principů používaných v OOP - základem je objet (prvek typu třída). Data se nazývají atributy, funkce metody (služby). Metoda je aktivována příchodem zprávy do jiného objektu.

Hlavní vlasntosti OOP: *zapouzdření dat*, *dědičnost*, *polymorfismus*

Objektově orientované modelování dat

Postup objektově orientovaného modelování dat lze stručně shrnout do následujících bodů:

- Vyhledají se objekty jako nositelé aktivit (např. metodou gramatické inspekce: podstatná jména představují objekty nebo třídy, přídavná jména představují hodnoty atributů a slovesa představují většinou aktivity).
- Identifikují se třídy zobecňování objektů se stejnými atributy (+ zkoumá se možnost uspořádat třídy hierarchicky podle dědičnosti) a vztahy.
- Stanoví se integritní omezení na hodnoty jednotlivých atributů a případně na typy atributů.
- Vyhotoví se seznam nabízených a požadovaných služeb pro všechny metody (přehled toku zpráv)
- Implementují se metody (teprve po jednoznačném vymezení všech funkcí systému)

Hlavní rozdíli mezi RDM a ODM:

	RDM	ODM
1)	▪ relační tabulka ▪ jeden záznam ▪ manipulace s atributy záznamu	▪ množina objektů ▪ jeden objekt ▪ přenos a zpracování zpráv
2)	normalizace relací (dekompozice) vede k rozptýlení popisu vlastností složitého objektu do mnoha tabulek	spojuje jednotlivé složky pomocí odkazů
3)	záznamy relací jsou omezeny na jednoduché datové typy	složité strukturované datové entity - objekty, které lépe vystihují prvky reálného světa
4)	manipulace s hodnotami atributů záznamů	operace posílání zpráv poskytuje větší možnosti
5)	každá tabulka musí mít identifikační klíč (ten nemusí odrážet požadavky zadání)	zabezpečuje identifikaci objektů vlastními systémovými prostředky
6)	při zpracování dotazů dochází často k získávání údajů z několika tabulek ⇒ narůstá čas potřebný k vyhodnocení dotazu	ke spojování množin dochází v daleko menší míře; dotazovací konstrukce lze díky polymorfismu aplikovat i na množiny obsahující různé typy objektů

Pozn.: RDM za určitých podmínek představuje zvláštní případ ODM.

Produkty: Objectivity, Versant, POET, CACHÉ, db4o, Ozone, GOODS, XL2, ZODB, Prevayler

Bariéry rozšíření objektových databázových systémů:

- neochota vývojářů a jejich klientů k přechodu od tradičního relačního přístupu k objektovému
- nedostatek kvalifikovaných vývojářů
- nízká podpora standardů
- nízká podpora dotazovacích jazyků
- neexistence mechanismu pro řízení přístupu k datům

--Dejw 17:52, 1. 6. 2011 (UTC)Dejw

Původní text (dle mého "mimo mísu")

Obsah

Objektový (konceptuální model)

- představuje statický model reality (businessu)
- popisuje z čeho je realita složena a jaké jsou základní (podstatné/**statické**) složky (objekty) a vazby mezi nimi.
- Akce (metody) a algoritmy, vázané k objektům v jejich životních cyklech, zde mají význam též statický (jsou podřízeny statickému - pohledu).
- K popisu lze využít specifický diagram – Diagram tříd (Class Diagram - základní diagram jazyka UML).
- Model (entitních, busssines, analytických) objektů (podstata struktury reality) sleduje základní stavební kameny, z nichž se realita (problémová doména) skládá.

Diagram tříd

- Původní strukturální přístup k analýze IS spočíval v rozdělení systému na funkční a datovou část (např. DFD - Data Flow diagramy a ER - Entity Relationship model).
- Přínosem byla funkční hierarchická dekompozice – psaní programu shora dolů a datové konceptuální modelování.
- Rostoucí složitost systémů (magická hranice 1000 entit a 10000 funkcí) znemožňuje soudržnost datové a funkční vrstvy.
- Objektový přístup čelí složitosti systému tím, že třída (objekt) jako nositel (funkční) odpovědnosti (dovednosti), plně odpovídá za svá data.
- Objekt má svou identitu, vlastnosti, chování a odpovědnost. Síla odpovědnosti spočívá v tom, že je nedělitelná – žádný jiný objekt nemůže odpovědnost sdílet – dělit se o ni, plést se do ní.
- Modelování tříd a objektů je klíčová aktivita objektově orientovaného vývoje.

Třída, Objekt

- **Třída** - popis množiny objektů sdílejících stejné vlastnosti (atributy), chování (operace/metody) a vztahy.
- **Objekt** - instance třídy (chybně se pojem třída a objekt volně zaměňují).

Definice – J. Rumbauh: objekt je diskrétní entita s jasně definovaným rozhraním, které zapouzdřuje stav a chování.

- Třídu si můžeme představit jako razítko, objekty jsou pak otisky tohoto razítka, které vidíme na papíře.
- Při návrhu třídy neuvažujeme o konkrétním naplnění atributů, pouze určíme jejich název a typ. Teprve při vzniku instance objektu se atributům přiřadí skutečné hodnoty.
- Třída je jednoznačně určena svým názvem (v příslušném názvovém prostoru – balíčku). Pro třídu je možno definovat vlastnosti - atributy (Attribute) a chování - operace (Operation).
- Hledání tříd, jejich atributů a kompetencí - vyberme z reality objekty, kandidáty pro zobecnění na třídy a prověřme jejich vhodnosti:
  - Potenciální třída je smysluplná, pokud je nezbytná pro funkci systému.
  - Potenciální třída je dostatečně stabilní a invariantní vůči vnějším změnám např. technologie, legislativy apod.

Hledání tříd na základě analýzy podstatných jmen a sloves.

Analyzujeme jazyk problémové domény, např. text sebraných požadavků. Podstatná jména a jejich spojení mohou označovat třídy nebo atributy. Slovesa mohou označovat odpovědnosti, chování tříd.

Pozor na skryté, utajené třídy, které nejsou v textu uvedeny.

Klasifikace, zařazení do třídy přenáší význam na formální objekt, je nositelem sémantiky. Klasifikace je jedním z nejdůležitějších způsobů, jímž lidé uspořádávají, vnímají, chápou okolní svět. Existuje mnoho způsobů jak klasifikovat okolní svět, proto je analýza tak náročná.

Atributy

Definice atributů

Atribut určuje vlastnosti objektu, je nositelem informace o objektu.

Atributy popisují hodnoty (stavy) udržované v jednotlivých objektech.

Objekty jsou vymezeny (popsány) množinou atributů.

Atributy popisují vlastnosti objektů, které potřebujeme k dosažení daného cíle. Reálný objekt ve své nekonečné složitosti nelze vymezit omezenou množinou atributů. Základní problém analýzy IS - výběr rozumného množství relevantních atributů.

S atributy mohou manipulovat výhradně služby daného objektu.

Klíčovou otázkou je zodpovědnost určitého objektu za uchování informací. Hledání atributů je řízeno otázkami: Jak je objekt popsán v kontextu zodpovědností daného systému. V jakých stavech se může objekt v průběhu svého životního cyklu nacházet

Specifikace atributů

Atribut je definován: jménem, typem (formátem) viditelností (veřejný – public, soukromý – private a chráněný – protected).

Každý atribut pečlivě pojmenujte. Volte názvy, které jsou běžné v aplikační oblasti a jsou rozumné délky a pevné struktury. Ke každému atributu připojte vysvětlující text.

Hledejte omezující podmínky pro hodnoty atributů. Omezení se vztahují na: formáty, rozsah, výčet přípustných hodnot, přesnost implicitní hodnoty, požadavek na nastavení výchozích hodnoty atributu prevalidační a postvalidační podmínky – podmínky, které musí být splněny před a po změně hodnoty atributu, za jakých podmínek je povolen přístup k atributu (např. v závislosti na hodnotách ostatních atributů) závislost atributů, viz datové modelování, jak změna jednoho atributu ovlivňuje

hodnoty jiných – závislých atributů, viz normalizace relačního modelu.

Identita objektu

Objekt je vedle svého stavu a chování jednoznačně určen, je jedinečný, má identitu, atribut, který jej jednoznačně identifikuje mezi všemi ostatními objekty dané třídy, má své ID. Atribut zajišťující identitu, se v datovém modelování nazývá primární klíč.

Čtenář (číslo čtenáře, jméno, adresa, kontakt) Kniha (isbn, autor, titul) Exemplář (číslo exemp, datum nákupu) Exempláře knihy se liší inventárním číslem a sledujeme u nich datum nákupu.

Problematika volby primárního klíče,

Umístění atributů

Ve třídách, které jsou vázány dědičností (generalizace) umístíme atribut do co nejvyšší třídy, ve které atribut platí pro všechny její generické podtřídy (specializace).

Hledání tříd, doporučení

Každá třída by měla mít 3-5 klíčových odpovědností První extrém - není dobré, když existuje velké množství malých tříd Druhý extrém – není dobré mít velké třídy Nezavádějte „funktiody“ – pro jednotlivé funkce systému nezavádějte třídy Vyhýbejte se stromům dědičnosti s mnoha úrovněmi

Vazby – relace mezi třídami

Vazba asociace (Association)

Vazba asociace mezi třídami je vyjádřením abstraktního vztahu mezi objekty (instancemi tříd). Asociace říká, že objekty mají mezi sebou přímý vztah, že o sobě ví.

Zaměstnanec pracuje v daném oddělení, mohu se ptát: V jakém oddělení pracuje zaměstnanec, mohu získat seznam všech zaměstnanců v oddělení. Vazba je nositelem významu – sémantiky, odpovídá – mapuje požadavky kladené na systém. Je trvalejšího charakteru.

Asociace je společný typ vazby pro:

- agregaci, vyjadřující vztah mezi celkem a částí
- prostou asociaci, vyjadřující prostou objektovou referenci.

Vazba asociace je specifikována řadou vlastností, z nichž některé jsou vázány přímo k vazbě asociace (například název), ostatní k zakončením vazby (například role). Podrobné určení vlastností až v okamžiku návrhu – specifikace návrhových tříd a jejich vazeb má „implementační“ důsledky.

Vazbu asociace lze zavézt jako orientovanou (Navigability), přičemž neorientovaná vazba je považována za obousměrnou (dva jednosměrné vztahy).

Třídy v asociaci mohou vůči sobě vystupovat v rolích (Role) (Objednávka – Zaměstnanec, Zaměstnanec vystupuje ve vztahu k objednávce v roli Prodejce). Každá strana asociace má své jméno – roli. Role popisuje vlastnost, funkci třídy „viděné“ z druhé strany.

V asociaci lze určit násobnost vazby, (kardinalitu) multiplicitu, která vyjadřuje počet možných vazeb objektů tříd v asociaci (0, 0..1, 0..*, 1, 1..*, *, M..N, …).

Násobnost vazby definuje, kolik může k jednomu objektu, tj. k jedné instanci třídy A na jedné straně vztahu existovat minimálně (parcialita) a maximálně (kardinalita) objektů ze třídy B na druhé straně vztahu a obráceně.

Standardně je vazba asociace implementována zavedením atributu třídy - role pro zachycení objektové reference (množiny referencí pro parcialitu 0..*) na objekty druhé třídy. Implementace vazby – objekt si sebou nese reference na asociované objekty.

S vazbami je třeba šetřit.

Na rozdíl implementace v relačním datovém modelu se jedná o explicitní vyjádření vazby. V relačním modelu se pro vyjádření vazby používají tzv. cizí klíče – implicitní implementace vazby.

Další vlastnosti vazeb související s implementací vazby (mimo UML, CASE):

Každý konec asociace, vazby se nazývá role. Pro roli můžeme definovat řadu vlastností.

Asociativní třída (Association Class) je vazba asociace, která je rozšířena přiřazením třídy pro zachycení informací nutných pro úplnou specifikaci této vazby.

Asociativní třída se používá například v případě oboustranně násobné vazby N:M. Asociativní třída nemá vlastní identitu, identitu přejímá od „asociovaných“ tříd

Vztah mezi třemi a více prvky popisují *vícenásobné asociace (N-ary Association)*. Pro vyjádření vícenásobné asociace se používá element modelu asociativní třída.

Vazba agregace (Aggregation)

Agregace je vyjádřením abstrakce vztahu mezi objekty (instancemi tříd), který odpovídá vztahu celku a části .

Agregace je speciálním případem asociace. Jazyk UML rozlišuje mezi dvěma typy agregací:

- prostou agregací (Simple Aggregation)
- kompozicí (Composition).

Vazba kompozice je silnější než vazba prosté agregace, jedná o vlastnictví částí celkem. Pokud je celek existenčně (tj. svou logikou) závislý na částech a nemůže bez nich fungovat, jedná se o kompozici – pokud se bez nich obejde, jedná se o agregaci.

Vazba generaliace (Generalization)

Vazba generalizace je vyjádřením vztahu mezi obecným elementem (Parent) a specifickým elementem (Child), který je konzistentní s obecným elementem a přidává k jeho definici další informace, je tedy bližší specifikací (specializací) obecného elementu.

Vazba generalizace mezi třídami je vyjádřením vlastnosti dědičnosti, jedné ze základních vlastností objektově orientovaného přístupu.

Jazyk UML povoluje vyjádřit vícenásobnou dědičnost zavedením více vazeb generalizace.

Vazba závislosti (Dependency)

- umožňuje znázornit jistou závislost mezi elementy modelu.
- je určena svým názvem a obvykle se používá s určitým stereotypem, který blíže specifikuje formu závislosti, zavádí její typ.
- je znázorněna orientovanou přerušovanou čárou, kde orientace je vyjádřena šipkou ve směru závislosti.

Závislost obvykle vzniká pouze dočasně pro potřeby poskytnutí služby klientskému objektu a poté tato vazba zaniká (implementační rozdíl od asociace).

Změna jednoho (nezávislého) elementu ovlivní druhý (závislý) element.

Chování objektů, předávání zpráv

Objekt poskytuje služby prostřednictvím operací (metod).

Rozhraní objektu je množina operací, které nabízí objekt k použití pro jiné objekty (nebo externí agenty). Objekty jsou známy jiným objektům pouze prostřednictvím svého rozhraní. Objekt má i své vnitřní – interní operace, které slouží k udržení vnitřní konzistence (stavu) objektu.

Objekt může poskytovat více rozhraní – mít více rolí, podle kontextu ve kterém se nachází. Stejně jako v reálném světě člověk vystupuje v různých rolích podle toho, v jakém kontextu se právě nachází (v zaměstnání se nachází v roli pracovníka, doma manželem, v automobilu řidičem)

Objekty tak odbourávají nevýhodu strukturálních metod, spočívající ve vzájemné izolaci funkční a datové vrstvy.

Objekty spolupracují proto, aby společně mohly vykonávat funkce poskytované systémem, viz modely spolupráce. Operace určující chování jsou definovány a rozpoznávány svojí signaturou – názvem, seznamem parametrů a návratových hodnot. Objekt přijme zprávu a vykoná operaci, jejíž signatura je shodná se signaturou zprávy.

Pro lepší pochopení myšlenky OODB: http://www.linuxexpres.cz/business/objektove-databaze

- Stránka byla naposledy editována 1. 6. 2011 v 17:52.

„Vnější“ programování (přes rozhraní/knihovny) – rozhraní ODBC, JDBC, rozhraní podporující objektově- relační mapování (Java Hibernate)

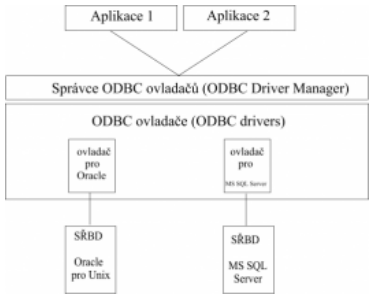
Z Na státnice zvesela!

ODBC

Je standardizované softwarové API pro přístup k databázovým systémům (DBMS). Snahou ODBC je poskytovat přístup nezávislý na programovacím jazyku, operačním systému a databázovém systému. Je to čistě C-čkové API, které nemá žádný objektový základ.

Navrženo Microsoftem, proto primárně přístupné pouze přes C/C++. Založeno na specifikaci X/Open a ISO: SQL Call Level Interface (SQL/CLI)

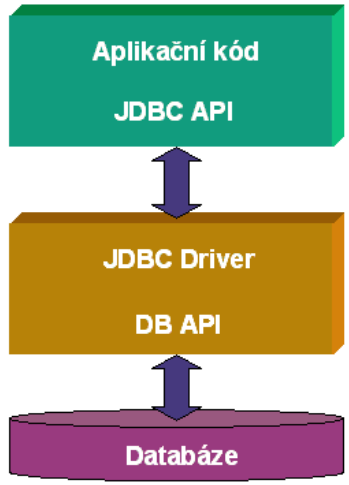
Model struktury ODBC se dá znázornit pomocí čtyř vrstev:



- V první nejvrchnější vrstvě se nachází samotná aplikace. Ta v případě, že potřebuje data, provede volání ODBC funkcí (ve formě SQL dotazu).
- Druhou vrstvou je tzv. "Správce ODBC ovladačů" (ODBC Driver Manager). Úkolem správce ovladačů je zajistit propojení mezi aplikací a příslušným ODBC ovladačem (ODBC ovladače tvoří třetí vrstvu modelu, podrobněji viz dále). Jakmile aplikace potřebuje data, správce ovladačů vyhledá a nahraje příslušný ovladač. (ve formě DLL knihovny). Správce ovladačů také zjistí jaké konkrétní funkce jsou podporovány jednotlivými ovladači a uschová si jejich adresy v paměti do tabulky. V případě, že aplikace volá konkrétní funkci, správce souborů zjistí, ke kterému ovladači funkce patří a zavolá ji. Tímto způsobem může být prováděn souběžný přístup k více ovladačům, což se hodí v případě programování aplikací přistupujících souběžně k několika zdrojům dat.
- Třetí vrstvou zde již zmíněnou vrstvou jsou ODBC ovladače. Ty provedou zpracování volané ODBC funkce, přeložení požadavku do SQL pro příslušný SŘBD (DBMS) a jeho následné poslání.
- Poslední vrstvou je SŘBD, který provede zpracování operace požadované ODBC ovladačem a výsledky této operací mu vrátí.

JDBC (Java Database Connectivity)

- jeho API poskytuje základní rozhraní pro unifikovaný přístup k databázím, aplikační programátor je tak odstíněn od specifického API databáze a může se naučit pouze jednotné rozhraní JDBC
- lze použít i mimo databáze – pro přístup k datům ve forme tabulek (CSV, XLS, ...)
- ovladače jsou k dispozici pro většinu databázových systémů



inspirováno rozhraním ODBC:

- objektové rozhraní
- strukturovanější a přehlednější
- možnost spolupráce s ODBC

JDBC ovladač

- zprostředkovává komunikaci aplikace s konkrétním typem databáze
- implementován obvykle výrobcem databáze
- dotazovací jazyk – SQL
  - předá se databázi
  - ovladač vyhodnotí přímo
- reprezentován specifickou třídou
  - sun.jdbc.odbc.JdbcOdbcDriver
  - com.mysql.jdbc.Driver

Typy JDBC ovladačů

Typ 1:

- využívá ODBC (pres JDBC-ODBC bridge)
- obtížne konfigurovatelné

Typ 2:

- komunikace s nativním ovladačem nainstalovaným na počítači

Typ 3:

- komunikuje s centrálním serverem (Network Server) sítovým protokolem
- pro rozsáhlé heterogenní systémy, velmi efektivní i díky poolingu připojení

Typ 4:

- založen ciste na jazyce Java
- prímý prístup do databáze

Java Hibernate

„Vnější“ programování (přes rozhraní/knihovny) – rozhraní ODBC,... http://wiki.zvesela.cz/index.php/„Vnější“_programování_(přes_rozh...

Hibernate je framework napsaný v jazyce Java, který umožňuje tzv. ORM – Objektově-Relační mapování. Usnadňuje řešení otázky zachování dat z objektů i po ukončení běhu aplikace. Provádí podobné věci jako např. JPA – Java Persistence API.

Co dělá hibernate

Hibernate poskytuje způsob, pomocí nějž je možné zachovat stav objektů mezi dvěma spuštěními aplikacemi. Říkáme tedy, že udržuje data persistentní. Dosahuje toho pomocí ORM, což znamená, že mapuje Javovské objekty na entity v relační databázi. K tomu používá tzv. mapovací soubory, ve kterých je popsáno, jakým způsobem se mají data z objektu transformovat do databáze a naopak, jakým způsobem se z databázových tabulek mají vytvořit objekty. Druhý způsob, jak mapovat objekty, je použít anotace místo mapovacích souborů. V Hibernate tedy pracujete se svými normálními business objekty, pouze pro každý atribut přidáte get/set metody a metody hashCode() a equals(). Nutno podotknout, že nelze použít EJB(viz.Java Bean), ale pouze tzv. POJO(Plain Old Java Object). Poté, co máte objekty uložené v databázi se na ně můžete dotazovat jazykem HQL (Hibernate Query Language), který je odvozen z SQL a je mu tedy velice podobný.

Výhody používání Hibernate

Hibernate, framework pro perzistentní vrstvu, usnadňuje programátorovi práci tím, že nemusí transformovat objekty do relací ručně, ale přenechá to perzistentní vrstvě. Zároveň jsou tím odstíněna specifika jednotlivých databází – programátor používá API Hibernate.

Citováno z „http://wiki.zvesela.cz/index.php/%E2%80%9EVn%C4%9Bj%C5%A1%C3%AD%E2%80%9C_programov%C3%A1n%C3%AD_%28p%C5%99es_rozhran%C3%AD/knihovny%29_%E2%80%93_rozhran%C3%AD_ODBC%2C_JDBC%2C_rozhran%C3%AD_podporuj%C3%ADc%C3%AD_objektov%C4%9B-_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD_%28Java_Hibernate%29“

- Stránka byla naposledy editována 17. 8. 2010 v 15:32.

Distribuované databáze, problémy replikace a fragmentace dat

Z Na státnice zvesela!

- Množina databází, která je uložena na několika počítačích
- uživateli se jeví jako jedna velká databáze
- V databázi neexistuje žádný centrální uzel nebo proces odpovědný za vrcholové řízení funkcí celého systému
- výrazně to zvyšuje odolnost systému proti výpadkům jeho částí

Charakterizována je vlastnostmi:

- transparentností – z pohledu klienta se zdá, že data jsou zpracovávána na jednom serveru v lokální databázi
- jsou syntakticky shodné příkazy pro lokální i vzdálená data, nespecifikuje se místo uložení dat (řeší to distribuovaný SŘBD)
- autonomností – s každou lokální bází dat zapojenou do distribuované databáze je možno pracovat nezávisle na ostatních databázích
- lokální databáze je funkčně samostatná, propojení do jiné části distribuované databáze se v případě potřeby zřizují dynamicky
- nezávislostí na počítačové síti – jsou podporovány různé typy architektur lokálních i globálních počítačových sítí (LAN, WAN)
- v distribuované databázi mohou být zapojeny počítače i počítačové sítě různých architektur, pro komunikaci se používá jazyk SQL

Obsah

Proč DDBS?

- 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 = přídavná režie, cena komunikace, zatížení sítě
- zvýšení výkonu (inherentní paralelismus rozdělením zátěže na více počítačů)
- spolehlivost (replikace dat, degradace služeb při výpadku uzlu, přesunutí výpočtů na jiný uzel)
- lepší rozšiřitelnost konfigurace (přidání procesorů, uzlů)

- větší schopnost sdílet informace integrací podnikových zdrojů
- uzly mohou zachovat autonomní zpracování a současně virtuálně zabezpečovat globální zpracování
- agregace informací (z více bází dat lze získat informace nového typu)

Problémy

- složitost (distribuce databáze, distrib. zpracování dotazu a jeho optimalizace, složité globální transakční zpracování, distribuce katalogu, paralelismus a uvíznutí, případná integrace heterogenních dat do odpovídajících schémat, složité zotavování z chyb)
- cena (komunikace je navíc)
- bezpečnost
- obtížný přechod (neexistence automatického konverzního prostředku z centralizovaných DB na DDB)

Fragmentace a replikace dat

Formy transparentnosti:

- Nezávislost:
  - Datová nezávislost = imunita uživatelské aplikace ke změnám v definici a organizaci dat. Je požadavkem i pro centralizované DB
  - Logická nezávislost (log. strukt. databáze)
  - Fyzická nezávislost (konkrétní způsob uložení dat)
- Síťová transparentnost = ukrytí síťových detailů = uživatel neví o síti
- Replikační transparentnost = neobtěžovat uživatele skutečností, že pracuje s daty existujícími ve více kopiích = uživatel neví o replikách
- Fragmentační transparentnost = uživatelův dotaz je specifikován na celou relaci, ale musí být vykonán na jejím fragmentu = uživatel neví o fragmentech

Fragmentace

Rozdělení DB na části k distribuci (data jsou uložena na místě, kde se často používají - "na nejbližším počítači")

- Horizontální - Výběr řádků určitého typu (jen rodinných domů z tabulky nemovitostí)
- Odvozená horizontální - Horizontální fragment, který je založen na horizontální fragmentaci rodičovské relace
- Vertikální - Výběr podle některých sloupců
- Smíšená - Zajišťuje, že fragmenty, které se používají často společně jsou uložené na stejném místě

více na [1] (http://uhk.xichtik.net/download.php?id=402)

Respektovat hlediska:

- Rozdělit relace do lokálních serverů tak, aby aplikace zatěžovaly servery stejnoměrně. (Musí být známa informace o předpokládaných přístupech k relacím).
- lokality zpracování (maximalizovat lokální)
- přístupnosti a spolehlivosti (např. replikací zlepšíme spolehlivost a read-only dostupnost)
- maximalizovat stupeň paralelismu zpracování dotazu
- dostupnosti a ceny paměti v jednotlivých uzlech

Problémy

- Určení vhodné fragmentace
- Minimalizovat počet fragmentů pro všechny relace

Replikace dat

Replikace primárních dat do lokálních databází vzdálených uživatelů

Výhody:

- zvýšení rychlosti a propustnosti aplikace
- zvýšení dostupnosti dat
- zajištění kompletní nezávislost

externí matroš (PDFko) (http://uhk.xichtik.net/download.php?id=402)

Rozdělení

- Podle okamžiku replikace
  - asynchronní (2 fázový commit)
  - synchronní (Pří výpadku proběhne commit po navázání spojení)
- Podle způsobu zacházení s daty
  - Replikace pouze pro čtení
  - Replikace pro transakční zpracování
    - Jednosměrné - master-slave
    - Obousměrné - peer-to-peer
- Podle vlastnictví dat
  - Master-slave
  - peer-to-peer
  - workflow

Problémy

- Problémy s aktualizací dat (v případě peer-to-peer)
- Problémy s výkonem (v případě peer-to-peer)

- Stránka byla naposledy editována 4. 9. 2010 v 14:49.

Temporální databáze, porovnání klasických a temporálních databází, modely času, vztah událostí a času (snapshot), temporální SQL.

Z Na státnice zvesela!

Obsah

Klasické vs. temporální

- Klasické DB
  - Neobsahují informaci o čase
  - V databázi zachycen pouze aktuální stav systému. V případě, že se v čase systém vyvíjí, změny se v databázi projeví přidáváním nových informací a mazáním starých.
  - V případě, že požadujeme uchovávání historie změn, či alespoň předchozího stavu, je nutné do databáze doplnit informaci o čase. Aktualizaci a operace s časem musí zajistit uživatel. Což není triviální.
- Temporální DB
  - Vhodný dotazovací jazyk zahrnující práci s časem
    - Výhodou jsou jednodušší dotazy v nichž se vyskytuje čas, což přináší méně chyb v aplikačním kódu

Temporální projekce

- V klasických DB projekce funguje tak, že z celé relace jsou vybrány hodnoty podle zadaných atributů.
- Obdobnou funkčnost požadujeme od projekce v temporálních databázových systémech (od temporální projekce). Navíc ale každá n-tice obsahuje čas. V případě, že dvě n-tice výsledku mají stejné hodnoty všech svých atributů a překrývají se nebo dotýkají se časem, srostou tyto dvě n-tice do jedné n-tice s časem odpovídajícím sjednocení časů obou n-tic
- Realizace temporální projekce v klasických databázích představuje netriviální problém a zesložiťuje údržbu aplikace,

Temporální spojení

- Stejné jako spojení (join) v klasických databázích
  - Klasické spojení funguje tak, že zadáme sloupce a podmínku, která říká, kdy jsou dva řádky tabulky spojeny.
- Navíc bere v úvahu čas události
  - V temporálních databázích nemusíme zadávat sloupce uchovávající čas, pouze podmínku na spojení pro čas.

Modely času

Podle uspořádání:

- Lineární
  - Čas roste od minulosti k budoucnosti lineárně
- Větvený (čas možných budoucností)
  - Lineární minulost až do teď, pak se větví do několika časových linií reprezentujících možný sled událostí
  - Každá linie se může dále větvit
- Cyklický
  - Opakující se procesy
  - Příklad: týden, každý den se opakuje po sedmi dnech

Podle hustoty:

- Diskrétní
  - Spolu s lineárním uspořádáním
  - Každý okamžik má právě jednoho následníka
  - Každé přirozené číslo odpovídá nerozložitelné jednotce času (chronon).
  - Chronon je nejmenší časová jednotka reprezentovatelná v diskrétním modelu. Není to okamžik, ale doba.
- Hustý
  - Isomorfní(převoditelný, zachovávající relace) s racionálními nebo reálnými čísly
  - Mezi každými dvěma okamžiky existuje nějaký další
- Spojitý
  - Isomorfní s reálnými čísly
  - Na rozdíl od racionálních čísel, neobsahuje „mezery“
  - Každé reálné číslo odpovídá bodu v čase (okamžiku)

Omezenost, absolutnost a relativnost času

- Omezený – nutnost zejména kvůli reprezentaci v počítači
- Neomezený
- Absolutní čas – vyjádří se hodnotou. Také ale potřebuje počátek.
- Relativní čas – vyžaduje nějaký počátek, čas se pak vyjádří jako vzdálenost a směr od počátku.

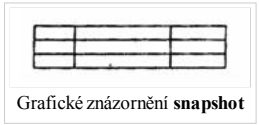
Vztah událostí a času

- Čas platnosti (valid time)
  - Čas, kdy byla událost pravdivá v reálném světě

- Může být v minulosti, přítomnosti i budoucnosti
- Transakční čas (transaction time)
  - Čas, kdy byl fakt reprezentován v databázi
  - Nabývá pouze aktuální hodnoty
  - Monotónně roste

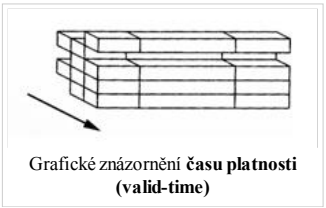
Datové modely

- snapshot
  - **„Jsem schopen zjistit co v tuhle chvíli platí o této chvíli. Nic víc.“**
  - Datový model nepodporující čas platnosti ani transakční čas
  - Klasický relační model
  - Každá n-tice je fakt platný v reálném světě
  - Při změně reálného světa jsou do relace prvky přidávány nebo z ní odebírány



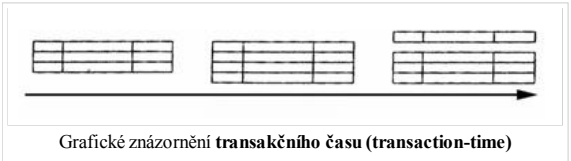
Grafické znázornění **snapshot**

- valid-time
  - **„Jsem schopen zjistit co v tuhle chvíli platí o kterémkoli uvažovaném čase.“**
  - Datový model podporující pouze čas platnosti
  - Cokoliv v relaci může být upraveno
    - Hodnoty n-tic
    - Čas události (začátek i konec)
  - Umožňuje klást dotazy o faktech platných v minulosti i budoucnosti

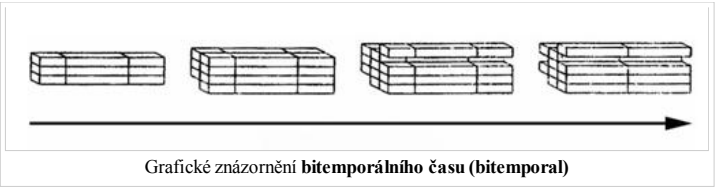


Grafické znázornění **času platnosti (valid-time)**

- transaction-time
  - **„Jsem schopen zjistit co v kterémkoli uvažovaném čase platilo o současnosti.“**
  - Datový model podporující pouze transakční čas
  - Posloupnost snapshot-ů indexované transakčním časem
  - Umožňuje získat informaci ze stavu databáze v nějakém okamžiku minulosti
  - Je možné uvažovat i větvení



Grafické znázornění **transakčního času (transaction-time)**



Grafické znázornění **bitemporálního času (bitemporal)**

- bitemporální
  - **„Jsem schopen zjistit co v kterémkoli uvažovaném čase platilo o kterémkoli uvažovaném čase.“**
  - Datový model podporující čas platnosti i transakční čas

  - append-only
- temporální
  - Datový model podporující čas platnosti nebo transakční čas

Obvykle založeno na relačním datovém modelu nebo objektově orientovaném datovém modelu.

Reprezentace „času platnosti“

- časový okamžik
- doba
- časový úsek
- množina okamžiků

Čas platnosti může být přidružen k:

- atributu
- množině atributů
- celé n-tici nebo objektu

Reprezentace „transakčního času“

- časový okamžik
  - nová n-tice se stejným klíčem => logické odstranění původní
- časový úsek
  - (teď, dokud nezměněno)
- tři časové okamžiky
  - čas zaznamenání začátku události v reálném světě
  - čas zaznamenání konce události v reálném světě
  - čas logického odstranění události z databáze
- Množina časových úseků

Temporální dotazovací jazyky

- Velké množství
- Nejčastěji založeny na SQL
- Typy
  - Relační
    - př.: HQL, HSQL, TDM, TQuel, TSQL, TSQL2
  - Objektově orientované
    - př.: MATISSE, OSQL, OQL, TMQL

Shrnutí

- Ideální temporální datový model
  - Minimální rozšíření existujícího DM
  - Souvislá prezentace chování měnícího se v čase
  - Snadná implementace
  - Vysoký výkon
- Dosažení ideálu je prakticky nemožné
- Hlavní cíl temporálního DM by měl být zachytit sémantiku dat měnící se v čase

    - Ale často se dostává do pozadí
- Mnoho nekompatibilních datových modelů s mnoha dotazovacími jazyky

Citováno z „http://wiki.zvesela.cz/index.php/Tempor%C3%A1ln%C3%AD_datab%C3%A1ze%2C_porovn%C3%A1n%C3%AD_klasick%C3%BDch_a_tempor%C3%A1ln%C3%ADch_datab%C3%A1z%C3%AD%2C_modely_%C4%8Dasu%2C_vztah_ud%C3%A1lost%C3%AD_a_%C4%8Dasu_%28snapshot%29%2C_tempor%C3%A1ln%C3%AD_SQL.“

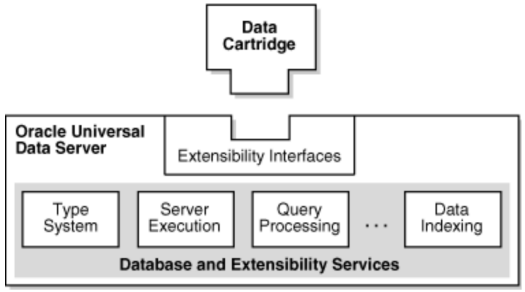
- Stránka byla naposledy editována 18. 8. 2010 v 09:52.

Uživatelské rozšíření databázových systémů – data cartridge, příklady použití.

Z Na státnice zvesela!

- rozšiřitelnost = možnost přidávání nových datových typů a programů (funkcí) zabalených do speciálního modulu
- uživatelsky definované:
    - typy = UDT
    - funkce = UDF
- problém: zapojení do relačního SŘBD včetně SQL
    - DB/2 = relační extendery
    - Informix = DataBlades
    - Sybase = Component Integration Layer
    - Oracle = cartridges

Oracle Extensibility Framework



- **Data cartridge**:
    - způsob uživatelského rozšíření databázového serveru Oracle
    - rozšíření je pouze na straně serveru
    - jsou integrované se serverem přes rozhraní
    - jsou v podobě balíků = instalují se jako celek
- **Extensibility interface**:
    - rozhraní, které umožňuje vytvářet cartridge jednotným způsobem
    - poskytuje přesně definovaný způsob, jak se serverem komunikovat
    - zpřístupňuje jednotlivé standardní služby, které lze v serveru rozšířit
- **Database and extensibility services**:
    - sada standardních služeb, které poskytuje server
    - lze je využívat a rozšiřovat pomocí cartridge
    - jednotlivé služby jsou:
        - Extensible type system:
            - podpora pro uživatelské typy, kolekce, reference (REF), LOBy
        - Extensible Server execution environment:
            - podpora vlastních procedur a funkcí v SQL, PL/SQL, C, Java
        - Extensible Indexing:
            - vlastní způsob indexování = tzv. doménové indegování pro doménově

specifická data
- Extensible Optimiser:
  - podpora pro vytávření uživatelských funkcí a indexů pro vlastní sběr statistik pro CBO

- pracuje na principu tzv. cartridge:
  - způsob uživatelského rozšíření databáze Oracle
    - integruje se pomocí sady rozhraní pro jednotlivé přístupy (k datům, indexům, …)
  - rozšíření může definovat:
    - nové datové typy (standardní, pole, hnízděné tabulky, LOBy) a jejich funkce
    - nové typy indexů
    - nové operátory
  - proč implementovat Data Cartridge:
    - nutnost zpracování komplexních dat, která neodpovídají standardním relačním informacím
    - nutnost snadné manipulace s takovými daty
  - **příklady**:
    - multioborové = datové, statistické výpočty, prostorové databáze, multimédia
    - specializované, finanční a právní systémy
- typická struktura cartridge:
  - definice nových objektových typů
  - implementace těl typů, balíky, procedury, funkce
  - případné DLL knihovny s implementací v C
  - operátory
  - doménové indexy pro podporu operátorů
- **př. standardní cartridge**: podpora indexování a vyhledávání v textech

LOB – Large Objects

- standardní typy pro ukládání objemných dat na serveru (až 4 GB)
- typy:
  - Externí
    - BFILE = samostatný binární soubor uložený vně databáze
  - Interní
    - CLOB = znakový typ v UTF
    - NCLOB = znakový typ v národní sadě
    - BLOB = binární typ
- ve sloupci tabulky uložen pouze deskriptor odkazující na samotná data
  - hodnoty pro xLOB sloupce = NULL, EMPTY_CLOB(), EMPTY_BLOB(), EMPTY_NCLOB()
- manipulace:
  - Oracle nabízí balík DBMS_LOB s řadou funkcí a procedur pro standardní manipulaci s daty
  - provádí se po částech pomocí bufferů
- indexace:
  - není standardně indexováno, ale je možné implementovat svá vlastní rozhraní pro indexaci

Rozšíření serveru

- server dovoluje psát implementace procedur v řadě jazyků:
  - nativním PL/SQL
  - Javě

- v čemkoliv s konvencí jazyka C a kompilací DLL

Data cartridge (Oracle) (http://download.oracle.com/docs/cd/B14117_01/appdev.101/b10800/dciwhatis.htm)

Citováno z „http://wiki.zvesela.cz/index.php/U%C5%BEivatelsk%C3%A9_roz%C5%A1%C3%AD%C5%99en%C3%AD_datab%C3%A1zov%C3%BDch_syst%C3%A9m%C5%AF_%E2%80%93_data_cartridge%2C_p%C5%99%C3%ADklady_pou%C5%BEit%C3%AD.“

- Stránka byla naposledy editována 18. 8. 2010 v 12:25.

Dokumentografické systémy, fulltextové vyhledávání, filtrace, disambiguace, lemmatizace, indexy, tezaury, dotazování.

Z Na státnice zvesela!

Obsah

Dokumentografické systémy (DIS)

- vznik 50. léta 20. stol. za účelem automatizace postupů používaných v knihovnictví
- Nyní samostatná podčást IS
  - *Faktografický IS* - informace s definovanou vnitřní strukturou (nejčastěji tabulky)
  - *Dokumentografický IS* - informace v podobě textu v přirozeném jazyce bez pevné vnitřní struktury
- **Dotazování**:
  1. Uživatel zadá dotaz způsobem, který je pro DIS či jeho nadstavbu akceptovatelný (jako nadstavbu si můžete představit libovolný nástroj, který spojuje více internetových vyhledávačů v jedinou stránku). V dotazu jsou obsaženy termy, o kterých uživatel předpokládá, že budou v dokumentech, které jsou pro tazatele relevantní. Zde vstupuje na scénu již uvedené kritérium predikce — čím „lepší“ termy budou zvoleny, tím více relevantních dotazů se dostane na výstup.
  2. Nyní dojde k vyhodnocování položeného dotazu (popis vyhodnocení viz dále v části Vyhodnocování dotazu).
  3. Pokud je uživatel s výsledkem spokojený, pokračuje následujícím bodem, jinak se snaží upravit (ladit) dotaz tak, aby výsledek více odpovídal jeho potřebám. Účelem ladění je přidat některé termy, které omezí velikost výslednou množiny, nebo naopak některé vypustit či modifikovat (tedy uvést synonyma, jiný pád apod.).
  4. V tomto kroku uživatel prochází některé dokumenty, na něž získal odkaz (a další sekundární informace) v druhé části (představte si to tak, že ve výsledku vyhledávače procházíte ty stránky, které vám — subjektivně — připadnou jako nejvíce vyhovující). Pokud není v DIS implementována část subsystém dodání dokumentu, je tento krok vynechán.
- Struktura DIS:

Dokumentografické informační systémy rozdělujeme do dvou relativně samostatných (nikoliv však nezávislých) částí. První je **subsystém zpřístupnění dokumentu**. Ten podle uživatelského dotazu poskytne tazateli seznam sekundárních informací (název dokumentu, autor, rok vydání, vydavatel, stručný obsah apod.) o dokumentech, které uvedenému dotazu vyhovují. Dále je ve výsledku informace o umístění dokumentu (při internetovém vyhledávání např. Googlem je to klasický odkaz na stránku). Druhou částí DIS je **subsystém dodání dokumentu**, který na základě výstupu ze subsystému zpřístupnění dokumentu poskytne tazateli celý požadovaný dokument. Jako příklad si můžeme vzít knihovnu. Pokud hledáte dostupné knihy od Karla Čapka, počítač vám zobrazí (či si v kartotéce najdete) seznam knih, které jsou v knihovně. Subsystémem dodání dokumentu je pak obsluhující personál, kterému povíte, jakou knihu chcete, a on vám ji donese (tzn. nemusí být realizován jako SW část DIS).

Vyhodnocení dotazu

- přímé porovnávání náročné na čas
- nutné vytvořit model dokumentu
  - ztrátový proces, založený na identifikaci slov v dokumentech
  - výsledkem strukturovaná data vhodná pro porovnávání
- dotaz se upraví do odpovídající podoby a porovná se s modelem dokumentů

Text

Předzpracování

- vyhledávání nad modelem efektivnější, ale lze použít jen informace z modelu
- cíl: vytvořit model, zachovávající nejvíce info z původního textu
- problem: nejednoznačností (ambiguity)
- dosud neřešené nároky na encyklopedické i asociativní znalosti

Porozumění textu

- Homonymie slov
  - jedno slovo může mýt stejný tvar pro různé pády a další gram. jevy
    - *kontroly*: 1.p.m.č., 2.p.j.č. - není zřejmé jestli více kontrol nebo jedna kontrola
  - jeden tvar může mít různý význam
    - *hnát* - sloveso, podst. jm.
    - *pět* - číslovka, sloveso
- přiřazení je závislé na osobě, která dokument píše nebo čte
  - dva lidé mohou jednomu slovu přikládat zcela nebo částečně jiný význam
  - dva lidé si i pod stejným významem mohou představit jiný konkrétní předmět nebo množinu
    - *máma, pokoj, …*
- výsledkem situace, kdy dva různí čtenáři nemusí přečtením získat stejnou informaci jako autor, ani navzájem.
- Homonymie a nejednoznačnosti narůstají při přechodu od slov k větám.

Přesnost a úplnost

- důsledek nejednoznačnosti: žádný existující DIS nedává ideální výsledky
- Pro zobrazení odpovědi na dotaz lze určit
  - **Nv** (počet vrácených dokumentů) - O nich si DB myslí, že jsou relevantní, odpovídající dotazu
  - **Nvr** (počet vrácenýcy relevantních dok.) - o nich si tazatel myslí, že uspokojí jeho požadavky
  - **Nr** (počet všech relevantních dok. v DB) - problematické u velkých DB
- Kvalita výsledné množiny se měří na základě:
  - *Přesnost (Precision):* P = Nvr / Nv
    - pravděpodobnost, že dokument zařazený v odpovědi je skutečně relevantní
  - *Úplnost (Recall):* R = Nvr / Nr
    - pravděpodobnost, že skutečně relevantní dokument je zařazený v odpovědi
- koeficienty jsou opět závislé na subjektivním názoru tazatele
- dokument vrácený na výstupu může uspokojovat požadavky dvou uživatelů, kteří položili stejný dotaz, různou měrou
- ideální případ: P == R == 1

Kritéria

- **Kritérium predikce**
  - při formulaci dotazů je třeba uhádnout, které termy (slova) byly v dokumentu autorem použity pro vyjádření dané myšlenky
    - problémy způsobují
      - synonyma - autor používá synonyma, které si tazatel nemusí při dotazu uvědomit
      - překrývající se význam slov
      - opisy jedné situace jinými slovy
  - částečné řešení - zařazení tezauru, který obsahuje
    - hierarchie slov a jejich významů
    - synonyma slov
    - asociace mezi slovy
  - tazatel může tezaurus využít při formulaci svých dotazů
  - při ladění dotazů má uživatel tendenci postupovat konzervativně
    - v dotazu často zůstávají ty části, které uživatele napadly na začátku a mění se jen podružné části, které nekvalitní výsledek nemusí zásadně ovlivnit
  - vhodné je uživateli pomoci s odstraněním nevhodných částí dotazu, které nepopisují relevantní dokumenty a naopak s přidáváním formulací, které relevantní dokumenty popisují
- **Kritérium maxima**
  - tazatel obvykle není schopen (ochoten) procházet příliš mnoho dokumentů do té míry, aby se rozhodl, zda jsou pro něj relevantní nebo ne
  - obvykle 20-50 podle velikosti
  - potřeba nejen dokumenty rozlišovat na odpovídající/neodpovídající dotazu, ale řadit je na výstupu podle míry předpokládané relevance

- v důsledku kritéria maxima se při ladění dotazu uživatel obvykle snaží zvýšit přesnost
  - malé množství dokumentů v odpovědi, obsahující co největší poměr relevantních dokumentů
- některé oblasti použití vyžadují co nejvyšší přesnost i úplnost
  - např. *právnictví*

Modely dokumentografických systémů

Úrovně modelů:

- rozlišují (ne)přítomnost slov v dokumentech
- rozlišují frekvence výskytů slov
- rozlišují pozice výskytů slov v dokumentech

Boolský model

- vznik 50. léta 20. stol., automatizace postupů používaných v knihovnictví
- Databáze obsahuje dokumenty, dokumenty popisovány pomocí termů, reprezentace dokumentu pomocí množiny termů (obsažených v dokumentu, popisujících význam dokumentu)
- **Indexace**
  - Přiřazení množiny termů, které jej popisují ke každému dokumentu
    - *Ruční* - nekonzistence
    - *Automatická* - konzistentní, ale bez porozumění textu
    - *Řízená* - předem daná množina termů
    - *Neřízená* - množina termů se mění s přibývajícími dokumenty
  - Tezaurus - vnitřně strukturovaná množina termů
    - Synonyma s preferovanými termy
    - Hierarchie užších/širších termů
    - Příbuzné termy
    - ...
  - Stop-list - nevýznamová slova
  - Příliš obecná slova nejsou pro identifikaci dokumentů vhodná, příliš specifická slova také ne
  - dotaz vyjádřen logickým výrazem: **AND, OR, NOT**
  - Příklad dotazu: *počítač AND NOT osobní*
  - Víceslovné termy: *počítač AND NOT osobní počítač*
  - *Organizace indexu*:
    - *Invertovaný seznam* - pro každý term je seznam dokumentů, ve kterých se vyskytuje
    - Zpracování dokumentů na vstupu - vnikne posloupnost dvojic *<dok_id,term_id>*
    - Setřídění dle term_id,dok_id
- **Nevýhody**
  - formulace dotazů je spíše uměním než vědou
  - nemožnost ohodnotit vhodnost vystupujících dokumentů
  - všechny termy v dotazu i v identifikaci dokumentu jsou chápány jako stejně důležité
  - nemožnost řízení velikosti výstupu
  - některé výsledky neodpovídají intuitivní představě

Vektorový model

- vznik 70. léta 20. stol., cca o 20 let mladší než Boolské DIS
- snaha minimalizovat nebo odstranit nevýhody Boolských DIS
- Struktura:
  - databáze obsahuje dokumenty
  - dokument popisován pomocí množiny termů
  - term je slovo nebo sousloví
  - reprezentace dokumentu pomocí vektoru vah termů

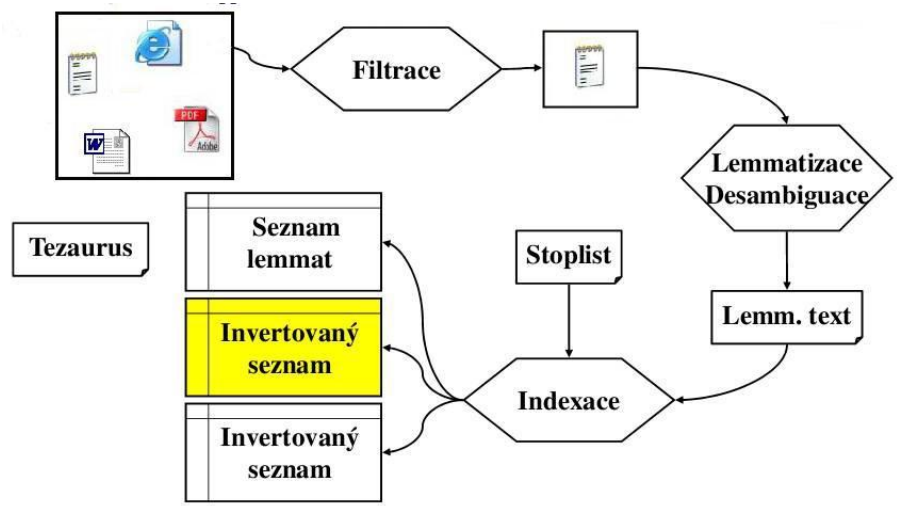
Fulltextové vyhledávání

- Filtrace
- Disambiguace - disambiguace je to, co rozhodne, jestli slovo stavení je 1. sg nebo 2. sg nebo 1. pl apod
- Lemmatizace - je proces, kdy je slovo převedeno do základního tvaru - tzv. lemma. Například slovo "počítačích" je převedeno na slovo "počítač". Umožňuje lepšímu strojovému porozumění textu a používá se pro vyhledávání fulltextem.
- Indexy
- Tezaury - je slovník, který uživateli nabízí seznam synonym, někdy i antonym.
- Dotazování

- Odlišné od principů běžného vyhledávání
  - neprohledávají se striktně strukturovaná data, kde má každý sloupec každé tabulky předem daný význam
  - prohledávají se volně psané texty, kde může být stejná událost popsána více autory rozdílně - různá slova stejného významu, různé slovní obraty a opisy
- DB systémy využívají svých prostředků rozšiřitelnosti a dodávají standardně prostředky, které vyhledávání v textových datech umožňují
- Rozdílné přístupy a možnosti
  - neexistuje objektivně nejlepší řešení
  - výsledky navíc podléhají subjektivním názorům tazatelů
- Samotná formulace dotazu, který by vrátil všechny dokumenty, které tazatele zajímají a žádné jiné, obvykle nelze zformulovat
  - spolu s vyhovujícími - relevantními - odpověďmi se obvykle vrací i odpovědi nerelevantní
- **Problémy**
  - *Hononyma* - ptá se tazatel dotazem "koruna" na finanční, lesnické či panovnické dokumenty?
  - *Synonyma*
    - Vyhovuje dokument o "krychlích" dotazu na dokumenty o "kostkách"?
    - Vyhovuje dokument o "stromech" dotazu na "souvislé grafy bez cyklů"?
  - *Hierarchie významů*
    - Zvíře - Savec - Šelma - Medvěd
    - Tiskovina - Časopis
  - *Ohebnost slov* - Jít, Jde, Jdu, Jdou, ...
- Striktní boolská logika není pro formulaci dotazů příliš vhodná
  - Dokument buďto vyhovuje nebo nevyhovuje
  - Dotazování v textech vyžaduje třídit odpovědi podle předpokládané vhodnosti pro tazatele - je potřebné mít možnost definovat míru shody dotazu s dokumentem

Předzpracování

- Databáze obvykle používají některý z boolských modelů reprezentace dokumentů
  - nejlépe odpovídá běžným dotazům
  - relativně snadno se implementuje
  - dotazy jsou ve formě boolských formulí, ve kterých operandy tvoří jednotlivá slova - řada různých modifikací



Jednotlivé kroky:

- *Filtrace* - odstraní formátovací značky a nechá čistý ASCII text
- *Desambiguace* - určí význam slova podle kontextu
  - "pět chválu" ... sloveso pět
  - "pět vozidel" ... číslovka pět
- *Lemmatizace* - určí základní tvar slova a gramatický tvar v dokumentu, často nahrazen pomocí stemmeru, který hledá kmen slova
- *Indexace* - vytvoří pomocné seznamy lemmat a dokumentů a invertovaný soubor
  - dvojice *[id_dok, id_lemmatu]* setříděné dle id_lemmatu a zbavené duplicit

- dnes obvykle více informací, např. pětice *[id_dok,č_odstavce,č_věty,č_slova,id_lemmatu]* - dovoluje vyhodnocování tzv. proximitních omezení na vzdálenost slov v dokumentu

Large OBjects (LOB)

- pro podporu vyhledávání je potřeba nad textovým sloupcem vytvořit index - invertovaný soubor
- běžné textové sloupce jsou pro tyto účely krátké a nevyhovující
  - obvykle se takto indexují sloupce některého z LOB (Large OBject) typů
- LOBy
  - standardní typy pro ukládání objemných dat na serveru, definováno v SQL-92 Full
  - až 4GB dat
    - BLOB - standardní binární typ
    - CLOB - znakový typ v univerzální znakové sadě
    - NCLOB - znakový typ v národní znakové sadě
  - v Oracle navíc externí typ BFILE
    - pouze pro čtení
    - samostatný binární soubor uložený vně databáe v OS
  - v MS SQL
    - Image - binární data do velikosti 2 GB
    - Text - textová data do velikosti 2 GB
    - NText - textová data v národní znakové sadě do vel. 1 GB
- Ve sloupcích tabulky je uložen pouze deskriptor (tzv. LOB lokátor), odkazující na samostatně uložená data

Oracle fulltext

- Filtrování vstupních dokumentů
  - NULL_FILTER - pro textové dokumenty TXT, HTML, XML
  - INSO_FILTER - pro binární dokumenty
  - CHARSET_FILTER - pro konverzi získaných dokumentů do znakové sady databáze
- Druhy fulltextových indexů
  - CONTEXT
    - základní typ indexu pro vyhledávání v textových datech
    - vhodný pro větší dokumenty
    - synchronizace indexu s daty je nutno provést explicitně
  - CTXCAT
    - vhodný pro menší dokumenty a jejich úryvky
    - může být zkombinován s dalšími netextovými sloupci pro kombinované dotazování
    - synchronizace indexu s daty se provádí automaticky se změnami v tabulce
  - CTXRULE
    - postaven na množině předdefinovaných dotazů
    - slouží pro klasifikaci dokumentů do skupin podle toho, kterým dotazům vyhovuje
- Uložení dokumentů
  - NORMAL_DATASTORE - text je v jednom sloupci jednoho rádku
  - MULTI_COLUMN_DATASTORE - text ve více sloupcích jednoho řádku
  - URL_DATASTORE - text je na internetu, dostupný přes URL ve sloupci
- Příklad vytvoření indexu nad textovým sloupcem:

```
CREATE INDEX myindex ON doc(htmlfile)
INDEXTYPE IS ctxsys.context
PARAMETERS('datastore ctxsys.default_datastore
  filter ctxsys.null_filter
  section group ctxsys.html_section_group');
```

- Spolu s novými typy indexů databáze implementují nové operátory pro porovnávání dotazu s textem
- Operátory vrací číslo - očekávanou míru shody obsahu textu s tazatelovými požadavky

ZDROJ - prednasky DB2 (2010)

Citováno z „http://wiki.zvesela.cz/index.php/Dokumentografick%C3%A9_syst%C3%A9my%2C_fulltextov%C3%A9_vyhled%C3%A1v%C3%A1n%C3%AD%2C_filtrace%2C_disambiguace%2C_lemmatizace%2C_indexy%2C_tezaury%2C_dotazov%C3%A1n%C3%AD.“

- Stránka byla naposledy editována 2. 6. 2011 v 15:05.

Možnosti tvorby datových skladů a metody dolování znalostí

Z Na státnice zvesela!

Základní problémy u běžných transakčních databázových systémů:

- nedosažitelnost dat skrytých v transakčních systémech
- dlouhá odezva při plnění komplikovaných dotazů
- složitá, uživatelsky nepříjemná rozhraní k databázovému softwaru
- cena v administrativě a složitost v podpoře vzdálených uživatelů
- soutěžení o počítačové zdroje mezi transakčními systémy a systémy podporujícími rozhodování

Cesta k řešení těchto problémů = datové sklady, tzv. Data Warehouse – DW

Obsah

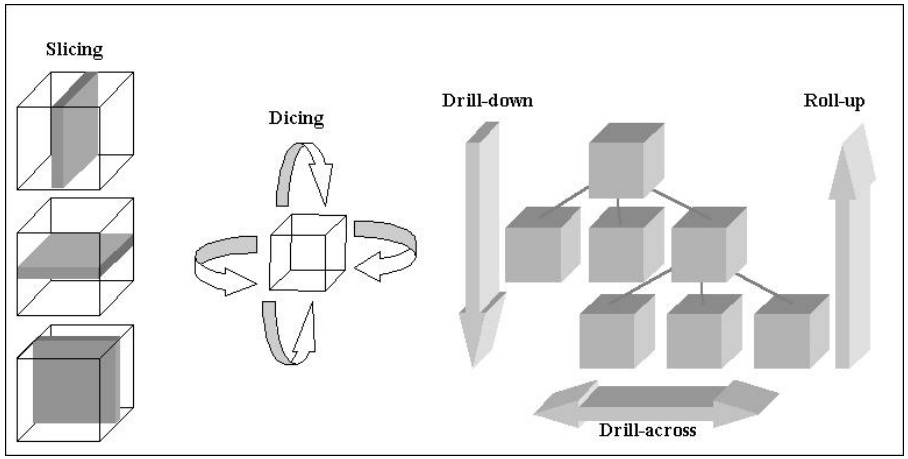
Datawarehouse

- samostatný informační systém postaven na již pořízených datech, určen především k jejich analýze
- architektura založená na relačním SŘBD, která se používá pro údržbu historických dat získaných z databází operativních dat, jenž byla sjednocena a zkontrolována před jejich použitím v databázi DW
- data z DW jsou aktualizována v delších časových intervalech, jsou vyjádřena v jednoduchých uživatelských pojmech a jsou sumarizována pro rychlou analýzu
- DW je obrovská databáze obsahující data za dlouhé časové období
- často slučuje data z více rozdílných zdrojů, které mohou obsahovat data různé kvality nebo používat nejednotné formáty a reprezentace
- objemově zabírá stovky GB až několik TB
- nemusí být databází v běžném smyslu, tj. pro přesné provádění transakcí
- je určen pro rychlé vyhledávání
- nejsou kladeny nijak důrazné požadavky na správnost a úplnost dat

Charakteristika

- data jsou uložena na různých místech ve formě relačních tabulek
  - uživatelé mohou tabulky jen číst
  - zapisovat může aktualizační program pravidelně udržující tabulky
- dotazy jsou většinou komplexní
  - podporují tzv. on-line analytické zpracování (OLAP)
    - výrazně se liší od on-line transakčního zpracování (OLTP)

- operační databáze je přizpůsobena pro podporu OLTP
  - složité OLAP dotazy by vyústily do nepřijatelné odezvy
- typické OLAP operace
  - roll-up (snížené stupně detailu -> zvýšení stupně agregace)
  - drill-down (zvýšení stupně detailu o informaci
  - slice-and-dice (selekce a projekce - změna kombinací dimenzí, které jsou zobrazovány)
  - pivot (přeorientování vícerozměrného pohledu na data)



- na základě dotazu se pospojují potřebná data do vícerozměrné tabulky (nebo více tabulek), do kterých lze klást SQL dotazy
- pro častější dotazy si uchovávají předem připravené vícerozměrné tabulky
- zátěž je většinou způsobena složitými dotazy, jež přistupují k miliónům záznamů a provádějí množství operací
- data bývají modelována vícerozměrně
  - v obchodním data warehouse mohou těmito rozměry být např. čas prodeje, místo prodeje, prodavač, výrobek, …
  - rozměry mohou být i hierarchické např. čas prodeje jako den-měsíc-čtvrtletí-rok, zboží jako výrobek-kategorie-průmysl
  - spojení více tabulek pomocí odkazu na řádky jednotlivých tabulek
  - používají speciální organizaci dat, přístupové a implementační metody, jež obecně nejsou v komerčních databázových systémech určených pro OLTP podporovány

Databázový systém – OLTP (Online Transaction Processing Systems)

- zákaznicky orientovaný
- aktuální data -- lze považovat i za slabinu, při výpadku (chybě), vznikají ztráty pro byznys
- ER schéma
- sofistikované atomické transakce i přes několik systémů(bank, po síti,...)
- velikost DB až několik GB
- jednoduché a efektivní
- příkladem je bankomat

DataWarehouse – OLAP (Online analytical Processing)

- orientovaný na trh, rychlé (oproti OLTP) získání výsledků na analytické dotazy
- historická data, *multidimenzionální* datový model
- agregovaná data (nenormalizovaná=redundantní)
- schéma hvězdy či vločky
- převážně pouze čtení
- velikost až TB

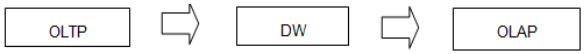
- použití: byznys reporty o prodeji, marketing, management reporty, rozpočty, finanční předpovědi a reporty

Použití DW

1. prezentace dat
2. testování hypotéz
3. objevování nových informací

Architektura DataWarehouse

- tři úrovně:
  1. klient
  2. OLAP server (MOLAP/ROLAP server)
  3. databázový server DW
- data lze organizovat v tzv. multidimenzionálním datovém modelu
  - odlišný od modelu relačního
  - odpovídá mu specializovaný software, multidimenzionální SŘBD (MDD)
  - model připomíná techniku spreadsheet ve více než dvou rozměrech
  - data jsou implementována pomocí vícerozměrných polí, jejichž dimenze odpovídají dimenzím podnikání organizace
- navržení a vytvoření DW je proces skládající se z následujících bodů:
  1. definovat architekturu, umístění a rozčlenění dat a fyzickou organizaci
  2. naplánovat kapacitu, vybrat OLAP servery a nástroje
  3. spojit servery, klientské nástroje, zdroje přes gatewaye, drivery ODBC, ...
  4. navrhnout schéma a pohledy, přístupové metody, některé složité dotazy
  5. mít skripty pro získávání, čištění, transformaci, ukládání a aktualizaci dat
  6. vytvořit koncové uživatelské aplikace
  7. spustit data warehouse i aplikace
- vytvoření je složitý proces trvající mnohdy i několik let
- mnoho organizací proto používá Data Mart umožňující rychlejší práci



Datová tržiště (Data Mart)

- DW slouží jako základna pro extrakci množin dat, resp. jejich agregaci do dílčích (replikovaných) MDD (Multidimenzionální DB)
  - MDD může pro DW sloužit ve dvou rolích
    - ”front-end” pro DW a poskytovat uživateli služby pro realizaci analytického zpracování (DW/OLAP)
    - “front-end” jednomu (několika) systémům OLTP - alternativa za DW, tj. poskytnout uživateli s OLTP data analytickým způsobem (OLTP/OLAP) – jde vlastně o datové tržiště

Systém OLAP (OnLine Analytical Processing)

- na databázové stroje jsou kladeny specifické požadavky
  1. objem zpracovávaných dat
     - transakční systém o velikosti gigabajtů dosáhne použitím jen jedné dimense velikosti desítek či stovek gigabajtů
  2. rychlost odezvy analytického systému *je* důležitá
  3. počet uživatelů současně pracujících s databází není zajímavý
     - počet pracovníků vyššího managementu je omezen
     - pro pracovníky nižších stupňů bývají údaje z datových skladů převedeny do menších specializovaných databází – datových tržišť
- s těmito omezeními se vyrovnává dvojím způsobem
  1. uzpůsobení stávajících systémů pro práci s vícerozměrovými daty
     - přidáním modulu, který to zajišťuje a prostředků pro jeho ovládání
     - v lepším případě mění způsob uložení dat, v horším“překládá” operace s vícedimenzionálními daty na operace s daty relačními

2. vytvoření speciálního systému správy dat, určeného pouze pro OLAP
    - umožňuje provést maximum optimalizací vzhledem k nárokům, jež jsou kladené analytickým způsobem práce - převažující způsob

Programy pro vytváření a plnění databáze

- převodní programy
    - načtení data z několika databází, či souborů a udělat z nich novou databázi, agregace se musí naprogramovat
- systémy znázorňující převodu dat graficky a administrátor dat namapuje zdrojová data do struktur vytvářeného datového skladu
    - výsledkem jsou buď programy (scripty) nebo přímo vykonání funkce
- moduly pro plánování jednotlivých akcí

Nástroje pro práci s daty - poslední trendy v architektuře klient/server

- nabízejí variantu tenkého klienta v podobě HTML prohlížeče

Reporting, monitorování, ad-hoc dotazy

- programy umožňující kladení dotazů a formátování odpovědí
    - nejčastěji jde o vizuální dotazovací nástroje
    - makra v tabulkovém procesoru
    - uživatelské rozhraní různě propracované:
        - zadání seskupení výsledku podle různých kriterií
        - formální kontrola dotazů
        - vytváření slovníků a metadat

MOLAP - Multidimenzionální OLAP

- datová krychle (obsahuje fakta)
- hierarchické dimenze (částečné či totální uspořádání)
    - vločkové schéma -- hlavní tabulka faktů je v relaci s dimezionálními tabulkami, přes cizí klíče, dimenzionální tabulky mohou být také v relaci s dalšími subdimenzionálními tabulkami podobně jako hlavní tabulka faktů; vytváří hierarchie dimenzí
    - hvězdové schéma -- je speciální případ vločkového, dimenzionální tabulky již nejsou v relaci s dalšími subdimenzionálními tabulkami; žádné hierarchie, jednodušší

ROLAP – Relační OLAP

- na relační architektuře založený model DW strukturou propojených DB tabulek - Relační OLAP (ROLAP) – pomalejší zpracování než MOLAP
- užívá relační nebo rozšířený relační DBMS, např server METACUBE Informix, pracuje s relačními tabulkami uspořádanými do hvězdy/vločky, adresuje pomocí klíče, data jsou neagregovaná)

Citováno z „http://wiki.zvesela.cz/index.php/Mo%C5%BEnosti_tvorby_datov%C3%BDch_sklad%C5%AF_a_metody_dolov%C3%A1n%C3%AD_znalost%C3%AD“

- Stránka byla naposledy editována 2. 6. 2011 v 16:56.

Informační systémy, jejich základní vlastnosti a typy

Z Na státnice zvesela!

Informační systém (IS) je systém pro sběr, udržování, zpracování a poskytování informací.

Funkce informačního systému

Konkrétní procesy (činnosti) podporující základní cíle informačního systému:

- získávání informací
- zpracování informací (evidence, organizace – pořádání, kategorizace, konverze –změna média, třídění, vyhledávání, agregace, odvozování nových informací, dolování znalostí)
- uložení informací (zaznamenávání a archivace dat, datová úložiště a datové sklady)
- přenos informací (v rámci počítačových sítí)
- zpřístupnění informací (tisk, zobrazení, vizualizace, šíření…)

Následující body vystihují vlastnosti, které by kvalitní IS s maximální výkonností měl splňovat.

- Musí obsahovat nutné informace, které uchovává, analyzuje a s potřebnou rychlostí předává procesům. Dané informace se týkají zejména vlastní činnosti firmy jako je výroba, evidence zákazníků, zásob, zaměstnanců, finance, stav a vývoj vlastních výrobků.
- Musí obsahovat informace o konkurenci, světovém trhu, trendech výroby, optimalizaci výrobních procesů, o místech působnosti firmy, o strategických cílech a podobně.
- Musí obsahovat moduly pro zjednodušení a urychlení výroby, čímž je míněno hlavně urychlení a zefektivnění návrhu výrobků, technologická příprava výroby a její řízení.
- Musí umožňovat rychlou komunikaci pracovníků firmy, jednotlivých pracovních úseků, ale musí také zahrnovat komunikaci se světem.
- Musí umožňovat z dostupných informací zpracovávat cíle a strategie firmy, koordinovat činnost různých procesů a tím přispívat k zefektivnění činnosti firmy.
- Musí nabízet rychlou komunikaci se zákazníkem přes počítačovou síť.
- Musí obsahovat další nutné moduly k vedení firmy jako jsou statistiky, mzdy, účetnictví, kompletní personalistika, sklad, oblast manažer – marketing, výroba a další.

Typy informačních systémů

Rozdělení dle funkce

- Systémy ERP
- Systémy na podporu rozhodování (např. BI - Business inteligence)
- Systémy na podporu plánování (např. SCM - supply chain management)
- Systémy řízení vztahů se zákazníky (např. CRM - Customer relationship management)
- Systémy pro tvorbu a správu dokumentů (např. PDM - product data management, PLM - product lifecccle management)
- Systémy na podporu návrhu a projekční činnosti (např. CAD systémy)

- CASE Systémy (např. CASE - computer aided software engineering systém)
- Knihovní systémy (např. eLibrary - hovořte o dokumentografických databázích a dalších deset minut v tahu)

Rozdělení dle technologie zpracování dat

- OLTP - umožňují skupině uživatelů vykonávat bezprostředně (online) velké množství transakcí
  - relační databáze
- OLAP - analýza velkého množství údajů, vetšinou jen pro čteni, nadstavba OLTP
  - např. BI

Rozělení IS dle uživatelů

- Veřejné informační systémy - Informační systémy, které jsou dostupné široké veřejnosti a poskytují veřejné informační služby. V tomto smyslu se jedná o jakékoli informační systémy bez ohledu na jejich provozovatele, obsah, typ, formu a příp. cenu poskytovaných informací a služeb.
- Privátní, uzavřené, neveřejné informační systémy (např. podnikové informační systémy, systémy zajišťující obranu státu, osobní informační systémy ad.).

- Stránka byla naposledy editována 11. 6. 2010 v 09:47.

Analýza informačních systémů (IS), role modelování a metodik při tvorbě IS

Z Na státnice zvesela!

Obsah

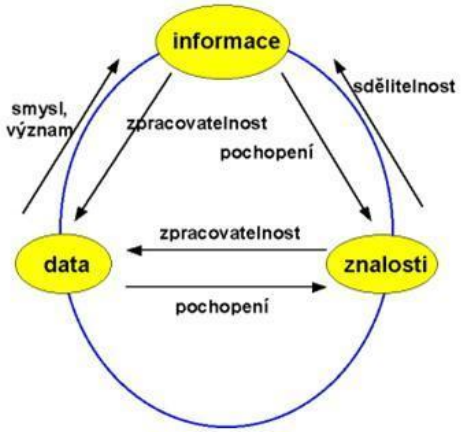
Informační systém

(IS) je systém pro sběr, udržování, zpracování a poskytování informací.

Informační systémy jsou založené na informačních a komunikačních technologiích. V poslední době se používá zkratka ICT (Information and Communication Technologies).

IS obsahuje data, znalosti a informace

- *Data* -jakékoli vyjádření (reprezentace) skutečnosti, schopné přenosu, uchování, interpretace či zpracování. Umožňují přenášet a zpracovávat odraz skutečnosti.
- *Znalosti* - výsledek poznávacího procesu, předpoklad uvědomělé činnosti. To, co jednotlivec ví po osvojení dat a po jejich začlenění do souvislostí. Účel znalostí - porozumět realitě.
- *Informace* - je definovaná pomocí dat a znalostí - data, která mají smysl (význam), sdělitelné (komunikovatelné) znalosti.



Role modelování a metodiky pří tvorbě IS

Složitost systému se promítá do složitosti jeho návrhu a realizace. (tedy i do modelování)

Role modelování a metodiky je taková, že odstraňuje tyto problémy (asi)

- Systém dělá něco jiného než by měl
- Systém řeší problémy lokálně. Důsledkem je, že jedna a tatáž věc je řešena na různých místech několikrát, po každé jinak.
- Opravy a změny systému jsou velmi obtížné a drahé. (Jestliže opravujeme chybu na základě lokálních znalostí, tak vlastně opravujeme výskyt chyby, ale ne její příčinu.)
- Systém nelze realizovat několika skupinami současně, paralelně. Bez plánu vznikají komunikační problémy.

Metodiky

Chceme-li se vyhnout potížím s lokálním rozhodováním, musíme postupovat **metodicky (ne chaoticky)**, strukturálně, dle „dobrých“ osvědčených vzorů.

Návod jak postupovat nám dávají **ověřené postupy** – vypracované metodiky.

Metodiky odrážejí určité náhledy na „realitu“, říkají „**jaké**“ kroky učinit v jakém pořadí a „**jak**“ je provádět. Dobré metodiky nám říkají i „**proč**“ to tak má být.

Metodiky jsou **konservovanou zkušeností** několika generací programátorů a projektantů. Zobecnění principů, zásad, které se osvědčily, viz historie UML.

Modely

Místo abychom se snažili popsat systém jako celek, vytváříme na něj **jednotlivé pohledy** – jeho jednotlivé, dílčí modely. Díváme se na systém postupně z jednotlivých „míst pozorování“, z jednotlivých perspektiv.

Díváme-li se na systém z jednoho místa, opomíjíme vlastnosti z tohoto místa „neviditelné“, nepodstatné a tím si práci zjednodušíme tak, že je mentálně zvládnutelná. Jednotlivé pohledy jsou jednodušší, zvládnutelné. Opomíjené vlastnosti se neztratí, jsou hlavními vlastnostmi v jiných pohledech - modelech.

Pohledy musíme volit tak, že postupně popíšeme všechny relevantní vlastnosti systému. Postupně popíšeme vše, co potřebujeme k dosažení stanoveného cíle.

Z jednotlivých pohledů lze zpětně zrekonstruovat celý systém (počítačová tomografie).

Pro tvorbu různých pohledů jsou obvykle **využity diagramy** – grafické objekty, jejichž kombinací lze tyto pohledy vytvářet.

Diagram je **graficky znázorněný model**. Diagram popisuje jistou část modelu pomocí grafických symbolů.

Tento přístup lze přirovnat k modelu stavby, který je tvořen syntézou dílčích stavebních plánů odpovídajících specifickým pohledům na stavbu – plán hrubé stavby, plánu rozvodů elektřiny, plánu rozvodů vody, … V každém z těchto plánů jsou zobrazeny pouze elementy modelu podstatné pro daný pohled, od ostatních elementů modelu je abstrahováno. **Pohledy nejsou nezávislé**, dohromady tvoří konzistentní pohled na systém, tedy konzistentní model.

Pro tvorbu diagramů systému, jejichž syntézou bude model, definuje např. **UML devět typů diagramů**.

- Stránka byla naposledy editována 2. 9. 2010 v 12:03.

Metodika návrhu a realizace informačního systému – strukturální a objektová analýza

Z Na státnice zvesela!

Existuje v zásadě několik metodik, které popisují analýzu, návrh a realizaci informačních systémů

Jedná se o více méně podrobný popis postupu, který vede návrháře jasně definovanými fázemi krok po kroku při vytváření IS

Metodiky jsou často obecné a každá firma si je může přizpůsobit pro vlastní potřebu a prostředí

Mezi nejpoužívanější patří Strukturální analýza (ta je nejstarší), SSADM (vznik 80. léta ve Velké Británii) a Objektová analýza (konec 80. let, 90. léta)

Obsah

Strukturální analýza

Postup se zaměřuje na data a jejich transformaci pomocí procesů systému (hlavními nástroji jsou tedy DFD popisující procesy a toky dat a ERD(Entity Relationship Diagram) popisující data a vztahy mezi nimi)

Nejznámější je Yourdonova strukturální analýza

Ta se zaměřuje na vytvoření logického modelu nového navrhovaného systému (esenciální model) a následně přizpůsobení implementačním požadavkům

Nedoporučuje vycházet z fyzického modelu nahrazovaného systému (např. oproti SSADM)

- Proces vytváření modelu současného systému může vyžadovat tolik času a úsilí, že uživatel se stane velmi frustrovaným a netrpělivým, což může vést až ke zrušení projektu.
- Někteří uživatelé, manažeři a programový analytici považují systémovou analýzu za plýtvání času před pravou prací (vytvářením zdrojového kódu).
- Někteří uživatelé pochybují o smyslu analýzy systému, který má být nahrazen novým.
- Někteří analytici se nechají unést modelováním a snaží se vytvořit naprosto detailní model současného systému, což vede k velkému plýtvání časem.

Esenciální model

Kompletní logický model (bez implementačních detailů) navrhovaného systému

Skládá se z:

- Model prostředí (Environmental model)
  - Definuje hranice systému a okolí
  - Obsahuje kontextový diagram a seznam událostí (event list)
- Model chování
  - Definuje vnitřní chování systému, tak aby plnil požadavky okolí
  - Používané modely (diagramy) DFD, ERD, DD (datový slovník), SP (specifikace procesů), případně STD (stavový diagram)

Toto je k tomu v materiálech PSDS

Funkční přístup chápe smysl modelu reálného světa v tom, že obsahuje souhrn stavů reálného světa a změn těchto stavů. Funkční přístup je dynamický.

Ke změnám stavů v modelu dochází prostřednictvím operací (to je abstraktní obraz událostí). Operace jsou podle různých hledisek a principu sdružovány do vyšších celků - funkcí. Příklad: Data Flow Diagram (DFD, Diagram datových toků).

Datový přístup se zaměřuje na vlastnosti (atributy) reálného světa, jejichž abstraktním obrazem v IS jsou data. Datový přístup je statický. Příklad: Entity-relationship model (ERM)

Datovým modelem reálného světa je potom systém entit, charakterizovaných jejich atributy, a jejich vzájemnými vztahy. Smyslem modelování z hlediska datového přístupu je především formulovat ideální (konceptuální) podobu uspořádání dat v informačním systému, která je „věrným“ obrazem reálného světa.

Objektová analýza

Zaměřuje se na objekty reálného světa a jejich třídy a komunikaci mezi objekty

Mezi 1989 a 1994 navrženo cca 40 nových OO metodik (např.: OMT, OOSE, OOD, aj.). Metody měli tolik společného, že se tři klíčoví autoři (Booch, Jacobson a Rumbaugh) rozhodli své návrhy integrovat do jedné metodiky RUP (Rational Unified Process)

Vychází ze specifikace požadavků, snažíme se v průběhu analýzy najít všechny objekty relevantní pro navrhovaný systém a vztahy mezi nimi.

Ze začátku jde pouze o hrubý nástin objektů (doménový model), v pozdějších fázích se třídy zpřesňují až na úroveň objektů reprezentujících fyzické třídy programovacího jazyka

Objektových metodik je více, mezi nejznámější patří RUP

Nástroje OA:

- Často se používá modelovací jazyk UML
  - Model případů užití
  - Doménový model
  - Diagram tříd

    - Stavové diagramy, sekvenční diagramy a další

- Existují i další pomocné nástroje:
    - CRC karty (Class-Responsibility-Collaboration cards)

Umožňuje zvládnout návrh i složitých a velkých systémů

- Hierarchie objektů – sdružovány podle logických souvislostí do balíků a podbalíků
- Přirozený přechod od analýzy k návrhu

Pro vytvoření objektového návrhu je zapotřebí:

- Identifikace objektů a tříd
    - Založeno často na analýze textového popisu řešeného problému, podstatná jména jsou kandidáti na objekty, slovesa na metody objektů a interakci objektů
- Příklady objektů – faktura, zaměstnanec, zakázka, atd.
- Nalezení chování (metod) objektů
- Nalezení vazeb mezi objekty a jejich uspořádání do hierarchií
    - Při pohledu na objekty získané shora uvedenými metodami je možno rozpoznat u některých společné rysy, opakující se metody, atd.
    - Tyto charakteristiky se v objektovém modelu převedou do hierarchie vazeb dědičnosti a/nebo celek-část (reprezentace charakteristik objektů), podle pravidel správného objektového návrhu.

Doménová analýza:

- Hledají se objekty, operace a vazby, které znalci z problémové oblasti (např. uživatelé nebo správci stávajících systémů, příslušní úředníci, ...) pokládají za důležité (často používají jejich názvy apod.).
- Konstrukce obecného modelu domény (tj. např. systému výplaty mezd) podle konzultací se znalci (bude obsahovat jak používané objekty a jejich vztahy, tak standardní vzory chování).
- Zkoumání stávajících systémů pro doménu -> inspirace. Identifikace podobností a odlišností mezi nimi podle konzultací.

Typy přístupů v návrhu IS

- Zdrojový (datový)

Stanovení **Datového modelu** reality a nového systému (zajímají nás uživatelské potřeby) -> informační (datová) analýza

- Funkční přístup

Stanovení **cílů**, které by měl nový systém splňovat (zajímá nás fungování systému) -> funkční analýza

- Top-down

Postup od abstraktního ke konkrétnímu

- Bottom-up

Postup od konkrétnímu k obecnému

- Strukturovaný

Aplikace má podobu hierarchie funkcí, která je realizována strukturovanými programy. Styl práce AKCE -> OBJEKT

- Objektově orientovaný

Funkce jsou realizovány pomocí "programových fragmentů" nad stabilními strukturami databází. Styl práce OBJEKT -> AKCE

- Rigorózní (striktní)
- Agilní (light)

- Plánovací
- Konstrukční
- Implementační

Metodologie

Strukturovaná metodologie

- Entitně relační model (ERA, ERD, E-R)

Grafický prostředek pro analýzu a zobrazení datového modelu systému

- Diagram datových toků (DFD - Data Flow Diagram)

Grafický prostředek návrhu a zobrazení funkčního modelu systému

Objektová metodologie

Objekt - cokoliv reálného či abstraktního s jasně vymezenou rolí v daném kontextu. Objekt má 3 charakteristiky:

- Stav
- Identita
- Chování

Vlastnosti objektů:

- Sebeidentifikace
- Zapouzdření (atributů a funkcí do objektu)
- Klasifikace (objekty jsou sdruženy do tříd)
- Dědičnost
- Polymorfismus (volání přetížených metod, metod se stejným jménem u různých stříd pomocí rozhraní)

Přínosy:

- Podpora složitých objektů
- Možnost opakovaného použití objektů

Krásný zdroj k celé otázce (http://web.sks.cz/users/ku/DOKUMENTY/pri_syl.pdf)

Citováno z „http://wiki.zvesela.cz/index.php/Metodika_n%C3%A1vrhu_a_realizace_informa%C4%8Dn%C3%ADho_syst%C3%A9mu_%E2%80%93_struktur%C3%A1ln%C3%AD_a_objektov%C3%A1_anal%C3%BDza“

- Stránka byla naposledy editována 5. 9. 2010 v 22:23.

Zpracování požadavků, objektové modelování, diagramy UML

Z Na státnice zvesela!

viz NIS

Citováno z „http://wiki.zvesela.cz/index.php/Zpracov%C3%A1n%C3%AD_po%C5%BEadavk%C5%AF%2C_objektov%C3%A9_modelov%C3%A1n%C3%AD%2C_diagramy_UML“

- Stránka byla naposledy editována 10. 6. 2010 v 10:26.

Datové modelování, perzistence objektů, konceptuální a fyzický datový model

Z Na státnice zvesela!

Obsah

Datové modelování

- jedna ze základních funkcí IS je ukládání a následné zpracování informací ve formě dat, proto se provádí při návrhu IS také datové modelování
- jeho úkolem je zvolit jaké objekty, jako nositele informace, potřebujeme ukládat do trvalé paměti a jaké jejich vlastnosti a vztahy mezi nimy chceme uchovávat
- při datovém modelování obvykle vytváříme konceptuální a fyzický datový model a také určujeme způsob perzistence objektů

Perzistence objektů

- zabývá se kam a jakým způsobem budou objekty trvale uloženy
- objekty se obvykle ukládají do relační databáze ve formě datových záznamů v tabulkách
- pro programový přístup do databáze se často používá standardizované softvarové API pro databáze jako ODBC nebo JDBC
- pro usnadnění programátorské práce při vytváření perzistentní vrstvy můžeme využít *objektově-relační mapování*, které nám zajistí automatickou transformaci ukládaných objektů do záznamů relační databáze - např. framework Hibernate pro Java aplikace

Konceptuální datový model

- vyjadřuje jaké objekty a jejich atributy budeme ukládat a vztahy mezi nimy
- pro grafické vyjádření se používají ERA modely (diagramy)

Fyzický datový model

- odvozuje se z konceptuálního modelu a vyjadřuje navíc jak přesně budou data v konkrétní databázi uložena
- také se popisuje ERA modely, které obsahují i přesné databázové typy atributů tabulek

ERA modely

- viz DB1

Objektově-relační mapování

- viz popis Java Hibernate v otázce 10. „Vnější“ programování (přes rozhraní/knihovny) – rozhraní ODBC, JDBC, rozhraní podporující objektově- relační mapování (Java Hibernate) (http://wiki.zvesela.cz/index.php/„Vnější“_programování_(přes_rozhraní/knihovny)_–_rozhraní_ODBC%2C_JDBC%2C_rozhraní_podporující_objektově-_relační_mapování_(Java_Hibernate))

Citováno z „http://wiki.zvesela.cz/index.php/Datov%C3%A9_modelov%C3%A1n%C3%AD%2C_perzistence_objekt%C5%AF%2C_konceptu%C3%A1ln%C3%AD_a_fyzick%C3%BD_datov%C3%BD_model“

- Stránka byla naposledy editována 2. 9. 2010 v 13:22.

Datové sklady (Data Warehousing), OLAP systémy, jejich význam a oblasti využití, základními principy, dimenze, agregace, extrakce a transformace dat, srovnání transakčních a analytických systémů (OLAP a OLTP technologií)

Z Na státnice zvesela!

Základní problémy u běžných transakčních databázových systémů:

- nedosažitelnost dat skrytých v transakčních systémech
- dlouhá odezva při plnění komplikovaných dotazů
- složitá, uživatelsky nepříjemná rozhraní k databázovému softwaru
- cena v administrativě a složitost v podpoře vzdálených uživatelů
- soutěžení o počítačové zdroje mezi transakčními systémy a systémy podporujícími rozhodování

Cesta k řešení těchto problémů = datové sklady, tzv. Data Warehouse – DW

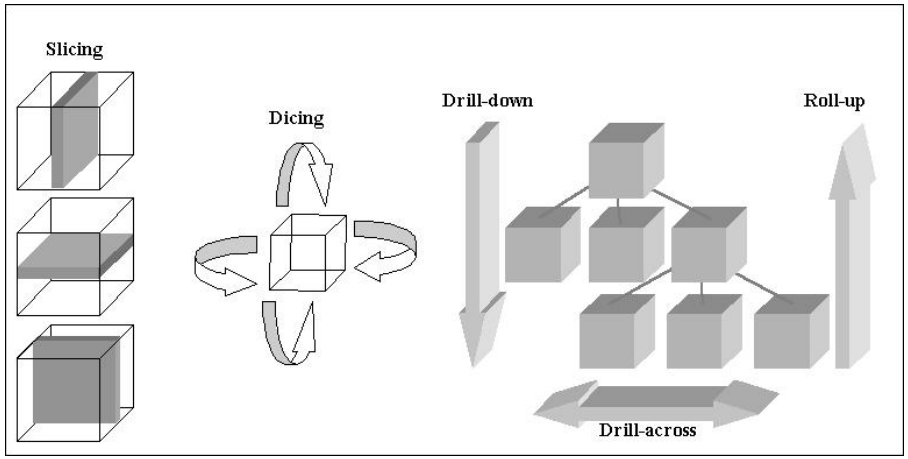
Obsah

Datawarehouse

- samostatný informační systém postaven na již pořízených datech, určen především k jejich analýze
- architektura založená na relačním SŘBD, která se používá pro údržbu historických dat získaných z databází operativních dat, jenž byla sjednocena a zkontrolována před jejich použitím v databázi DW
- data z DW jsou aktualizována v delších časových intervalech, jsou vyjádřena v jednoduchých uživatelských pojmech a jsou sumarizována pro rychlou analýzu
- DW je obrovská databáze obsahující data za dlouhé časové období
- často slučuje data z více rozdílných zdrojů, které mohou obsahovat data různé kvality nebo používat nejednotné formáty a reprezentace
- objemově zabírá stovky GB až několik TB
- nemusí být databází v běžném smyslu, tj. pro přesné provádění transakcí
- je určen pro rychlé vyhledávání
- nejsou kladeny nijak důrazné požadavky na správnost a úplnost dat

Charakteristika

- data jsou uložena na různých místech ve formě relačních tabulek
  - uživatelé mohou tabulky jen číst
  - zapisovat může aktualizační program pravidelně udržující tabulky
- dotazy jsou většinou komplexní
  - podporují tzv. on-line analytické zpracování (OLAP)
    - výrazně se liší od on-line transakčního zpracování (OLTP)
    - operační databáze je přizpůsobena pro podporu OLTP
      - složité OLAP dotazy by vyústily do nepřijatelné odezvy
    - typické OLAP operace
      - roll-up (snížené stupně detailu -> zvýšení stupně agregace)
      - drill-down (zvýšení stupně detailu o informaci
      - slice-and-dice (selekce a projekce - změna kombinací dimenzí, které jsou zobrazovány)
      - pivot (přeorientování vícerozměrného pohledu na data)



- na základě dotazu se pospojují potřebná data do vícerozměrné tabulky (nebo více tabulek), do kterých lze klást SQL dotazy
- pro častější dotazy si uchovávají předem připravené vícerozměrné tabulky
- zátěž je většinou způsobena složitými dotazy, jež přistupují k miliónům záznamů a provádějí množství operací
- data bývají modelována vícerozměrně
  - v obchodním data warehouse mohou těmito rozměry být např. čas prodeje, místo prodeje, prodavač, výrobek, …
  - rozměry mohou být i hierarchické např. čas prodeje jako den-měsíc-čtvrtletí-rok, zboží jako výrobek-kategorie-průmysl
  - spojení více tabulek pomocí odkazu na řádky jednotlivých tabulek
  - používají speciální organizaci dat, přístupové a implementační metody, jež obecně nejsou v komerčních databázových systémech určených pro OLTP podporovány

Databázový systém – OLTP (Online Transaction Processing Systems)

- zákaznicky orientovaný
- aktuální data -- lze považovat i za slabinu, při výpadku (chybě), vznikají ztráty pro byznys
- ER schéma
- sofistikované atomické transakce i přes několik systémů(bank, po síti,...)
- velikost DB až několik GB
- jednoduché a efektivní
- příkladem je bankomat

DataWarehouse – OLAP (Online analytical Processing)

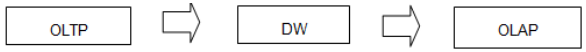
- orientovaný na trh, rychlé (oproti OLTP) získání výsledků na analytické dotazy
- historická data, *multidimenzionální* datový model
- agregovaná data (nenormalizovaná=redundantní)
- schéma hvězdy či vločky
- převážně pouze čtení
- velikost až TB
- použití: byznys reporty o prodeji, marketing, management reporty, rozpočty, finanční předpovědi a reporty

Použití DW

1. prezentace dat
2. testování hypotéz
3. objevování nových informací

Architektura DataWarehouse

- tři úrovně:
  1. klient
  2. OLAP server (MOLAP/ROLAP server)
  3. databázový server DW
- data lze organizovat v tzv. multidimenzionálním datovém modelu
  - odlišný od modelu relačního
  - odpovídá mu specializovaný software, multidimenzionální SŘBD (MDD)
  - model připomíná techniku spreadsheet ve více než dvou rozměrech
  - data jsou implementována pomocí vícerozměrných polí, jejichž dimenze odpovídají dimenzím podnikání organizace
- navržení a vytvoření DW je proces skládající se z následujících bodů:
  1. definovat architekturu, umístění a rozčlenění dat a fyzickou organizaci
  2. naplánovat kapacitu, vybrat OLAP servery a nástroje
  3. spojit servery, klientské nástroje, zdroje přes gatewaye, drivery ODBC, ...
  4. navrhnout schéma a pohledy, přístupové metody, některé složité dotazy
  5. mít skripty pro získávání, čištění, transformaci, ukládání a aktualizaci dat
  6. vytvořit koncové uživatelské aplikace
  7. spustit data warehouse i aplikace
- vytvoření je složitý proces trvající mnohdy i několik let
- mnoho organizací proto používá Data Mart umožňující rychlejší práci



Datová tržiště (Data Mart)

- DW slouží jako základna pro extrakci množin dat, resp. jejich agregaci do dílčích (replikovaných) MDD (Multidimenzionální DB)
  - MDD může pro DW sloužit ve dvou rolích
    - ”front-end” pro DW a poskytovat uživateli služby pro realizaci analytického zpracování (DW/OLAP)
    - “front-end” jednomu (několika) systémům OLTP - alternativa za DW, tj. poskytnout uživateli s OLTP data analytickým způsobem (OLTP/OLAP) – jde vlastně o datové tržiště

Systém OLAP (OnLine Analytical Processing)

- na databázové stroje jsou kladeny specifické požadavky
  1. objem zpracovávaných dat
     - transakční systém o velikosti gigabajtů dosáhne použitím jen jedné dimense velikosti desítek či stovek gigabajtů
  2. rychlost odezvy analytického systému *je* důležitá
  3. počet uživatelů současně pracujících s databází není zajímavý
     - počet pracovníků vyššího managementu je omezen
     - pro pracovníky nižších stupňů bývají údaje z datových skladů převedeny do menších specializovaných

databází – datových tržišť
- s těmito omezeními se vyrovnává dvojím způsobem
  1. uzpůsobení stávajících systémů pro práci s vícerozměrovými daty
     - přidáním modulu, který to zajišťuje a prostředků pro jeho ovládání
     - v lepším případě mění způsob uložení dat, v horším“překládá” operace s vícedimenzionálními daty na operace s daty relačními
  2. vytvoření speciálního systému správy dat, určeného pouze pro OLAP
     - umožňuje provést maximum optimalizací vzhledem k nárokům, jež jsou kladené analytickým způsobem práce - převažující způsob

Programy pro vytváření a plnění databáze

- převodní programy
  - načtení data z několika databází, či souborů a udělat z nich novou databázi, agregace se musí naprogramovat
- systémy znázorňující převodu dat graficky a administrátor dat namapuje zdrojová data do struktur vytvářeného datového skladu
  - výsledkem jsou buď programy (scripty) nebo přímo vykonání funkce
- moduly pro plánování jednotlivých akcí

Nástroje pro práci s daty - poslední trendy v architektuře klient/server

- nabízejí variantu tenkého klienta v podobě HTML prohlížeče

Reporting, monitorování, ad-hoc dotazy

- programy umožňující kladení dotazů a formátování odpovědí
  - nejčastěji jde o vizuální dotazovací nástroje
  - makra v tabulkovém procesoru
  - uživatelské rozhraní různě propracované:
    - zadání seskupení výsledku podle různých kriterií
    - formální kontrola dotazů
    - vytváření slovníků a metadat

MOLAP - Multidimenzionální OLAP

- datová krychle (obsahuje fakta)
- hierarchické dimenze (částečné či totální uspořádání)
  - vločkové schéma -- hlavní tabulka faktů je v relaci s dimezionálními tabulkami, přes cizí klíče, dimenzionální tabulky mohou být také v relaci s dalšími subdimenzionálními tabulkami podobně jako hlavní tabulka faktů; vytváří hierarchie dimenzí
  - hvězdové schéma -- je speciální případ vločkového, dimenzionální tabulky již nejsou v relaci s dalšími subdimenzionálními tabulkami; žádné hierarchie, jednodušší

ROLAP – Relační OLAP

- na relační architektuře založený model DW strukturou propojených DB tabulek - Relační OLAP (ROLAP) – pomalejší zpracování než MOLAP
- užívá relační nebo rozšířený relační DBMS, např server METACUBE Informix, pracuje s relačními tabulkami uspořádanými do hvězdy/vločky, adresuje pomocí klíče, data jsou neagregovaná)

- Stránka byla naposledy editována 2. 6. 2011 v 17:56.

Vlastnosti CASE nástrojů a jejich význam v analýze a návrhu informačních systémů.

Z Na státnice zvesela!

CASE - Computer Aided Software (System) Engineering

Modely softwarových systémů jsou příliš složité

- nutná podpora různých úrovní abstrakce, různých pohledů
- nutnost rozdělení mezi jednotlivé vývojáře

Obsah

CASE nástroje

- slouží pro standardizaci používaných postupů
- nástroj na podporu práce analytiků a programátorů při vývoji informačních systémů, zejména ve fázi analýzy a návrhu – tvorba modelů
- mezistupeň mezi analýzou problému a programováním
- označení pro integrovanou tvorbu programových aplikací pomocí programových prostředků s minimální potřebou manuálního psaní zdrojového kódu programu

obsah: databáze (repository, systémová encyklopedie) navrhovaných prvků informačního systému

funkce: podpora realizace projektu informačního systému

základ: metodika = návod na vytváření modelů a určení závislostí mezi nimi

Dělení CASE systémů

- Nižší CASE – podpora tvorby software
  - návrhy obrazovek, formulářů, menu
  - jazyková podpora
- Vyšší CASE – podpora analýzy a návrhu
  - tvorba diagramů a modelů
  - kontrola konzistence modelu
  - Příklad: AxiomSys: strukturovaná analýza
- Integrované CASE – podpora živ. cyklu
  - od analýzy po generování kódu
  - round-trip engineering
  - podpora tvorby dokumentace
  - Příklad: Oracle Designer
- Komponentové CASE – otevřenost
  - repository s rozhraním (SCM, testování)
  - integrace nástrojů od různých výrobců
  - Příklad: Rational Suite Enterprise

Dělení CASE systémů dle rozsahu možností

- Jedna fáze
  - podpora jedné fáze ŽC (analýza, prog.)
- Jedna metodika
  - podpora dané metodiky přes ŽC
- Více fází, více metodik
  - transformace modelů, vlastní metodiky
- Vývoj + management
  - včetně podpůrných funkcí pro řízení

Vlastnosti CASE systémů

Kladné

- Zvýšení produktivity
  - automatizace prací - čas na podstatné věci
  - lepší přehled o modelu a implementaci
  - podpora rozdělení práce
  - snazší údržba dokumentace i systému
- Zvýšení kvality
  - podpora analýzy - lepší znalost požadavků
  - kontroly konzistence a úplnosti
  - synchronizace reality a dokumentace
  - podpora používání standardů

Záporné

- Cena
  - CASE jsou drahé (řádově 100000 Kč)
  - vybírat s rozvahou (reklamní slogany vs. skutečné možnosti vs. skutečné potřeby)
- Doba návratnosti investice
  - na počátku potřebné zaškolení a zaučení
  - přínosy obvykle až od 2-3 projektu
- Změna stylu práce
  - nástroj bez používání metodiky k ničemu
  - nutnost podřídit se CASE (techniky, metoda)
  - podpora managementu nutná
  - jen kód je aktuální (pomalé updatování modelu při změnách)

Použití CASE systémů

- automatizovaná evidence vytvořených objektů a specifikací, dokumentace vývoje systému
- grafická podpora modelování (notace)
- kontrola správnosti modelů podle zvolené metodiky, zajištění integrity a konzistence návrhu (předem definovaná integritní omezení a jejich kontrola, automatické uplatnění změn vytvořených v jedné části ve všech souvisejících částech návrhu)
- podpora týmové práce (identifikace osob a týmů zodpovědných za jednotlivé modely, tvorba více modelů současně, současná práce více osob na jednom modelu)
- správa verzí
- automatický převod definovaných modelů do konkrétního logického a fyzického návrhu (generování programu, popisu databáze, příp. prototypu)
- reverse engineering – zpětné generování konceptuálního modelu z existující aplikace

Některé příklady CASE systémů

- Powerdesigner (Sybase)
- Together (Borland)
- Oracle designer (Oracle)
- SELECT (LBMS)
- Rational Rose (Rational Software Corporation)
- Enterprise Architect (Sparx)

- Stránka byla naposledy editována 18. 5. 2010 v 08:37.