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

Embedded SQL

Přímý přístup programovacího jazyka do databázových struktur – **jazyk (překladač) je obohacený o konstrukce pro SQL**

- 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 klasicky velké databázové systémy

- 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 nedostačovaly a tak si začali implementovat vlastní procedurální rozšíření SQL.
- Příklad: kursor

Motivace

- Šetření komunikačního kanálu
    - Menší množství odesílaných povelů

    - v jednom povelu je větší množství příkazů
- Podstatně menší objem přenesených 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 špatně, bude zadáno špatně
- Integritní omezení, triggery
    - Později je čištění nekonzistentních dat namáhavé a opravy často nemožné
    - Lépe ohlídat vše

Procedurální rozšíření v SQL:1999

SQL:1999 standardizuje procedurální rozšíření. Rozšíření se jmenuje *SQL/PSM - SQL/Persisted Stored Modules*. **Nikdo to moc nedodržuje a všichni si dělají vlastní standardy/implementace**

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

Podpora v databázích

SQL:1999 (SQL/PSM)

- IBM DB2
- MySQL
- PostgreSQL

Vlastní (proprietární) rozšíření

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

Jazyk PL/SQL

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

PL/SQL přidává k jazyku SQL konstrukce procedurálního programování.

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 část
EXCEPTION
 ošetření výjimek
```

```
END;
```

Pouze výkonná sekce je povinná, ostatní jsou doporučené.

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.

Kurzory – definice, klasifikace, použití kurzorů.

Definice

- 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.

Klasifikace

- explicitní kurzor
  - nutno deklarovat, otevřít, načíst data a uzavřít

```
DECLARE CURSOR <jméno kurzoru>IS <dotaz>;
OPEN <jméno kurzoru>;
FETCH <jméno kurzoru>INTO <jméno proměnné1>, <jméno proměnné2>, …;
CLOSE <jméno kurzoru>;
```

- 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
  - musí se shodovat datové typy sloupců a proměnných
  - implicitní kurzor SELECT musí vracet pouze jeden řádek (**SELECT INTO** !)

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

Použití kurzorů

Když je potřeba iterovat přes hromadu položek a pro každou položku něco provést (tedy ne pro všechny najednou, ale pro každou zvlášť)

- v triggerech
- v uložených procedurách

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;
```

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

Procedury a funkce

- POJMENOVANÁ posloupnost příkazů, část programu, kterou můžeme opakovaně volat,
- jsou si poměrně dost podobné,
- subrutina uložená v datové struktuře databáze,
- přístupná aplikacím přistupujícím k databázi (sdílení kódu),
- s jejich pomocí můžeme řadu úloh vyřešit přímo v databázi bez potřeby instalace dalšího dodatečného software - generování testovacích dat, filtrování, transformace a podobně,
- **Opravitelnost:** Mnohé chyby v kódu uložené procedury se dají opravovat bez nutnosti distribuovat a instalovat nové verze aplikace přistupující k databázi.
- **Jednoduché rozhraní**
- **Menší přesuny dat**

Nevýhody

- Jazyky uložených procedur jsou obecně mezi různými databázovými servery navzájem nekompatibilní (SQL/PSM definované standardem zdaleka není široce rozšířeným jazykem).
- Některé databázové servery nepodporují uložené procedury vůbec.
- Většinou jsou prostředky pro ladění kódu uložených prostředků chudší než nástroje k ladění ve vyšších jazycích.

Procedura

- nemusí mít vstupy,
- nemusí vracet výstup, a když vrací, tak RESULT SET (pro zpracování kursorem),
- často se používá pro periodické (časované) volání ,

- vždycky se volá pomocí CALL proc nebo EXECUTE proc (funkci lze zavolat i uvnitř dotazu),
- externí procedury (v deklaraci je uveden pouze odkaz do externí knihovny),

Funkce

- Vrací právě jednu návratovou hodnotu (ne SET)

Balíky

- Roztřídění a propojení logicky/funkčně příbuzných funkcí a procedur a proměnných;
- možnost deklarace public a private proměnných, fcí a procedur konstant, kursorů…;
- vyšší výkon;
- nejdřív se definuje hlavička (názvy fcí a procedur, konstanty) a pak BODY;
- volání přes tečkovou notaci BALÍK.FUNKCE;
- **implicitní balík STANDARD**

Kompilace a spuštění

Vynucení uložená procedura k překompilovat z jiného důvodu je v případě potřeby vyvážení chování "parametr šňupání" kompilace uložená procedura. (Microsoft)

- PL/SQL procedury a funkce lze urychlit jejich kompilací do nativního kódu,
- defaultní způsob provádění procedur je v interpretovaném režimu (dá se změnit),
- urychlení práce s daty není moc velké, proto se vyplatí kompilovat jenom procedury s velkým podílem výpočetního kódu a málo voláním SQL
- kompilace se provádí ručně voláním ALTER PROCEDURE[FUNCTION] jmproc COMPILE;

Aktivní databáze – Oracle triggery, klasifikace a spouštění triggerů.

Databáze je aktivní když má v sobě triggery. Tak je totiž schopná pouze na základě akce vložení/úpravy dat nad nimi sama vyvolat nějakou akci.

Databáze, které umožňují automatickou propagaci akcí mimo jiné k udržení platnosti jistých typů integritních omezení.

- databázový trigger je procedura PL/SQL spojená s tabulkou
- trigger se automaticky provede při provádění některého příkazu SQL, je-li splněna podmínka triggeru.
- v těle řádkového triggeru je k dispozici jak OLD, tak NEW hodnota aktuálního řádku

Odlišnosti triggerů od uložených podprogramů

- jsou implicitně spouštěny při modifikaci tabulky
- definují se pouze pro databázové tabulky
- nepřijímají argumenty
- lze je spustit pouze při 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

Klasifikace a spouštění

Klasifikace podle voleb spuštění

- Podle akce: INSERT, UPDATE, DELETE
- Podle následnosti: BEFORE, AFTER, INSTEAD OF
- Rozsah: řádkový (FOR EACH ROW), příkazový

Transakce, dvoufázový uzamykací protokol, detekce uváznutí

Databázové transakce musí splňovat tzv. vlastnosti **ACID**

- ***A - Atomicity***

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

- ***C - Consistency* - konzistentnost**

Při a po provedení transakce není porušeno žádné integritní omezení.

- ***I - Isolation* - 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).

- ***D - Durability* - 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 zamykání

- 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.

Prováděné 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

- Zase **Best Practice**
- V první fázi se jen zamyká, v druhé jen odemyká (takže ne Lock-Unlock-Lock-Unlock)
- 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)**
  - uvolňuje zámky až po skončení transakce (COMMIT). Zřejmá nevýhoda je omezení paralelismu. S2PL navíc stále nevylučuje možnost deadlocku.
- **Konzervativní dvoufázový protokol (C2PL)**
  - 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).

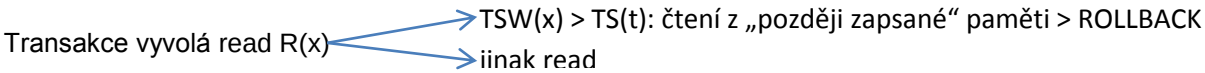
Detekce uváznutí a zotavení

- 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
- Hledá se takový plán transakcí, aby se co nejmíň kryly a tak, aby byl dodržen princip ACID (hlavně Isolation), když se takový plán povede najít, nazývá se **uspořadatelný**
- **Well formed transakce** (správně zamykat a odemykat)

- **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 oběť 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: **metoda kontrolních bodů** (checkpointing) – konzistentní mezistavy

Zajištění uspořadatelnosti pomocí pořadových čísel transakcí

Transakce vyvolá write W(x)
- TSR(x) > TS(t): zápis do „později přečtené“ paměti > ROLLBACK
- TSW(x) > TS(t): zápis do „později přepsané“ paměti > ROLLBACK
- jinak proveď zápis



Optimalizace dotazu – Rule Based optimalizace (RBO), Cost Based optimalizace (CBO), podstata optimalizátoru, přínos optimalizace.

- 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
  - přenesená data
- u malých databází optimalizace nepatrná, projeví se až u objemných, nebo u často navštěvovaných webů může při špatně formulovaných dotazech vzrůst trafic v obou směrech

zpracování příkazů se skládá z následujících komponent:

- parser
- optimalizátor
- generátor řádkových zdrojů
- vlastní provádění SQL

CBO/RBO

Oracle doporučuje používat pouze CBO, který je stále vylepšován a RBO je implementován hlavně kvůli zpětné kompatibilitě.

RBO

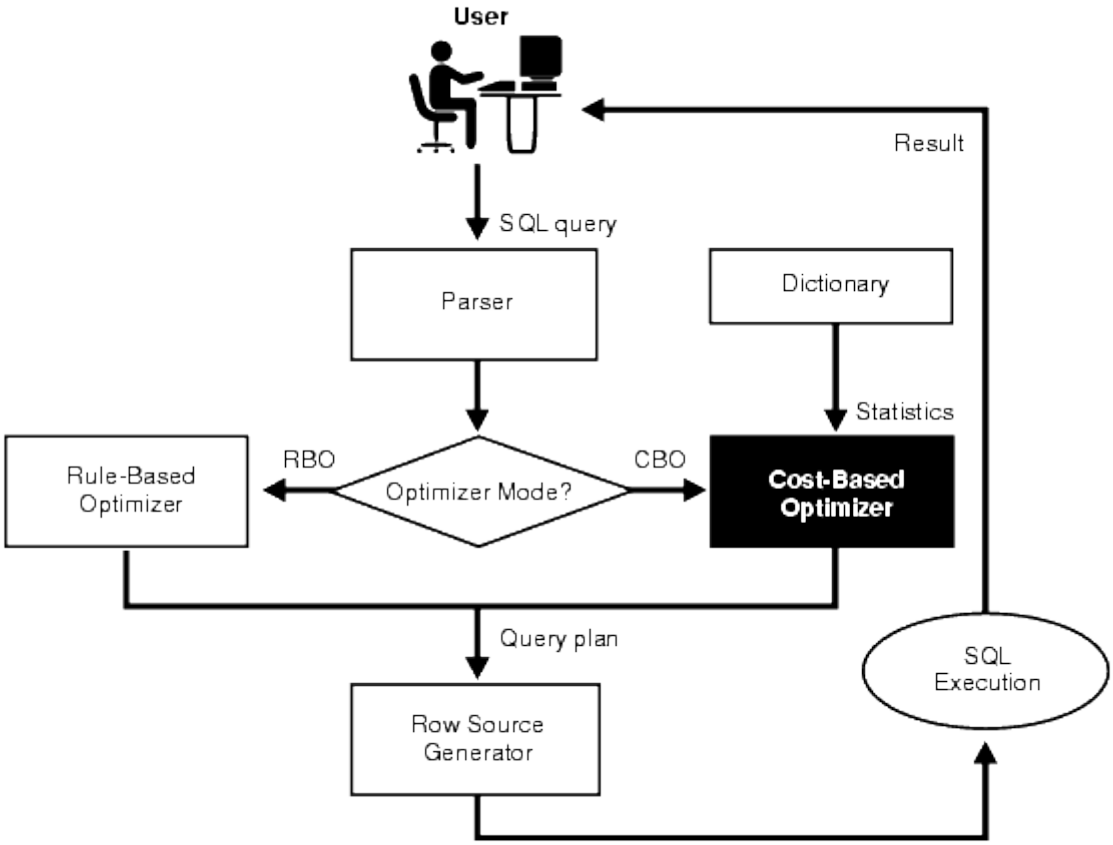
- Starší přístup, dnes často deprecated (Oracle),
- řídí se předem sestavenou sadou pravidel, která nezohledňují např. velikost tabulky:
  - **Malá tabulka** (obsahuje 5 řádků a vejde se do jednoho datového bloku), je rychlejší jí přečíst celou než hledat podle indexu (1 IO operace vs. čtení bloku indexů a pak dat)

CBO

- Novější přístup, pořád se vyvíjí,
- založený na statistikách a měření rychlosti různých typů dotazů (velikost, histogram, počet unikátních hodnot),
- zohledňuje aktuální stav (velikost, clustering, hustota NULL polí),
- nevýhoda je větší náročnost a nutnost udržovat statistiky (zase ve formě tabulek, se zamykáním atd.)

Podstata optimalizátoru a přínos

Podstata: stejná data lze z databáze získat různými dotazy (SELECT * vs. SELECT col1, col2…), výsledek bude stejný, ovšem zpracování se může lišit potřebným časem a systémovými nároky.



SQL Hint – nápověda optimalizátoru

Přínos:

- zdrojový čas
- kapacitu paměti či prostoru
- programátorskou práci
- přenesená data

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

Evoluce relačních databází. Každá současná databáze je postrelační, žádná už není holá RDB, např. aktivní databáze (triggery) už jsou postrelační databáze.

- **RDB** - relational database
- **ORDB** - object-relational database
- **OODB** - object oriented database

Výhody a nevýhody

Co je v jednom případě výhoda, může být jinde nevýhoda.

Relační model

- jednoduchý a elegantní,
- 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.

Objektově orientovaný model

- založen na objektech, což jsou struktury, které kombinují daný kód a data,
- objektově orientovaný přístup je bližší programátorovi (nemusí v hlavě přepínat kontext)
- ODB jsou výborné pro manipulaci s daty,
- větší flexibilita struktury (nemusí být jen sloupce a řádky)
- některé typy dotazů jsou efektivnější než v RDB díky dědičnosti a referencím,
- mapování může vést k výrazné neefektivitě pokud je aplikováno slepě a bez znalosti zákonitostí DB

Mapování

Mezivrstva mezi OO jazykem a RDB, poskytuje OO přístup programátorovi ale jako úložiště využívá RDB.

- Objekty reálného světa jsou v aplikaci reprezentovány jako entity. Zatímco je v relační databázi entita reprezentována jako řádek, resp. množina řádků v databázových tabulkách, tak v objektově orientovaném jazyce je entita zpravidla reprezentována instancí nějaké třídy.
- Hlavním cílem ORM je synchronizace mezi používanými objekty v aplikaci a jejich reprezentací v databázovém systému tak, aby byla zajištěna persistence dat. Vývojář potřebuje persistentně uchovávat objekty, ale nepotřebuje se starat, jak se tato persistence provede.
- Rozdíl mezi relační databází a objekty může přinášet komplikace.
    - Rozdíly mezi datovými typy v databázi a v objektech
    - Rozdíly ve struktuře dat a integritních omezeních
    - Rozdíly v prováděných operacích nad databází a nad objekty
    - Rozdíly v provádění transakcí
    - Deklarativní vs. imperativní přístup

RDB

- 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.
- Deklarativní jazyk (SQL)

ORDB

- Pořád tabulky a řádky, ale možnost ADT (přidávají objektovost do tabulek),
- Krok k objektům ale pořád založeno na RDB, proto je objektovost omezená,
- "Rozšířená relační" a "objektově-relační" jsou synonyma pro databázové systémy, které se snaží sjednotit rysy jak relačních, tak objektových databází.
- Informix, IBM, Oracle a Unisys.
- Jazyk SQL s rozšířením pro přístup k ADT je stále hlavním rozhraním pro práci s databází. Příma podpora objektových jazyků stále chybí, což nutí programátory k překladu mezi objekty a tabulkami

OODB

- Datový model má objektově orientované aspekty jako třídy s atributy a metodami a integritními omezeními;
- poskytují objektové identifikátory (OID) pro každou trvalou instanci třídy (automaticky, v RDB musí uživatel zavést klíč); 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.

ANSI/ISO normy SQL – objektové vlastnosti jazyka SQL99 !!! biflovačka

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

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

Ostatní

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 obvykle dodržují 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

Objektově relační databáze

Nové prvky přinášejí změnu v přístupu návrhu datové základny.

	RDB	**ORDB**	**OODB**
Definovaný standard	SQL2 (ANSI X3H2)	SQL3/4 (in process)	ODMG-V2.0

ODMG: Object Data Management Group

- zajištění kompatibility s existujícími relačními databázemi
- ponechání jazyka SQL
- volitelně primární klíče
- existenci pohledů
- mechanizmus tvorby relací.

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 tečka (.)(.)
- dědičnost, polymorfismus

Hnízděné tabulky

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);

CREATE TABLE FilmovyHerec OF typherec;
```

Kolekce

- **Kolekcemi** se zde rozumí **proměnná pole a vnořené tabulky**. Vnořená tabulka je vyjádřena jako sloupec jiné (hlavní) tabulky.

Abstraktní datové typy

- umožňují zapouzdření atributů a operací (na rozdíl od řádkových typů)
- hodnoty jejich typů mohou být umístěny do sloupců tabulek.
- možná i tečková notace jmeno_objektu.jmeno_atributu

```
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;
```

velké datové objekty
možnost uložení velkého datového objektu (obvykle obrazové informace, zvukové sekvence, textu či prostorového obrazce) o rozsahu do 4 GB. Objekty mohou být jednoho ze čtyř typů:

- BLOB - binární objekt
- CLOB - znakový objekt

- BFILE - binární data uložená mimo databázi a zpracovatelná jen pro čtení
- NCLOB - sloupec typu CLOB podporující vícebytovou množinu znaků.

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)

Reference

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

Vlastnosti objektově orientovaného datového modelu

- Všechno je objekt,
- začlenění do OO jazyka

	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

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).

Tvorba datového modelu

- **Určení tříd a metod: Metoda gramatické inspekce**: podstatná jména představují objekty nebo třídy, přídavná jména představují hodnoty atributů a slovesa představují většinou aktivity.
- Identifikují se třídy zobecňováním 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
- Implementují se metody (teprve po jednoznačném vymezení všech funkcí systému)

Objektový (konceptuální model) – UML diagram

Identita objektu: OID vs. primární klíč

Vazby – relace mezi třídami

- Vazba **asociace** (Association)
- Vazba **agregace** (Aggregation) – speciální případ asociace
- Vazba **generalizace** (Generalization) – dědičnost specifického elementu od obecného elementu
- Vazba **závislosti** (Dependency) – změna nezávislého elementu ovlivní závislý element

Chování objektů

- Objekt poskytuje služby prostřednictvím operací (metod), izolace funkční a datové vrstvy

Bariéry rozšíření

- 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

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

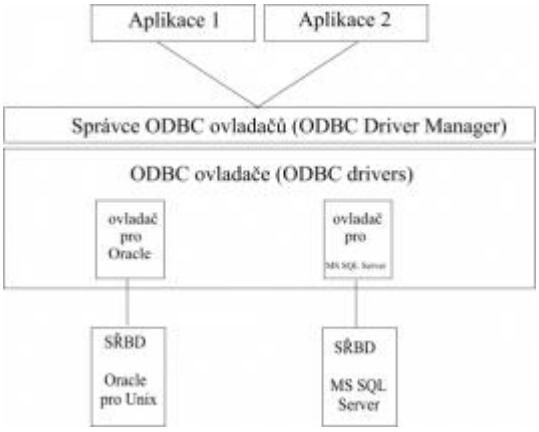
ODBC (Open Database Connectivity)

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.

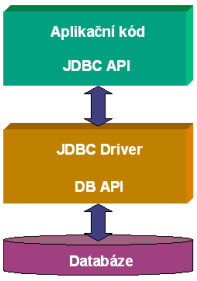
Navrženo Microsoftem, proto primárně přístupné pouze přes C/C++. Založeno na specifikaci X/Open a ISO: SQL Call Level Interface (SQL/CLI)

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). 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

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

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.

Distribuované databáze – koncepce distribuovaného databázového systému, replikace a fragmentace dat, distribuovaná správa transakcí

- 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

Charakteristické vlastnosti:

- transparentnost – 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)
- autonomnost – 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
- nezávislost 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

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) – jak kdy, někdy DDBS zlevňuje
- bezpečnost
- obtížný přechod (neexistence automatického konverzního prostředku z centralizovaných DB na DDB)

Fragmentace a replikace dat

- **Replikace:** Systém (jako celek) udržuje několik kopií dat uložených v různých uzlech distribuovaného systému kvůli rychlejším přístupům, odolnosti proti chybám a minimalizaci přenosů dat
- **Fragmentace:** Relace je rozdělena do několika fragmentů uložených na různých strojích
- **Replikaci a fragmentaci lze kombinovat:** Relace je rozdělena do několika fragmentů a systém tyto fragmenty replikuje

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

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 (podřazená tabulka půjde s fragmentem nadřazené)
- **Vertikální** - Výběr podle některých sloupců
- **Smíšená** - Zajišťuje, že fragmenty, které se používají často společně jsou uložené na stejném místě

Výhody fragmentace

- **Horizontální**: umožňuje paralelní zpracování na fragmentech relace, které jsou umístěny tam, kde se k nim nejčastěji přistupuje
  - např. údaje o lokálních účtech se používají na pobočce nejčastěji
- **Vertikální**: umožňuje umístit atributy tam, kde se nejčastěji potřebují
  - na přepážce se nejčastěji potřebuje zůstatek účtu
  - přidaný atribut tuple_id dovoluje snadné spojování vertikálních fragmentů a tím i paralelní zpracování

- **Vertikální a horizontální fragmentaci lze kombinovat:** Fragmenty mohou být nadále fragmentovány do libovolnéhloubky

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

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)

Distribuovaná správa transakcí

Transakce může přistupovat k datům v různých uzlech

- Každý uzel má svého **správce** transakcí odpovědného za
  - vedení žurnál pro účely zotavení
  - účast na koordinaci souběžných transakcí vyhodnocovaných v lokálním uzlu
- Každý uzel obsahuje **koordinátor** transakcí, který odpovídá za
  - Spouštění běhu transakcí, které vznikly v tomto uzlu
  - Distribuci sub-transakcí do jiných uzlů
  - Koordinaci ukončení transakcí vzniklých v tomto uzlu, což může mít za následek potvrzení (commit) či zrušení (abort) sub-transakcí v mnoha jiných uzlech

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

Porovnání klasických a temporálních DB

- **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

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, např: 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

- **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

- **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

- **bitemporální**
  - **„Jsem schopen zjistit co v kterémkoli uvažovaném čase platilo o kterémkoli uvažovaném čase.“**
  - Datový model podporující čas platnosti i transakční čas
  - append-only
  - Obvykle založeno na relačním datovém modelu nebo objektově orientovaném datovém modelu.

Temporální dotazovací jazyky

- Velké množství: Mnoho nekompatibilních datových modelů s mnoha dotazovacími jazyky
- Nejčastěji založeny na SQL
- Typy
  - Relační
    - př.: HQL, HSQL, TDM, TQuel, TSQL, TSQL2
  - Objektově orientované
    - př.: MATISSE, OSQL, OQL, TMQL

Uživatelské rozšíření databázových systémů – data cartridge, příklady použití.

rozšiřitelnost = možnost přidávání nových datových typů a programů (funkcí) zabalených do speciálního modulu

Oracle je rozšiřitelný DB systém – poskytuje podporu pro rozšíření vnitřní logiky a je pro podporu rozšiřování přímo navržen.

uživatelsky definované:

- typy = UDT
- funkce = UDF

- Každý výrobce má svůj název, Oracle má Data Cartridge

Data cartridge:

- uživatelské rozšíření databázového serveru Oracle
- pouze server-side
- integruje se pomocí sady rozhraní pro jednotlivé přístupy (k datům, indexům, …)
- 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

Obory rozšiřitelnosti Oracle DB

- 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 indexování pro doménově specifická data
- Extensible **Optimiser**:
  - podpora pro vytváření uživatelských funkcí a indexů pro vlastní sběr statistik pro **CBO**

Proč implementovat Data Cartridge:

- nutnost zpracování komplexních dat, která neodpovídají standardním relačním informacím
- nutnost snadné manipulace s takovými daty

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

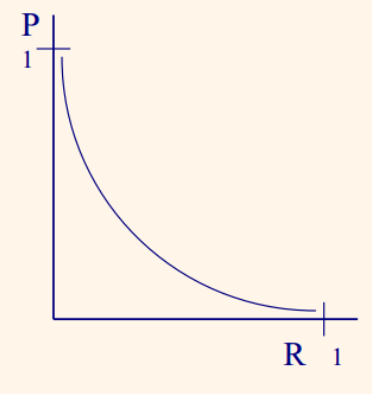
Dokumentografické systémy, fulltextové vyhledávání, filtrace, disambiguace, lemmatizace, indexy, tezaury, dotazování.

Nosná myšlenka: problém úplnosti (R - recall) a relevance (P - precision)

(když vrátím úplně všechny možné správné odpovědi na dotaz „auto jede do lesa“, jen málo z nich bude relevantních mému dotazu):

důsledek nejednoznačnosti: žádný existující DIS nedává ideální výsledky

ideální případ: P == R == 1



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, ...
  - Atd…

Jednotlivé kroky

Filtrace

odstraní formátovací značky a nechá čistý ASCII text

Disambiguace

disambiguace je to, co rozhodne, jestli slovo "stavení" je 1. sg nebo 2. sg nebo 1. pl apod. určí význam slova podle kontextu.

- "pět chválu" ... sloveso pět
- "pět vozidel" ... číslovka pět

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. Určí základní tvar slova a gramatický tvar v dokumentu, často nahrazen pomocí stemmeru, který hledá kmen slova.

Stop-list

Odstranění nevýznamových slov

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

Tezaurus

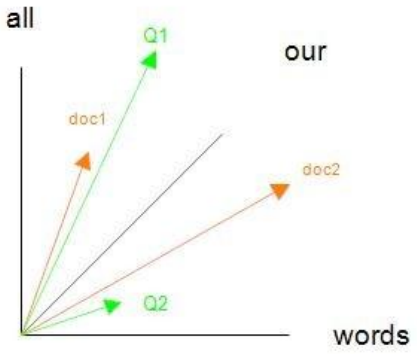
obsahuje

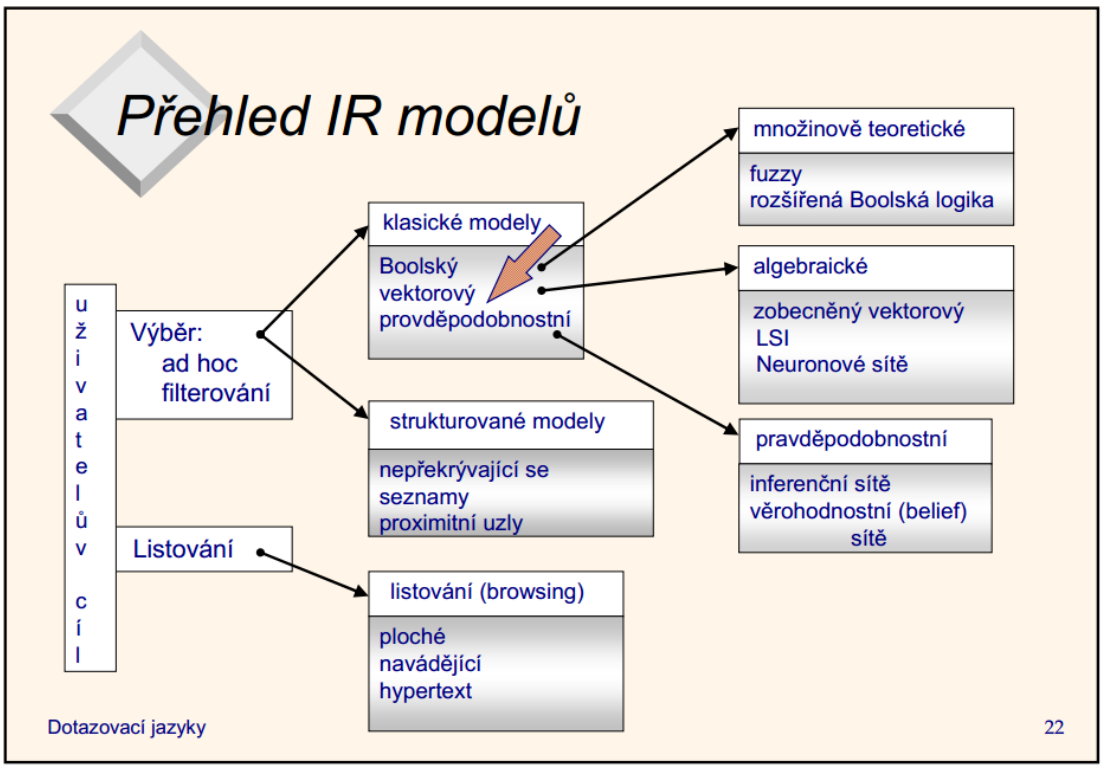
- hierarchie slov a jejich významů
- synonyma slov s preferovanými termy
- asociace mezi slovy
- Hierarchie užších/širších termů

- Důležitý pro booleovský model

Dotazování

- **Boolský model:**
  - formálně: pomocí Boolských výrazů
  - způsob: na přesnou shodu
  - Prakticky:
    - dotaz pouze na významová slova (odstranění nevýznamových)
    - Problém synonym - obecný jazyk, nelze podchytit tezaurem: Př.: nehoda, neštěstí srážka, karambol, „něco se tam stalo“,
    - Všechny D vyhovující dotazu jsou chápány jako stejně důležité, není možné je uspořádat podle hodnoty relevance.
    - formulace dotazů je spíše uměním než vědou
- **Vektorový model**
  - formálně: pomocí vektoru dotazu
  - dotazování na částečnou shodu pomocí funkce (koeficientu) podobnosti
  - Termy jsou reprezentovány vektory





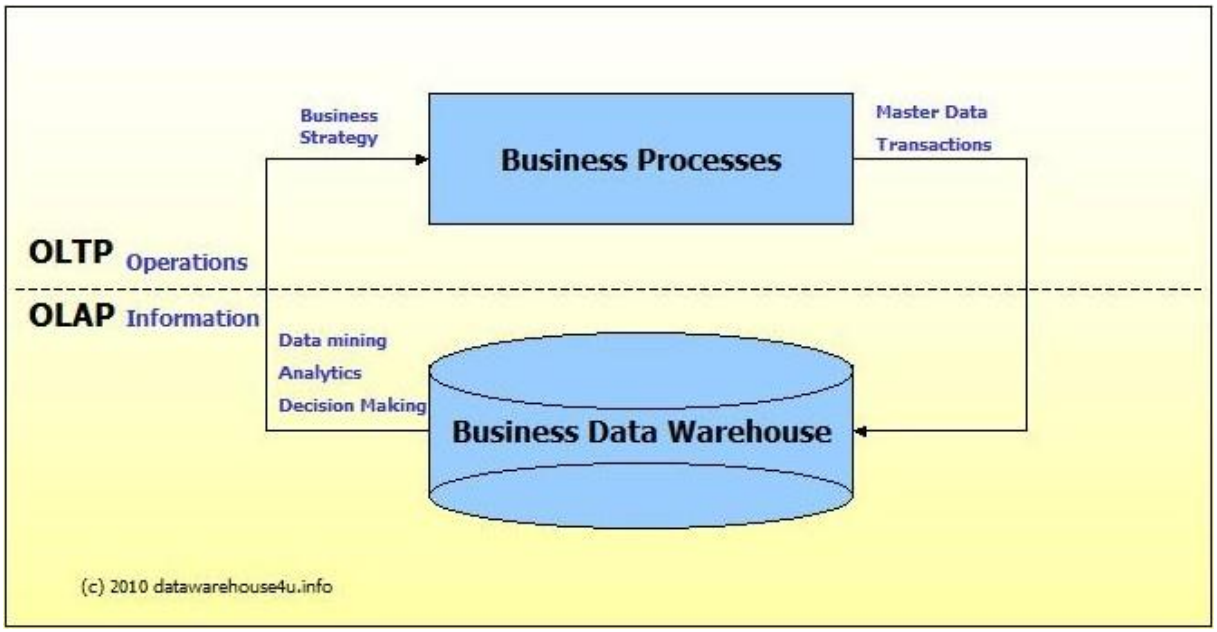
Možnosti tvorby datových skladů a metody dolování znalostí

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, Data Mart – DM

OLTP vs. OLAP



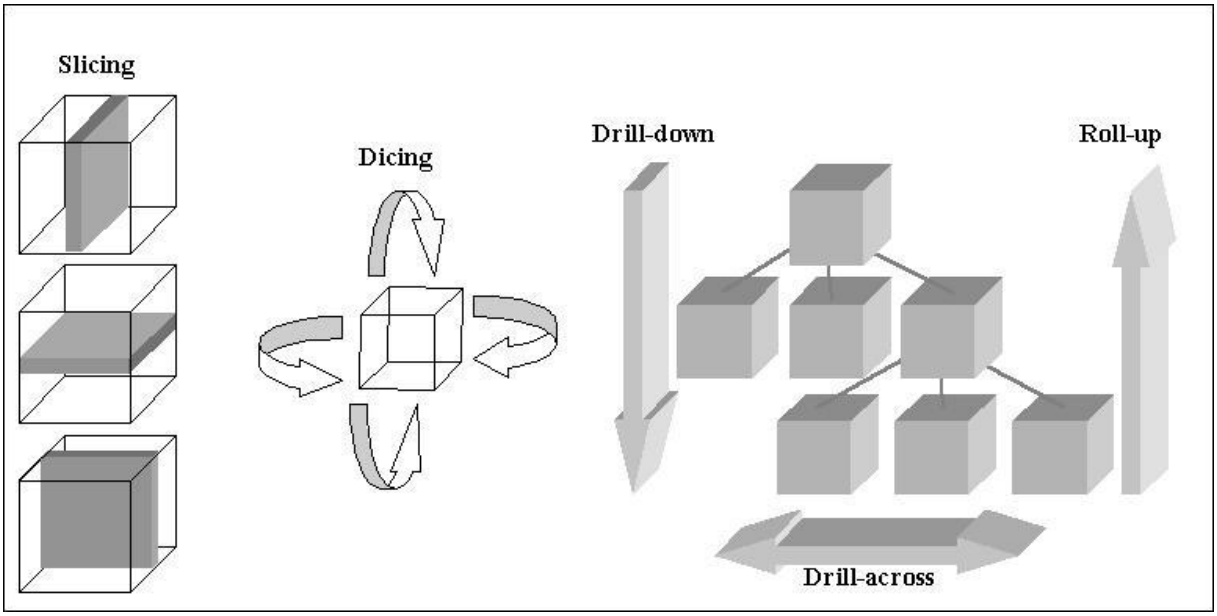
	OLTP	OLAP
Source of data	Operational data; OLTPs are the original source of the data.	Consolidation data; OLAP data comes from the various OLTP Databases
Purpose of data	To control and run fundamental business tasks	To help with planning, problem solving, and decision support
What the data	Reveals a snapshot of ongoing business processes	Multi-dimensional views of various kinds of business activities
Inserts and Updates	Short and fast inserts and updates initiated by end users	Periodic long-running batch jobs refresh the data
Queries	Relatively standardized and simple queries Returning relatively few records	Often complex queries involving aggregations
Processing Speed	Typically very fast	Depends on the amount of data involved; batch data refreshes and complex queries may take many hours; query speed can be improved by creating indexes
Space Requirements	Can be relatively small if historical data is archived	Larger due to the existence of aggregation structures and history data; requires more indexes than OLTP
Database Design	Highly normalized with many tables	Typically de-normalized with fewer tables; use of star and/or snowflake schemas
Backup and Recovery	Backup religiously; operational data is critical to run the business, data loss is likely to entail significant monetary loss and legal liability	Instead of regular backups, some environments may consider simply reloading the OLTP data as a recovery method

Datawarehouse

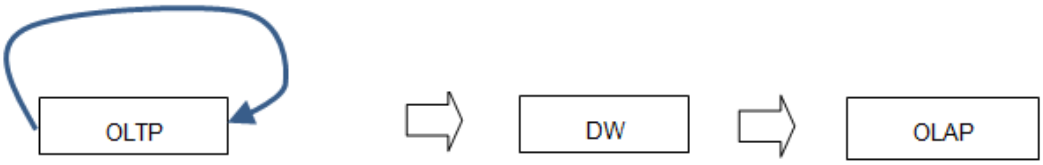
- samostatný informační systém postaven na již pořízených datech, určen především k jejich analýze
- architektura založená na relačním SŘBD, která se používá pro údržbu historických dat získaných z databází operativních dat, jenž byla sjednocena a zkontrolována před jejich použitím v databázi DW

- 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
- vkládání dat: **ETL, MDM**

Typické operace OLAP:



Příklad zaplnění DW:



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