SSZ 2016 SWI

Databázové technologie

DB

Část: databáze

Seznam otázek

1. „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 povelův 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
  ošetrení 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.

2. 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

```
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;
```

3. Uložené procedury a funkce, balíky (packages), 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 atabázi (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;

4. Aktivní databáze – klasifikace a spouštění triggerů, oblast použití.

Status: nutná revize, otázka zcela neodpovídá.

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

Trigger

- uživatelsky definovaný blok PL/SQL sdružený s určitou tabulkou
- je implicitně spuštěn, jestliže je nad tabulkou prováděn aktualizační příkaz

Typy triggerů a podoba

- BEFORE / AFTER – **kdy se spustí**
- INSERT / UPDATE / DELETE– **na jaký dotaz**
- omezení triggeru (nepovinná klauzule **WHEN**)
- akce triggeru – **blok PL/SQL**
- může být buďto příkazový a nebo řádkový – klíčové slovo **FOR EACH ROW**
- řádkový se spustí pro každý aktualizovaný řádek tabulky
- hodnoty v triggeru – **:OLD** a **:NEW**
    - logicky OLD u insert null a NEW u delete null

Vytvoření:

```
CREATE [OR REPLACE] TRIGGER jmeno
  typ_triggeru_kdy spousteci_akce [OF sloupec, sloupec] ON tabulka
  [FOR EACH ROW] [WHEN podmínka]
BEGIN
  ...
END;
```

Použití

- zajištění referenční integrity – vyhodíme výjimku
- aktualizace údajů (při změně data narození chceme automaticky aktualizovat věk, atp.)
- logování (chceme vědět, že uživatel provedl aktualizaci nějaké tabulky, atp.)
- ohlídání business rules

Postup spouštění triggeru

- do ORACLE DB předán příkaz INSERT/UPDATE/DELETE
- provede se příkazový trigger BEFORE
- pro každý řádek, kterého se SQL týká se provede
    - řádkový trigger BEFORE
    - změní se řádek a provedou se kontroly integritního omezení
    - řádkový trigger AFTER
- dokončí se odložené kontroly IO s ohledem na přechodná porušení
- provede se příkazový trigger AFTER
- návrat do aplikace

aktivace a deaktivace

- po vytvoření defaultně aktivní
- **jednoho** – ALTER TRIGGER trigrname ENABLE | DISABLE
- **všech pro tabulku** – ALTER TABLE tablename ENABLE | DISABLE ALL TRIGGERS

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

5. 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

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 (relační)

- 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 (objektově-relační)

- 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 (objektově orientovaná)

- 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.

Příklady postrealčních DBS:

prostorové DB

- rozšířeny o práci s prostorovými objekty a vztahy mezi nimi
- složité na návrh a školení uživatelů, drahá implementace
- jsou schopné zpracovávat prostorová data v podobě jednoduchých geometrických entit (obrovského množství)
- objektově orientované DB – rozšířeny o objektový model dat a vazby
- nedostatek standardů, nedostatek zkušeností, chybí univerzální datový model

deduktivní DB

- rozšířeny o funkce pro analýzu dat
- poskytují matematickou logiku (vztahy) spolu s relačními databázemi (znalosti) a využívají znalostí a vztahů k dedukci závěrů

temporální DB

- rozšířeny o temporální (časovou) logiku
- temp. db jsou databáze rozšířené o časovou dimenzi
- používají se v bankovnictví, účetnictví a podobně
- mají dobré vlastnosti pro archivaci a monitorování změn

multimediální DB

- rozšířeny o fce pro práci s multimediálním obsahem (např. Oracle interMedia)
- vyžaduje zkušené uživatelé, high end hardware, zbytečný overhead

aktivní DB

- rozšířeny o aktivní pravidla

6. Objektové vlastnosti jazyka SQL99, rozšíření datových typů.

Status: nutná revize otázka zcela neodpovídá + rozšíření datových typů je nové.

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 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

Rozšíření datových typů

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 … body of method … 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

7. Vlastnosti objektově orientovaného datového modelu, možnosti použití.

Status: možnosti použití jsou nové.

- 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

8. 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
  - 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

9. 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 ystému
- 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

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

Výhody:

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

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

10. Transakce, dvoufázový uzamykací protokol, detekce uváznutí.

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

- **A - Atomicity = atomičnost**
  - 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
- Transakce vyvolá write W(x)
  - TSW(x) > TS(t): čtení z „později zapsané“ paměti ⇒ ROLLBACK
  - jinak read

11. Optimalizace dotazu, jednotlivé přístupy (např. 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 (Rule Based Optimalisation)

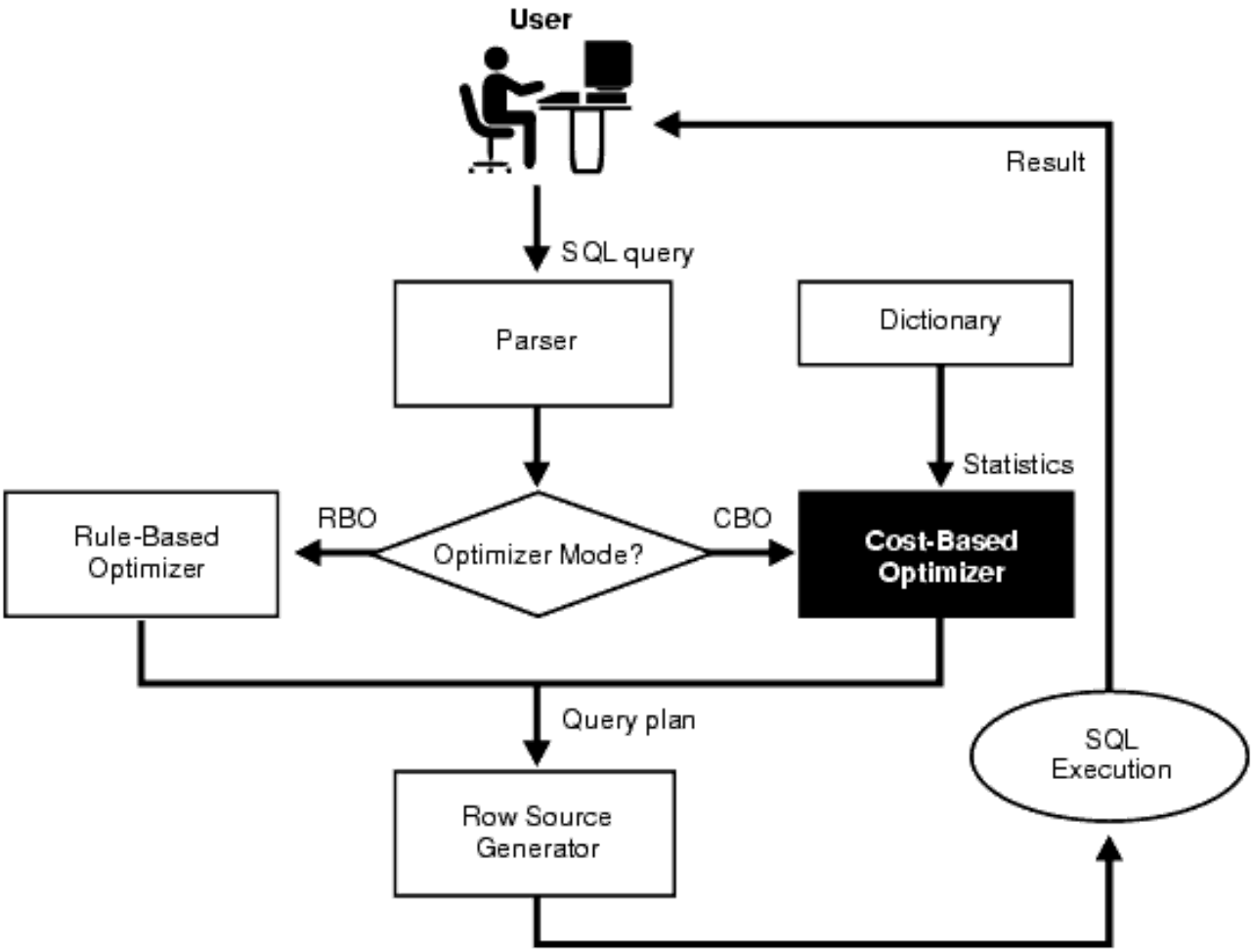
- Starší přístup, dnes často deprecated (Oracle),
- řídí se předem sestavenou sadou pravidel, která nezohledňují např. velikost tabulky:
  - **Malá tabulka** (obsahuje 5 řádků a vejde se do jednoho datového bloku), je rychlejší jí přečíst celou než hledat podle indexu (1 IO operace vs. čtení bloku indexů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

CBO (Cost Based Optimalisation)

- 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

12. Webové a nativní databázové aplikace.

Status: Zcela nová otázka.

Na nejnižší úrovni lze pro komunikaci se systémem pro řízení báze dat (SŘBD) používat jeho nativnírozhraní.Je specifické pro každou databázovou platformu a připojuje se vpodobě dynamických knihoven. Využívá se výjimečně vnízkoúrovňových systémech extrémně náročných na rychlost.

Obecná rozhraní

- jsou nezávislá na použité databázové platformě
- poskytují omezenou množinu služeb
- umožňují relativně jednoduchou změnu databázové platformy i současné využití více typů SŘBD vrámci jedné aplikace
- *příkladem mohou být ODBC (JDBC) ovladače*

Mezi obecná rozhraní patří například ODBC a vyznačují se tím, že dokáží spolupracovat s více databázovými platformami, pro které poskytují určitou společnou (bohužel jen základní) množinu služeb. Tato rozhraní tak usnadňují tvorbu aplikací, které mají být přenositelné mezi více databázovými platformami. Vzhledem k tomu, že se jedná o obecná rozhraní, jsou tato podporována celou řadou dalších aplikací, pro které není využívání externích databází primárním účelem – u již zmiňovaného ODBC se tak jedná například o Excel.

Obecná rozhraní ale mají celou řadu mnohdy jen obtížně zvládnutelných nevýhod – některá existují pouze ve variantách pro konkrétní operační systém (na straně klienta), mnohdy tato rozhraní mají výkonnostní problémy. Uvedené problémy přetrvávají, i když mnohdy jsou tyto obecné ovladače částečně přizpůsobeny konkrétnímu databázovému serveru.

Přímá (nativní) rozhraní

- jsou závislá na konkrétní databázové platformě (někdy i verzi) a mnohdy také klientské straně (vývojovému nástroji)
- výhodou je možnost využití všech dostupných funkcí databázového stroje a jeho výkonu
- *příkladem může být OCI (Oracle Call Interface) ovladač*

Nabízejí více možností než rozhraní obecná a jsou zcela přizpůsobena konkrétní databázové platformě a mnohdy také klientské straně (zpravidla vývojovému nástroji). Hlavní výhody nativního přístupu je tedy především možnost maximálního využití vlastností databázového serveru a zpravidla také mnohem vyšší výkon. Nevýhodou je neuniverzálnost, ze které plyne především velmi obtížná tvorba klientů, kteří by měli spolupracovat s více databázovými servery.

Zpravidla ale platí – tvoříte-li aplikace pracující s konkrétní databázovou platformou a nebrání vám v použití nativního přístupu jiné okolnosti, pak jej využijte.

Perzistentní spojení

Na rozdíl od desktopových aplikací, kde lze udržovat platnost objektu po celou dobu běhu programu, u webových aplikací ztrácí objekt(např. instance databázového spojení)platnost po dokončení požadavku (načtení stránky).

Ztoho vyplývá, že i vpřípadě dynamických stránek (využívajících AJAX) musí při každém požadavku vznikat nová instance databázového spojení a tedy i novárelace vdatabázi.

Tento problém řeší perzistentní spojení, které vpřípadě opakované žádosti (stejná adresa, uživatelské jméno a heslo) o připojenípřidělí již otevřenou volnou instanci. Vše probíhá na pozadí (v režii serveru).

13. „Vnější“ programování (přes rozhraní/knihovny). Porovnání přístupu JDBC a OCI.

Status: nutná revize, otázka zcela neodpovídá.

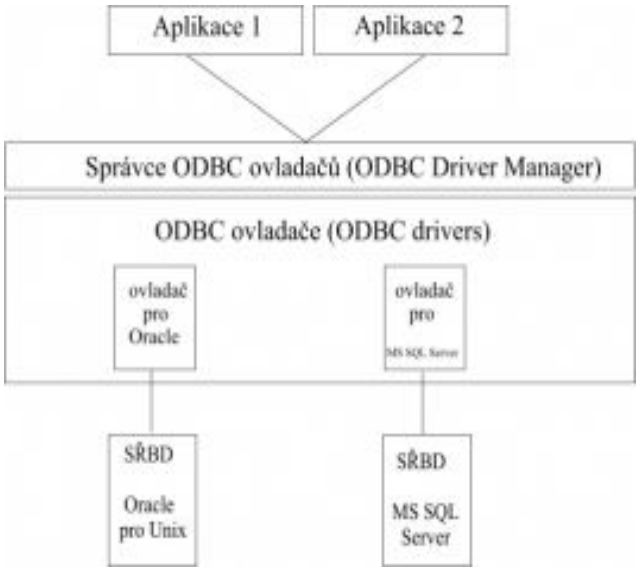
ODBC (Open Database Connectivity)

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)

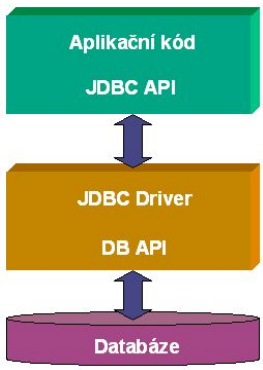
Model struktury ODBC se dá znázornit pomocí čtyř vrstev:

- 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
  - *sem patří OCI*
- 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
  - *Oracle JDBC Thin driver*

ODBC (Open Database Connectivity)

Představuje aplikační rozhraní pro přístupkdatůmvrelační databázi nezávislena databázové platformě (Oracle, Microsoft SQL Server, Microsoft Access, DB2, Sybase, MySQL, PostgreSQL atd.) i operačním systému(Windows, Linux).

```
$database_resource = odbc_connect($dsn, $user, $password);
```

- **$dsn** → adresa databázového zdroje, např. "localhost"
- **$user** → uživatelské jméno (název schématu)
- **$password** → uživatelské heslo

ODBC Driver Manager (Object Linking and Embedding Databases)

- Správce ovladačů (ODBC DM) tvoří mezivrstvu mezi aplikací a ODBC ovladačem.
- Zajišťuje načítání potřebných knihoven ODBC ovladače.
- Umožňuje současnou inicializaci více ovladačů a tedy i jejich přepínání vrámci jedné aplikace.
  - Je tedy možné přistupovat současně kvícezdrojům dat.

OCI (Oracle Call Interface)

- Rozhraní OCI určené pro platformu Oracle nabízí širokou paletu funkcí pro komunikaci sdatabázovým serverem a zároveň poskytuje vyšší výkon, než např. ODBC rozhraní.
- Rozhraní OCI obsahuje (kromě základních funkcí pro práci s db) podporu pro volání PL/SQL, transakce,načítání a ukládáníobjektu typuLOB/CLOB, funkce pro správu.

```
$database_resource = oci_connect($username , $password [,$connection_string
                                    [,$character_set]] )
```

- **$username** → uživatelské jméno (název schématu)
- **$password** → uživatelské heslo
- **$connection_string** → adresa databázového zdroje, např. "localhost:1528"
- $**character_set** → znaková sada, např."UTF8"

14. SQL/MM - základní rámec normy, full-textová data, prostorová data, obrázky (statické i videa).

Status: Zcela nová otázka.

Rámec normy

- **full-textová data**
- **prostorová data**
- **obrázky (statické i videa)**

Framework

- jednotlivé části SQL/MM jsou na sobě poměrně nezávislé
- část společná a závazná zbytku světa
- **poskytuje**
  - definici společného konceptu užitého v dalších částech SQL/MM
  - hlavní rysy přístupu k definici těchto částí

Fulltext

- **def**: textová data, lišící se od běžných znakových řetězců
  - obvykle delší záznamy
  - specifické operace
  - způsob indexování
    - index slov v dokumentu
    - index vzájemných vzdáleností slov a frází
- **konstruktor** – řetězce znaků [+ zadání jazyka]
- **konverze** do běžných SQL znak. řetězů – FullText_to_Character
- **vyhledávací metody** – ano/ne CONTAINS, rank
- **vyhledávání**
  - vzorek + wild cards
  - odvozená slova(STEMMED)
  - slova s podobným nebo odvozeným významem(SYNONYMS)
  - stejně znějící slova(SOUNDS, LIKE)
  - dle pozice v textu (NEAR)
  - dle konceptu textu
- **podpora** jazyků
  - počítá se především s podporou jazyků, kde je snadné výpočetně rozeznat jednolité tokeny
- příklad: `CREATE TABLE informace ( číslo_doc INTEGER, dokument FULLTEXT);`
- dotaz:
  - `WHERE dokument.CONTAINS( 'STEMMED FROM OF "standard" IN SAME PAR. AS SOUNDS LIKE "sequel") = 1`

Spatial (SQL/MM Spatial)

- **definuje**
  - ukládání, výběr, dotazování a aktualizaci jednoduchých prostorových objektů
  - reprezentaci prostorových objektů pomocí **prostorových datových typů**
  - **funkce** pro práci s prostorovými objekty – 0,1,2 dimenzionální
- **rozdělen** do několika klauzulí
  - prostorové datové typy a jejich metody
  - datový katalog

- **datové typy** ⇒ funguje dědičnost
  - **body** – pouliční lampa, lavička
  - **křivky** – koryto řekni, hranice pobřeží
  - **polygony** – půdorysy budov, území států
- vychází z geometrického modelu OGC
  - rozdíly – místo `Line` a `LinearRing` zavádí nový typ `ST_LineString`
  - nové typy pro repr. křivek a povrchů tvořených kruhovými oblouky – `ST_CircularString`
  - u kolekcí není zřejmé z jakých se skládá jednoduchý typů – `ST_MultiPoint je z ST_Point`??
- prostorový objekt – svázán s nějakým prostorovým systémem souřadnic – **Spatial Reference Systém** – některé resp. i různé zakřivení země
  - **`ST_SpatialRefSys`** – typ který určí systém souřadnic
  - všechny hodnoty sloupců musí být ve stejným systému souřadném
- pro reprezentaci prázdného prostorového objektu neexistuje samostatný typ – instance každého prostorového typu může být prázdný prost. Objekt
  - pokud návratový typ není spec., pak návratovou hodnou bod
- **operace**
  - průnik, sjednocení, sousedi
  - relace – objekt A obsahuje B?
  - vzdálenost
- **4 kategorie metod**
  - konverze do/z externích datových formátů
    - **WKT** – well known text (point(10,10), multipolygon(1 1 , 2 2, 1 2, 11)
    - **WKB** – well known binary
    - **GML** – geography markup lang – xml zápis
  - práce s atributy prostorových objektů
    - datové typy mají společné atributy (např. Dimenze)
    - podtyp si přidávají některé specifické atributy
    - `ST_X` – souřadnice X, `ST_Length` – délka křivky, `ST_Boundary` – hranice
  - porovnávání prost. Objektů
    - `ST_Equals` – prost. rovnost, `ST_Touches` – dotýkání, `ST_Within` – obsah
  - generování nových prost. objektů – výsledkem operací
    - `ST_Intersection`, `ST_union`, `ST_difference`
    - algoritmu na jeden objekt- `ST_ConvexHull`

Still Image

- **obrázky** = hodnotná data
- ukládání obrázků
- úprava obrázků
- vyhledávání obrázků – dle vizuálních vlastností
- datové typy
  - `SI_StillImage` – obrazová data
  - `SI_Feature` – vlastnosti obrázku
    - užitečné při vyhledávání, má podtypy
      - `SI_AverageColor`, `SI_ColorHistogram`, `SI_PositionalColor`, `SI_Texture`
    - metoda `SI_Score` – vrací podobnost obrázku <0, 1>
  - `SI_FeatureList` – seznam pro vlastnosti obrázku (i všechny třeba)
  - konstruktory
  - BLOB [+FORMAT] – JPEG, TIFF, GIF

- metody
  - pro přeformátování – SI_ChangeFormat
  - vytvoření miniatury,
  - změna velikosti,
  - ořezání **rotace**

15. Zpracování full-textových dat.

Status: Zcela nová otázka. Dobré pojednání mimo jiné nabízí wikipedia (https://cs.wikipedia.org/wiki/Fulltextové_vyhledávání)

- patří do rámce (frameworku) SQL/MM

Framework

- jednotlivé části SQL/MM jsou na sobě poměrně nezávislé
- část společná a závazná zbytku světa
- poskytuje
  - definici společného konceptu užitého v dalších částech SQL/MM
  - hlavní rysy přístupu k definici těchto částí

Fulltext

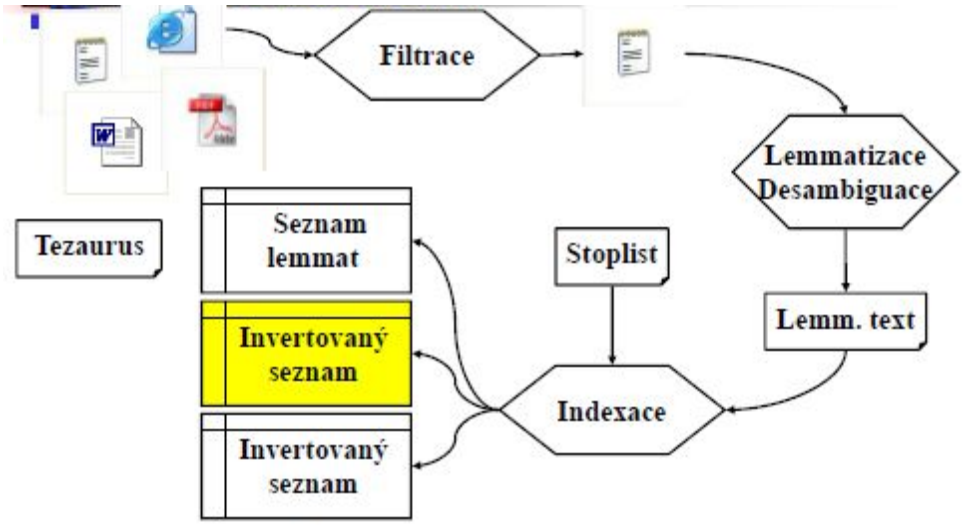
- je tedy souhrn algoritmů, které dokáží z daného dokumentu vytvořit podrobnou statistiku výskytu jednotlivých pojmů a tu zanést do databáze (říkáme, že dokument naindexují)
- definice: textová data, lišící se od běžných znakových řetězců
  - obvykle delší záznamy
  - specifické operace
  - způsob **indexování**
    - index slov v dokumentu
    - index vzájemných vzdáleností slov a frází
- 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 rozdíln
    - různá slova stejného významu, různé slovní obraty a opisy
- DB systémy využívají svých prostředkůrozšiřitelnosti a dodávají standardněprostředky, 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
- **konstruktor** – řetězce znaků [+ zadání jazyka]
- **konverze** do běžných SQL znak. řetězů – FullText_to_Character
- **vyhledávací metody** – ano/ne CONTAINS, rank
- **vyhledávání**
  - vzorek + wild cards
  - odvozená slova(STEMMED)
  - slova s podobným nebo odvozeným významem(SYNONYMS)
  - stejně znějící slova(SOUNDS, LIKE)
  - dle pozice v textu(NEAR)
  - dle konceptu textu
- **podpora jazyků**
  - počítá se především s podporou jazyků, kde je snadné výpočetně rozeznat jednolité tokeny

příklad:

```
CREATE TABLE informace (číslo_doc INTEGER, dokument FULLTEXT);
... WHERE dokument.CONTAINS ('STEMMED FROM OF "standard" IN SAME PAR. AS SOUNDS LIKE "sequel") = 1
```

Zpracování full-textových dat

- je třeba mít možnost třídit odpovědi podle předpokládané vhodnosti pro tazatele
  - je nutné mít možnost vyjádřit míru shody
- hledají se homonyma, synonyma, slova stejně znějící, slova s nějakou vazbou na hledané slovo – hierarchie (auto = BMW) či ohnutá slova (jdu = jdou)
- **obvyklý postup předzpracování**
  - DB obvykle používají nějaký z boolovských modelů reprezentace modelů
    - snadná implementace
    - nejlépe odpovídá běžným dotazů
    - dotazy ve formě boolovských formulí, kde operandy tvoří jednotlivá slova



  - **postup**
    - filtrace – odstraní se formátovací značky a nechá se čistý ASCII text
    - desambiguace – určí význam slova podle kontextu
      - pět chválu – sloveso pět
      - pět vozidel – číslovka
    - lemmatizace – určí se základní tvar slova a gramatický tvar v dokumentu
      - nahrazen často pomocí stemmeru => hledá kmen slova
    - indexace
      - vytvoří se pomocné seznamy lemmat a dokumentů a invertovaný soubor
      - dvojice[id_dok, id_lemmatu] setříděné dle id lemmatu a zbavené duplicit
      - dnes vice info – ještě č. odstavce, věty, slova…
- fulltext vyhledávání
  - třeba vytvořit nad textovým sloupcem index – invertovaný soubor
  - běžné textové sloupce pro tyto účely krátká a nevyhovující
    - indexují se obvykle sloupce některého z LOB typů
      - BLOB, CLOB, NCLOB (native) – až 4gb
    - ve sloupcích pouze deskriptor – LOB lokátor, odkazuje na samostatně uložená data

Problémy

- Homonyma
  - 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"?
- Synonyma
  - Zvíře -Savec - Šelma - Medvěd
  - Tiskovina - Časopis
- Ohebnost slov
  - Jít, Jde, Jdu, Jdou, ...

Fulltextové vyhledávání

Zdroj: DP, Fulltextové vyhledávání v rozsáhlém informačním systému, Bc. Miroslav Prachař, Brno, 2008

- Fulltextové vyhledávání nebo jen krátce fulltext bychom mohli denovat jako metodu vyhledávání uvnitř zpravidla rozsáhlých textových souborù nebo v sadě takových souborù. Hybnou silou vzniku a vývoje fulltextu byl nárùst objemu dat publikovaných prostřednictvím sítě internet, kde vznikla potřeba efektivně hledat nebo lokalizovat webové stránky. V první fázi toto řešily tzv. katalogy, kam uživatelé ukládali informace o svých stránkách. Jelikož ale tyto katalogy přestaly požadavk ùm uživatelù dostačovat, vznikly fulltexty. Softwarové stroje procházely jednotlivé HTML stránky, stahovaly je a transformovaly je do vhodné podoby tak, aby bylo umožněno efektivní vyhledávání v obsazích stránek.
- V dnešní době se fulltext uplatňuje všude tam, kde se vyskytují rozsáhlejší textové dokumenty. Některé údaje o dokumentu (např. název, autor nebo třeba klíčová slova) jsou obvykle uvedeny jednak uvnitř samotného dokumentu, ale také i vně dokumentu jako pomocné informace o dokumentu sloužící k jeho vyhledání. Nazývejme tyto pomocné informace jako metadata. Fulltext umožňuje vyhledávat dokumenty nejenom podle jejich metadat, což v mnoha případech nedostačuje, ale prochází samotný obsah dokumentu a porovnává jej se zadaným výrazem.
- Textové soubory v nějaké dokumentové sadě obsahují texty týkající se nejrůznějších témat a cílem fulltextu je zprostředkovat nalezení takového textu nebo dokumentu, jehož obsah co nejvíce koresponduje se zadaným vyhledávacím dotazem, tedy hledaným slovem nebo slovním spojením.

Technologie fulltextu od Seznam.cz

http://vyhledavani.sblog.cz/technologie-fulltextu/

Přehled

Následující text uvádí stručný popis procesů, které proběhnou potom, jak zadáte dotaz do vyhledávacího pole.

Fráze je nejprve zpracována webovým rozhraním a rozšiřovačem dotazu (Query Processor) – to znamená, že se provede kontrola pravopisu, případně se doplní diakritika atd. Výsledná fráze (dotaz) je zaslána do agregátorů hledání (Search Aggregator). Server indexu vyhledá dokumenty v indexu a vrátí nejrelevantnější výsledky agregátoru hledání. Ten vybere 10 nejvíc relevantních výsledků a zašle požadavek Snippet Serveru, který ke každému z nich přidá titulek webové stránky a úryvek. Tyto výsledky jsou pak vráceny jako výsledek na váš dotaz. Všechny tyto procesy se dějí online.

Data jsou stahována z Internetu robotem (web crawler) a poté jsou uložena do databáze dokumentů. Z té jsou pak dokumenty postupně vybírány na indexaci a uložení v Indexu. Obnovování databáze dokumentů probíhá pravidelně (týdenně, denně a každých několik minut).

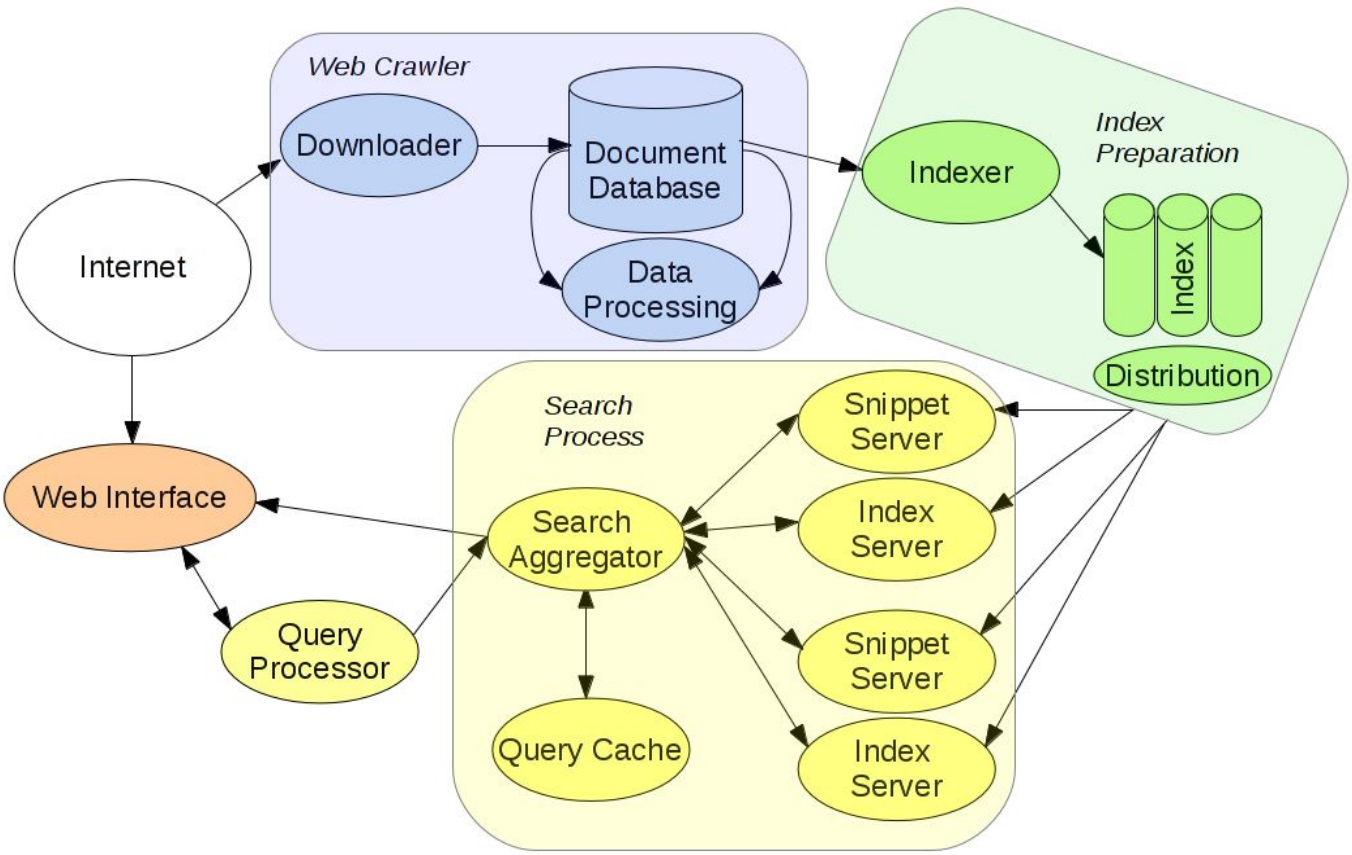
Query Processor

Query Processor zpracovává frázi, kterou jste zadali. Proběhne kontrola pravopisu, jsou rozvinuty zkratky a doplní se případná synonyma a diakritika. Vznikne tak rozšířený dotaz, který je poslán Search Aggregatoru. Tato implementace umožňuje získat širší rozsah relevantních výsledků a také vrátit relevantní výsledky i v případě, že fráze byla zadána nesprávně.

Query Processor používá sadu procesů aby upravil zadanou frázi nasledovným způsobem:

- hledá dvojslovná spojení
- určí tematiku fráze
- vykoná rozšíření zkratek nebo vytvoří zkratky
- rozdělí nebo vytvoří složená slova
- zjistí, jestli člověk zadal přesnou frázi
- vytvoří související slova
- vytvoří alternativní zápis čísel nebo zkratek
- identifikuje slova v hledané frázi, která můžou být vynechána z hledacího procesu
- hledá čárku a rozdělí hledanou frázi
- hledá strukturu doménového jména v hledané frázi
- převede čísla na jejich tokenovou reprezentaci anebo na text
- převede ligatury na běžné znaky
- spojí tokeny oddělené znakem + nebo &

Následující diagram ukazuje tok dat:



Search Aggregator

Search Aggregator dostane dotaz z Query Processoru a pošle ho Query Cache pro ověření, jestli se podobný dotaz nezpracoval již dříve.

Query Cache ukládá poslední relevantní výsledek připravený na zobrazení uživateli. Tyto uložené výsledky obsahují také úryvky (snippety).

Pokud Query Cache neobsahuje vhodné výsledky, dotaz je zaslán Index Serveru. Po zpracování dotazu Index Serverem je 10 nejrelevantnějších výsledků vrácených Search Aggregatoru.

Agregátory hledání mají stromovou strukturu. Každý agregátor dostane výsledky z několika serverů indexu a vybere 10 nejlepších výsledků. Jiný agregátor hledání je připojen k těmto agregátorům a provede výběr 10 nejvhodnějších výsledků z agregátorů hledání – viz obrázek nahoře.

Poslední agregátor hledání ve stromu pošle požadavek Snippet Serveru pro úryvky. 10 výsledků je potom přímo posláno webovému rozhraní.

Index Server

Index Server vykonává hledání v databázích indexu. V provozu je vždy vícero (přibližně 200) serverů indexu. Každý server indexu vykonává hledání v odlišné indexové databázi. Taková implementace vede k odlišným výsledkům z odlišných serverů indexu.

Jestliže hledaná fráze obsahuje více než jedno slovo, výsledek je průnikem. Server indexu určí, jestli je výsledek relevantní na hledanou frázi, např. pokud hledaná fráze je „bílá kočka“, nejenom nalezené dokumenty musí obsahovat obě slova „bílá“ a „kočka“, ale navíc také musí tato slova být v dokumentu blízko u sebe.

Snippet Server

Snippet Server přidá všechny informace o dokumentu – url, titulek stránky, obsah, odkazy atd. k 10 výsledkům. Také vytvoří úryvky(snippety) – dynamické popisky a texty které obsahují zvýrazněné části textu odpovídající dotazu.

I v případě, že je počet snippet serverů a serverů indexu stejný, snippet servery se použijí pouze na tom serverovém stroji, kde servery indexu najdou relevantní výsledky. Tato implementace snižuje síťový provoz na serverových strojích, protože snippet server se používá zřídka (snippet server v porovnání se servery indexu pracuje s velkým objemem dat).

Index

Clusterová databáze indexu ukládá indexované dokumenty. Současná implementace obsahuje následující části:

- **word barrel** – ukládá seznam dokumentů pro každé známé slovo a také ukládá počet výskytů (hitů) daného slova v dokumentech
- **document barrel** – ukládá seznam všech dokumentů ve svazku
- title barrel – ukládá obsah zpracovaných webových stránek a metadata
- **query site barrel** – ukládá informaci o tom, kolik unikátních dotazů zpracovaných z jednoho webu obsahovalo specifické slovo
- **site barrel** – ukládá seznam dokumentů pro weby
- **link barrel** – ukládá hypertextové odkazy ukazující na webové stránky
- **qds barrel** – ukládá hodnoty signálů pro rank zpětné vazby pro specifický dotaz
- **queryurl barrel** – ukládá informaci o tom, které unikátní dotazy vedou na stejnou URL

Indexer-worker

Hlavním účelem indexer-workeru je připravit dokumenty z databáze dokumentů na uložení v indexu.
Indexer-worker vykonává na dokumentech následující operace:

- zjistí, jestli dokument je ve formátu HTML → určí jazyk dokumentu → zkontroluje, jestli stránka není spam

Indexer-worker rozdělí dokument na jednotlivá slova a vykoná inverzní indexaci – každé slovo dostane seznam dokumentů, kde se nachází.

16. Prostorové databáze, modelování prostorových dat.

Status: Zcela nová otázka.

Příkladem systém Oracle Spatial

SQL/MM Spatial

Definuje

- ukládání, výběr, dotazování a aktualizaci jednoduchých prostorových objektů
- reprezentaci prostorových objektů pomocí **prostorových datových typů**
- funkce pro práci s prostorovými objekty – 0,1,2 dimenzionální
- **rozdělen** do několika klauzulí
    - prostorové datové typy a jejich metody
    - datový katalog

Datové typy

- **body**: pouliční lampa, lavička
- **křivky**: koryto řekni, hranice pobřeží
- **polygony**: půdorysy budov, území států
- funguje **dědičnost** – definovaná určitá hierarchie prostorových datových typů
- **0 dimenzionální objekty**
    - ST_Point – X a Y souřadnice vzhledem k nějakému SS (souřadný systém)
    - ST_MultiPoint – kolekce bodů
- **1 dimenzionální objekty** – křivky
    - ST_LinearString
    - ST_CircularString
    - ST_CompoundCurce – kombinace 2 předchozích – složeniny
    - ST_MultiLineString
- **2 dimenzionální objekty** – povrchy
    - ST_CurvePolygon, ST_MultiPolygon
    - **hranice** –reprezentovaná uzavřenou křivkou (více, když jsou díry)

- vychází z **geometrického modelu OGC** (Open Geospatial Consortium)
    - **rozdíly** – místo Line a LinearRing zavádí nový typ **ST_LineString**
    - **nové** typy pro repr. křivek a povrchů tvořených kruhovými oblouky – ST_CircularString
    - u kolekcí **není zřejmé**, z jakých se skládá jednoduchý typů
        - ST_MultiPOint je z ST_Point??
- prostorový objekt – svázán s nějakým prostorovým systémem souřadnic – **Spatial Reference System** – některé resp. i různé zakřivení země
    - ST_SpatialRefSys – typ který určí systém souřadnic
    - všechny hodnoty sloupců musí být ve stejným systému souřadném
- pro reprezentaci prázdného prostorového objektu neexistuje samostatný typ – instance každého prostorového typu může být prázdný prost. objekt
    - pokud návratový typ není spec., pak návratovou hodnou bod

Operace

- průnik, sjednocení, sousedi
- relace – objekt A obsahuje B?
- vzdálenost

4 kategorie metod

- **konverze** do/z externích **datových formátů**
  - **WKT** – well known text (point(10 10), multipolygon((1 1, 2 2, 1 2, 1 1), (2 3, 1 3, 2 4, 2 2))
    - textový značkovací jazyk pro reprezentaci vektorových objektů na mapě
  - **WKB** – well known binary (pro ukládání do DB)
  - **GML** – geography markup lang – xml zápis (od OGC)
- **práce s atribut**y prostorových objektů
  - datové typy mají společné atributy (např. dimenze)
  - podtyp si přidávají některé specifické atributy
  - ST_X – souřadnice X, ST_Length – délka křivky, ST_Boundary - hranice
- **porovnávání prost. objektů**
  - ST_Equals – prost. rovnost, ST_Touches – dotýkání, ST_Within – obsah
- generování nových prost. objektů – výsledkem operací
  - ST_Intersection, ST_union, ST_difference
  - algoritmu na jeden objekt – ST_ConvexHull

- vlastní typ – složením z již existujících
- nedostatky geometrického modelu
  - pokud chceme například volat něco nad `ST_Point` a `ST_Multipoint`, můžeme použít pouze nějaké obecné funkce, protože se každý typ nachází úplně v jiné větvi
- views
  - `ST_Geometry_Columns` – záznamy o všech sloupcích, kt. jsou deklarované jako prostorový datový typ
    - ke každému sloupci může být přidružen SS
    - záznam obsahuje – katalog, schéma, tabulku, název sloupce a také SS
  - `ST_Units_Of_Measure` – info o matematických jednotkách
    - jednotka má – název, typ, konverzní faktor (vůči základní jednotce daného typu)

17. XML databáze – charakteristické vlastnosti, výhody a nevýhody.

Status: Zcela nová otázka.

Vlastnosti

- oproti mnoha malých XML dokumentů několik XML databází
- obvykle dynamické
- explicitní struktura – schéma
- záznamy
- přizpůsobený stroj
- obsah – schéma, data, metody
- paradigmata – atomicita, souběžnost, izolace, trvanlivost
- metadata – popis schématu

Výhody

- univerzální formát provýměnu informací s mnoha výhodami
- snadná transformace do jiných formátů
- ideální pro ukládání dokumentů
- univerzální pro datové modelování
- snažší (přirozenější) reprezentace objektů

Nevýhody

- tam kde stačí relační, raději relační
- aplikace se specifickými požadavky na výkon – XML pomalejší
- v plenkách
  - nedosahují takové robustnosti – transakce, souběžný přístup, škálovatelnost
  - chybí podpora standardů
  - metody pro indexování a optimalizaci se zatím vyvíjí

Další

- **práce s databází**
  - aktualizace
  - čistění dat
  - dotazování
  - skládání / transformování
- **mezi dokumenty a databázemi** neexistuje jasná hranice
- **legitimní** oba přístupy – někde uprostřed semi-strukturovaná data, která jsou neuspořádaná či neúplná
  - struktura se může měnit, dokonce nepredikovatelně
  - HTML, Bibtexovské soubory
- **XML** – značkovací jazyk rodiny SGML – založen na logickém vyznačování
  - stromová struktura
  - jediný kořenový element
  - součástí mohou být atributy – dvojic klíč - hodnota
  - tag obsah /tag
- **DTD** – Document Type Definition → novější **XSD** (XML Schema Definition)
- **termíny**
  - **XML** – specifikace XML jazyka
  - **XSD/DTD** – způsob specifikace XML dokumentu; dokumenty splňující strukturu se označují jako validní

- **XSLT** – způsob transformace XML dokumentů – do mnoha výstupních formátů XML, HTML, WML, PDF…
- **XLink/XPointer** – specifikuje, jak jsou jednotlivé XML dokumenty (případně jejich části) spojeny dohromady
- **XPath/XQL/XML-QL/Quilt/XQuery** – dotazovací jazyky určené pro hledání v XML dokumentu dle určitých vyhledávacích kritérií

XML-QL

- dotazovací jazyk, určen pro elektronickou výměnu dat(EDI – electronic data interchange)
- vychází z předpokladu že XML dokument odpovídá databázi a jeho databázovému DTD
- více **datově** a **databázově** orientovaný, než ostatní dotazovací jazyky
- vychází z jazyka SQL a byl navržen na standard, ale nebyl schválen
- níže uvedená konstrukce vrací struktury, které mají v NAME Amazon.com a vrací jejich telefonní číslo
  - **CONSTRUCT** – vytváří výsledek (jako resultSet v SQL)

XQL

- podobný XPath, něco navíc, něco naopak není třeba
- navržen jako jednoduchý dotazovací jazyk použitelný v různých XMl prostředích
- pokouší se dívat na několik XML dokumentů jako na databázi
- výsledkem dotazu množinu uzlů, které lze zpracovat nebo použít pro další dotazy
- navržen Microsoftem, nakonec konsorcium vybralo jako doporučení k implementaci XPath
- př: Book[Author=”Karel Čapek”][1]

XPath

- standard pro procházení XML dokumentů
- přesná definice polohy prvku v XML dokumentu
- podobné cestě k souboru
- hledaným prvkem může být i komentář či jmenný prostor
- př: //zakaznik[@id=”2001”]

XQuery

- připravovaný standard W3C
- vychází z Quilt jazyka
- inspirace u XPath, XQL, XML-QL…

18. NoSQL databáze – chakteristika, porovnání ACID a BASE.

Status: Zcela nová otázka.

Charakteristika

- databáze, které nejsou založené na datovém modelu – **nejsou relační**
- dovedou zpracovávat **obrovské množství dat** v reálném čase
- neřeší se transakční zpracování (**ACID**) , ale data jsou snadno dostupná vždy i v částečně konzistentním stavu
- podporují **nerelační datový model**
- podporují **distribuovanou architekturu** – lze použít jako centrální DB, ale síla v distr.
- většina DB NoSQL je open source
- řeší většinou **CAP** omezením konzistence dat - **BASE**

Architektura

- používá se distribuovaná architektura
- např: u MongoDB se zavádí pojmy jako je **PRIMARY** a **SECONDARY** uzel
  - do PRIMARY se zapisuje a do ostatních se replikují data
  - uzly musí být ve stejné replikační množině
  - zajišťuje dostupnost dat při výpadku nějakého uzlu, rychlejší
- **existující projekty a řešení**
  - jednotlivé implementace NoSQL DB systémů se od sebe hodně liší
  - některá architektonická řešení se lze však vysledovat
  - u jednotlivých implementací se potom vyskytují určité rozdíly

ACID vs BASE

ACID

- **Atomic** (=atomicita) – atomičnost transakcí, žádný rozpracovaný stav a to i ve vztahu k možné chybě OS nebo HW (celá transakce proběhne nebo nic)
- **Consistency** (=konzistence) – v DB jsou pouze platná data podle daných pravidel. izolovaná transakce zachovává konzistenci DB
- **Isolation** (=izolovanost) – souběžné transakce se neovlivňují – serializace, pořadí není zajištěno
- **Durability** (=trvalost) – uskutečněná transakce nebude ztraceni, a to ani po pádu SW nebo HW

- nevýhody => netriviální, omezuje změny dat (zamykání) a přístup k datům (rychlost)

CAP teorém u sdílených distribuovaných systémů

- **Consistency** – každý uzel/klient vidí ve stejnou chvíli stejná data
  - konzistentní nezávisle na běžících operacích či jejich umístění
- **Availability** (=dostupnost) – každý request dostane odpověď, jestli byl nebo nebyl úspěšný
  - nepřetržitý provoz, vždy možnost zapsat a číst data
- **Partition tolerance** (=odolnost proti rozdělení) – funkční navzdory chybám sítě nebo výpadkům uzlů
  - možnost výpadku části infrastruktury (odstávka pro údržbu), databáze je schopna korektně fungovat, i když je přerušena komunikace mezi jednotlivými servery.

teorém ⇒ u sdílených systémů je možné uspokojit pouze 2 ze 3 požadavků

- **CA** – Postresql, MySQL (relační), Vertica (sloupcová)
- **CP** – BigTable (sloupcová), Mongo (dokumentová), Redis (Key-Value)
- **AP** – Cassandra (sloupcová), CouchDB (dokumentová), SimpleDB (dokumentová)

Požadavky na moderní DB

- **cloud, distribuovaná DB**
  - decentralizace úložiště dat, redundance pro odolnost proti výpadkům a rychlost, velké objemy dat a velké množství operací
- **problematické datové typy**
  - údaje klíč-hodnota, objekty, nestrukturované dokumenty…
- **iterativní vývoj**
  - časté změny schématu DB, nebo dokonce žádné schéma
- **vysoké požadavky na škálovatelnost**
  - mobilní zařízení jako klienti
  - nerovnoměrné zatížení prostorové i časové
  - spec. požadavky na dostupnost
- **relační DB přestává odpovídat relačnímu konceptu**
  - nejedná se o matematické relace, ale spíše kolekce/množiny/ grafy nestr. dat
- **dodržování ACID omezuje práci s DB**
  - úmyslné zanedbání/odpuštění A , C, I nebo D – pro vznik rychlosti a dostupnosti dat

Base

- řeší CAP omezením konzistence dat
  - Basically Available – aplikace pracuje víceméně pořád
  - Soft state – nemusí být stále konzistentní
  - Eventual consistency – nakonec se do konzistentního stavu dostanou
- nekonzistence řešeny při čtení (verzování, nevalidní cache), zápisu (distribuce změn) nebo asynchronně (replikace dat)

ACID	BASE
silná konzistence	slabá konzistence
izolovanost	dostupnost na prvním místě
orientace na commit	přibližné odpovědi jsou ok
vnořené transakce	jednodušší, rychlejší
dostupnost	dodávka jak to jen půjde
konzervativní (pesimistické)	agresivní (optimistické)
složitá evoluce (schématu..)	jednodušší evoluce

19. Databáze typu key-value, dokumentové databáze, grafové databáze.

Status: Zcela nová otázka.

Klíč-hodnota (key-value)

- eden klíč, jedna hodnota, žádný duplikát
  - klíč může být složený (z hlavní a upřesňující části)
- přístup podle klíče přes hash tabulky => brutálně rychlé
- hodnotou BLOB – databáze se nesnaží chápat
  - zpracování hodnoty na aplikaci
- pokud nás zajímá jen část, pak neefektivní
- *zástupci: Oracle NoSQL, Dynamo, Redis, Riak*

Sloupcové (big table)

- jako u RDB, akorát že sloupce se mohou pro jednotlivé řádky vzájemně lišit
- supercolumn –adresář – řádek může obsahovat kolekci supersloupců, kde každý obsahuje kolekci sloupců
- řídká, více dimenzionální, uspořádaná mapovací funkce
- *zástupci: Cassandra (Facebook), BigTable (Google – nepoužívá ani Google)*

Dokumentové (document)

- jako klíč hodnota, akorát že hodnota je strukturovaná
  - DB vidí dovnitř, hodnota pochopena, analyzována
- hodnotou např. XML/JSON – nebo jako objekt
  - vnořování, reference, kolekce
- dotazy i složitější než přes klíče
  - XPath nebo jako v objektových DB
- *zástupci: MongoDB, Elasticsearch*

Grafové (graph)

- uzly, vlastnosti a hrany
- různé implementace úložiště – nastavitelné, generické, uživatelovo
- pro reprezentaci sítí a jejich topologií (sociální, dopravní..)
- hlavním přínosem jsou implementované algoritmy pro prohledávání grafů a vyhledávání příslušných uzlů, neporovnatelně rychlejší než běžná DB
- *zástupci: Neo4j, HyperGraphDB*
- **RDF** specifickou kategorií grafových NoSQL
  - orientovaný ohodnocený graf
  - hrana začíná v **subjektu** , končí v **předmětu** a ohodnocena **predikátem**
  - subjekt a predikát jsou reprezentovány **URI**
  - předmět je hodnota nebo URI odkazující na nějaký předmět

Partitioning

1) Virtuální nódy
   - velký počet serverů lišící se konf.
   - z důvodu zohlednění a využití rozdílného výkonu serverů se používají v konf. DBS tzv. **virtuální nódy**
   - na jednom fyzickém serveru může běžet několik virtuálních nodů, v závislosti na jeho konfiguraci
   - *používá MongoDB, BigTable či Cassandra*
2) Sharding (dělení)
   - metody, která rozděluje velké soubory dat na několik serverů
   - **Shard** v pojetí MongoDB je skupina serverů, které udržují identické kopie určené části dat
   - *používá MongoDB nebo Redis*
3) Hashing
   - nejjednodušší způsob rozdělení dat spočívá v rozdělení rozsahu primárního klíče
   - pomocí hash fce se dá zjistit, na kterém serveru leží data
   - *například používá systém Memcached*
   - **nevýhoda** - v případě výpadku jednoho ze serverů nebo při přidání serveru, musí být rozdělení dat provedeno znovu a data podle nového rozdělení opět rozdistribuována
   - u Memcached – tolik nevadí, pouze malý objem dat v paměti
   - u DB, kde jsou objemná data uložena na HDD by to znamenalo kopírování velkých objemů dat mezi servery
4) Consistent Hashing
   - *používají systémy jako Cassandra nebo MongoDB*
   - rozsah klíče na pomyslném kruhu
   - každá hodnota má svého správce, kterým je první nod na pomyslné kružnici
   - výhodou – změna se dotkne jen sousedních nodů a pouze mezi nimi je nutná redistribuce dat
5) Replikace
   - používá se k zajištění proti výpadků serveru (při vysokých počtech v infrastr. hor. škál. velmi časté) a k zajištění vysoké dostupnost
   - data udržována ve více kopiích
   - současně umožňuje **rozkládání** zátěže

20. Možnosti tvorby datových skladů a metody dolování znalostí.

Data Warehouse (DW, datový sklad)

- měl by poskytovat nikoliv operativní data, ale data přeměněná ve strategické informace

Problém s OLTP (Online Transaction Processing)

- nedosažitelnost dat vytvořených či skrytých v transakčních systémech
- dlouhé prodlevy, když se nedostatečně silné systémy snaží provést komplikované dotazy

Skladování dat

- kolekce technologií podporujících rozhodování, s cílem umožnit řídícímu pracovníku učinit lepší a rychlejší rozhodování

datový sklad

- označuje DB architekturu používanou pro údržbu historických dat, která jsou získána z jedné nebo více operativních DB
- tato data jsou typicky vyčištěna a restrukturována pro podporu dotazů, agregací a analýz
- **klíčové** – integrace vlastních a externích dat

DW v informačním prostředí organizací

- **výroba dat** – selektivní dotazy nad OLTP
- **skladování dat** - DW, datová tržiště, dotazy intenzivní na data
- **prodej dat** – OLAP – Online Analytical processing
- **klíčové** – multidimenzionalita

Modelování DW

- **OLTP databáze** – normalizované tabulky, optimalizace pro insert, update
- **OLAP databáze** – odvozené tabulky, redundantní data, **optimalizace pro dotazy**, procesní logika ve schématech
  - většinou se z ní čte, dlouhé komplexní dotazy, GB-TB dat,. sumarizovaná a konsolidovaná data, jako uživatel nějaký vedoucí pracovník, analytik
  - soubor operací **drill-down**, **roll-up** – různé pohledy na data

Přístupy k modelování

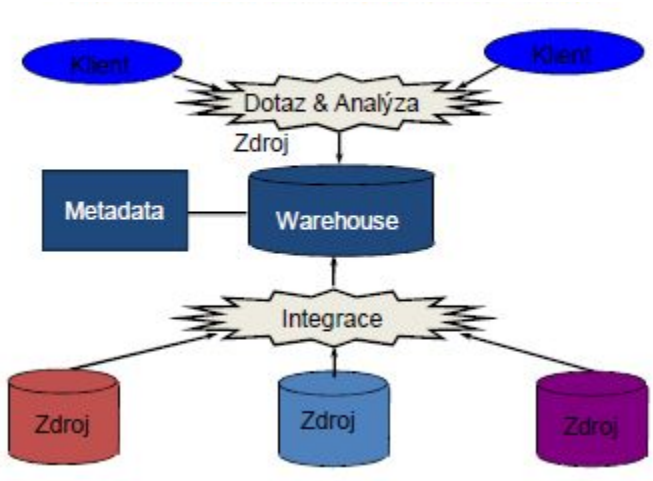
- konceptuální struktury založené na tabulkách (dimenzionální a tabulky faktů)
  - organizované do tzv hvězdicových schémat (cizí klíč odkaz na tabulku)
- konceptuální struktury založené na hyperkostkách (kostkách, multidimenzionálních polích)
  - reprezentují data jako multidimenzionální strukturu

Hyper kostka (hyperqube)

- najde se těžiště systému a jako jednotlivé dimenze se vezmou cizí klíče
- schéma MD databáze je množina vícerozměrných polí
- MD DB je dána vícerozměrnými množinami dat uloženými v těchto polích
- **výhody**
  - pole nabízí přímo jisté informace – např: počet pozic v každé dimenzi
  - jednodušší vyhledávání
  - v poli se přirozeně seskupují data (řezy kostkou)
  - seskupování dat => agregace
- **objem** – množina dat
- nejvýhodnější pokud existují multizávislosti, efektivnější rozložení do kostky
- **agregáty** – jenom se agreguje podle nějaký vlastnosti (odebere se dimenze) a vrátí se výsledek

pomocí tabulek a vztahů

- jako standardní schéma
- hierarchie dimenzí – zakresluji se hrany tam, kde je cizi klic, jako uzly nazvy tabulek



Architektura warehouse

Data Mining

= netriviální proces identifikace nových, platných, potenciálně použitelných a snadno pochopitelných vzorů v datech.

- neustále generujeme a ukládáme stále více dat,
- DB technologie jsou rychlejší, levnější, data rozsáhlejší,
- ale vyvodit užitečné závěry je stále složitější → smysl: dát uloženým datům význam
- poznatky z několika oborů matematiky a informatiky – umělá inteligence, vizualizace, databázové systémy, statistika

OLAP vs. Data Mining

- Významní dodavatelé: SAS, IBM, SPSS, Silicon Graphics, Angnoss Software, StatSoft

Data Mining

- Hledání nových vzorů, znalostí, které nejsou explicitně uvedeny
- Dosahováno pomocí sofistikovaných algoritmů

OLAP

- Soubor operací (drill-down, roll-up, ...) – poskytují různé pohledy na data
- Dosahováno pomocí sumačních a předdefinovaných operací

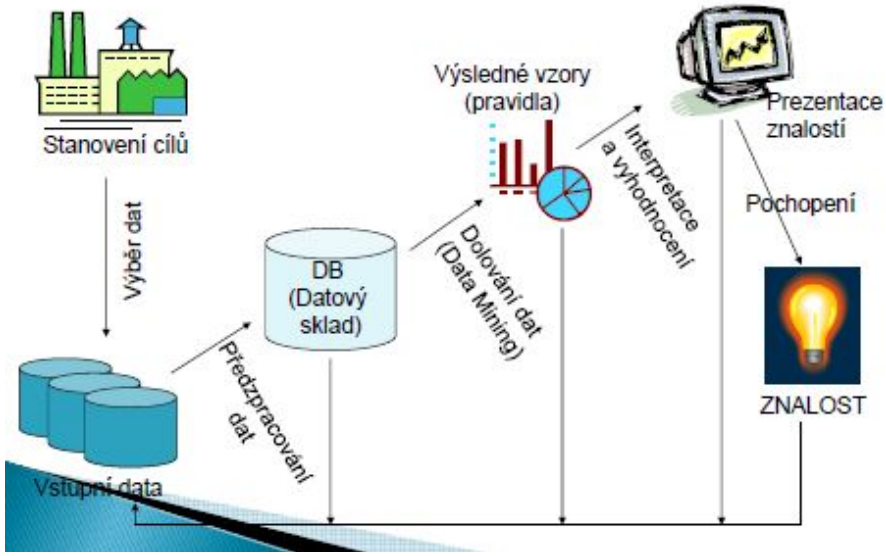
Vlastnost	OLAP	Data mining
motivace použití	Co se děje v podniku?	predikce budoucnosti, skryté znalosti
granularita dat	sumační data	data na úrovni záznamu
počet obchodních dimenzí	omezený	velký (až ∞)
počet vstupních atributů	spíš velmi nízký	mnoho atributů
velikost dat pro jednu dimenzi	ne velká pro každou dimenzi	obvykle velmi rozsáhlá pro každou dimenzi
přístup k analýze	řízený uživatelem → interaktivní analýza	řízený daty → automaticky
techniky analýzy	multidimenzionální, drill-down, slice-and-dice	příprava dat, použití nástrojů pro získávání znalostí
stav technologie	známý a rosáhle využívaný	stále se vyvíjející, některé metody již využívány v praxi

Proces získávání znalostí

Stanovení cílů

- Co chceme nalézt, v čem, bude to k něčemu, je problém řešitelný?
- Jak zobrazíme výsledky, jsou data pro metodu vhodná?

Výběr zdrojů dat



- **Typy dat z hlediska zaměření**
  - Demografická
  - Behaviorální
  - Psychografická
- **Typy databází z hlediska obsahu**
  - Zákaznické
  - Transakční
  - Databáze historie nabídek
  - Datový sklad
- **Typy dat z hlediska formátu**
  - Relační a transakční databáze
  - OO databáze
  - Multimediální databáze
  - WWW, textové dokumenty, prostorová, časová data, ...
- **Externí data**

Předzpracování dat

- vybrat z objemné databáze relativní data
- **Čistění dat**
  - položky obsahující neúplné nebo chybné hodnoty, nekonzistentní data

    - řešení: ruční opravení, nebo opravné rutiny
- **Integrace dat**
    - více zdrojů do jedné databáze -> redundance, určení ekvivalentních entit?
    - Detekce a řešení konfliktů hodnot atributů (různé kódování, jednotky, …)
- **Transformace dat**
    - Do formátu vhodného pro dolování dat
        - Slučující techniky – sumační operace (více hodnot -> jedna hodnota)
        - Generalizace – nižší úroveň nahrazena vyšší (ulice -> město)
        - Normalizace – přepočítání hodnot do daného intervalu
    - Přidávání nových (odvozených) atributů
    - Diskretizace hodnot numerických atributů
        - Rozdělení numerických hodnot na intervaly, ekvidistantní, do hloubky, pokročilé metody
- **Redukce dat**
    - Agregace v kostce – redukce sumačními operacemi
    - Redukce rozměrů – detekce a odstranění nadbytečných a nepoužívaných atributů
    - Komprese dat – zmenšení objemu dat
    - Numerosity – data nahrazena alternativní menší reprezentací

Dolování dat

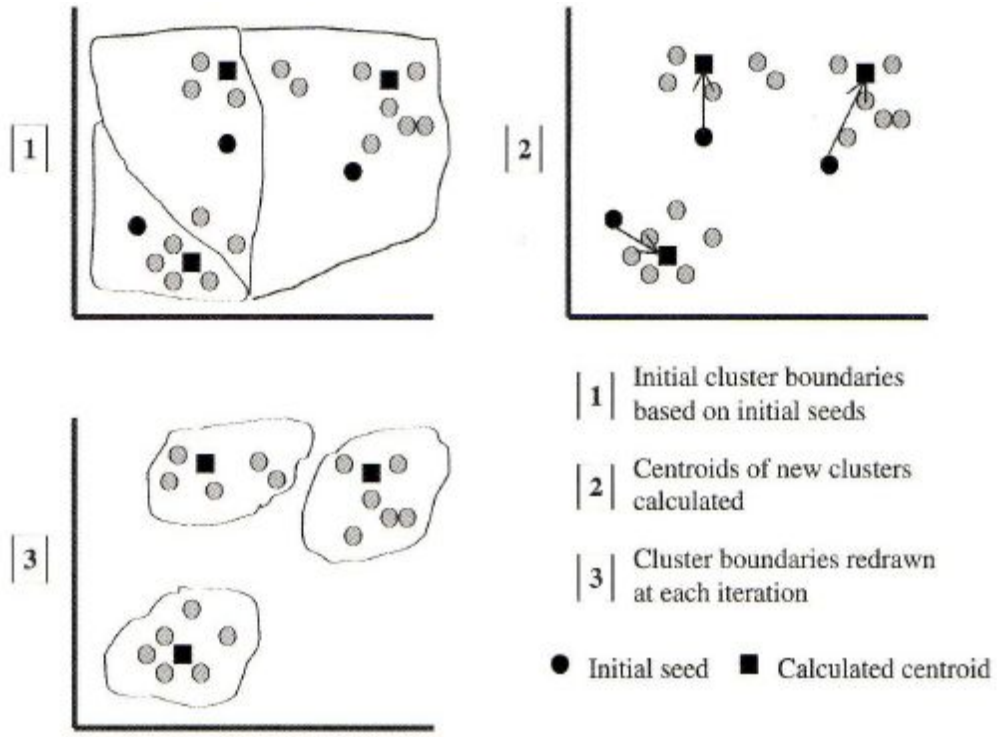
- aplikace zvoleného algoritmu na předzpracovaná data, dle typu znalosti a dat:

Asociační pravidla A => B

- *„Jestliže transakce obsahuje položky z množiny A, pak také pravděpodobně obsahuje položky z B“*
- Dva ukazatele:
    - Podpora (support) – ppst, že se vyskytují v db položky z A i B
    - Spolehlivost (confidence) – podmíněná ppst, že se v transakci vyskytuje B za předpokladu, že se tam vyskytuje A
- Silné asociační pravidlo – má podporu a spolehlivost vyšší než uživatelem zadaní hodnota – málo silných asociačních pravidel -> víceúrovňová asociační pravidla (položky sdruženy do skupin)
- Frekventovaná množina – množina, která má podporu vyšší než minimální hodnota
- Postup: výpočet frekventovaných množin (alg. Apriori), generování asociačních pravidel z frekventovaných množin
- Asociační pravidla v relačních databázích
    - kategorické atributy
        - konečný počet hodnot, lze použít Apriori
    - kvantitativní atributy
        - nemají konečný počet hodnot, nutnost diskretizace

Shlukování

- Nejstarší nástroje
- Roztřídění objektů do skupin (shluků), které nejsou předem stanoveny a nemají tedy význam (ten je potřeba zjistit)
- Rozdíly uvnitř shluků minimální, rozdíly jednotlivým shluků maximální
- Problém: Jakou metriku použít?
- Metody:
    - **Rozdělovací**
        - Rozdělení na předem daný počet shluků
        - Např. alg. K-means – optimalizuje těžiště shluků a dané prvky pak přiřadí nejbližšímu těžišti



- **Hierarchické**
  - Postupné rozdělování velkých shluků nebo postupné slučování malých shluků → hierarchická struktura
  - Ukončení procesu při splnění určité podmínky (např. minimální počet shluků)
- Další metody – **neuronové sítě**, **mřížky**, apod.

- *Příklady aplikací: marketing, plánování města, studie zemětřesení, pojištění, geografie*

Klasifikace

- Rozdělování do předem známých skupin
- Úspěšnost se měří procentem úspěšně klasifikovaných objektů
- Klasifikátory (modely)
  - Nejčastěji rozhodovací stromy
    - 1. Konstrukce stromu na základě vzorku dat
    - 2. Klasifikace objektů na základě stromu
      - Vnitřní uzel – test hodnoty jistého atributu
      - Koncový uzel – třída, kam je klasifikován
    - Pravidla – IF ... THEN ..., lze převést na rozhodovací strom
    - Neuronové sítě
- Predikce

Využití data miningu

- Členění (segmentace) zákazníků – porozumět zákazníkovi a jeho chování
- Analýza nákupního košíku – nalezení závislostí mezi různým zbožím
- Management rizik – odhalení rizikových zákazníků (např. u pojišťoven)
- Detekce podvodů – např. hledání extrémních útrat na kreditce
- Odhalování zločinnosti – odhalení potenciálních neplatičů půjček
- Predikce požadavků – předpověď zájmu zákazníků o různé zboží

Dotazovací jazyky pro data mining – součásti dotazu

- **Relevantní data** – jméno DB/DB skladu, DB tabulky/kostky, podmínky pro selekci dat, kritéria pro seskupování, ...
- **Typ znalosti** – asociační pravidla, shlukování, klasifikace, ...
- **Doménová znalost** – typické využití: konceptuální hierarchie (stromová, seskupovací, odvozená z operace, ...)
- Metriky zajímavosti
    - Jednoduchost – počet prvků pravidla, velikost rozhodovacího stromu
    - Použitelnost – např. podpora a spolehlivost
    - Jedinečnost – odstranění podobných znalostí
- Vizualizace/prezentace získaných znalostí
    - Různé formy – grafy, tabulky, konceptuální hierarchie