Západočeská univerzita
FAKULTA APLIKOVANÝCH VĚD

Okruhy otázek ke státní závěrečné zkoušce z předmětu
Databázové technologie (DB)

Databázové systémy 1 (DB1)
Databázové systémy 2 (DB2)
Případové studie databázových systémů (PSDS)

Studijní program:	3902	Inženýrská informatika
Obor:	2612T025	Informatika a výpočetní technika – Softwarové inženýrství
	3902T031	Softwarové inženýrství

Akademický rok: 2005/2006

Obsah:

1 Databázové jazyky

- Jazyky pro definici dat (prostředky pro popis dat, jazyky typu DDL – Data Definition Language)
- Jazyky pro manipulaci s daty (prostředky pro popis algoritmů, jazyky typu DML – Data Manipulation Language)
- Jazyky pro řízení dat (prostředky pro popis ochrany přístupu k datům, jazyk DCL – Data Control Language)

DDL – Data Definition Language - jazyky pro definici dat

- jazyky na úrovni definice schématu databáze svou složitostí úměrné složitosti vlastního databázového modelu
- využívány spíše správcem dat
- neposkytují pouze prostředky pro popis struktury dat, ale u komerčních systémů i pravidla pro uložení dat, přístup k nim, případně funkce

Základní komponenty

1. Jméno domény
2. Popis obsahující přesný popis množiny hodnot, které tvoří doménu
3. Uspořádání určující specifikaci hodnot v doméně sloužící pro možnost srovnání jejích prvků
4. Akce při narušení sémantické integrity sloužící k specifikaci akce, která se má provést v případě, že údaj nepatří do dané domény

Další možnosti komponent

5. Výčet prvků domény
6. Množinové operace na již definované části domény pomocí jiných domén

DML – Data Manipulation Language – jazyk pro manipulaci s daty

- základní uživatelský prostředek pro manipulaci s daty v databázi a v uživatelském schématu, pokud možno bez ohledu na fyzické cesty přístupu k jednotlivým datům
- tyto jazyky se vždy vztahují k jednomu schématu databáze daného databázového modelu a zpravidla transformují jeden stav databáze do jiného stavu databáze
- další možností jazyků tohoto typu jsou jazyky pro manipulaci schématu databáze, tj. definice změn schématu

DML je množina výrazů popisujících následující operace s daty: AKCE a VÝBĚRY

AKCE (lze uvažovat jednotlivě nebo jejich kombinace)

a) Přiřazení bodu zpracování v logické struktuře (např. uzlu grafu, řádku tabulky, odpovídající relaci, apod.)
b) Získání (zpřístupnění) části databáze
c) Vložení (přidání) dalších dat do databáze
d) Rušení (odebírání) dat z databáze
e) Změna dat v databázi

VÝBĚRY (výběr části databáze můžeme specifikovat následujícími způsoby)

a) Logickou pozicí (např. bodu zpracování)

b) Specifikací hodnot dat podmínkou
c) Relatibilitou (vzájemnou souvztažností dat)

Logickou pozici získáme jak explicitně (např. speciálním indikátorem zpracování pro daný typ objektu), tak implicitně (přechod na další záznam).
Specifikace hodnot dat se provádí vhodným logickým jazykem (např. podmnožina jazyka logiky 1. řádu). V DML se takovému výrazu říká kvalifikace. Kvalifikace může existovat i v operacích typu změna (změna množiny hodnot v dané struktuře).
Relatibilita umožňuje pomocí základních konstruktů popisujících vztahy mezi daty definovat výběr přímo.

Kombinace uvedených typů strategie výběru - dvě složky – výběrová a aktualizační
Aktualizační složka – mění obsah databáze
Výběrová složka – stav databáze zůstává zachován

Podle způsobu zpracování v logické struktuře databáze rozlišujeme DML do dvou skupin:

- Navigační
- Specifikační

Jiná terminologie

- Procedurální
- Neprocedurální

A ještě jiná terminologie

- Prekriptivní
- Deskriptivní

2 Dotazovací jazyk SQL – jazyky DDL, DML a DCL

Dotazovací jazyk
Jazyk určený ke komunikaci uživatele s vyhledávacím programem, umožňující vyhledávání a případnou úpravu požadovaných dat. Základními typy jsou jazyky procedurální vyžadující algoritmický popis přístupu k hledaným informacím a jazyky neprocedurální, zadávající pouze podmínky, jež mají vyhledané údaje splňovat. Z hlediska uživatelského se rozlišují jazyky umožňující výběr z předdefinovaných dotazů (např. menu), jazyky pro listování (browsing, navigaci) a strukturované jazyky s klíčovými slovy (např. SQL); ve stadiu výzkumů je využití přirozeného jazyka ve funkci dotazovacího jazyka.

Vyjadřovací síla dotazovacích jazyků
Programování versus relační algebra – relační algebra je jazyk velmi vysoké úrovně
Dotazovací jazyk, který umožňuje realizovat relační algebru se nazývá relačně úplný.
Komerční svět: SQL, jazyky formulářů, obrázkové jazyky

Jazyk SQL
Historie databázového jazyka SQL (tehdy se tak ještě nejmenoval) začíná v roce 1974, kdy byly uveřejněny první práce Dr. Codda o relačních databázových modelech. První použitelnou verzi databázového jazyka vytvořila firma IBM pod názvem SEQUEL (Structured English Query Language). Už tehdy byla patrná snaha vytvořit databázový jazyk, který by vycházel z přirozeného jazyka – angličtiny. Postupně se k tomuto standardu přidávaly další firmy (Oracle, Sybase, Informix), a tak vznikl "nepsaný standard" databázové jazyka s názvem SQL. Postupně byly přijaty vylepšené a upravené standardy jazyka SQL s názvem SQL-86 a dále SQL-92. Pro verzi SQL-92 se vžil zkrácený název SQL-2. Jazyk SQL můžeme použít jednak jako dotazovací jazyk pro práci s údaji v relační databázi, případně jako část hostitelského jazyka pro vývoj databázových aplikací. Například databázová platforma Oracle má implementovaný procedurální jazyk PL/SQL, anebo Microsoft SQL Server 2000/2005 má implementovaný jazyk Transact SQL (T-SQL).

- SQL je neprocedurální jazyk
- Je určen pro relační databázové systémy
- SQL lze využít i jako rozhraní jiných, nerelačních systémů
- Možnosti dělení SQL – interakční verze (příkazy jsou zadávány samostatně z terminálu) X hostitelská verze (příkazy jsou součástí hostitelského jazyka, nutno zajistit vazby mezi proměnnými a výsledky dotazů)
- Další možné dělení – dle práce s dotazem – kompilační X interpretační systémy

Verze jazyka SQL

Rok	Jméno	Alias	Komentář
1986	SQL-86	SQL-87	Poprvé publikováno ANSI. Prohlášeno za ISO v roce 1987.
1989	SQL-89		Malé úpravy
1992	SQL-92	SQL2	Rozsáhlé úpravy
1999	SQL:1999	SQL3	Přidány regulární výrazy, rekurzivní dotazy, triggery, neskalární typy a některé objektově orientované záležitosti (poslední dvě věci jsou kontroverzní a nejsou plně podporovány)
2003	SQL:2003		Představeny XML vlastnosti, okenní funkce, standardizovány

	sekvence a sloupce a auto-generovanými hodnotami (včetně identity-sloupců).

Jazyk SQL lze rozdělit na několik částí

Definice dat – Data Definition Language

Umožňuje definovat struktury a vytvářet v databázi objekty, jako tabulky, pohledy, indexy a podobně, případně měnit jejich strukturu anebo je rušit (odstaňovat). Na tomto místě je třeba si uvědomit rozdíl mezi vymazáním a zrušením. Vymazat můžeme například všechny, nebo jen některé údaje z databázové tabulky, přičemž tato tabulka jako objekt zůstane zachována a je připravená pro případně vložení nových údajů. Když ale tabulku zrušíme (odstraníme) například pomocí příkazu DROP TABLE, bude z databáze odstraněna natrvalo (pokud nezasáhne transakční nebo zálohovací logika) databázová tabulka současně se všemi údaji a objekty, které obsahuje. Do této skupiny patří například příkazy:

CREATE DATABASE	CREATE INDEX	DROP VIEW
CREATE TABLE	DROP INDEX	DROP INDEX
ALTER TABLE	CREATE VIEW	CREATE SEQUENCE
DROP TABLE	ALTER VIEW	ALTER SEQUENCE

Manipulace s daty – Data Manipulation Language

Umožňuje manipulaci s údaji, to znamená výběr a vkládání údajů a jejich aktualizaci, vymazání údajů a samozřejmě velmi mocný příkaz SELECT pro výběr údajů. Patří sem čtyři hlavní příkazy:
SELECT – získávání dat
INSERT – vkládání dat
UPDATE – úprava dat
DELETE – mazání dat

Získávání dat (Data retrieval)

Nejčastější operací prováděnou dotazovacím jazykem nad databázemi je získávání dat. K této operaci se používá příkaz SELECT.
SELECT je dotaz, ve kterém uživatel specifikuje popis výsledné sady výsledků, ale nespecifikuje fyzické operace nutné k získání takovéto sady výsledků. Zbytek práce je ponechán na databázovém systému.

SELECT

- FROM – z kterých tabulek brát data a jak tabulky spojovat (JOIN)
- WHERE – identifikace řádek v tabulce
- GROUP BY – sjednocení skupiny dat do jednoho záznamu
- HAVING – funguje jako WHERE nad výsledky GROUP BY a tudíž může používat agregační funkce
- ORDER BY – způsob řazení

Příklad:

```
SELECT * FROM knihy
  WHERE cena > 500.00
  ORDER BY nazev
```

Řízení dat - Data Control Language

Obsahuje speciální příkazy pro řízení provozu a údržbu databáze. Také můžeme přidělovat a odebírat uživatelská privilegia jednotlivým uživatelům a skupinám uživatelů. Patří sem příkazy:

CREATE USER	DROP USER	REVOKE
ALTER USER	GRANT	

Transakce (Data transaction) – Transaction Control Commands (TCC)

Je podmnožinou řídících příkazů. Do této podmnožiny patří například příkazy:

SET TRANSACTION	ROLLBACK
COMMIT	SAVEPOINT

3 Integritní omezení

Nejdříve si prosvištíme pár základních pojmů z databází:
Doména je množina hodnot stejného významového typu.
Relace je podmnožina kartézského součinu nad několika doménami, množina vztahů mezi prvky několika domén, dá se zachytit jako tabulka
Atribut relace představuje jméno použité hodnoty z domény v relaci
Relační schéma se dá vyjádřit jako jméno relace + jména atributů, v čase je stálé na rozdíl od těla relace
Klíč – atribut (či skupina atributů), kterým hodnota (kombinace hodnot) identifikuje n-tici relace

Integritní omezení
Vymezují hodnoty objektů databáze tak, aby mohly mít v reálném světě smysl. Integritní omezení můžeme zavést na třech úrovních:

I. **Entitní integrita** – zajištění jednoznačné identifikace každého řádku relace – jednoznačný primární klíč
II. **Doménová integrita** – zajištění, aby každá hodnota atributu byla v souladu s množinou přípustných hodnot
III. **Referenční integrita** – Cizí klíče (tj. atributy nebo skupina atributů tvořící v jiné relaci primární klíč nemůže nabývat hodnoty, které jsou v rozporu s hodnotami odkazovaného primárního klíče – různé způsoby zajištění)

Primární klíč jsme definovali jako jednoznačný identifikátor každé entity, nebo jinak řečeno, každého záznamu. Měl by se skládat z minimálního počtu atributů. Může to být sloupec, případně kombinace více sloupců, které slouží k jednoznačné identifikaci každého řádku tabulky. Hodnota pole primárního klíče musí být v rámci tabulky jedinečná. Například občana bychom jednoznačně mohli identifikovat rodným číslem. Bohužel systém přidělování rodných čísel nebyl dokonalý, a tak všichni musíme ve většině důležitých formulářů vyplňovat kromě rodného čísla i jméno, příjmení, datum a někdy i místo narození.

Pole primárního klíče musí obsahovat konkrétní hodnotu, tedy nesmí nikdy nabývat hodnoty NULL. Bez primárního klíče není možné definovat relace mezi tabulkami. Při výběru atributů, které budou tvořit primární klíč, je potřeba dbát na to, aby vybrané atributy skutečně plnily úlohu identifikačního klíče, ale na druhé straně, aby jejich použití bylo efektivní i z časového a paměťového hlediska.

Cizí klíč je sloupec, případně kombinace více sloupců, které jsou propojeny s primárním klíčem v jiné tabulce. Zavedením relací mezi tabulkami pomocí cizího klíče se minimalizuje objem údajů, které jsou v databázi uloženy, protože namísto kompletních údajů o zákazníkovi je v tabulce objednávek uveden jen klíč do jiné tabulky.

Způsoby zajištění integrity

1. **Restriktivní**

- nedovolí zrušit řádku nadřízené tabulce, pokud k ní existují řádky v podřízené tabulce
- rovněž nepřipustí změnu hodnoty klíče

2. **Kaskádovitý**

- zruší záznam a zruší všechny jeho podřízené záznamy
- povolí přepsat klíč a všude povolí přepsat podřízené klíče

3. **Dosazení prázdné hodnoty**

Poznámka:
Integritní omezení na úrovni dat lze uskutečnit i pomocí napsaných „programů“ – uložené procedury a triggery
Uložená procedura – program uložený na serveru
Trigger – program uložený na serveru, který se spouští při určité činnosti

Integritní omezení v jednotlivých verzích SQL

SQL 86

- NOT NULL
- UNIQUE

SQL 89

- PRIMARY KEY
- CHECK – CREATE TABLE kredit (počet kreditů INTEGER CHECK BETWEEN 1 AND 6);
- REFERENCES a FOREIGN KEY – CREATE TABLE predmet (... FOREIGN KEY (garant) REFERENCES ucitel(cislo_ucitel));

SQL 92

- definice cizího klíče doplněna o
    - ON UPDATE CASCADE
    - ON DELETE CASCADE
    - ON DELETE NULL
    - ON UPDATE NULL
    - ALTER TABLE ADD CONSTRAINT
    - ALTER TABLE DROP CONSTRAINT
- lze provést změnu v definici tabulky omez. podmínek
    - SET CONSTRAINT OFF (ON)

4 Procedurální prostředky v rámci jazyka SQL, jazyk PL/SQL

Hlavním omezením jazyka SQL je skutečnost, že se jedná o neprocedurální jazyk. V praxi to znamená, že příkazy jazyka SQL se vykonávají sekvenčně, bez možnosti použití klasických programátorských konstrukcí, jako jsou například cykly, podmínky, využití procedur a funkcí a podobně. Proto má skoro každá moderní databázová platforma implementované určité procedurální rozšíření jazyka SQL. Potom se takto rozšířený jazyk SQL stává mocným nástrojem, který umožňuje naprogramovat i ty nejsložitější algoritmy pro práci s údaji, případně pro jejich zpracování a vyhodnocování.

MS SQL Server 2005 – T-SQL

- Rozšíření má název Transact SQL (T-SQL)
- Používá se u platforem Sybase a Microsoft SQL Server pro tvorbu částí aplikační logiky, uložených procedur a triggerů

ORACLE – PL/SQL

- Rozšíření má název PL/SQL (Transaction Processing Language)
- Součástí systému ORACLE, specifické rozšíření o procedurální rysy
- Poskytuje prostředky pro definování a spouštění programových jednotek PL/SQL (procedury, funkce, balíky-package)

PL/SQL

Nástroje pro tvorbu a spouštění PL/SQL – SQL*Plus, SQL Worksheet

BLOK

Blok v jazyce PL/SQL je základní kódový segment jazyka

Zápis bloku:

```
DECLARATION
  Deklarace – deklarační sekce
BEGIN
  Výkonné příkazy – výkonná sekce
EXCEPTION
  Výkonné příkazy – sekce pro zpracování vyjímek
END;
```

Příklad:

```
DECLARE
  Vypis_prom char(14);
BEGIN
  Vypis_prom := 'Vypis promenne';
END;
```

Příklad (výpis do konzole):

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('AHOJ SVETE');
END;
```

Anonymní blok
V předcházejících příkladech byl použit tzv. anonymní blok (nepojmenovaný blok). Anonymní blok je použitelný pouze jednou – v místě jeho definice. Tento blok lze uvést do příkazové dávky SQL, definiční části databázového triggeru, do definice prvku aplikačního menu.

Proměnné
Proměnné je potřeba před prvním použitím deklarovat. Navíc je možné proměnnou také inicializovat (přiřazení počáteční hodnot nebo pomocí klíčového slova DEFAULT). Pro kombinaci statického tisku a obsahu proměnných lze použít operátor zřetězení ||.

Skalární datové typy
BINARY INTEGER, DEC, DECIMAL, NUMBER, NUMERIC, REAL, INTEGER, …
CHAR, CHARACTER, LONG, STRING, ROWID, …
BOOLEAN
DATE, INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH, …

Kompozitní typy
RECORD, TABLE, ARRAY

Referenční typy
REF_CURSOR, REF obj_typ

LOB typy
BFILE, BLOB, CLOB, NCLOB

Příklad (proměnná + zřetězení+ typ proměnné dle sloupce):
```
SET SERVEROUT ON SIZE 10000 -- vyrovnavaci pamet nastavena na 10000 znaku

DECLARE
  v_p1 number(3) NOT NULL DEFAULT 10;
  v_misto zamestnanec.adresa%TYPE;
BEGIN
  DBMS_OUTPUT.PUT_LINE('v_p1 = '||v_p1);
  v_misto = 'Plzeň';
END;
```

Syntaxe bloku PL/SQL – vnořené bloky
Existuje možnost vytvářet vnořené bloky
- Platí zásady obdobné blokové struktuře Pascalu – globální a lokální proměnné
- Ve vnitřním bloku lze používat (vidět) proměnné deklarované ve vnějším bloku
- Ve vnějším bloku nelze používat (vidět) proměnné deklarované ve vnitřním bloku

Řízení toku programu

IF – THEN – ELSE – END IF
```
IF ACCOUNT_BALANCE <0 THEN
  ACCOUNT_STATUS = 'V MINUSU';
ELSIF ACCOUNT_STATUS = 'NEZAPORNE';
END IF;
```

CYKLY – LOOP

LOOP
Loop block
END LOOP;

WHILE *condition* LOOP
Loop block
END LOOP

FOR *countvar* IN [REVERSE] *start_num..end_num* LOOP
Loop block
END LOOP;

Kurzory

- prostředek pro získání informace z databáze a předání do programu v jazyce PL/SQL
- explicitní kurzor – nutno deklarovat, otevřít, načíst, data a uzavřít
- implicitní kurzor – je deklarován a prováděn přímo v těle programu, v tomto typu kurzoru jsou povoleny pouze příkazy SQL, které vrací jednotlivé řádky nebo nevrací žádné řádky, příkazy SELECT, UPDATE, INSERT a DELETE obsahují implicitní kurzory

Atributy kurzoru poskytují informace o stavu kurzoru

<jméno_kurzoru>%FOUND
- vrací TRUE, pokud při posledním FETCH načetl řádek, jinak FALSE

<jméno_kurzoru>%NOTFOUND
- opak FOUND

<jméno_kurzoru>%ISOPEN
- hodnota TRUE, pokud je kurzor otevřen, hodnota FALSE před otevřením nebo po uzavření kurzoru

<jméno_kurzoru>%ROWCOUNT
- atribut sdělí počet již načtených řádek, před načtením nějakého řádku má hodnotu 0. Vpříkaze UPDATE má hodnotu počtu aktualizovaných řádek.

Explicitní kurzor

Deklarace kurzoru:
DECLARE CURSOR <jméno kurzoru>IS <dotaz>;
Otevření kurzoru:
OPEN <jméno kurzoru>;
Načtení dat, pouze jeden řádek
FETCH <jméno kurzoru>INTO <jméno proměnné1>, <jméno proměnné2>, …;
Zavření kurzoru
CLOSE <jméno kurzoru>;

Implicitní kurzor

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

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

Kurzor pro cyklus LOOP
FOR *rowname* IN *cursorname* LOOP
Loop block
END LOOP;

Vyjímky – EXCEPTION
EXCEPTION WHEN <*jméno vyjímky*> THEN
<*exception block>;*
WHEN <*jméno vyjímky*> OTHERS THEN
<*exception block>;*
END;

Vyjímky (předdefinované)– NO_DATA_FOUND, TOO_MANY_ROWS, DUP_VAL_ON_INDEX, ZERO_DIVIDE, VALUE_ERROR, INVALID_NUMBER

Vlastní vyjímky
Deklarace:
DECLARE <jméno vyjímky>exception;
Vyvolání vyjímky:
RAISE <jméno vyjímky>;
Obsluhu uvést na konci PL/SQL programu
EXCEPTION <jméno vyjímky> THEN
<handler-commands>;
END:

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

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

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

Uložené samostatné podprogramy

- podprogramy mohou být trvale uloženy do databáze
- mohou být použity jakoukoliv aplikací, která s databázovým strojem komunikuje
- jednou přeložený a uložený podprogram patří mezi databázové objekty
- může být referován libovolným počtem aplikací
- uložené podprogramy mohou být samostatné nebo součástí balíku

Podprogramy lze volat z

- databázových triggerů
- jiných uložených podprogramů
- aplikačních programů zapsaných ve vyšším programovacím jazyce, pro nějž existuje předkompilátor (Oracle*PRO)
- interaktivně (SQL*Plus) : EXECUTE jméno_procedury(parametry);

PACKAGE – uložené balíky

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

Package:

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

Volání podprogramů

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

5 Aktivní databáze (Oracle triggers)

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

Odlišnosti triggerů od uložených podprogramů:
- Jsou implicitně spouštěny při modifikaci tabulky
- Definují se pouze pro databázové tabulky
- Nepřijímají argumenty
- Lze je spustit pouze při DML příkazech UPDATE, INSERT, DELETE

Výhody triggerů
- Nepovolí neplatné datové transakce
- Zajišťují komplexní bezpečnost
- Zajišťují referenční integritu přes všechny uzly v integrované databázi
- Vytvářejí strategická a komplexní aplikační pravidla
- Zajišťují audit (sledování)
- Spravují synchronizaci tabulek
- Zaznamenávají statistiku často modifikovaných tabulek

Trigger (zálohování dat o smazaných knihách při jejich smazání):
```
CREATE OR REPLACE TRIGGER zrusit
BEFORE DELETE ON kniha
FOR EACH ROW
BEGIN
  INSERT INTO zrus_knihy(isbn,autor,nazev,zrusil,datum)
  VALUES(:OLD.isbn,:OLD.autor,:OLD.nazev,USER,SYSDATE);
END;
```

Části triggeru

CREATE OR REPLACE – může být nahrazen bez použití DROP – jedinečné jméno triggeru

Načasování triggeru – BEFORE|AFTER|INSTEAD OF - UPDATE|INSERT|DELETE - ON <tabulka>

Volba INSTEAD OF – od Oracle 8 – nemůže obsahovat množinové operátory, skupinové funkce, spojení, klauzuli DISTINCT

Volba FOR EACH ROW – určení, zda se jedná o řádkový či příkazový trigger
- Je-li klauzule uvedena, pak se trigger spouští jednou pro každý řádek
- Není-li uvedena, spustí se trigger pouze jednou bez ohledu na množství řádek
- V těle řádkového triggeru je k dispozici jak OLD, tak NEW hodnota aktuálního řádku

Volba WHEN – podmínka určující, zda se má trigger spustit (hodnota booleovského příkazu musí být TRUE)

Triggery jsou plně zkompilované po spuštění příkazem CREATE TRIGGER a po uložení p-kódu v systémovém katalogu.

6 SQL – XML

Dokumenty vs. databáze

Mnoho malých dokumentů	Několik rozsáhlých databází
Obvykle statické	Obvykle dynamické
Implicitní struktura – sekce, paragraf, věta	Explicitní struktura – schéma
Značkování	Záznamy
Přizpůsobený člověku	Přizpůsobený stroji
Obsah – formát/rozvržení na médiu, anotace	Obsah – schéma, data, metody
Paradigmata – "ulož jako", wysiwyg	Paradigmata – atomicita, souběžnost, izolace, trvanlivost
Metadata – autor, jméno, datum, předmět	Metadata – popis chématu

Způsob práce (dokumenty vs. databáze)

Editace	Aktualizace
Tisk	
Spell-checking	Čištění dat
Počítání slov	
Výběr informací (IR)	Dotazování
prohledávání	
	Skládání/transformování

Mezi dokumenty a databázemi neexistuje jasná hranice. V řadě případů jsou legitimní oba přístupy. Někde uprostřed se nacházejí formátovací jazyky a semistrukturovaná data. Semistrukturovaná data jsou definována jako data, která jsou neuspořádaná či neúplná, jejich struktura se může měnit, dokonce nepredikovatelným způsobem - př. data ve webovských zdrojích, HTML stránky, Bibtexovské soubory, biologická data. XML data jsou instancí semistrukturovaných dat.

XML (eXtensible Markup Language)

```
<?xml version="1.0" encoding="UTF-8"?>
  <zakaznik id="2000">
    <jmeno>
      <krestni>Pavel</krestni>
      <prijmeni>Novák</prijmeni>
    <adresa />
  </jmeno>
</zakaznik>
```

XML je značkovací jazyk z rodiny SGML. Na rozdíl od jazyka HTML je založen na logickém vyznačování. Skládá se z elementů. Element tvoří obvykle počáteční a koncová značk (tag) a obsah elementu. XML má stromovou strukturu, tzn. vždy musí existovat jediný kořenový element. Součástí elementu může být tzv. atribut reprezentovaný dvojicí klíč-hodnota.

DTD (Document Type Definition)

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Osoba [
```

```
<!ELEMENT Osoba (Jmeno,Adresa,Datum_narozeni) >
<!ATTLIST Osoba id CDATA #REQUIRED >
<!ATTLIST Osoba stav CDATA #REQUIRED >
<!ELEMENT Jmeno (#PCDATA) >
<!ELEMENT Adresa (#PCDATA) >
<!ELEMENT Datum_narozeni (#PCDATA) >
]>
```

Prosvištíme si pár termínů z XML

- **XML** - specifikace jazyka XML samotného
- **XSD/DTD** – způsob specifikace struktury XML dokumentu, všechny dokumenty splňující strukturu (šablonu) jsou označovány jako validní
- **XSLT** – způsob transformace XML dokumentů, tyto dokumenty mohou být transformovány do mnoha výstupních formátů (XML, HTML, WML, PDF, RTF, …).
- **XLink/XPointer** - specifikuje, jak jsou jednotlivé XML dokumenty, případně jejich části spojeny dohromady
- **XPath/XQL/XML-QL/Quilt/XQuery** - dotazovací jazyky, které jsou určeny pro hledání v XML dokumentu dle určitých vyhledávacích kritérií (těmi se budeme dále zabývat)

XML-QL

XML-QL (Query Language for XML data) je dotazovací jazyk, jehož použití se předpokládá především při řešení problémů elektronické výměny dat (EDI – electronic data interchange). Vychází se z předpokladu, že XML dokument odpovídá databázi a jeho DTD databázovému schématu: XML-QL je tedy více datově a databázově orientovaný, než ostatní dotazovací jazyky. Vychází z jazyka SQL, byl navržen na standard, ale nebyl schválen.

```
WHERE
   <VENDOR_LIST>
      <VENDOR>
         <NAME>Amazon.com</NAME>
         <PHONE>$ph</ PHONE>
      </VENDOR>
   </VENDOR_LIST> IN  "http://pillango.sirma.bg/vendors.xml"
CONSTRUCT <VENDOR_PHONE>$ph</VENDOR_PHONE>
```

Poznámky:

- vyhledává struktury uvedené uvnitř klauzule WHERE, které mají v elementu NAME hodnotu Amazon.com, vrací jejich telefonní čísla do proměnné $ph
- název dokumentu je v “” po slově IN
- klauzule CONSTRUCT vytváří výsledek (jako result set v SQL), tzn. Zde bude například výsledkem
  <VENDOR_PHONE>(800)555-1212 </VENDOR_PHONE>
  <VENDOR_PHONE>(800)555-1313 </VENDOR_PHONE>

XQL

XQL je jazyk velmi podobný XPath, liší se v některých částech, které pro XQL dotazy nejsou potřeba a naopak, jiné jsou zde navíc. XQL byl navržen jako jednoduchý dotazovací jazyk, použitelný v různých XML prostředích, se syntaxí použitelnou jako hodnota XML atributů. XQL se pokouší dívat na několik XML dokumentů jako na databázi, výsledkem dotazu má být množina uzlů, se kterou lze dále pracovat, nebo jí použít jako vstup pro další dotazy.

Jazyk XQL byl původně navržen firmou Microsoft a jejími spolupracovníky v době, kdy neexistovala jiná specifikace pro dotazovací jazyk nad XML. Časem konsorcium W3C vybralo jazyk XPath jako doporučení k implementaci a jazyk XQL neschválilo.

```
Book[Author="Karel Čapek"][1]
```

XPATH

Jazyk XPATH se stal standardem pro procházení XML dokumentů. Tento jazyk dokáže přesně definovat polohu prvku v XML dokumentu. Výrazy jazyka XPATH připomínají zadávání cesty k souboru v souborovém systému. Jde tedy o způsob, jak nalézt prvek v XML dokumentu. Tímto prvkem nemusí být jen element či atribut, ale například i komentář či jmenný prostor.

```
//zakaznik[@id="2001"]
```

XQUERY

Jedná se o připravovaný standard W3C. Vychází z jazyka Quilt, inspirace z XPath 1.0, XQL, XML-QL, SQL, OQL.

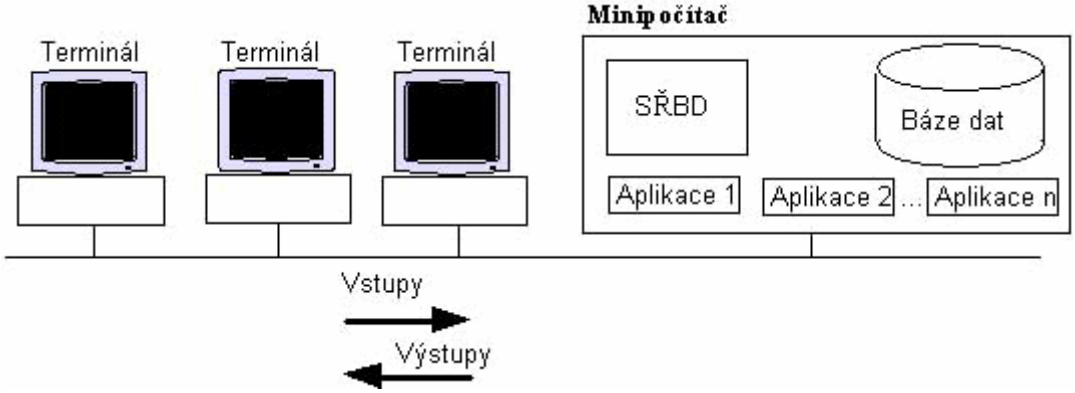
-

7 Architektura klient – server

Poznámka: otázka je naprosto stejná jako v PSDS

Architektury databázových systémů

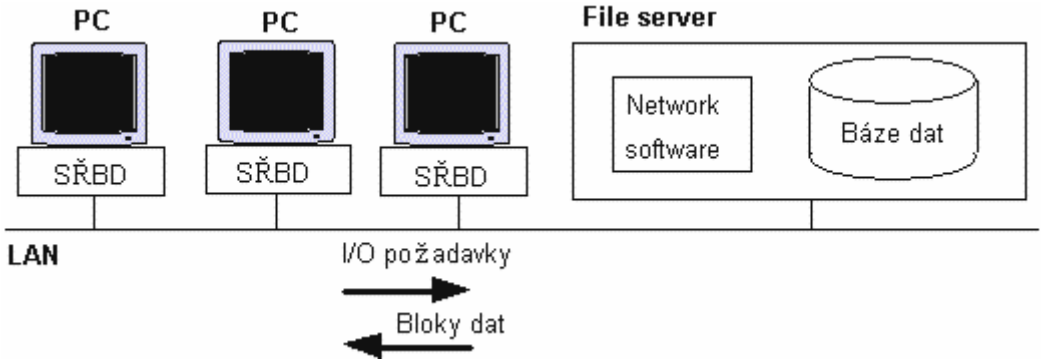
Centrální architektura
V této architektuře jsou data i SŘBD v centrálním počítači. Tato architektura je typická pro terminálovou síť, kdy se po síti přenáší vstupní údaje z terminálu na centrální počítač do příslušné aplikace, výstupy z této aplikace se přenáší na terminál. Protože aplikační program i vlastní zpracování probíhá na centrálním počítači, který může zpracovávat více úloh, mají odezvy na dotazy určité zpoždění.



Architektura file-server
Tato metoda souvisí zejména s rozšířením osobních počítačů a sítí LAN. SŘBD a příslušné databázové aplikace jsou provozovány na jednotlivých počítačích, data jsou umístěna na file-serveru a mohou být sdílena. Aby nedocházelo ke kolizím při přístupu více uživatelů k jedněm datům, musí SŘBD používat vhodný systém zamykání (položek nebo celých tabulek).

Komunikace uživatele se systémem probíhá následujícím způsobem:

- ✓ uživatel zadá dotaz
- ✓ SŘBD přijme dotaz, zasílá požadavky na data file-serveru
- ✓ file-server posílá bloky dat na lokální počítač, kde jsou data zpracovávána podle zadaného dotazu (vyhledávání, setřídění atd.)
- ✓ výsledek dotazu se zobrazí se na obrazovce osobního počítače.



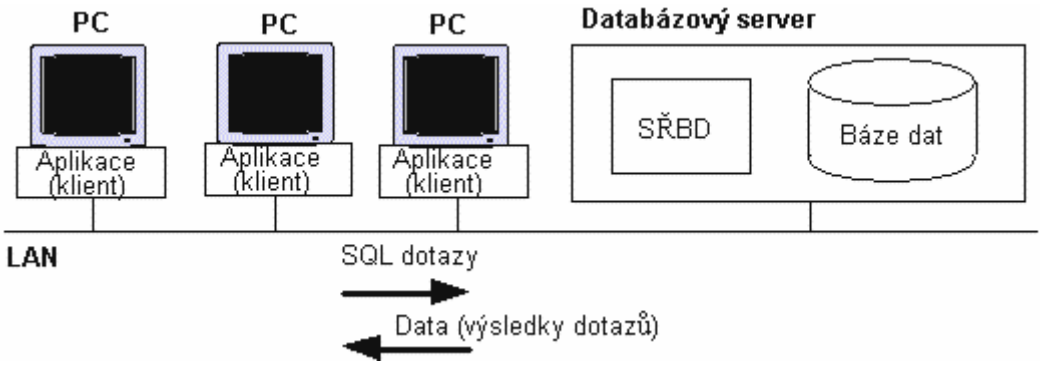
>>**Architektura Klient-Server** <<

V podstatě je založena na lokální síti (LAN), personálních počítačích a databázovém serveru. Na personálních počítačích běží program podporující např. vstup dat, formulaci dotazu atd. Dotaz se dále předává pomocí jazyka SQL (Structured Query Language) na databázový server, který jej vykoná a vrátí výsledky zpět na personální počítač. Databázový server je tedy nejvíce zatíženým prvkem systému a musí být tvořen dostatečně výkonným počítačem. Celá komunikace probíhá tímto způsobem:

- ✓ uživatel zadává dotaz (buď přímo v SQL nebo musí být do tohoto jazyka přeložen),
- ✓ dotaz je odeslán na databázový server,
- ✓ databázový server vykoná dotaz,
- ✓ výsledek dotazu je poslán zpět na vysílací počítač, kde je zobrazen.

Architektura klient-server redukuje přenos dat po síti, protože dotazy jsou prováděny přímo na databázovém serveru a na personální počítač jsou posílány pouze výsledky. Např. pokud je mezi 10 000 záznamy pouze 100 záznamů, které splňují podmínku dotazu, pak na personální počítač putuje pouze těchto 100 záznamů. V případě architektury file-server je však nutné poslat všech 10 000 záznamů na personální počítač, tam se teprve provede dotaz a zpracuje nalezených 100 záznamů.

Architektura klient-server vyhovuje i náročným aplikacím a je využívána většinou renomovaných databázových firem.



Charakteristika
Rozdělení aplikace na klientskou a serverovou část
GUI – může být rozdílné na různých klientech
Jednoduchý přenos dat mezi aplikacemi
Metodika vývoje produktu – často přírustková – postup: rozdělení na části klient a server, implementace části server, implementace části klient, testování a intalace přírůstku

Architektura klient – server
prvky architektury: hardwarová platforma, operační systémy, databázový systém, vývojové prostředí, standardy architektury
architektura klient-server znamená dekompozici funkcionality a tedy ideální model např. pro online zpracování transakcí (OLTP) v distribuovaném prostředí
síla architektury: pružnější rozdělení práce (klientům lze zpřístupnit i více serverů), aplikace běží na levnějších zařízeních, klientem může být oblíbený prezentační software

(PowerBuilder, Excel), standardizovaný přístup pomocí SQL, centralizace dat – lepší a účinnější ochrana dat

Databázový server
dedikovaný=slouží jistému účelu
databázový server - středně velký 4 procesory, velký 8 procesorů
databázově - aplikační server (super server – 16 procesorů)
aplikační server – několik procesorů

Mapování tří vrstev

1. serverová data – DBMS schéma – sdílené entity >serverové funkce (triggery, uložené procedury)
2. uživatelské objekty >přemístitelné funkce
3. okna >klientské funkce > (klientská reprezentace)

Mapování datového modelu je spojeno se třemi aspekty – rozdělení dat, duplicita dat a realizace integritních omezení

Problém – distribuce data mezi několik serverů
Východisko – výhodnější je udržování duplicitních dat – je nutno procesně zajistit konzistenci duplicitních údajů

Problém – integritní omezení (zajištění referenční integrity, kardinality vazeb, doménové integrity, atd.)
Východiska –
mohou být mapována na server jako součást jazyka DDL
na server jako součást databázového systému ve formě uložených procedur
na klientovi jako součást zpracování (tento způsob implementace je obtížně udržovatelný)

Model uživatelských objektů
Funkčnost systémů – akce s objekty
Většina systémů klient-server zatím nepoužívá objektovou technologii
Uživatelské objekty mohou být mapovány na server, na klienta nebo jako přemístitelný model
Server: objekty implementovány pomocí uložených procedur
Klient: prostřednictvím popisu v jazyce 4GL (SAL) nebo DLL knihoven
Pravidlo: čím větší část aplikačního zpracování je obecná a realizovatelná na serveru, tím menší změny je nutno provádět ve zpracování klienta při modifikaci uživatelských úkolů

8 Nedostatky relačního modelu

Přehled nedostatků relačních databází

- **Neschopnost modelovat komplexní datové struktury**
- **Podpora omezené množiny atomických dat**
- **Nezahrnuje schopnosti pro generalizaci a agregaci dat**
- **Malá výkonnost pro náročné aplikace**
- **Nepodporuje hledisko času a verzí objektu**
- **Impedance mismatch (impedanční nesoulad)**

Impedance mismatch (impedanční nesoulad)

Impedanční nesoulad je termín původně z elektrotechniky, který v softwarovém světě označuje rozdíl plynoucí z podstatné odlišností mezi relačním a objektovým modelem databáze.

Jednoduše řečeno: **relační model** uspořádá data do řádků a sloupců. Každý řádek představuje záznam, sloupce představují různé datové položky v záznamu. Jsou-li data příliš složitá, než aby se dala reprezentovat v dvourozměrné mřížce, musí se vytvořit dodatečné tabulky, které uchovají vztahové informace. Každá tabulka v takovém relačním schématu pak obsahuje některé, ale ne všechny, datové položky pro velmi mnoho záznamů.

Objektový model není omezen na uchovávání dat v řádcích a sloupcích. Místo toho návrhář vytvoří definici (šablonu), která úplně popisuje určitou třídu informací. Každý objekt (záznam) je specifickou instancí této třídy. Každý objekt tedy obsahuje všechny datové položky právě jednoho záznamu. To ale není všechno. Definice tříd mohou také obsahovat úseky kódu (tzv. metody), které pracují s daty popsanými třídou. V relačním modelu podobný způsob záznamu neexistuje.

Možná řešení

- ✓ Rozšířená relační technologie
- ✓ Vytvoření nového datového modelu
- ✓ Rozšíření OO programovacího jazyka o databázové prostředky
- ✓ Vytvořit rozšiřitelné DBMS knihovny
- ✓ Zahrnutí konstrukcí OODB jazyka do konvenčního jazyka
- ✓ Vývoj aplikačně doménových prostředků používajících OODB technologie

9 Vlastnosti objektově orientovaného datového modelu

Datový model je logická organizace objektů reálného světa, jejich omezení a relací mezi objekty. OO datový model navíc zahrnuje OO koncepty. Oproti relačnímu modelu OO model není formalizován.

Manifest OODS98

Povinné

- **Podpora persistence** (zřejmě nejdůležitější vlastnot relačních i OO databázových systémů, obecně ji lze definovat jako schopnost systému trvale uchovávat data nejen po dobu svého běhu, ale i mezi vypnutím a opětovným spuštěním)
- **Správa vnější paměti**
- **Paralelismus**
- **Zotavování**
- **Prostředky pro kladení adhoc dotazů**
- **Komplexní objekty** (na rozdíl od zjednodušených objektů v relační DB)
- **Identifikace objektů** (identifikace na základě OID, skryta uživateli i programátorovi)
- **Zapouzdření** (skrývání informací – public – protected – private)
- **Typy nebo třídy** (předpis pro třídu)
- **Dědičnost** (dědění atributů a metod od jiné třídy – jednoduchá či vícenásobná)
- **Pozdní vazba** (úzce souvisí s polymorfismem – pozdní vazbou se označuje způsob volání polymorfních operací, kdy aplikace při volání operace dynamicky za chodu programu zvolí kód metody (vybere příslušnou implementaci) na základě třídy objektu, nad kterým je metoda volána)
- **Výpočtová úplnost**
- **Rozšiřitelnost**

Volitelné

- Vícenásobná dědičnost
- Typová kontrola
- Distribuovanost
- Verzování
- Dlouhotrvající transakce

Otevřené (výběr z alternativ)

- Programovací paradigma
- Rozšiřitelnost základních typů
- Typový systém (zapouzdřenost, genericita)
- Uniformita (implementace, program. Jazyk, rozhraní)

Základní OO datový model

1. Objekt a identita objektu
2. Atributy a metody
3. Zapouzdřenost a předávání zpráv
4. Třídy
5. Dědičnost a hierarchie tříd

Ad1, každá entita reálného světa je objektem, objekt je popsán – jménem, dobou trvání a OID (je jedinečný)

Relační model	OO model
Identifikace klíčem	Identifikace OID)*
Klíč je modifikovatelný	OID nelze měnit
Klíč je jedinečný v rámci relace	OID je jedinečný v databázi

)* OID generuje systém – ukazatel na objekt

10 Základní konstrukce jazyků ODL a OQL

Objektově orientované databáze potřebují stejně jako relační databáze prostředky pro definici dat a dotazování na data. Těmito prostředky jsou jazyky ODL (Object Definition Language) a OQL (Object Query Language).

ODL (Object Definition Language)

Účel jazyka: umožnit OO návrh databáze a překlad do OODBMS, jde o obdobu jazyka DDL pro objektově orientované databáze
ODL podporují programovací jazyky C++, Java, Smalltalk
Jednotlivé objekty jsou seskupovány do tříd, přičemž každý objekt má jedinečné OID. OID vytváří systém – v centralizovaném prostředí z času vytvoření objektu, v distribuovaném prostředí z času vytvoření objektu a z identifikace hostitele
Při popisu ODL tříd určujeme atributy, relace a metody (signatury). Metody jsou psány v hostitelském jazyce.

Deklarace objektových typů
interface – popis abstraktního chování – použitelné jen k dědění operací
interface <jméno třídy> {<seznam vlastností>}

class – k popisu abstraktního chování i abstraktního stavu (k dědění i vytváření objektů). Class slouží k popisu databázového schématu.
class <jméno třídy> {<seznam vlastností>}

Atributy
Jsou nejjednodušším typem vlastností
Př.

```
class Film {
      attribute string titul;
      attribute short rok;
      attribute short delka;
      attribute enum Efilm {barevny, cernobily} typfilmu;
      };
```

Př.

```
class Herec {
      attribute string jmeno;
      attribute short rok;
      attribute short delka;
      attribute Struct Adr {string ulice,string mesto} adresa;
      };
```

Relace v ODL
Prostředek pro spojování objektů

```
class Film {
      …
      relationship Set <Herec> herci;
        inverse Herec :: hrajev; /*vazba M-N Herec-Film */
      };

class Herec {
      …
```

```
relationship Set <Film>hrajev
  inverse Film :: herci
};
```

Objekty a literály
Literály (konstanty) – atomické, kolekce, strukturované
Objekty
- mají OID (a popř. i jméno)
- mohou mít komplexní strukturu danou konstruktorem

Atomické objekty se specifikují pomocí class. Většina objektů má komplexní strukturu s atributy, operacemi apod.

Konstruktory pro objekty typu kolekce:
a, množiny – Set <libovolný typ>
b, multimnožiny – Bag <libovolný typ>
c, seznamy – List <libovolný typ>
d, pole – Array <T, i>
e, slovník – Dictionary <k, v>
vytváří asociaci dvojic key-value
o.bind(k,v) – vytvoří asociaci k-v v objektu o
o.unbind(k,v) – zruší asociaci k-v v objektu o
v=o.lookup(k) – přiřadí s k asociovanou hodnotou v o

Konstruktor struktury má tvar Struct N {T1 F1, T2 F2, …, Tn Fn}

Podtřídy a dědičnost
```
class Muzikal extends Film {
…
}
```

Deklarace klíčů
OO model klíče nepotřebuje, umístění objektu se určí z OID
ODL umožňuje definovat klíče pro návaznost na ER model a relační databáze

Př.
```
class Zamestnanec (key (osobniCislo, cisloOP)) {
…
}
```

Deklarace signatur metod v ODL
Signatura určuje
- jméno metody sdružené s třídou
- vstupní / výstupní typy metody

Kód metody je zapsán v hostitelském jazyce, není součástí ODL
Syntax podobná funkcím C mimo
- specifikace parametrů in, out, inout
- fce může způsobit vyjímky – raises (seznam vyjímek)

Každá ODL třída může mít deklarován extent
extent = (rozsah) pojmenování současné množiny objektů této třídy
je obdobou jména relace

OQL dotazy se týkají extent, ne samotné třídy

Př.
```
class FILM (extent Filmy, key (osobniCislo, cisloOP)) {
…
}
```

OQL (Object Oriented Query Language)

OQL – přenos SQL do OO prostředí
SQL3 (alias SQL-99) – přenos OO do relačního prostředí

Základní konstrukce OQL

- OQL není úplným jazykem (obdoba SQL92)
- Je chápán jako rozšíření hostitelského jazyka (C++, Smalltalk, Java)
- Poskytuje deklarativní přístup k objektům
- Umožňuje práce s objekty (mají OID) i s literály (identické se svou hodnotou)
- Neobsahuje explicitně UPDATE

Tvorba objektů

a, objekt bez identity
Struct (a: 10, b: “Jan”)

- vytvoří objekt bez identity (konstantu) – strukturu se dvěma položkami

b, pomocí SELECT
c, pomocí konstruktorů – Set, Bag, List, Array, jména tříd

y = Bag (x, x, Struct (a:3, b:4));
x = Film (titul: ”X-Files”, rok: 1999, …);

Typový systém

Proměnné užívané v OQL jsou většinou deklarovány v hostitelském jazyce
Oproti ODL potřebuje OQL i konstanty
Základní typy (jako v SQL)

- atomické
- vyjmenované

složené typy

- Set (…)
- Bag (…)
- List (…)
- Array(…)
- Struct (…)

K odkazům na části objektů se používá tečková notace, synonymem je ->
Pokud je a objekt, pak a.p značí:

- Je-li p atribut ,pak hodnotu atributu
- Je-li p relace, pak objekt nebo kolekci objektů, které jsou v relaci p s objektem a
- Je-li p metoda, pak výsledek aplikace p na objekt a

Př.
mujFilm.delka, mujFilm.herci

SELECT v OQL

Jednoduchý dotaz
SELECT f.rok FROM Filmy f WHERE f.titul “X-Files”

Eliminace duplikátů
SELECT DISTINCT s.jmeno FROM Filmy f, f.herci h WHERE
f.vlastnene.jmeno=”Paramount”

Dále jde použít podobně jako u SQL : kvantifikátory, agregační operátory, množinové operátory, operátor ORDER BY, operátor GROUP BY, operátor HAVING, poddotazy

Kvantifikované výrazy FOR ALL x IN S: C(X) a EXISTS x IN S:C(X)
Př. SELECT h FROM Herci h WHERE EXISTS f IN h.hrajev :
m.vlastneny.jmeno=”Paramount”

Agregační funkce
AVG, SUM, COUNT, MIN, MAX

Množinové operátory
UNION, INTERSECT, EXCEPT

Přiřazování objektů v OQL
Dotazy OQL produkují objekty, které lze přiřadit proměnným hostitelského jazyka

Př. (pro neznalé – tohle je kód v C++)

```
stareFilmy = SELECT DISTINCT f FROM Filmy f WHERE f.rok <1945;
pocetFilmu = COUNT (Filmy);
for ( i = 0 ; I < pocetFilmu; i++)
{
  film = seznamFilmu[i];
  cout << film.titul << „\n“;
}
```

Vazba na C++
Knihovna tříd C++ poskytuje třídy a operace pro implementaci ODL konstrukcí
Způsob práce s objekty v C++ popisuje jazyk OML

11 Objektově relační databáze – objektové vlastnosti jazyka SQL99

Za posledních 25 let vývoje počítačů a programování aplikací můžeme vysledovat mohutný trend přechodu od strukturovaného k objektově orientovanému programování. Toto platí i v oblasti zpracování dat a databází. V osmdesátých létech způsobily revoluci relační databáze, v létech devadesátých s mohutným nástupem objektově orientovaného programování začaly vznikat i objektově orientované databáze, které si kladou za cíl ulehčit a zrychlit práci s daty. Ovšem situace není zdaleka tak jednoduchá, protože relační a objektový přístup je od základu rozdílný. Existuje jak mnoho výhod, tak i mnoho nevýhod pro relační i objektové databáze. Na současném databázovém trhu existují tři základní typy – relační databáze (Relational Database Management System, RDBMS), objektově-relační ("Object Relational" Database Management System, ORDBMS) a objektové (Object Database Management System, ODBMS).

RDBMS – Oracle 7.x, DB2
ORDBMS – Oracle 8.x, 9.x, 10.x
ODBMS – Jasmine, Gemstone. O2

Objektově relační databáze (ORDBMS)

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

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

Dotazovací jazyk
ORDBMS podporuje rozšířenou verzi SQL — SQL3 (SQL-99). Důvodem je podpora objektů (tj. dotazy obsahující atributy objektů). Typická rozšíření zahrnují dotazy obsahující vnořené objekty, atributy, abstraktní datové typy a použití metod. ORDBMS je stále relační, protože data jsou uložena v řádcích a sloupcích tabulek a SQL, včetně zmíněných rozšíření, pracuje právě s nimi.

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

Objektové vlastnosti SQL99 (SQL3)

Kompatibilita s existujícími jazyky
OID
Hnízděné tabulky
Uživatelem definované typy

- **Abstraktní datové typy (jsou typem atributu relace)**
- **Řádkové typy (jsou typem relace)**
- **Odlišující typy (musí být FINAL)**

Uživatelem definované typy mohou být organizovány do hierarchie s děděním.
Chování uživatelem definovaných typů je realizováno pomocí procedur, funkcí a (metod u ADT).
Objekty v SQL pracují s relacemi.

Řádkové typy

Vytvoření 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);

Příklady tvorby tabulky s řádkovými typy

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

Tvorba dotazů

```
SELECT FilmovyHerec.jmeno, FilmovyHerec.adresa.ulice
        FROM FilmovyHerec WHERE FilmovyHerec.adresa.mesto = ‘Plzeň’;
```

Abstraktní datové typy

Umožňují zapouzdření atributů a operací (na rozdíl od řádkových typů). Hodnoty jejich typů mohou být umístěny do sloupců tabulek.

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

```
CREATE METHOD mzda
        lze uvést Language program. jazyk
        FOR typZamestnanec
        BEGIN … END;
```

Instance ADT vznikají

1. Konstruktorem *jmenoTypu()*
2. Operátorem NEW jmeno hodnota,
   např. WHERE vedouci = NEW typZam(1012,'Pavel Novák',…)
3. Příkazem INSERT,
   např. INSERT INTO osoby VALUES(1012,'Pavel Novák',…);

Pro každý atribut jsou k dispozici funkce

- Implicitně či explicitně zavedené porovnání
- Zjištění hodnoty atributu z objektu *jmeno_atributu(jmeno_objektu)* – stejné i pro aplikaci metod, možná i tečková notace *jmeno_objektu.jmeno_atributu*

Funkce, procedury a metody

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

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

Volání procedury: `CALL zjisti_cenu(12134,z);`

Podtypy a podtabulky

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

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

Reference

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

Dereference

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

OID hodnota
Zpřístupnění klauzulemi REF IS SYSTEM GENERATED a REF IS USER GENERATED

Technická poznámka: k tomuto tématu je možné ještě doplnit praktickou implementaci SQL99 v systému Oracle8, ale to už je jen pro fanatiky ;)

12 Deduktivní databáze – základní vlastnosti, jazyk Datalog

- Deduktivní databáze, systémy řízení znalostní báze, expertní databázové systémy

Rozšiřují vyjadřovací sílu relačního jazyka a zachovávají neprocedurální styl vyjadřování.

Společné vlastnosti logického programování a databází
Logické programování
- Malé, jednouživatelské databáze
- Zpracování řetězcem dedukcí, výsledek dotazu yes/no

Databáze
- Velké, víceuživatelské databáze, persistentní data
- Efektivní přístup k datům na disku

Motivace pro použití deduktivních databází
- Dovolují přirozené vyjadřování rekurzivních pravidel
- Logická pravidla jsou vhodnou bází pro aplikace informačních systémů
- Dovolují redukovat rozsah tabulek relační databáze

Datalog je podmnožinou prologu
miluje (pijak, pivo).
prodava (hospoda, pivo, cena).
navstevuje (pijak, hospoda).

```
stastny(X) :- navstevuje (X, Hospoda),
              miluje (X, Pivo),
              prodava (Hospoda, Pivo, P).
```

hlava :- telo {predikát a argumenty}

Vyhodnocování: výsledkem jsou všechny úspěšně dosažené hodnoty

Pro bezpečnost (smysluplnost) vyhodnocení musí proměnná X, která se vyskytuje buď v hlavě pravidla, v negovaném podcíli nebo v porovnávacím predikátu, se také vyskytovat v normálním pozitivním podcíli těla.
Př. s(X) :- r(Z).
s(X) :- not r(X).
s(X) :- r(Z), Z>X.
Je-li pravidlo bezpečné, může být vyhodnoceno obdobně SQL pravidlu.

Datalogovský program = kolekce pravidel
Extenzionální databáze (EDB) = relace uložené v databázi. EDB predikáty se mohou vyskytovat pouze v těle pravidel
Intenzionální databáze (IDB) = relace definované pomocí pravidel. IDB predikáty se mohou vyskytovat v těle a v hlavě pravidel

Korespondence pojmů

Logické programování	Databáze
Predikát	Relace
Argumenty predikátu	Atributy relací
Fakt	Entice
Pravidlo	Pohled
Cíl	Dotaz
Cíl	omezení

Nevýhody prologu
- Vrací 1 hodnotu
- Procedurální zpracování
- Speciální predikáty
- Funkční symboly

Datalog
- Dotaz vytváří virtuální relace
- Rekurzí lze vyjádřit i to, co relačními prostředky nejde
- Pravidla datalogu jsou transformovatelná do rovnic relační algebry

Prolog vyhodnocujeme odshora dolů, datalog odspodu nahoru.

Nahražením „if“ v datalogovském programu symbolem „=“ získáme datalogovské rovnice.Řešení datalogovských rovnic pro danou EDB se nazývá pevným bodem (může existovat více řešení).
Nejmenším pevným bodem množiny rovnic je takové řešení, jemuž odpovídající relace jsou nejmenší vlastní podmnožinou všech relací řešení (neexistuje žádný další pevný bod, který je jeho vlastní podmnožinou).

Shrnutí

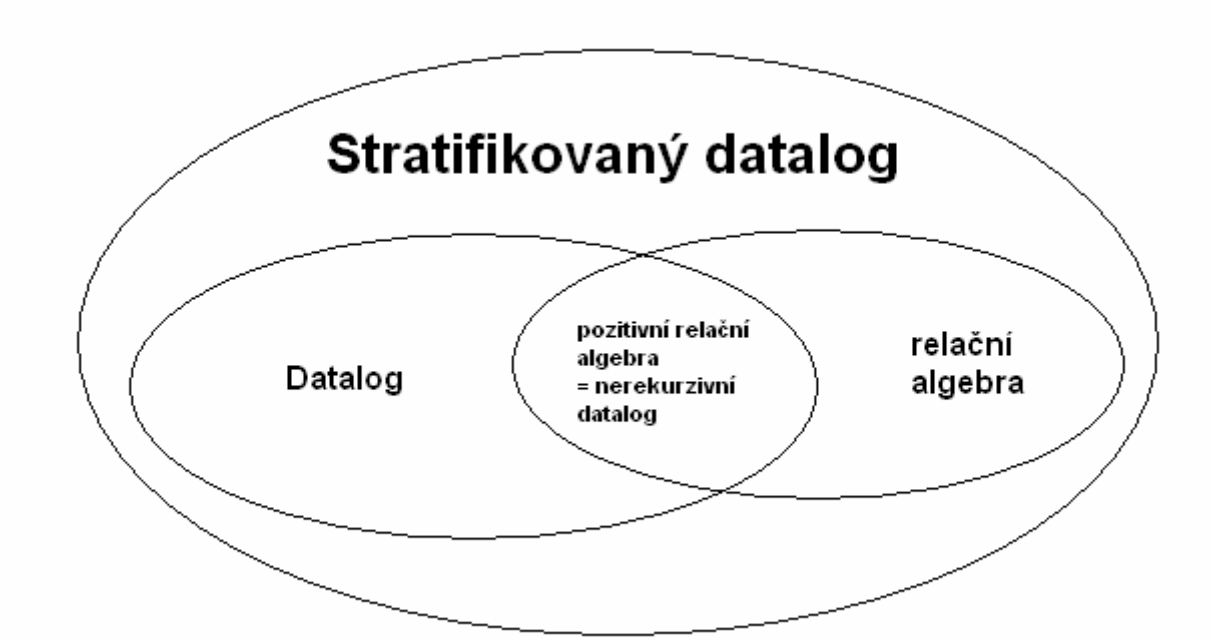
Deduktivní databáze jsou založeny na podpoře teorie dokazování a jsou schopné dedukce dodatečných faktů z extenzionální databáze. Mají zabudovány specifikované deduktivní axiomy a pravidla dedukce. Deduktivní axiomy společně s integritními omezeními jsou obvykle označovány jako intenzionální databáze. Deduktivní databáze se tedy skládá ze dvou složek, extenzionální databáze (EDB) a intenzionální databáze (IDB).

Vezmeme-li za základ relační datový model, deduktivní přístup rozšiřuje tento model (jak známo založený na jediném konstruktu, nazývaném databázová relace) tím, že využívá dvou typů relací. Základní relace jsou uloženy v databázi (EDB) a odpovídají relacím v relačním datovém modelu. Odvozené relace nemusí být uloženy v databázi. Jsou obvykle dočasné a uchovávají mezivýsledky operací. Deduktivní pravidla, která tyto relace odvozují, společně s integritními omezeními, jsou označována jako IDB. Jazyk SQL a obdobné jazyky založené na relační algebře a relačním kalkulu nemají dostatečnou vyjadřovací sílu a nejsou schopny vyjádřit uzávěr tranzitivních závislostí v relačním datovém modelu. Logická pravidla, využívající funkčních symbolů, mohou vyjádřit jakýkoliv výpočet tak, že může být zapsán v konvenčním programovacím jazyku. Dokonce logická pravidla bez funkčních symbolů (např. jazyk DATALOG) mají sílu vyjádřit výpočet větší než konvenční DQL (Data Query Language).

Deduktivní databáze se v komerční podobě prakticky nepoužívají. Jejich nasazení je maximálně na akademické půdě. Důvodem je stejně jako u OO databází fakt, že se příliš odlišují od relačních databází. Některé jejich vlastnosti jsou však přenášeny do relačních i OO databází a v tom je jejich smysl. Některé jejich vlastnosti jsou například využity v jazyce SQL-99(SQL-3).

13 Vztah Datalogu a relační algebry, problém stratifikace

Pokud datalog neobsahuje rekurze, je ho možné zapsat pomocí relační algebry.



Stratifikace
Deduktivní pravidla mohou na sobě navzájem záviset různým způsobem, buď hierarchicky nebo rekurzivně. Tato závislost může být vyjádřena tvrzením, že koncepty odvoditelné na vyšší úrovni mohou výt vyhodnoceny pouze tehdy, pokud koncepty na nižší úrovni, na nichž jsou tyto koncepty vyšší úrovně závislé, již byly dříve vyhodnoceny. Každá úroveň je označována „strata“ a celkové vytváření úrovní se nazývá stratifikace množiny pravidel. Tento proces musí být pečlivě kontrolován, pokud se v pravidlech objeví negace.

Negativní literály v deklarativních výrazech nemohou produkovat žádná nová tvrzení, ale mohou pouze potvrdit nebo vyvrátit tvrzení vytvořená v průběhu vyhodnocení jejich pozitivních výskytů se stejnými proměnnými. Před vyhodnocením negace je třeba znát všechna pozitivní fakta, ze kterých může být vyjádřen závěr ve formě negace. Máme-li např. definována tvrzení kdo je koho potomkem, pak pouze jejich plný výčet umožňuje potvrdit či vyvrátit tvrzení kdo koho potomkem není. Stratifikace nemůže být dosaženo, pokud existuje odvozený koncept, který závisí jak rekurzivně, tak negativně na sobě samém. Množina pravidel obsahující takovýto rekurzivní cyklus, který zahrnuje negace, se nazývá nestratifikovatelná.
Např.
poslanec(X) ← not podnikatel(X)
podnikatel(X) ← not poslanec(X)

Obě pravidla je třeba umístit na stejnou úroveň, protože jsou rekurzivně závislá. Zároveň by však měla být umístěna do různých úrovní, protože jeden na druhém negativně závisí. Taková situace je označována jako logický paradox a musí jí být zabráněno. Další paradoxní situace může nastat, pokud k odvození konceptu je třeba vypočítat agregační funkci, jejímž

elementem je odvozovaný koncept. Stratifikovaná množina pravidel je pak množina pravidel, která vylučuje negační a agregační paradox.

Jednoduše řečeno stratifikace určuje pořadí vyhodnocení predikátů. Pro určení stratifikace se používá rozšířený graf závislostí predikátů. Rozšířeným grafem závislostí je graf závislostí, ve kterém jsou symbolem „~“ označeny všechny hrany vedoucí z uzlu Q do uzlu P, kde P reprezentuje predikát, jehož některé pravidlo obsahuje v těle negovaný literál Q. Program je stratifikovaný, jestliže rozšířený graf závislostí neobsahuje žádné cykly s hranou označenou „~“.

Všeobecné principy dobře formovaných deduktivních pravidel

Pravidla musí splňovat několik syntaktických a sémantických požadavků, aby mohla být vyhodnocena. Tyto požadavky lze shrnout do následujících bodů:

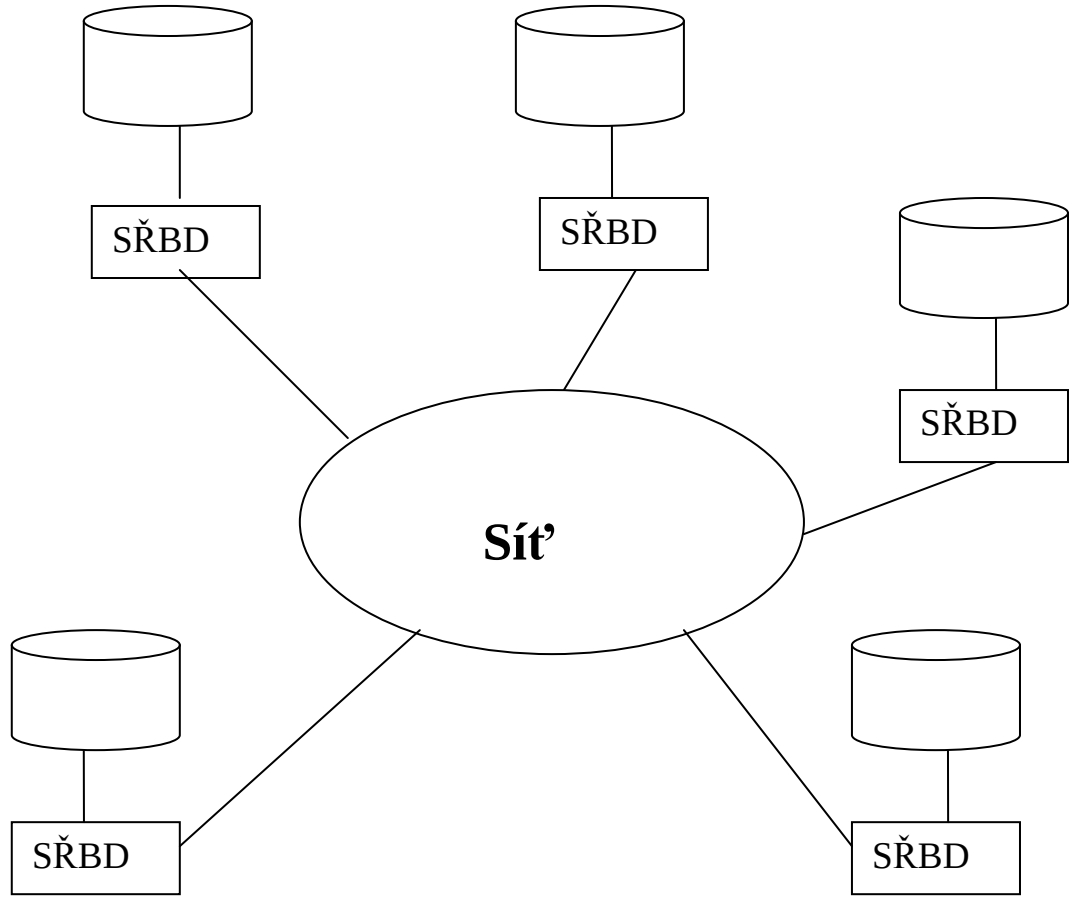
- Pravidla musí být v omezeném rozsahu, tzn. všechny jejich proměnné musí být omezeny vhodnými formulemi.
- Pravidla musí být bezpečná, tzn. všechny proměnné v hlavě pravidla se musí objevit minimálně v jedné pozitivní formuli v těle pravidla.
- Pravidla s negativními formulemi musí být stratifikována, tzn. nesmí být rekurzivně použita v těle jiných pravidel tak, aby vytvářela cyklus rekurzivních pravidel, která vyvolávají negativní formuli.
- Pravidla s termy musí být stratifikována, tzn. že nesmí být rekurzivně použity v těle ostatních pravidel tak, aby tvořila cyklus rekurzivních pravidel, které volají množinu termů.

14 Distribuované databáze – přednosti, nedostatky

Distribuované databáze spojují dohromady databázové systémy a počítačové sítě.
Distribuované databázové systémy poskytují integrovaná data bez centralizace.

Distribuovaný výpočetní systém má vlastnosti

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



Přednosti 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)

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

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

Požadavky DSŘBD formuloval Date:
1. lokální autonomie – každé místo má lokální SŘBD
2. vše je distribuované – ve všech službách se nespoléhá na žádné centrální místo
3. kontinuálnost – akce v jednom místě (např. odstranění tabulky) by neměla příliš narušovat provoz DDBS jako celku
4. nezávislost na místě – uživatel nemusí vědět, kde jsou uložena potřebná data
5. nezávislost na fragmentaci – nemusí vědět, kde jsou fragmenty
6. nezávislost na replikaci
7. možnost distribuovaného zpracování dotazu – nemělo by být nutné přesouvat data ke zpracování dotazu do jednoho místa
8. možnost distribuovaného zpracování transakcí – může dojít ke konfliktu s 1. Pro zajištění korektnosti se používá 2 fázový potvrzovací protokol
9. nezávislost na hardware
10. nezávislost na OS
11. nezávislost na síti
12. nezávislost na DBMS

15 Fragmentace a replikace dat

Fragmentace

- **Horizontální**
  - Dle selekční podmínky rozdělíme tabulku na dvě části horizontálním řezem, tedy např. na knihy s ISBN nižším než 122 a na knihy s ISBN větším nebo rovným 122 (viz příklad).
  - Zápis fragmentace : F_i = Selection $_{Pi}$ R
  - Zápis spojení do celé tabulky: R = $\cup F_i$
- **Odvozená horizontální**
  - Opět dochází k rozdělení tabulky na dvě a více částí horizontálním řezem, v tomto případě však je fragmentace založena na jiné relaci.
  - Máme například tabulky DODAVATELE a KNIHY. Základ fragmentace provedeme v tabulce DODAVATELE, kdy rozdělíme jednotlivé dodavatele do fragmentů. Na základě těchto fragmentů provedeme fragmentaci v tabulce KNIHY, která je spojena relací s tabulkou DODAVATELE (viz příklad).
  - Zápis fragmentace - polospojení = $R<t_1 \theta t_2] S = (R [t_1 \theta t_2] S) [Atr(R)]$
- **Vertikální**
  - Vertikální fragmentace spočívá v rozdělení tabulky podle sloupců na dvě a více částí. V jedné skupině jsou jedny sloupce, v druhé jiné.
  - Zápis fragmentace: $F_i = R(A_i)$
  - Zápis spojení do celé tabulky: R = F_1 join F_2 … join F_n
- **Smíšená**
  - Tabulku například rozdělíme dle sloupců (vertikální fragmentace) a následně v jednotlivých částech provedeme horizontální fragmentaci.

Horizontální fragmentace

F2000

ISBN	INV_C	CENA	R_VYD	VYDAVATEL
1590	120	590	2000	KOP
1910	121	720	2000	Springer

F2002

ISBN	INV_C	CENA	R_VYD	VYDAVATEL
0235	122	900	2002	AP
0235	123	900	2002	AP
1488	124	899	2002	Alfa

Odvozená horizontální fragmentace
relace KNIHY
relace VYDAVATELÉ(VYDAVATEL, ZEMĚ, OBRAT$)

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

K1 = KNIHY Semijoin (VYDAVATEL=VYDAVATEL) VYDAVATELÉ1

K2 = KNIHY Semijoin (VYDAVATEL=VYDAVATEL) VYDAVATELÉ2

např.
VYDAVATELÉ:

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

VYDAVATELÉ1:

AP	USA	200
BC	NL	120
Springer	D	500

VYDAVATELÉ2:

Alfa	SK	20
KOP	CZ	15

K1:

ISBN	INV_C	CENA	R_VYD	VYDAVATEL
1910	121	720	2000	Springer
0235	122	900	2002	AP
0235	123	900	2002	AP

K2:

ISBN	INV_C	CENA	R_VYD	VYDAVATEL
1590	120	590	2000	KOP
1488	124	899	2002	Alfa

Vertikální fragmentace relace KNIHY

předp. funkční závislosti INV_C → { ISBN, CENA }
ISBN → {R_VYD, VYDAVATEL }

bezztrátová vertikální fragmentace je:

FR1 (ISBN, R_VYD, VYDAVATEL)

ISBN	R_VYD	VYDAVATEL
1590	2000	KOP
1910	2000	Springer
0235	2002	AP
1488	2002	Alfa

FR2 (INV_C, ISBN, CENA)

INV_C	ISBN	CENA
120	1590	590
121	1910	720
122	0235	900
123	0235	900
124	1488	899

Smíšená fragmentace

FR1_2000

ISBN	R_VYD	VYDAVATEL
1590	2000	KOP
1910	2000	Springer

\+ FR2

FR1_2002

ISBN	R_VYD	VYDAVA
0235	2002	AP
1488	2002	Alfa

Replikace dat

Při replikaci dat se nachází kopie množiny objektů v každém uzlu, ve kterém je využívána. V systému tedy existuje několik kopií každého objektu. Výhodou tohoto řešení je kvalitní dostupnost, rychlý přístup ke každému objektu a menší nároky na komunikaci mezi uzly. Nevýhodou je problém duplicity, díky níž je nutné trvale zajišťovat konzistenci všech kopií.

Zajišťování konzistence je klíčovým problémem replikace dat. Je nutné implementovat systém, který určí správné pořadí provedených operací a při replikaci rozdistribuje správnou kopii. Také je nutné zabránit současné modifikaci dvou kopií objektu. Správné pořadí provedených operací je určováno většinou časovými razítky, současné modifikaci dvou kopií jednoho objektu mohou zabránit klasické paralelní synchronizační prostředky (zámky, semafory, apod.). U transakčních systémů je možné razítky označovat jednotlivé transakce, výhodou je jednodušší a spolehlivější zajištění konzistence, nevýhodou pak nutnost archivovat všechny transakce až do doby další synchronizace. Jinak je možné označovat celé objekty, což má ale za následek zvýšení rizika poškození konzistence dat. Další variantou může být označování větších resp. menších celků, tj. např. skupiny objektů resp. jednotlivé atributy. V některých případech může jít o vhodnou variantu, to je však nutné pečlivě zvážit.

Doposud jsme předpokládali asynchronní replikaci, tj. že jsou operace pro zajištění konzistence prováděny nezávisle na modifikaci replikovaných objektů. Spolehlivější může být provádění operací pro zajištění konzistence bezprostředně před modifikací objetu nebo po ní. V prvním případě je před modifikací objektu nalezena jeho aktuální kopie, která je použita během modifikace. Tato metoda je spolehlivější než ostatní, problémem je však hledání aktuální kopie. V druhém případě je po každé modifikaci objekt zkopírován do všech uzlů, tj. každý uzel má vždy aktuální kopii každého objektu. Tato varianta je vhodná pro případ, kdy jsou objekty mnohem častěji pouze čteny, než modifikovány a kdy je nutné zajistit aktuálnost kopie objektu při každém čtení. Obecně však hrozí riziko přetížení komunikační sítě příliš častou výměnou kopií objektů.

Ani v těchto případech se však nezbavíme nutnosti vyloučit současnou změnu dvou kopií replikovaného objektu. To muže být zejména v geograficky rozsáhlejších systémech problém, který je nutné řešit dalšími prostředky. Mezi ně patří např. systémy přístupových práv, priorit či distribuce objektů pouze pro čtení.

16 Taxonomie distribuovaných databází

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

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

2. distribuce dat
3. heterogenity systémů

Architektura distribuovaných databází (UNOFFICIAL)
Distribuované databáze jsou vystavěny na následujících třech principech

1. autonomie jednotlivých částí vytvářejících dohromady systém
   integrované – jednotlivé části systému/stroje jsou úzce propojené a nemohou pracovat nezávisle
   částečně(polo) autonomní – jednotlivé části/stroje mohou pracovat nezávisle, ale jejich spolupráce požaduje určité změny, které musí být provedeny, pokud by měly pracovat samostatně
   plně autonomní – jednotlivé části/stroje pracují v naprosté izolaci nevěda nic o ostatních částech a neovlivňovány ostatními částmi
2. distribuce dat celým systémem
   bez distribuce dat – všechna data jsou uložena na jednom místě/stroji
   částečná distribuce dat – data jsou distribuována přes některá místa, ale ne přes všechna, existují rozdíly mezi jednotlivými místy, některá jsou servery, jiná klienty
   plná distribuce dat – neexistuje žádný rozdíl mezi jednotlivými místy, alespoň ne ve smyslu šíření dat
3. heterogenita hardwaru, operačního systému, datového modelu a/nebo DBMS jednotlivých částí
   homogenní – všechna místa/stroje jsou identická co se týče hardwaru a softwaru vyjma nepodstatných detailů jako je například velikost harddisku
   heterogenní – přinejmenším dvě místa se liší podstatnými rozdíly (např. rozdílnými DBMS (SQL Server 2005 x Oracle 10g) či datovým modelem (relační x objektově orientovaný))

Zajímavé příklady architektur

- integrované + částečná distribuce dat + homogenní = klient/server, s mnoha servery
- poloautonomní + plná distribuce dat + homogenní = P2P distribuovaná databáze
- plně autonomní + plná distribuce dat + heterogenní = federační (multi-)databáze

Třívrstvá architektura dle ANSI/SPARC
Architektura centralizovaného databázového systému může být zobrazena pomocí 3-úrovňového schématu definovaného sdružením ANSI/SPARC roku 1977.

Jednotlivé části architektury:

- Externí (uživatelské) schéma
- Konceptuální schéma
- Interní (fyzické) schéma

Interní schéma popisuje soubory a jejich strukturu (např. sekvenční indexování, hash, řazení) a způsob fyzického uložení dat. Konceptuální schéma může být chápáno jako celkový datový model – ER diagram či OO – který konceptuálně popisuje celou strukturu databáze. Pro relační databáze zde mohou být též popsány definice tabulek a referenční integrity, ačkoliv to není úplně vpořádku, protože konceptuální schéma je zamýšleno tak, aby bylo implementačně nezávislé. Externí (uživatelské) schéma jsou formuláře (okna), které uživateli umožňují pracovat s databází včetně filtrace jednotlivých dat pro daného uživatele (jeho roli).

17 Data Warehousing a OLAP

Databázový systém – OLTP (Online Transaction Processing Systems)

- Zákaznicky orientovaný
- Současná data
- ER schéma
- Atomické transakce
- Velikost DB - až GB

DataWarehouse – OLAP (Online analytical Processing)

- Orientovaný na trh
- Historická data
- Agregovaná data (nenormalizovaná=redundantní)
- Schéma hvězdy či vločky
- Read only
- DB až TB

Datawarehouse:

DW je architektura založená na relačním SŘBD, která se používá pro údržbu historických dat získaných z databází operativních dat, která byla sjednocena a zkontrolována před jejich použitím v databázi DW.

DW je strukturované rozšiřitelné prostředí navržené pro analýzu neměnících se dat, která byla logicky transformována z více zdrojových aplikací tak, aby byla uvedena do souladu se strukturou podniku. Data z DW jsou aktualizována v delších časových intervalech, jsou vyjádřena v jednoduchých uživatelských pojmech a jsou sumarizována pro rychlou analýzu.

Datová tržiště (Data Mart)
k řešení části podniku

Stadia zpracování dat: OLTP → DW → OLAP

Použití DW:

- Prezentace dat
- Testování hypotéz
- Objevování nových informací

Předchůdcem DW jsou Decision Support Systems (podpora rozhodování)
=Nadstavba nad operativními daty podniku

Architektury DW

Data lze uspořádávat:

- Klasicky pomocí speciálně navržené relační databáze
- Ve vícerozměrném datovém modelu (zcela odlišné od relačního modelu)

Vícerozměrný datový model implementuje data vícerozměrnými poli (vícerozměrný spreadsheet). Dimenze odpovídají dimenzím podnikání organizace.

MOLAP - Multidimenzionální OLAP
Datová krychle (obsahuje fakta)
Hierarchické dimenze (částečné či totální uspořádání)
- hvězdové schéma
- vločkové schéma

ROLAP –Relační OLAP
Na relační architektuře založený model DW strukturou propojených DB tabulek - Relační OLAP (ROLAP) – pomalejší zpracování než MOLAP

Tříúrovňová architektura DW
- Klient
- OLAP server (MOLAP/ROLAP server)
- databázový server DW

Datawarehouse (UNOFFICIAL)
Datový sklad je podnikový strukturovaný depozitář předmětově orientovaných , vzájemně provázaných, časově neměnných, historických dat používaný na získávání informací a podporu rozhodování. Bill Inmon

Datový sklad je centrální úložiště různorodých dat firmy
- Řešení integračních efektů
- Jednotné místo uložení dat

Určeno pro analytickou podporu rozhodování (drill-up, drill-down)
Zahrnuje nejen data v databázi, ale i nástroje pro extrakci dat, nástroje pro reporting, analýzu dat, data mining, …
Prezentace dat uživatelsky příjemným způsobem zaměřeným na střední a vyšší management

OLAP

OLAP (**Online Analytical Processing**) je technologie zpracování databáze na serveru, který umožňuje uspořádat velké objemy dat tak, aby byla data přístupná a srozumitelná uživatelům zabývajícím se analýzou obchodních trendů a výsledků. Tyto databáze uspořádávají kategorie dat do skupin polí (nazývaných také dimenze) a úrovní podrobností. Ty mohou být i několikadimenzionální.

Typické OLAP operace na multidimenzionálních datech
Roll up – srolování – spojení řádek/sloupců do jednoho, např. Praha, Plzeň -> Čechy
Drill down – zavrtání – opak roll up, např. 1.čtvrletí -> leden, únor, březen
Dice – výřez – výřez 3D krychle z původní krychle
Slice – řez – 2D řez původní krychlí

Architektura OLAP serverů
1. Specializovaný SQL server (má rozšířený SQL pro dotazy nad hvězdovým/vločkovým schématem)

2. ROLAP server (užívá relační nebo rozšířený relační DBMS, např server METACUBE Informix, pracuje s relačními tabulkami uspořádanými do hvězdy/vločky, adresuje pomocí klíče, data jsou neagregovaná)

3. MOLAP server (přímo zobrazuje multidimenzionální pohledy na struktury datové krychle –vícerozměrného pole, dovoluje rychlý přístup – přímé adresování pole indexem, data v krychli bývají včetně agregovaných v kuboidech = podkrychlích, horší využívání paměti – řídká krychle)
4. HOLAP server (kombinace 2 a 3, zdrojová data jsou v relačních databázích, výsledky výpočtů a agregací jsou v multidimenzionální databázi)

Efektivní zpracování OLAP dotazů

1) Určit, které dotazy se budou realizovat na materializovaných kuboidech
2) Transformovat slicing/dicing (selekce, projekce), roll-up (group by), drill-down do SQL nebo OLAP operací.
3) Vybrat vhodné kuboidy pro operace tak, aby cena operace byla minimální

18 Získávání asociačních pravidel z transakčních databází

Typickým příkladem dolování asociačních pravidel je tzv. analýza nákupního košíku (market basket analysis). Tento proces analyzuje zvyky nakupujících a hledá závislosti mezi různým zbožím, které si zákazníci vloží do svého nákupního košíku. Znalost informace o tom, které výrobky zákazníci obvykle kupují současně, mohou v praxi pomoci v tvorbě katalogů, pomáhají při definování strategie rozmístění zboží v prodejně atd.

Zkoumaná data pochází z transakční databáze, tj. z databáze, která zaznamenává všechny (obchodní) transakce, např. podrobnosti o každém nákupu. Přídavné jméno transakční zde v žádném případě neznamená pojem transakce, jak jej známe z databázových technologií (skupina databázových operací, které musí být provedeny všechny (nebo žádná), aby nedošlo k porušení konzistence databáze).

Analyzovaná data sestávají z řady booleovských atributů (tj. mohou nabývat jen hodnot 0 nebo 1) stejné dimenze. Typickým příkladem může být dimenze koupil s atributy odpovídajícími jednotlivým položkám zboží. Atribut bude mít v záznamu provedeného nákupu hodnotu 1, pokud si zákazník dané zboží koupil, a 0, pokud ne.

Výsledkem analýzy nákupního košíku jsou asociační pravidla obecně tvaru:
$i1 \wedge i2 \wedge \ldots \wedge i_{n-1} \Rightarrow i_n$ [podpora = p; spolehlivost = s]
s následujícím významem: jestliže si nějaký zákazník koupí zároveň zboží $i_1 \ldots i_{n-1}$, pak si koupí i zboží i_n s podporou p a spolehlivostí s.

Příkladem konkrétního objeveného vztahu může být např.:
Tequila => citróny [podpora = 1 %; spolehlivost = 80 %]

Symbolům i_n v asociačním pravidle se ve shodě s pojmy matematické logiky říká predikáty. Později si ukážeme, že se dá vytěžovat asociace s predikáty i trochu jiného (obecnějšího) tvaru.

Zmiňovaná podpora (support) a spolehlivost (confidence) jsou základní mírou relevance (zajímavosti) asociačního pravidla.

Podpora představuje pravděpodobnost výskytu záznamu, který splňuje všechny predikáty $i_1 \ldots i_n$, v datech. Jinými slovy je to podíl počtu záznamù, obsahujících všechny druhy zboží určené predikáty $i_1 \ldots i_n$, a všech záznamù.

$$s(X \Rightarrow Y) = \frac{\text{Počet transakcí obsahujících } X \text{ a } Y}{\text{Celkový počet transakcí}} = P(X \text{ a } Y)$$

Spolehlivost určuje tzv. podmíněnou pravděpodobnost jevu i_n v datech za podmínky, že platí $i_1 \ldots i_{n-1}$. Je to tedy poměr počtu záznamů o koupi zboží $i_1 \ldots i_n$ vůči počtu záznamů obsahujících pouze $i_1 \ldots i_{n-1}$.

$$c(X \Rightarrow Y) = \frac{\text{Počet transakcí obsahujících } X \text{ a } Y}{\text{Počet transakcí obsahujících } X} = P(Y \mid X)$$

Klasifikace asociačních pravidel:

- Dle dimenze (jednorozměrná / vícerozměrná)
  Položky v koši / položky v koši + čas nákupu + typ zákazníka
- Dle typů hodnot (booleovská / kvantitativní)
  Existence položky v koši / číselný údaj o položkách v koši
- Dle úrovně abstrakce (jednoduchá úroveň / násobná úroveň)
  Nehierarchické členění položek / hierarchické členění

Asociace, které překročí uživatelem nastavené prahy pro podporu i spolehlivost, jsou považovány za zajímavé a jsou zahrnuty do zprávy o výsledcích vytěžování.

Algoritmy hledání asociací z transakčních databází

Na řádky datové matice tvořené jen booleovskými hodnotami se můžeme dívat také jako na množiny položek, pro které je na řádcích v příslušných sloupcích hodnota 1. Předpokládejme, že naše datová matice má m řádků a n sloupců s booleovskými hodnotami. Můžeme zavést množinu všech položek $I = \{i_1, i_2, \ldots i_n\}$ a jednotlivé transakce T_k (pro k = 1… m) chápat jako podmnožiny $T_k \subseteq I$. Hledaná asociační pravidla jsou potom tvaru A => B, kde $A \subset I$, $B \subset I$ a $A \cap B = 0$. Podpora p je pravděpodobnost výskytu množiny položek $A \cup B$ (tedy všech položek obsažených v A i B) v množině transakcí (datové tabulce). Spolehlivost s je pravděpodobnost výskytu B v transakcích, které obsahují A. Tedy: p = podpora(A => B) = $P(A \cup B)$; s = spolehlivost(A => B) = P(B|A). Množina položek M se nazývá frekventovaná, jestliže dosahuje minimální stanovené podpory p_p, tedy $P(M) \geq p_p$. Asociační pravidlo A => B se nazývá silné, jestliže množina položek $A \cup B$ je frekventovaná, ($P(A \cup B) \geq p_p$) a spolehlivost pravidla dosahuje minimální stanovené hodnoty s_p ($P(B|A) \geq sp$). Je tedy jasné, že výsledkem algoritmu hledání asociací by měl být seznam všech silných asociačních pravidel.

Generování silných pravidel z častých množin položek

- Pro každou častou množinu položek l generuj všechny neprázdné podmnožiny l
- Pro každou neprázdnou podmnožinu $m \subset l$ vytvoř pravidlo
  m =>(l – m), platí-li $p(l) / p(m) \geq \underline{c}$
- Za silná prohlas ta pravidla, jejichž věrohodnost (confidence) překračuje prahovou hodnotu

Př.
Nechť transakční databáze má tvar:

TID	seznam položek
T1	I1, I2, I5
T2	I2, I3, I4
T3	I3, I4
T4	I1, I2, I3, I4

Předpokládejme, že mezi časté množiny položek patří { I2, I3, I4}
Její neprázdné podmnožiny jsou:
{I2,I3}, {I2,I4}, {I3,I4}, {I2}, {I3}, {I4}

Generovaná pravidla z { I2, I3, I4}:
I2 ^ I3 => I4 c = 100%

I2 ^ I4 => I3 c = 100%
I3 ^ I4 => I2 c = 66%
I2 => I3 ^ I4 c = 66%
I3 => I2 ^ I4 c = 66%
I4 => I2 ^ I3 c = 66%

Pokud c bude 80% pak silná jsou prvá dvě.

Vyhledání častých množin položek (algoritmus apriori)
- najde množinu L_1 častých 1-množin položek
- pomocí L_1 najde množinu L_2 častých 2-množin položek
- pomocí L_2 najde množinu L_3 častých 3-množin položek
  …
-až žádné další k-množiny položek nelze najít

Apriori vlastnost: Je-li $p(I) < konst$, pak $p(I \cup J) < konst$

Postup:
1-Prohlédnutím D zjistíme množinu C_1 kandidátních 1-položek
2-pro zadané s určíme z C_1 množinu L_1
3-z L_{k-1} generujeme joinem C_k množinu kandidátních k-množin položek
4-z C_k odstraníme ty členy, které obsahují (k-1) podmnožinu, která není častá. Zbylé členy porovnáme s databází. L_k tvoří ty, co splňují s .
5-opakujeme od 3 pokud pro zadané s vznikají nové L_k

pozn. k 3:
L_{k-1} dva členy považujeme za spojitelné, pokud mají (k-2) společných položek.

$$L_k \text{ join } L_k = \{ A \text{ join } B \text{, kde } A, B \in L_k, |A \cap B| = k - 1 \}$$

Variace apriori algoritmu:
Hash associate
Redukce počtu průchodů databází
Redukce transakcí
Partition associate
Sample associate

Př. pokračování předcházejícího příkladu – vyhledání častých množin položek pro s = 50% využitím apriori algoritmu

C1:

množ. pol.	p = s
{ I1 }	2
{ I2 }	3
{ I3 }	3
{ I4 }	3
{ I5 }	1

L1:

množ. pol.	p = s
{ I1 }	2
{ I2 }	3
{ I3 }	3
{ I4 }	3

C2 = L1 join L1

C2:

množ. pol.	p = s
{ I1, I2 }	2
{ I1, I3 }	1
{ I1, I4 }	1
{ I2, I3 }	2
{ I2, I4}	2
{ I3, I4}	3

L2:

množ. pol.	p = s
{ I1, I2 }	2
{ I2, I3 }	2
{ I2, I4 }	2
{ I3, I4 }	3

C3 = L2 join L2
= { { I1, I2, I3}, {I1, I2, I4}, {I2, I3, I4} }
apriori vlastnost splňuje pouze třetí množina položek

C3:

množ. pol.	p = s
{ I2, I3, I4 }	2

L3:

množ. pol.	p = s
{ I2, I3, I4 }	2

Poznámka:
Algoritmy generování silných pravidel z častých množin položek a apriori metoda pravděpodobně nebudou vyžadovány. Ale pro jistotu jsem je zde uvedl, aby o nich člověk alespoň věděl.

19 Metody dolování znalostí

- Rozhodovací stromy a pravidla
- Asociační analýza
- Induktivní logické programování
- Nelineární regrese
- Bayesovské metody
- Neuronové sítě
- Metody založené na příkladech

UNOFFICIAL-1

Lineární regrese: Klasický statistický model, který předpokládá, že spojité závislé proměnné jsou lineární kombinací prediktorů

Logistická regrese: Model pro kategorizované výstupy, který předpokládá, že šance kategorií závislé proměnné lineárně závisí na hodnotách prediktorů.

Diskriminační analýza: Statistický predikční model, pro separaci kategorií výstupní proměnné ve vícerozměrném prostoru pomocí oddělovacích ploch.

Seskupovací (klastrová) analýza: Model, který vytváří skupiny případů tak, aby případy spadající do jedné skupiny si byly co nejvíce podobné, zatímco případy z různých skupin se co nejvíce odlišují.

Faktorová analýza: Postup pro redukci dimezionality dat. Metoda se snaží nahradit velký počet proměnných menším počtem tzv. faktorů takovým způsobem, aby ztracená informace byla minimální.

Asociační pravidla: Nacházení typických vzorů v datech, tj. určení které kombinace atributů se spolu vyskytují nejčastěji

Indukce logických podmínek: Nalezení nezávislých implikačních podmínek typu "když je splněno ... pak ..." pro předpověď hodnot výstupní proměnné.

Rozhodovací stromy: Hierarchický systém pravidel pro klasifikaci a vysvětlení variability cílové vlastnosti.

Neuronové sítě: Zjednodušený model nervové soustavy. Skládá se z vrstev neuronů, které si pomocí synapsí vyměňují informace. Model se učí tak, že vhodně nastavuje citlivost na podměty přicházející po synapsích.

Metoda nejbližších sousedů

Genetické algoritmy: algoritmy, které řídí proces učení modelu. Zpravidla hledají nejvhodnější strukturu modelu pro konkrétní data. Vychází ze vzoru biologické evoluce, kdy následující generace modelu je dokonalejší než jeho rodič. Například ve spojení s neuronovými sítěmi mohou genetické algoritmy vybrat nejvhodnější metodu učení nebo architekturu sítě.

Bayesovské metody

Bayesovská síť

- Orientovaný acyklický graf
- Uzly jsou náhodné proměnné
- Hrany reprezentují vztahy závislosti mezi nimi
- Je definována svojí topologií
- Dva uzly jsou spojeny jen pokud jeden přimo ovlivňuje čizpůsobuje druhý
- Orientace grafu indikuje směr působení mezi proměnnými (Markovova vlastnost)

UNOFFICIAL-2

- **Lineární regrese**: Klasický statistický model, který předpokládá, že spojité závislé proměnné jsou lineární kombinací prediktorů.
- **Logistická regrese**: Model pro kategorizované výstupy, který předpokládá, že šance kategorií závislé proměnné lineárně závisí na hodnotách prediktorů.
- **Diskriminační analýza**: Statistiský predikční model, pro separaci kategorií výstupní proměnné ve vícerozměrném prostoru pomocí oddělovacích ploch.
- **Seskupovací (klastrová) analýza**: Model, který vytváří skupiny případů tak, aby případy spadající do jedné skupiny si byly co nejvíce podobné, zatímco případy z různých skupin se co nejvíce odlišují.
- **Faktorová analýza**: Postup pro redukci dimezionality dat. Metoda se snaží nahradit velký počet proměnných menším počtem tzv. faktorů takovým způsobem, aby ztracená informace byla minimální.
- **Asociační pravidla**: Nacházení typických vzorů v datech, tj. určení které kombinace atributů se spolu vyskytují nejčastěji. Vhodné především pro anlýzy nákupních košíků nebo web mining.
- **Indukce logických podmínek**: Nalezení nezávislých imlikačních podmínek typu "když je splněno ... pak ..." pro předpověď hodnot výstupní proměnné.
- **Rozhodovací stromy**: Hierarchický systém pravidel pro klasifikaci a vysvětlení variability cílové vlastnosti. o hodnotě výstupní proměnné se rozhoduje podobným způsobem jako při určování rostlin podle botanického klíče.
- **Neuronové sítě**: Zjednodušný model nervové soustavy. Skládá se z vrstev neuronů, které si pomocí synapsí vyměňují informace. Model se učí tak, že vhodně nastavuje cilivost na podměty přicházející po synapsích.
- **Metoda nejbližších sousedů**: Model komparuje zpracovávaný případ se známými případy, které se novému nejvíce podobají ve vstupních proměnných.

Sdružování modelů

Vlasnosti modelu závisí jednak na použité metodě, ale též na datech, na kterých se model učí. Různá tréninková data mohou indukovat modely dávající různé výsledky. V praxi se osvědčilo kombinování modelů jednak vytvořených na základě různých algoritmů a jednak naučených na různých datech. Pokud se výsledky několika takových modelů odlišují, modely pak mezi sebou "hlasují" o nejlepší predikci. Situaci bychom mohli přirovnat k týmovému rozhodování odborníků.

Jinou možností je sériové spojení modelů. Výstup (predikce) jednoho modelu se stává vstupem do dalšího modelu atd. Následující model se buď může přiklonit k predikci svého

předchůdce nebo ji změnit. Toto uspořádání lze přirovnat k nadřízenému, který má neomezené rozhodovací právo, ale zaměstnává poradce, kteří mají své poradce atd.

Interpretace výsledků, které pochází ze společného uspořádání různých modelů je většinou ztracena, ale výrazně lze tímto způsobem zlepšit kvalitu předpovědí.

Genetické algoritmy

Genetické algoritmy jsou alogoritmy, které řídí proces učení modelu. Zpravidla hledají nejvhodnější strukturu modelu pro konkrétní data. Vychází ze vzoru biologické evoluce, kdy následující generace modelu je dokonalejší než jeho rodič. Například ve spojení s neuronovými sítěmi mohou genetické algoritmy vybrat nejvhodnější metodu učení nebo architekturu sítě.

20 Textová databáze a dokumentografické informační systémy, hypertextové systémy

Textová databáze - Systematicky organizovaná digitální reprezentace dat spolu s pomocnými datovými strukturami

Související pojmy
- dokumentografické informační systémy
- digitální knihovny
- rešeržní systémy
- hypertextové systémy

Efektivita vyhledávání
VR vybrané relevantní dokumenty
VN vybrané nerelevantní dokumenty
NR nevybrané relevantní dokumenty
NN nevybrané nerelevantní dokumenty

Úplnost R = VR / (VR + NR)
Přesnost P = VR / (VR + VN)

Dokument = (deskriptor, vlastní dokument)
Term = slovo (nebo fráze)

Datové struktury
- **sekvenční soubor**
- **invertovaný soubor**
- **signaturový soubor**

Invertovaný soubor
abecedně setříděný seznam termů (tzv. index = obdoba indexů v relační databázi), každý term má přiřazeny ukazatele na dokumenty obsahující tento term. Deskriptorem dokumentu je seznam jeho termů.

zdokonalování invertovaného souboru:
TF term frequency
DF document frequency
IDF inverse document frequency

Použití inv. souboru pro zodpovězení konjunktivního dotazu
-dokumenty s prostým výskytem termu
-uspořádání výsledku dle vah

$$\sum_{term \in Querry} (TF_{term} \times IDF_{term})$$

Konstrukce invertovaného souboru:

1. Rozpoznání významových slov textu, jejich uložení (spolu s ukazatelem na dokument do pomocného souboru
2. Setřídění pomocného souboru
3. Odstranění duplicit (příp. výpočet frekvencí) a vytvoření invertovaného souboru

Vytváření negativního slovníku (stop-slova, nevýznamová slova)

Lze využít Zipfův zákon :
f * r = konstanta
f - frekvence výskytu slova v textech
r - pořadí slova v setříděné tabulce výskytů
konstanta - je-li N počet všech výskytů slov v textu, pak ~ N/10

Signaturový soubor
zakódování dokumentu do 0/1
Signatura S je d bitový řetězec přiřazený dokumentu resp. dotazu

$$S_Q \text{ AND } S_D = S_Q$$

pak D je částí odpovědi

Vytváření S:

1. řetězením binárních reprezentací termů dokumentu nebo
2. vrstvením binárních reprezentací termů dokumentu nebo fcí OR

Příklad (pro bod 2)

dokument:	výpočetní	100 000 101 010
	technika	001 010 100 000
	---	---
	signatura	101 010 101 010

dotazy Q1: technika 001 010 100 000 shoda

Q2: dobytek 000 110 011 001 neshody (odfiltruje)

Q3: virus 001 000 101 000 chybný výběr

lze eliminovat chybný výběr dodatečným textovým porovnáním

Q4: výpočetní AND technika 101 010 101 010 shoda

Techniky tvorby signatur:

- triviální (napevno přiřazená pozice bitu pro term)
- rozptylování - umístění 1 do pozice hash(term)

Modely textových databází

- **Booleovský model**
- **Rozšířený booleovský model**
- **Vektorový model**

Booleovský model

- dokument reprezentován množinou termů,
- dotaz vyhodnocuje Booleovský výraz

Tvary dotazů

- s Booleovskými operátory,
- s proximitivními operátory
- s metasymboly
- v přirozeném jazyce

V základní podobě používají index v podobě lineárního seznamu termů.
V dokonalejší podobě - složitější indexy (tezaury)

Textový výraz může obsahovat specifikaci

- metasymboly
- zúžení pojmu
- rozšíření pojmu
- synonyma
- příbuzné termy

Nedostatky Booleovského modelu

- nepřesné výsledky
- nedovoluje rozlišit důležitost termů dotazu
- nelze řídit velikost výstupu dotazu
- nelze automaticky modifikovat dotaz na základě odpovědi
- nedokonalá formulovatelnost dotazu

Zdokonalení - **Rozšířený Bool. model** - zavádí váhy termů

Vektorový model

- dokumenty i dotazy jsou reprezentovány vektory

Př.	D1:	text slovo přesnost	(1 1 1 0 0)
	D2:	úplnost přesnost	(0 0 1 1 0)
	D3:	text prohledávání slovo	(1 1 0 0 1)
	Q:	prohledávání textu	(1 0 0 0 1)

hitem v Bool. modelu je D3

Vektorový model používá skalární součin D . Q
D1 . Q = 1 D2 . Q = 0 D3 . Q = 2
pořadí vybraných D3, D1

Vektorový model bere v úvahu počet výskytů termů v dokumentech
např. (2 1 0 0 3)

nevýhody: nerespektuje závislost vah termů na délce dokumentu
zdokonalení: kosinová míra

***Dokumentografický informační systém* (DIS)** je informační systém, který slouží k uchovávání textů v přirozeném jazyce. Texty se zde nazývají *dokumenty*. Jako příklad DIS si můžeme uvést knihovnu. Jednotlivé dokumenty zde představují knihy, časopisy, sborníky a další tiskoviny, které knihovny obvykle poskytují.
V dnešní době se používá několik různých implementací DIS. Jednotlivé implementace se od sebe liší organizací indexu, reprezentací dotazu a tedy celkovým přístupem k vyhledávání dokumentů. Dva nejčastěji používané přístupy k implementaci DIS, a to *boolský model* a *vektorový model*, jsme si uvedli výše.

Hypertextové systémy

- nelineární reprezentace textů v podobě indexové struktury reprezentované orientovaným grafem

Uzly grafu:
Jednotlivé dokumenty nebo jejich části, které popisují jednu myšlenku, koncept, ideu (forma = okno).
Méně obsáhlé než tradiční dokumenty.
Podpora modulárního myšlení.

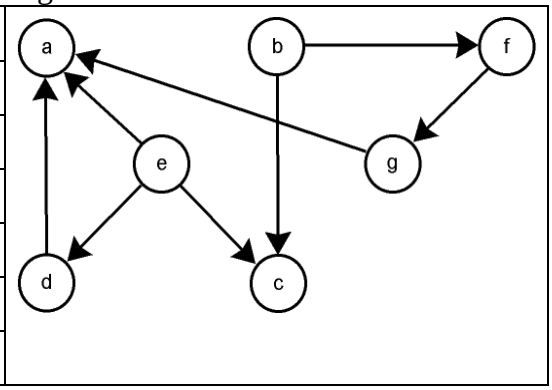
Typy vazeb grafu:
Referenční - vytvářejí nehierarchické struktury. Referenční vazby spojující body nebo spojovací oblasti textu jsou v zásadě jednosměrné.
Organizační - realizují nějaký způsob organizace. Např. vazby hierarchického typu umožňují vytvářet hierarchie uzlů jako logických jednotek, které mohou vyjadřovat generalizace/specializace k vytváření struktur známých jako hierarchie.

Funkce vazeb:
- spojují odkazy z jednoho dokumentu na druhý
- spojují poznámku nebo anotaci s textem, kterého se týká
- vykonávají organizační funkci, tj. například spojení několik souvisejících částí textu vyjadřujících jeden koncept.
- spojují bezprostřední následníky jednotlivých částí textu
- spojují různé typy uzlů (např. obrázek s jeho popisem apod.)

Statické vazby - pouze založení a zrušení
Dynamické vazby - mění své určení pomocí přidruženého skriptu
Křížové odkazy - výjimky z hierarchie

Reprezentace struktury dokumentu:
maticí vzdálenosti grafem

	a	b	c	d	e	f	g
a	0	∞	∞	∞	∞	∞	∞
b	3	0	1	∞	∞	1	2
c	∞	∞	0	∞	∞	∞	∞
d	1	∞	∞	0	∞	∞	∞
e	1	∞	1	1	0	∞	∞
f	2	∞	∞	∞	∞	0	1



g	1	∞	∞	∞	∞	∞	0	

Centralita uzlu:
součet vzdáleností z uzlu do všech ostatních uzlů. (Nekonečno nakrazujeme číslem K, tzv. konverzní konstantou. Nejčastěji K volíme rovno počtu uzlů.

Matice konvertovaných vzdáleností

Hypertextové metriky
Cp (Kompaktnost dokumentu) - míra propojenosti uzlů

$$Cp = \frac{Max - Sum}{Max - Min}$$

kde
Max je maximum, kterého může dosáhnout součet konvertovaných vzdáleností,
Sum je aktuální součet všech vzdáleností v matici konvertovaných vzdáleností
Min je minimum, kterého může dosáhnout součet všech vzdáleností

Cp plně propojeného dokumentu = 1
Cp zcela nepropojeného dokumentu = 0

Prestiž uzlu i
- suma konečných hodnot řádku i minus suma konečných hodnot sloupce i v matici vzdáleností (nekonvertované)

Absolutní prestiž dokumentu
- součet absolutních hodnot prestiží všech uzlů

Lineární absolutní prestiž dokumentu s n uzly
- prestiž lineárně uspořádaného dokumentu se stejným počtem uzlů

Stratum (vrstevnatost) - míra volnosti čtení dokumentu uživatelem
Stratum (D) = Absolutní_prestiž(D) / Lineární _abs_prestiž(D)
Např. žádná volnost v pořadí čtení, pak lineární dokument = (stratum = 1)

Dolování Webu

1. **Web content mining (analýza obsahu)**
2. **Web usage mining (analýza logů)**
3. **Web structure mining (analýza topologie web stránek)**

1.
Vyhledávací mechanismy založené na IR technikách dolování v textech (klíčová slova, hierarchie pojmů, synonyma).

A. Vyhledávací agenti (robots, crawlers, spiders) - shromažďují data
B. Indexátor - ukládá data
C. Dotazovací mechanismus – poskytuje informace uživateli

Související úlohy:
- Filtrace dokumentů
- Vyhledávání uživatelských profilů (personalizace)

- Sumarizace vyhledaných informací
- Klasické dolovací techniky (shlukování a klasifikace dokumentů)

2.
Vyhledání vzorů přístupu uživatelů ke stránkám z Web log záznamů.
Web server zaznamenává:

- Požadované URL
- IP adresu odkud pochází žádost o přístup
- Timestamp.

-využitelné ke zlepšení návrhu stránek (strukturu, návštěvnost)
- využitelné ke zlepšení efektivity zpracování (page caching, swapping)
- využitelné ke zlepšení vyhledání potenciálních zákazníků (e-obchod)

Obvyklá technika – hledání tzv. traversal patterns v podobě např:
-asociačních pravidel –které stránky jsou při seanci společně navštěvovány
-sequential patterns –posloupnosti stránek, časté vzhledem k počtu zákazníků,
kteří se dle takového vzoru chovají (lze pak shlukovat i zákazníky)

3.
Analýza topologie webových stránek

- Analýza z grafu (matice) stránek

Metoda HITS (Hyperlink-Induced Topic Search)
Hledání vysoce relevantních stránek (pro dané téma)
Hledání stránek obsahujících kolekce odkazů na autority

- Dotazem získá PS -počáteční množinu (cca 200) stránek
- Expanduje PS na BS přidáním všech stránek, na které PS odkazuje a které odkazují na PS a omezí BS na cca x000 stránek
- Odstraní spoje mezi stránkami z jedné domény (první úroveň URL slouží pro navigaci a neobsahuje autority).
- Přiřadí počáteční konstantu váze autority a_p a hub váze h_p stránky p. Váhy jsou normalizovány tak, že součet jejich kvadrátů je 1
- Iteračně provádí propagaci vah mezi stránkami

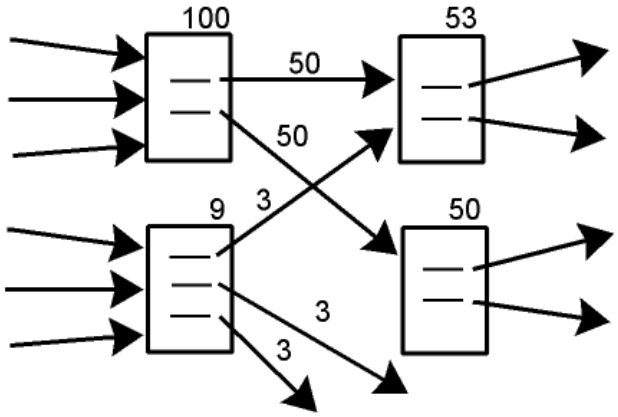
$a_p = \sum_{(q \text{ takové, že } q \rightarrow p)} h_q$, $h_p = \sum_{(q \text{ takové, že } q \leftarrow p)} a_q$

Bude-li X matice sousednosti stránek, a, h vektory vah, lze popsat maticově:
$a = X^t.h = X^tXa = \ldots = (X^tX)^2h$, $h = X.a = XX^t h = \ldots = (XX^t)^2h$
Konverguje k vlastním vektorům. Výsledkem jsou velké hubs a autority.

Ohodnocení stránek metodou PageRank (Google)
Důležitá stránka

- vede k ní mnoho odkazů,
- odkazují na ní vysoce ohodnocené stránky

Ohodnocení stránky R(u) je v iteracích rozdělováno rovnoměrně mezi stránky na které odkazuje, a tím těmto stránkám přispívá na jejich vlastní ohodnocení.



Poznámka: Google hodnotí stránky pomocí metody PageRank od 1 do 10, příčemž nejlepší je 10. Příklady: iDnes.cz (7/10), msdn.com (8/10), yahoo.com (9/10), google.com (10/10).

Zpracoval: Jan Kupka (kupka.jan@centrum.cz – ICQ:136594891)
Verze: 1
Zpracováno dle přednášek z DB2, internetu a několika knih
Upozornění:
Některé přednášky (XML) byly zaktualizovány dle nejnovějších technologií.
Sekce označené jako UNOFFICIAL nekopírují přednášky a jsou poskytovány bez záruky.