Deduktivní databáze

- Porovnejte vlastnosti logického programování a deduktivních databází

LP: malé, jednouživatelské databáze; zpracování řetězcem dedukcí; výsledkem dotazu yes/no
DDB: velké, víceuživatelské databáze; persistentní (stálá) data; efektivní přístup k datům na disku (analogie relace – predikát)

DDB rozšiřují vyjadřovací sílu relačního jazyka a zachovávají neprocedurální styl vyjadřování. Dovolují přirozené vyjadřování rekurzivních pravidel. Logická pravidla jsou vhodnou bází pro aplikace informačních systémů. Dovolují redukovat rozsah tabulek relační databáze.

- Jaké jsou podmínky bezpečnosti vyhodnocení datalogovských pravidel

Pro bezpečnost (smysluplnost) vyhodnocení se musí proměnná X, která se vyskytuje buď
- v hlavě pravidla,
- v negovaném podcíli
- v porovnávacím predikátu

vyskytovat také 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.

- Zapište datalogovským pravidlem integritní omezení …

Vyjadřování integritních omezení
např. incorrectDB :- rodič(X,X).
?- incorrectDB.

```
incorrectDB :- matka(X,Z), matka(Y,Z), not(X=Y).
incorrectDB :- rodic(X, _), not(osoba(X,_,_)).
incorrectDB :- rodic(_,Y), not(osoba(Y,_,_)).
incorrectDB :- predek(X, X).

incorrectDB('samsoberodic',[X]):-rodic(X, X).
incorrectDB('dvematky',[X,Y,Z]):-matka(X,Z),matka(Y,Z), not(X=Y).
incorrectDB('rodicneniosobou',[X]):-rodic(X,_), not(person(X,_,_)).
```

např. transakce
insert into parent(mary, karel), not(incorrectDB(X,Y))
odpověď: X=rodicneniosobou Y=[karel]

- Charakterizujte intenzionální a extenzionální databázi

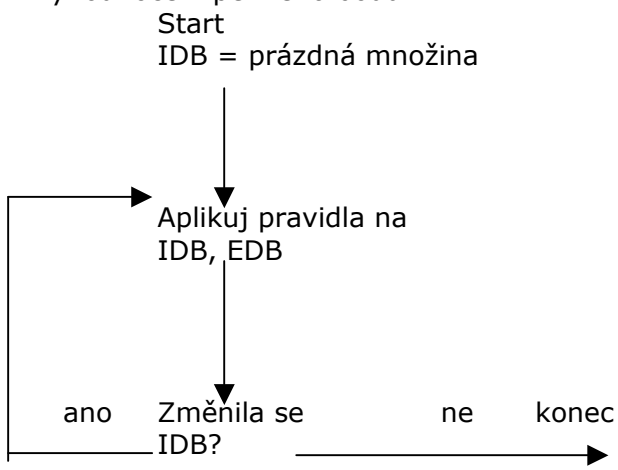
Extenzionální databáze = relace uložené v databázi. EDB predikáty se mohou vyskytovat pouze v těle pravidel.
Intenzionální databáze = relace definované pomocí pravidel. IDB predikáty se mohou vyskytovat v těle a v hlavě pravidel.

- Popište princip vyhodnocení pevného bodu

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).

Iterační vyhodnocení pevného bodu:



- Jaký je vztah mezi Datalogem a relační algebrou

Stratifikovaný Datalog

Datalog relační — pozitivní algebra = nerekurzivní Datalog — relační algebra

- Co rozumíte pod pojmem stratifikace logického programu

Stratifikace je „rozvrstvení", „vrstvení".
Graf závislosti predikátů:
- uzly jsou jména predikátů
- hrana vede z uzlu Q do uzlu P, obsahuje-li program pravidlo P :- ... Q ... .

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.
Stratifikací logického programu se nazývá rozklad množiny intenzionálních predikátů na podmnožiny P1, P2, ... , Pn tak, aby bylo splněno:

- jesliže $p \in Pi$, $q \in Pj$ a <q, p> je hranou rozšířeného grafu závislostí, pak $i \geq j$
- jesliže $p \in Pi$, $q \in Pj$ a <q, p> je hranou rozšířeného grafu závislostí označená „~", pak $i > j$

Stratifikace určuje pořadí vyhodnocování predikátů
př.) o(X) :- r(X), s(X).
p(X) :- r(X), not o(X).
q(X) :- s(X), p(X).
předpokládáme, že r, s jsou extenzionální
strata: P1 = {o}
P2 = {p, q}

Program je stratifikovaný, jestliže rozšířený graf závislostí neobsahuje žádné cykly s hranou označenou „~".

- Formulujte podmínku jednoznačné vyhodnotitelnosti datalogovského programu s negacemi

Stratum IDB predikátu A je maximální počet ~ hran na cestě do A z EDB predikátu v grafu závislosti predikátů. Datalogovský program je stratifikovaný, jestliže každý IDB predikát má konečné stratum. Je-li datalogovský program stratifikovaný, můžeme vyhodnotit IDB relace v pořadí od nejnižšího strata.

- Jaká omezení musí splňovat rekurzivní relace v dotazech SQL99

Založeno na rekurzi v Datalogu, předpokládá pouze lineární rekurzi, stratifikovanou negaci a obdobná omezení pro agregáty

Def: Predikát *s* implikuje predikát *r* ($s \Rightarrow r$), obsahuje-li IDB klauzuli s hlavou *r* a s predikátem *s* v těle, nebo existuje predikát *t* takový, že $s \Rightarrow t$ a $t \Rightarrow r$. Predikát *r* je rekurzivní jestliže

$r \Rightarrow r$. Jestliže $r \Rightarrow s$ a $s \Rightarrow r$, pak r, s jsou vzájemně rekurzivní a jsou na stejné dedukční úrovni. Jinak, pokud $r \Rightarrow s$ a not $s \Rightarrow r$, *pak* r je v nižší dedukční úrovni než s.

Def: Pravidlo je lineárně rekurzivní, obsahuje-li jeho tělo právě jeden rekurzivní predikát a tento predikát je definován ve stejné dedukční úrovni jako hlavový predikát.

OO a OR databáze

- Zdůvodněte nedostatečnost relačních DB pro současné aplikace

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 objektů
impedance mismatch
Možné 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.

- Popište pojem impedance mismatch

nesoulad typových systémů DBS a PJ
nesoulad vyhodnocovací strategie DBS a PJ

- Jaké vlastnosti zahrnuje 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

ad 1 Každá entita reálného světa je objektem, který je popsán jménem, dobou trvání, OID (jedinečné)
Relační model: identifikace klíčem, klíč je modifikovatelný, klíč je jedinečný v rámci relace
OO model: identifikace OID, OID nelze měnit, OID je jedinečný v databázi, OID generuje systém (ukazatel na objekt)
OR model: soužití OO a relačních DB, vliv výrobců, kontinuita (spojitost)

- K čemu slouží a jaké výhody má OID

Každý objekt má jedinečné a nemodifikovatelné OID. OID vytváří systém, v centralizovaném systému jej vytvoří z času vytvoření objektu, v distribuovaném prostředí navíc identifikace hostitele, OID nelze měnit, OID je jedinečný v databázi, OID generuje systém (ukazatel na objekt)

- Jaké prostředky má ODL k popisu vlastností a jak popisuje relace mezi objekty

Deklarace objektových typů
interface: k popisu abstraktního chování, použitelné jen k dědění operací
interface <jméno třídy> {<seznam vlastností>}
class: k popisu abstr. chování i abstr. stavu (k dědění i k vytváření objektů), class slouží k popisu databázového schéma
class <jméno třídy> {<seznam vlastností>}
Atributy v ODL jsou nejjednodušším typem vlastností (atomické, složené)
Př.

```
Interface Obrazec {
attribute enum Tvar {Obdelnik, Kruh, Trojuhelnik} Druh;
attribute struct Bod {short x, short y} Ref_bod;
float plocha();
};
class Obdelnik: Obrazec {
attribute struct Bod {short x, short y} Ref_bod;
attribute short Delka;
attribute short Sirka;
};
class Kruh: Obrazec {
```

attribute struct Bod {short x, short y} Ref_bod;
attribute short Radius;
};
Př.
class Film {
attribute string titul;
atribute short rok;
attribute short delka;
attribute enum EFilm {barevny, cernobily} typfilmu;
};
Př. složeného atributu
class Herec {
attribute string jmeno;
attribute Struct Adr {string ulice, string mesto} adresa;
};
Typ atributu nesmí být třídou

Relace v ODL prostředek spojování objektů
Př. chceme přidat k třídě Film vlastnost - množinu jeho herců
=> je nutné vytvořit relaci mezi třídami Film - Herec
relationship Set <Herec> herci ;
a vložit jí do třídy Film
Př. Chceme vyjádřit relaci jediného objektu s třídou
relationship Herec hlavníherec;
ODL zavádí jen binární relace (n-ární lze transformovat)
Binární relace musí být obousměrná

- Jaké prostředky a možnosti má ODL k popisu dědění

Podtřídy a dědičnost
interface jméno_subinterface : jméno_interface1,: jméno_interface2…
{ seznam vlastností subinterface }

class jméno_třídy : jméno interface1, : jméno_interface2 …
{ seznam vlastností třídy }

class jméno_podtřídy extends jméno třídy
{ seznam vlastností podtřídy }

Př.
class Muzikál extends Film {
relationship Set < Herec > hlasy;
} ;
class Krvák extends Film {
attribute long long po_etmrtvol;
} ;
Násobná dědičnost je možná pouze od interface!

- Co je extent třídy v ODL a k čemu slouží

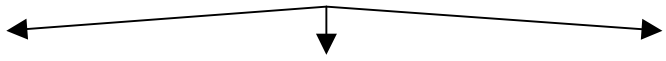
Deklarace signatur metod v ODL
Signatura určuje jméno metody sdružené s třídou, vstupní / výstupní typy metody
Kód metody je zapsán v hostitelském jazyce, není součástí ODL
Syntax obdobná funkcím C mimo:
- specifikace parametrů in, out, inout
- fce může způsobit výjimky, raises (seznam výjimek)

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 (titul, rok))
{
attribute string titul;

```
    attribute short rok;
    attribute short delka;
    attribute enum (barevny, cernobily) typfilmu;
    relationship Studio vlastneny inverse Studio :: vlastní;
    relationship Set <Herec> herci inverse Herec :: hrajev;
    float naklady ( ) raises(nenalezenynaklady);
    void jmenahercu(out Set <String>);
    void jinefilmy(in Herec, out Set <Film>) raises(nenitakovyherec);
};
class Herec (extent Herci key jmeno)
{
    attribute string jmeno;
    attribute Struct Adr {string ulice, string m_sto} adresa;
    relationship Set <Film> hrajev inverse Film :: herci;
};
class Studio
(extent Studia key jmeno)
{
    attribute string jmeno;
    attribute string adresa;
    relationship Set <Film> vlastní inverse Film :: vlastneny;
};
```

- Popište objektové konstrukce SQL99

Kompatibilita s existujícími jazyky
OID
Hnízděné tabulky
Uživatelem definované typy (definují třídy)



Abstraktní datové typy (jsou typem atributu relace)	řádkové typy (jsou typem relace)	Odlišující typy (musí být FINAL)

- UDT mohou být organizovány do hierarchií s děděním
- chování UDT je realizováno pomocí procedur, funkcí a (metod u ADT)

Objekty v SQL99 pracují s relacemi

- Napište v SQL99 řádkový typ, který …

CREATE ROW TYPE jméno (deklarace komponent)
př. řádkový typ reprezentující herce

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

Deklarace relace s pojmenovaným řádkovým typem
př. CREATE TABLE Filmovýherec OF typherec; (obdoba extent třídy: Herec ---- typherec)

Deklarace relace s nepojmenovaným řádkovým typem
př.

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

Zpřístupnění komponent řádkového typu
př. Najdi jméno a ulici každého herce z Plzně

```
SELECT Filmovýherec.jméno, Filmovýherec.adresa.ulice FROM Filmovýherec
WHERE Filmovýherec.adresa.město = ´Plzeň´;
```

- Napište v SQL99 abstraktní typ, který …

Na rozdíl od řádkových typů umožňují zapouzdření atributů a operací. Hodnoty jejich typů mohou být umístěny do sloupců tabulek.
Definice ADT:
CREATE TYPE <jméno typu> AS (seznam atributů a jejich typů, nepovinná deklarace metod, údaje o děditelnosti a instalovatelnosti);
Další funkce lze deklarovat vně příkazu CREATE TYPE, ty pak nejsou svázány s ADT.

př.
```
CREATE TYPE typzamestnanec AS (
    c_zam INTEGER,
    jmeno CHAR (20),
    adresa typadresa,
    vedouci typzamestnanec,
    datum nástupu DATE,
    základní plat DECIMAL(6,2))
    INSTANTIABLE NOT FINAL,
    METHOD odpr_leta( ) RETURNS INTEGER; /*jen signatury*/
    METHOD mzda ( ) RETURNS DECIMAL;

    CREATE METHOD odpr_leta FOR typzamestnanec BEGIN … END ;
    CREATE METHOD mzda FOR typzamestnanec BEGIN … END;
```

Instance ADT vznikají:
1. konstruktorem *jménotypu( )*
2. operátorem *NEW jméno hodnota* (př. WHERE vedoucí = NEW typzam(10234, ` Petr Nový...)
3. příkazem INSERT (př. INSERT INTO osoby VALUES (10234,`Petr Nový`,…);)

Pro každý atribut jsou k dispozici funkce:
- implicitně či explicitně zavedené porovnání
- zjištění hodnoty atributu z objektu
    jméno atributu(jméno objektu)
    stejné i pro aplikaci metod
    př. odpracovaná_léta(X) -virtuální atribut
    možná i tečková notace X.odpracovaná_léta

- Napište v SQL99 podtyp … typu …, který

CREATE TYPE typkulisak UNDER typzamestnanec AS (*Další atributy a metody*);
- jen jednoduché dědění
- dědí atributy i metody svých nadtypů
- strukturované typy musí být NOT FINAL
- odlišující typy musí být FINAL

- Definujte v ORACLE definici typu pro …

Definice typů je podobná SQL99. Syntax má tvar:
CREATE TYPE t AS OBJECT (list of attributes and methods);/
př. CREATE TYPE PointType AS OBJECT (x NUMBER, y NUMBER);/
Objekt může být použit jako ostatní typy v deklaracích objektových typů nebo tabulkových typů.
př.
CREATE TYPE LineType AS OBJECT (end1 PointType, end2 PointType);/

- Definujte v Oracle tabulku, jejíž sloupce obsahují odkazy na hodnoty typu …

CREATE TABLE Lines (lineID INT, line LineType);
Odstranění typů: DROP TYPE Linetype;
Vytváření hodnot objektů - konstruktory:
INSERT INTO Lines VALUES(27, LineType(PointType(0.0, 0.0), PointType(3.0, 4.0)));

- Definujte v Oracle tabulku, jejíž sloupce obsahují hnízděné tabulky

Hnízděné tabulky - typ sloupce může být tabulka
CREATE TYPE PolygonType AS TABLE OF PointType;/
př. Deklarace relace, jejíž sloupce mají hodnoty polygonů

```
CREATE TABLE Polygons (
    name VARCHAR2(20),
    points PolygonType
) NESTED TABLE points STORE AS PointsTable;
```

Hodnotami sloupce points jsou tabulky dvojic bodů. Řádky této vnořené tabulky jsou uloženy ve zvláštní tabulce definované konstrukcí NESTED TABLE jméno_sloupce STORE AS uložení_sloupce. Při použití více sloupců ve tvaru vnořených tabulek, je třeba definovat takovou "ukládací tabulku" pro každý typ vnořené tabulky. Data vnořené tabulky jsou uložena mimo rodičovskou tabulku. Propojení si Oracle zajistí sám.

Vkládání do relací se sloupci typu hnízděné relace je prováděno pomocí konstruktoru typu hnízděné relace (zde PolygonType). Vkládané hodnoty jsou rovněž označeny typem.
př. Vložení polygonu "square"
INSERT INTO Polygons VALUES('square', PolygonType(PointType(0.0, 0.0), PointType(0.0, 1.0), PointType(1.0, 0.0), PointType(1.0, 1.0)));

Dotaz na rohy čtverce: SELECT points FROM Polygons WHERE name = 'square';
Dotaz na body polygonu (čtverce), ležící na hlavní diagonále (tj. x=y)
SELECT ss.x FROM THE (SELECT points FROM Polygons WHERE name = 'square') ss
WHERE ss.x = ss.y;

Distribuované databáze

- Jaké jsou výhody a nevýhody distribuovaných databází oproti centralizovaným

výhody:

- 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)
- nevýhody:
- 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)

- Uveďte obecné požadavky kladené na distribuované databázové systémy

obecné:

- transparentnost distribuce (míra viditelnosti distribuce dat pro uživatele
- autonomie (distribuce řízení)
- heterogennost
- výkon (vysoká průchodnost krátká odezva)

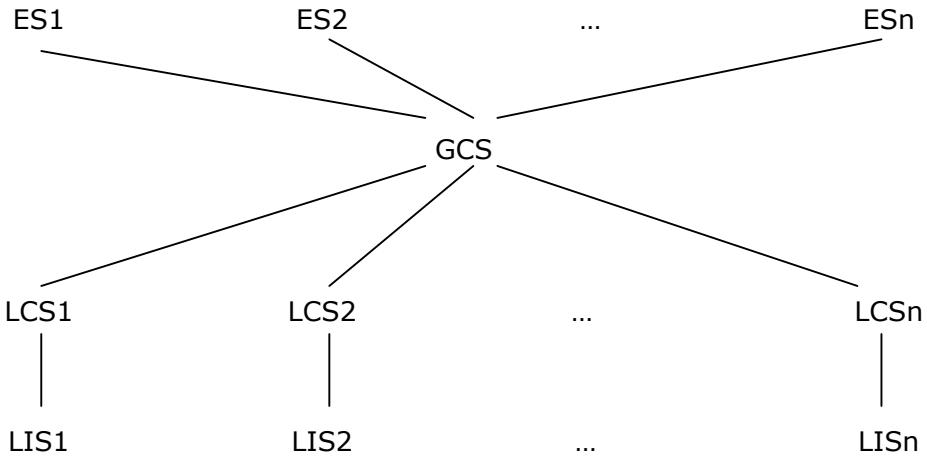
Požadavky DSŘBD formuloval Date: 1. lokální autonomie – každé místo má lokální SŘBD, 2. vše je distribuované – ve všech službách se nespoléhá na žádné centrální místo, 3. kontinuálnost – akce v jednom místě (např. odstranění tabulky) by neměla příliš narušovat provoz DDBS jako celku, 4. nezávislost na 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
Implementace realizují jen část idejí Date. (Ingres Star, IBM DRDA, Oracle 7 a vyšší)

- Uveďte formy transparentnosti požadované v distribuovaných databázích

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

- Jaké části obsahuje ANSI/SPARC referenční model pro distr. Databáze

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



- Jmenujte hlavní části obsažené v architektuře distribuovaného databázového systému

globální transakční monitor
- globální katalog
- komunikační interface
  - lokální transakční monitor
    - LSŘBD
      - lokální DB
      - lokální katalog

- Podle jakých hledisek je zavedena taxonomie distribuovaných databázových systémů?

Důsledky ANSI/SPARC architektury - hlediska (vzájemně ortogonální):
1. autonomie lokálních systémů
2. distribuce dat
3. heterogenity systémů

- Na jaké typy se člení DistrDBS podle hlediska autonomie lokálních systémů?

1. 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
2. semiautonomní systémy (lokální DBMS pracují nezávisle a sdílejí svoje lokální data v celé federaci). Jen část jejich dat je sdílena
3. zcela autonomní = izolované. Lokální DBMS pracují nezávisle a neví o ostatních DBMS. Pro vzájemnou komunikaci potřebují softw. vrstvu pracující nad jednotlivými DBMS

- V jakých stavech mohou být podtransakce v DistrDBS

Transakce v DDBS rozděleny na podtransakce (prováděny na různých místech). Transakční zpracování v jednotlivých uzlech nestačí.
Podtransakce má stavy: A (Active), C (Commited), AB (ABorted), RC (Ready to Commit) a F (Failed)

- Popište dvoufázový potvrzovací protokol pro Distr.DBS

Transakční provoz zajišťují zprávy: PREPARE to COMMIT, READY to COMMIT, COMMIT, COMMIT SUCCESFULL, ABORT, ABORT COMPLETED

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

Pravidlo o potvrzení lze formulovat také: 1. Koordinátor zruší transakci právě když alespoň jeden participant hlásí zrušení transakce. 2. Koordinátor potvrdí transakci právě když všichni participanti hlásí, že jsou připraveni ji potvrdit. Pozn. Nevrací-li se odpověď, je po uplynutí time-out transakce zrušena.

- Napište fragmentační formuli a formuli pro rekonstrukci relace pro případ horizontální fr. (odvozené, vertikální)
  - Horizontální
    F_i = Selection $_{Pi}$ R
    R = $\cup F_i$
  - Odvozená horizontální, založena na horizontální fragmentaci jiné relace
    Používá polospojení = R<$t_1\theta\, t_2$] S = (R [$t_1\theta\, t_2$] S) [Atr(R)]
    př.
    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
  - Vertikální
    F_i = R(A_i)
    R = F_1 join F_2 … join F_n
    př.
    předp. funkční závislosti INV_C $\rightarrow$ {ISBN, CENA}, ISBN $\rightarrow$ {R_VYD, VYDAVATEL}
    bezztrátová vertikální fragmentace je:
    FR1 (ISBN, R_VYD, VYDAVATEL) FR2 (INV_C, ISBN, CENA)
  - Smíšená

- Předpokládejte … fragmenty … zapište v relační algebře dotaz … Nakreslete strom dotazu

Zjistit jména vydavatelů knih levnějších než 600
- v centralizované DB: KNIHY (CENA < 600) [Vydavatel]
  Projection $_{Vydavatel}$ (Selection $_{CENA<600}$ KNIHY)
  select Vydavatel from KNIHY where CENA<600
- v horiz. fragm. DB: (F2000 $\cup$ F2002) (CENA<600) [Vydavatel]
- v vertik. fragm. DB: (FR1 * FR2) (CENA<600) [Vydavatel]

Strom dotazu
Dokreslit, strana 12

- Předpokládejte relace … umístěné v uzlech … Navrhněte strategii pro výpočet spojení … prostřednictvím jednoduchého zpracování spojení (paralelního zpracování spojení, polospojení)

Problém přenosu dat mezi uzly lze redukovat na problém řešení spojení relací umístěných v různých uzlech sítě.
1. Jednoduché zpracování spojení - spojení se zpracovává v jednom uzlu sítě
2. Paralelní zpracování spojení - ve více uzlech se současně zpracovávají podvýrazy dotazu obsahující spojení

3. Zpracování pomocí polospojení - mezi uzly se přenáší pouze ty n-tice, které budou operací spojení skutečně spojeny

př.
DB s relacemi, KNIHA, EXEMPLÁŘ, VÝPUJČKA v uzlech S1, S2, S3
KNIHA (ISBN, AUTOR, TITUL)
EXEMPLÁŘ (ISBN, INV_C, R_VYD, CENA, VYDAVATEL)
VÝPUJČKA(INV_C, C_CT, D_VRAC)

př. strategie 1:
předp. v dotazu potřebu spojení KNIHA* EXEMPLAR * VYPUJCKA v S3. Nechť je |EXEMPLAR| >=|KNIHA|>=|VYPUJCKA|

S2: T1 ← VYPUJCKA
S2: T2 := T1 * EXEMPLAR
S1: T3 ← T2
S1: T4 := T3 * KNIHA
S3: T5 ← T4

př. strategie 2:
předp. dotaz "najdi tituly knih, jejichž exempláře se mají vrátit do 1.6.

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

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

$\text{projection}_{TITUL}$
(projection $_{INV.C}$ (selection $_{D.VRAC<1.6.}$ VYPUJCKA)
join
(projection $_{TITUL, INV.C}$ (EXEMPLAR join KNIHA))
)

př. strategie 3:
polospojení ($R_{[A=B]}$ S)[R]($R_{<A=B]}$ S)
R Semijoin$_F$ S = Projection$_{Atrib(R)}$ (R join$_F$ S)
platí ($R_{<A=B]}$ S) ≠ ($S_{<B=A]}$ R)

Chceme vyčíslit EXEMPLAR * KNIHA výsledek
S2 S1 S2

Postup: S2: T1 := EXEMPLAR [ISBN]
S1: T2 ← T1
S1: T3 := KNIHA * T2
S2: T4 ← T3
S2: T5 := EXEMPLAR * T4

v S1 vyhodnoceno KNIHA $_{<ISBN=ISBN]}$ EXEMPLAR

převedení EXEMPLAR * KNIHA
na (KNIHA $_{<ISBN=ISBN]}$ EXEMPLAR) * EXEMPLAR

- Jaká obecná hlediska je třeba respektovat při návrhu fragmentů distr.DB
  - rozdělit relace do lokálních serverů tak, aby aplikace zatěžovaly servery stejnoměrně. (Musí být známa informace o předpokládaných přístupech k relacím).
  - lokality zpracování (maximalizovat lokální)
  - přístupnosti a spolehlivosti (např. replikací zlepšíme spolehlivost a read-only dostupnost)
  - maximalizovat stupeň paralelismu zpracování dotazu
  - dostupnosti a ceny paměti v jednotlivých uzlech

Nelze vyhovět všem (složité optimalizace), nutno vyřešit dva problémy:
- co má fragment obsahovat
- kam fragment umístit

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

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

- Jaké vlastnosti zachycuje graf distribuovaného spojení? Definujte jednoduchý graf d. sp.

Návrh fragmentů při odvozené horizontální fragmentaci
Odv. hor. fragmentace se používá k usnadnění spojení mezi fragmenty
Distribuované spojení relací R a S = spojení mezi horizontálně fragmentovanými relacemi. Provádí se porovnání všech fragmentů R_i se všemi fragmenty S_j.

Graf distribuovaného spojení - hrany spojují fragmenty, jejichž spojení nejsou prázdná.
Navrhnout fragmenty tak, aby vzniknul:
- Jednoduchý graf spojení
  R join S má jednoduchý graf spojení, platí-li Ri = R Semijoin Si
- Nesouvislý graf spojení

- Pro atributy … uzly … a procesy … navrhněte fragmenty a jejich alokaci v síti

Návrh fragmentů při vertikální fragmentaci (které atributy do kterého fragmentu)
Pro bezztrátovost musí každý fragment obsahovat primární klíč. Fragmentace ostatních atributů závisí na frekvenci současného přístupu k nim z aplikace (užívá se statistické shlukování atributů).
1. Minimalizovat počet fragmentů potřebných ke splnění požadavku
2. Minimalizovat zátěž sítě

Alokace fragmentů a replik
- Plně replikovaná DB
- Parciálně replikovaná DB (problém - stupeň replikace, odkud číst)

DW

- Charakterizujte rozdíly mezi DW a DBS

DBS (OLTP): zákaznicky orientovaný, současná data, ER schéma, atomické transakce, DB až GB
DW (OLAP): orientovaný na trh, historická data, agregovaná data (nenormalizovaná=redundantní), schéma hvězdy či vločky, read only, DB až TB

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.

- Jaké úkoly plní OLAP

„on line analytic processing" – popisuje zpracování ve warehousu (získávání dat).
Většinou čte, dlouhé komplexní dotazy, Gb-Tb dat, summarizovaná a konsolidovaná data, vedoucí pracovník a analytik jako uživatel. Jedná se o specializovaný SQL server, který má rozšířený SQL pro dotazy nad hvězdovým/vločkovým schématem.

- Jaké jsou alternativní modely pro uspořádání dat v DW, jejich výhody a nevýhody

Data lze uspořádávat:
- Klasicky pomocí speciálně navržené relační database
- Ve vícerozměrném datovém modelu (zcela odlišné od relačního modelu)

Vícerozměrný datový model implementuje data vícerozměrnými poli (vícerozměrný spreadsheed). Dimenze odpovídají dimenzím podnikání organizace. Multidimenzionální OLAP (MOLAP). Datová krychle (obsahuje fakta), hierarchické dimenze (částečné či totální uspořádání).

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

- Jaké typy tabulek obsahuje hvězdové schéma a jaký je jejich účel

tabulka faktů obsahuje mimo jiné i primární klíče ostatních tabulek, kolik tabulek „jde" z tabulky faktů, toliko je to dimenzionální krychle, oproti vločkovému schématu zde mohou být redundantní záznamy

- Charakterizujte pojem galaktické schéma DW

Konstelace faktů, obsahuje více tabulek faktů.

- Jak se odlišuje hvězdové schéma od vločkového

Vločkové schéma odstraňuje redundance, tabulky dimenzí podle potřeby dále rozkládá (např. výrobce bude vícekrát u více výrobků → rozložit)

- Vysvětlete sémantiku klauzule GROUP BY CUBE (GROUPING)

SELECT … GROUP BY CUBE (seznam seskupovaných sloupců)
Dává multidimenzionální přehled všech možných kombinací podle vybraných dimenzí.

např.
```
SELECT rok, zbozi_jm, město, sum(obrat_v_Kc) AS "obrat"
FROM prodeje, místo, cas, zbozi
WHERE    prodeje.cas_klic = cas.cas_klic AND
         prodeje.misto_klic = místo.misto_klic AND
         zbozi.zbozi_klic = prodeje.zbozi_klic
GROUP BY CUBE (rok, zbozi_jm, město);
```

Pomocí funkce GROUPING lze vytvořit masky pro dimenze.
```
SELECT rok, zbozi_jm, město,  sum(obrat_v_Kc) AS "obrat",
GROUPING (rok) AS r,
GROUPING (zbozi_jm) AS z,
GROUPING (mesto) AS m
FROM  prodeje, místo, cas, zbozi
WHERE    prodeje.cas_klic = cas.cas_klic AND
         prodeje.misto_klic = místo.misto_klic AND
         zbozi.zbozi_klic = prodeje.zbozi_klic
GROUP BY CUBE (rok, zbozi_jm, město);
```

Masky je možné použít v konstrukci HAVING pro ROLLUP.
```
SELECT  rok, zbozi_jm, město,  sum(obrat_v_Kc) AS "obrat"
GROUPING (rok) AS r,
GROUPING (zbozi_jm) AS z,
GROUPING (mesto) AS m
FROM  prodeje, místo, cas, zbozi
WHERE    prodeje.cas_klic = cas.cas_klic AND
         prodeje.misto_klic = místo.misto_klic  AND
         zbozi.zbozi_klic = prodeje.zbozi_klic
GROUP BY CUBE (rok, zbozi_jm, město)
HAVING   (GROUPING (rok) = 1) OR (GROUPING (zbozi_jm) = 1) OR (GROUPING (mesto) = 1);
```

Vypíše jen agregovaná data (mající alespoň jednu 1 v masce)

- Popište typické OLAP operace

roll-up (srolování): sroluje jeden rozměr…, např. města sroluje na Čechy a Morava
drill-down (zavrtání): jakoby opak roll-up, např. kvartály (čtvrtletí) rozdělí (zjemní) na menší časové jednotky měsíce
dice (výřez): vybere jen to co chceme,např. Praha, Plzeň; 1. a 2. kvartál a jen dva druhy zboží
slice (řez): např. pro jeden čas, řezem získáme jen města a zboží v daném čase

- Jaké jsou kroky návrhu DW

Shora dolů, zdola nahoru. Z hlediska SWI obsahuje kroky: plánování, studie požadavků, analýza problému, návrh DW, integrace a testování dat, spuštění systému DW.

Metodologie používané při vývoji:
vodopád - provádí analýzu v každém kroku, před zpracováním dalšího kroku
spirála - vytváří postupně funkčně dokonalejší verze systému

Administrace DW: obnovování dat, synchronizace datových zdrojů, zajištění bezpečnosti, zajištění výkonnosti, …

Konkrétní kroky návrhu:
1. Výběr modelovaného procesu
2. Volba granuity
3. Výběr dimenzí aplikovaných na každý záznam tabulky faktů
4. Výběr měřítek pro záznam v tabulce faktů

- Popište 3 úrovňovou architekturu DW
  - klient
  - OLAP server (MOLAP/ROLAP server)
  - databázový server DW

- Kolik kuboidů lze vytvořit ve … rozměrné krychli, mají-li dimenze … úrovní

Počet kuboidů (variací GROUP BY) z n-rozměrné krychle = 2^n.

S uvážením i hierarchií, je-li Li počet úrovní i-té dimenze, pak počet všech kuboidů = $\prod_{i=1}^{n}(L_i+1)$

- Z materializovaných kuboidů … vyberte použitelné pro dotaz na …

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

ad 3): Kuboid P může být generátorem dotazu Q když:
a) Rozměry Q jsou podmnožinou dimenzí P (či stejné)
b) Selekční klauzule v Q implikuje selekční klauzuli v P
c) Pro každou dimenzi v Q je úroveň abstrakce této dimenze hrubší (vyšší) než v P.

Z množiny kuboidů se vybere rozsahem minimální, s nejvhodnější indexovou strukturou.

př.
Dána krychle: prodeje[čas,zboží,místo]: sum(prodej-vKč)
Dány hierarchie dimenzí:
den < měsíc < kvartál < rok
jméno_zboží < skupina < typ
ulice < město < země < stát

Dotaz je na {skupina, země}, selekční konstantou je rok 2004
Materializované kuboidy jsou:
k1: {jméno_zboží, město, rok}
k2: {skupina, stát, rok}
k3: {skupina, země, rok}
k4: {jméno_zboží, země, 2004}

Pak k2 nemůže být generátorem. k1, k3, k4 mohou být, liší se ale jejich ceny použití: k1 je nejdražší, je-li ve skupina počet hodnot pro rok < počet hodnot pro jméno_zboží pak vybereme k3, má-li k4 efektivní indexování pak lze vybrat k4.

Datamining

- Jaké jsou fáze KDD procesu

KDD proces - příprava dat
- hledání vzorů
- vyhodnocení znalosti
- modifikace a iterace

Kde data jsou množina faktů F a vzor je výraz E v jazyce L popisující fakta podmnožinou $F_E \subset F$
Vlastnosti vzoru: platnost C, novost N, užitečnost U, srozumitelnost S
Zajímavost vzoru Z(E,F,C,N,U,S). Vzor $E \in L$ nazýváme znalostí, platí-li pro uživatelem stanovený práh i: Z(E,F,C,N,U,S) > i
KDD proces je iterační a interaktivní

- Jmenujte základní DM tehniky
  - Charakterizace dat
  - Hledání asociací
  - Klasifikace
  - Predikce
  - Shluková analýza
  - Analýza odchylek
  - Vývojová analýza
  - Vyhledávání podobností

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

- Určete metodou ind. log. programování nespecifičtější zobecnění klauzulí …

Nejspecifičtější zobecnění (lgg)
Jsou-li dvě klauzule c1, c2 pravdivé, je jejich lgg(c1, c2) také pravdivé.
lgg dvou literálů se zjistí jejich porovnáním a nesouhlasné části se nahradí proměnnými
např. je-li c1 = dcera(m,a) ← žena(m), rodič(a,m).
c2 = dcera(e, a) ← žena(e), rodič(a,e).
pak lgg(c1,c2) = dcera(X,a) ← žena(X), rodič(a,X).

- Podle jakých hledisek klasifikujeme asociační pravidla
  - 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

- Uveďte definici supportu a konfidence asociačních pravidel

Získávání asociačních pravidel z databází
Support:

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

Confidence:

Počet transakcí obsahujících X a Y

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

- V transakční databázi … najděte časté množiny položek pro s = …

Častá nazýváme položky vyhovující minimální podpoře s.

Pro vyhledání častých množin položek se používá 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 join L_k = { A join B , kde A, B $\in L_k$, $|A \cap B| = k - 1$ }

př. (pokračování) vyhledání častých množin položek
Předpokládejme s = 50%

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

- V transakční databázi … vyhledejte silná pravidla pro c = … , s = …

Silná nazýváme pravidla splňující minimální prahové hodnoty s, c.
Platí: c (A⇒ B) = p (A a B) / p (A), kde p (A) je počet transakcí obsahujících A

Postup:
- Pro každou častou množinu položek generuj všechny neprázdné podmnožiny

- Pro každou neprázdnou podmnožinu *m* z *l* vytvoř pravidlo
  m $\Rightarrow$ *(l - m),* platí-li *p (l) / p (m)* $\geq$ c

Algoritmus generování pravidel: vstupem je množina častých množin položek L a minimální conf. C, výstrupe jsou silná asociační pravidla.
Metoda: pro každou častou množinu položek l [k] z L , k $\geq$ 2 vyvolej genrules(l [k], l[k])

```
procedure genrules ( l[k] : častá k-množina položek, m[j] : častá j-množina položek ) ;
begin
    S := { ( j - 1 ) - množiny položek  m [ j - 1], kde   m [ j - 1]  ∈ m [ j] } ;
    pro každé  m [ j-1] ∈ S dělej
            begin  c = p ( l [k] ) /  p ( m [ j -1] );
                    if  c  ≥  c  then
                            begin
                                    print (pravidlo m[ j-1] s podporou s=p(l[k]) s důvěryhodností  c) ;
                                    if  j -1 > 1  then genrules ( l [k] ,  m [ j-1]);
                            end ;
            end;
end;
```

př. 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)
- Za silná prohlas ta pravidla, jejichž věrohodnost (confidence) překračuje prahovou hodnotu

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.

- Jaké jsou obecné požadavky na shlukovací metody
  - schopnost zpracovávat rozsáhlá data
  - možnost pracovat s různými typy dat (numerická, binární, kategorická, ordinální)
  - malé požadavky na doménové znalosti (např. počet shluků)
  - schopnost nalézt shluky různého tvaru (nejen sférické)
  - schopnost pracovat za přítomnosti chyb v datech
  - necitlivost k uspořádání vstupních dat
  - schopnost práce s daty s vysokou dimenzionalitou
  - schopnost shlukovat i s ohledem na specifikovaná omezení
  - interpretovatelnost a užitečnost výsledků

- Definujte euklidovskou vzdálenost objektů s numerickými atributy (manhatanskou)

Vyhodnocování nepodobnosti pro měřitelné proměnné (numerické = váha, délka, …). Data nutno standardizovat, tj. dát všem proměnným stejnou váhu.

Postup standardizace (normalizace) proměnné f:
1. vypočítat střední absolutní odchylku s_f
   $s_f = (| x_{1f} - m_f | + | x_{2f} - m_f | + \ldots + | x_{nf} - m_f |) / n$, kde
   $m_f = (x_{1f} + x_{2f} + \ldots + x_{nf}) / n$
2. vypočítat standardizované hodnoty (z score), nahrazující v matici hodnoty proměnných
   $z_{if} = (x_{if} - m_f) / s_f$

Po (nebo bez) standardizaci počítáme nepodobnosti objektů

- Euklidovskou vzdáleností
  $d(i, j) = \sqrt{ (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ip} - x_{jp})^2 }$
  kde $(x_{i1} .. x_{ip})$, $(x_{j1} .. x_{jp})$ jsou dva p rozměrné datové objekty
- Manhattanskou vzdáleností
  $d(i, j) = | x_{i1} - x_{j1} | + | x_{i2} - x_{j2} | + \ldots + | x_{ip} - x_{jp} |$

- Navrhněte způsob porovnání míry rozdílnosti objektů s binárními atributy
  Kontingentní tabulka pro p binárních proměnných objektů i a j

	object tj			
object ti		1	0	sum
	1	q	r	q+r
	0	s	t	s+t
	sum	q+s	r+t	p=q+r+s+t

q počet proměnných rovných 1 u obou objektů
t počet proměnných rovných 0 u obou objektů
r počet proměnných rovných 1 pro object i, ale 0 pro j
s počet proměnných rovných 0 pro object i, ale 1 pro j
je-li binární proměnná symetrická (0 i 1 je stejně hodnotná), např. atribut pohlaví, pak:
d(i, j) = (r+s) / (q+r+s+t)

je-li nesymetrická (hodnoty nejsou stejně důležité), např. HIV pozitivní = 1, HIV negativní = 0, používá se Jaccardův koeficient (negativní shody nejsou důležité, ignorují se)
d(i, j) = (r+s) / (q+r+s)

Př.

Jméno	pohlaví	horečka	kašel	test1	test2	test3	test4
Jan	M	A (1)	N(0)	P(1)	N	N	N
Marie	Z	A	N	P	N	P	N
Josef	M	A	A	N	N	N	N
…							

d(Jan, Marie) = (0 + 1) / (2 + 0 + 1) = 0,33 dle Jaccarda
d(Jan, Josef) = (1 + 1) / (1 + 1 +1) = 0,66 „
d(Josef,Marie) = (1 + 2) / (1 + 1 +2) = 0,75 „
Takže Marie a Josef asi nemají stejný neduh

- Jak je definována vzdálenost mezi shluky dat: single / complete / average / medoid link

Single link – nejmenší vzdálenost mezi prvkem ze shluku 1 a shluku 2
Complete link – největší vzdálenost mezi prvkem ze shluku 1 a shluku 2
Average link – průměrná vzdálenost mezi prvkem ze shluku 1 a shluku 2
Medoid link – vzdálenost mezi reprezentativními prvky (medoidy)
Centriod link – mezi středy shluků

- Uveďte taxonomii shlukovacích metod a jejich základní principy
  - Nehierarchické metody (partitioning methods), rozkládají data, pro malé a střední databáze k nalezení sférických shluků.
    k-means: každý shluk je reprezentován střední hodnotou objektů ve shluku. Časová náročnost je O(t k n), kde t je počet iterací.
    k-medoids: každý shluk je reprezentován jedním z objektů, umístěným blízko středu shluku.

další: neuronové sítě, Kohonenovy sítě, metody založené na hustotě (shluk narůstá pokud počet objektů v sousedství překračuje zadanou mez), metody založené na mřížkách (prostor objektů rozparcelují mřížkou a shluky hledají na buňkách mřížky = rychlé)

- Hierarchické metody
  Vytváří stromovou strukturu – dendogram, prostorová náročnost je $O(n^2)$ – matice rozdílnosti, časová náročnost je $O(kn^2)$ – pro každou úroveň dendogramu jedna iterace
  aglomerativní (bottom-up): na počátku každý objekt je shlukem, Postupně se shluky sdružují, dokud není splněna ukončovací podmínka
  divisivní (top-down): na začátku jsou všechny objekty v jednom shluku a postupně jsou shluky štěpeny na menší.

- Z matice vzdálenosti objektů … vytvořte dendrogram (aglomerativně s použitím single link) (divisivně)

Single link se používá v různých variacích.
Uvedeme způsob založený na minimální kostře grafu (Minimum Spanning Tree)
Kostra grafu spojuje všechny všechny vrcholy a neobsahuje cyklus. Shluky jsou sdružovány dle vzestupného pořadí ohodnocení větví kostry grafu, kterou produkuje procedura MST z matice A.
Vstup: D = { t_1, t_2 , …, t_n } množina objektů, A matice vzdálenosti (nepodobnosti)
Výstup: DE dendogram tvaru <d, k, K>, kde d je prahová vzdálenost, k je počet shluků, K je množina shluků
Postup:

```
d = 0; na začátku je každý object shlukem, vzdálenost objektu je 0
k = n;
K = {{ t1 }, { t2 }, …, { tn }};
DE = {<d, k, K>};
M = MST( A );
repeat
  oldk = k;
  Ki , Kj  jsou dva shluky navzájem nejbližši dle MST;
  K = K – { Ki } - { Kj } ∪ { Ki ∪ Kj };
  k = oldk - 1;
  d = dis( Ki , Kj ) ;
  DE = DE ∪ <d, k, K>;  přidání nových shluků k dendogramu
  dis( Ki , Kj ) = ∞ ;
until k = 1;
```

Divisivní algoritmy
např. na základě MST: Postupně odsekáváme hrany z MST od největší k nejmenší

- Popište princip metody k-means

Přesouvá objekty mezi shluky, dokud není dosaženo konvergence, překročen daný počet iterací, min.kvadratické chyby apod.
Vstup: D = { t_1, t_2 , …, t_n } množina objektů, k = počet požadovaných shluků
Výstup: K = množina shluků
Postup:

```
Přiřaď počáteční hodnoty k středům m 1, m 2, …m k  (libovolně);
repeat
        přiřaď každý objekt ke shluku s nejbližším středem;
        vypočti nové středy shluku
until  konvergenční kriterium je splněno;
```

Časová náročnost je O(pkn), kde p je počet iterací. Nalezne lokální optimum. Pro různé inicializace dojde většinou k různým řešením. Vyžaduje číselná data. Vyhledává jen konvexní shluky. Špatně se vyrovnává s úlety.

- Popište princip metody k-medoids (Partitioning Around Medoids)

Lépe zpracovává úlety.
Vstup: D = { t_1, t_2 , …, t_n } množina objektů, k = počet požadovaných shluků
Výstup: K = množina shluků

Postup:
Libovolně vyber k medoidů z D;
repeat
for každé t_X které není medoidem do
for každý medoid t_i do
vypočti cenu změny CZ_{iX};
najdi i, X, pro něž CZ_{iX} je nejmenši;
if $CZ_{iX} < 0$ then
nahraď medoid t_i medoidem t_X;
until $CZ_{iX} \geq 0$;
for každé $t_i \in D$ do
přiřaď t_i k tomu K_j, pro něž je $dis(t_i, t_j)$ minimální pro všechny medoidy t_1 .. t_k;

Není vhodný pro větší databáze, časová náročnost je $O(t\,k\,(n-k)^2)$. Pro rozsáhlejší data se používají rafinovanější, byť na klasických metodách založené algoritmy.

- Pro množinu číselných objektů . . . a daný počet shluků . . . najděte shluky metodou k-means

Objekty jsou { 2, 4, 10, 12, 3, 20, 30, 11, 25 }
k = 2, tj. chceme 2 shluky
Zvolme inicializaci středu např. $m_1 = 2$, $m_2 = 4$ a pracujme s euklidovskou vzdáleností.
Přiřadíme objekty shlukům:
K_1 = {2, 3 }; K_2 = { 4, 10, 12, 20, 30, 11, 25 }
Vypočteme:
$m_1 = 2{,}5$; m_2 = suma/7 = 112/7 = 16
Znovu přiřadíme objekty shlukům:
K_1 = {2, 3, 4 }; K_2 = { 10, 12, 20, 30, 11, 25 }
Vypočteme:
$m_1 = 3$; m_2 = suma/6 = 108/6 = 18
Znovu přiřadíme objekty shlukům:
K_1 = {2, 3, 4, 10 }; K_2 = { 12, 20, 30, 11, 25 }
Vypočteme:
$m_1 = 4{,}75$; m_2 = suma/6 = 98/5 = 19,6
Znovu přiřadíme objekty shlukům:
K_1 = {2, 3, 4, 10, 11, 12 }; K_2 = { 20, 30, 25 }
Dokonvergováno – hotovo.

- Definujte základní vlastnosti rozhodovacích stromů

Dána databáze $D = \{t_1, t_2, \ldots, t_n\}$, kde $t_i = < t_{i1}, t_{i2}, \ldots, t_{ih} >$, množina tříd $C=\{C_1, C_2, \ldots, C_m\}$ a databázové schéma obsahující atributy $\{ A_1, A_2, \ldots, A_h \} = A$. Hodnoty atributu A_i jsou z $\{a_{i1}, a_{i2} .. a_{iv}\}$ Tj. máme n záznamů, m tříd, h atributů, i-tý atribut má iv hodnot
Rozhodovací strom pro D má pak vlastnosti:
- Každý vnitřní uzel je označen atributem
- Každá hrana je označena predikátem jehož proměnnou je atribut rodičovského uzlu
- Každý list je označen třídou

Výhody rozhodovacích stromů:
- Snadno a efektivně použitelný
- Umožňuje generovat snadno interpretovatelná pravidla
- Velikost stromu nezávisí na rozsáhlosti dat
- Čas nutný pro klasifikaci je přímo úměrný fixní hloubce stromu

- Popište způsob výběru atributu pro rozvětvení rozhodovacího stromu

Základním problémem vytváření stromu je výběr pořadí atributů pro větvení. Cílem je minimalizovat počet porovnávání. Zásada: Ptát se otázkou, odpověď na kterou poskytne nejvíce informace. Obdoba hry „Zjisti otázkami jakou věc nebo osobu si myslím".

Výběr atributu pro testování (větvení) pomocí informačního zisku (redukce entropie)
Je dána: - množina tříd $C = \{ C_1, C_2, \ldots, C_m \}$
- množina záznamů (vzorků) S mohutnosti s, rozdělených dle příslušnosti ke třídě na disjunktní množiny $S_1, S_2, \ldots, S_m$, o mohutnostech $s_1, s_2, \ldots, s_m$.

- množina atributů A

Informace (v počtu bitů) potřebná k rozdělení S na m tříd je

$$I(s_1, s_2, \ldots, s_m) = \sum_{i=1}^{m} p_i \log(1/p_i) = -\sum_{i=1}^{m} p_i \log(p_i)$$

kde p_i je pravděpodobnost, že záznam z S patří do třídy C_i . $p_i = s_i / s$

Nechť atribut A_x má hodnoty $\{a_{x1}, a_{x2} .. a_{xv}\}$
Takže může rozdělit S na $S^1, S^2, \ldots, S^v$, kde S^j obsahuje ty prvky z S, které mají hodnotu $A_x = a_{xj}$
S^j korespondují větvím vycházejícím z uzlu, který obsahuje S

Nechť s_{ij} je počet záznamů z S^j, které patří do třídy C_i
Entropie založená na rozvětvení podle A_x je

$$E(A_x) = \sum_{j=1}^{v} ((s_{1j} + \ldots + s_{mj}) / s) * I(s_{1j}, \ldots, s_{mj})$$

Čím menší je hodnota entropie, tím lepší rozdělení atribut dovoluje.
Informační zisk se spočte pro každý atribut. Vybere se ten s největším ziskem.

- Diskutujte odlišnosti klasifikace rozhodovacími stromy a pravidly

Problém u rozhodovacích stromů pro rozsáhlé databáze je, že se trénovací data S nevejdou do paměti. Řešení: rozdělit S na části a tvořit strom pro každou část separátně. Pak je zkombinovat do finálního stromu (přesnost klasifikace se sníží); diskově rezidentní struktury. Např. pro každý z atributů seznam tvaru (hodnota atributu, třída, identifikátor záznamu) Po rozvětvení uzlu jsou rozděleny i seznamy atributů mezi potomky uzlu.

Rozhodovací pravidla
If – then pravidla r = <a, c> antecedent a, consequent c
Korespondují rozhodovacím stromům s rozdíly:

- oproti stromům neimplikují pořadí rozdělování (nemají pořadí)
- při generování pravidel je uvažována v daném okamžiku vždy jen jedna třída (u stromu všechny).

Lze je generovat snadno ze stromu (jedno pravidlo pro každou z cest od kořene k listu - evidentní).
Existují i metody bez vytvoření stromu:

- 1R vytváří strom hloubky 1 výběrem atributu, který způsobí nejméně chyb v trénovacích datech (nemá 100% přesnost)
- PRISM pro každou z tříd C_j zkoumá postupně dvojice (atribut, hodnota) a jako vhodnou do antecedentu vybírá dvojici, která ze vzorků S obsáhne největší počet patřících do třídy C_j. Pokud neobsáhne celou třídu, hledá další vhodnou dvojici (atribut, hodnota) a přidá ji do antecedentu.

Existují metody generování pravidel z neuronové sítě.

- Popište základní princip Bayesova klasifikátoru (kNN klasifikátoru)

Bayesův klasifikátor:
Vychází z Bayesova teorému $P(H \mid X) = \frac{P(X \mid H)\ P(H)}{P(X)}$

Kde:
X je datový vzorek, jehož třídu neznáme
H je hypotéza (že X patří k určité třídě)
P(H | X) je *posteriorní pravděpodobnost* platnosti H pro vzorek X
P(H) je *priorní pravděpodobnost* hypotézy H
P(X | H) je *posteriorní pravděpodobnost* , že se jedná o X, platí-li H

knn klasifikátor (k- Nearest Neighbors):
Založen na vzdálenosti záznamů v n-rozměrném prostoru numerických atributů. Klasifikátorem jsou sama trénovací data. Zkoumaný záznam se zařadí do třídy, která přísluší většině z K nejbližších záznamů trénovací množiny.

- Definujte pojem „přesnost klasifikace"

Hodnocení kvality klasifikace
Označme:

a	počet záznamů správně zatříděných klasifikátorem do třídy C
b	počet záznamů nesprávně zatříděných klasifikátorem do třídy C
c	počet záznamů patřících do třídy C, které klasifikátor nezařadil do C
d	počet záznamů nepatřících do třídy C, které klasifikátor nezařadil do C

Zavádí se míry:

citlivost = $\frac{a}{a+c}$ specifičnost = $\frac{d}{b+d}$ přesnost = $\frac{a}{a+b}$

accuracy = $\frac{a+d}{a+b+c+d}$

Míry vyhodnocované makroprůměrováním/mikroprůměrováním

Další kriteria: rychlost, robustnost (šumy), schopnost práce s rozsáhlými daty, interpretovatelnost výsledků

Textové databáze

- Definujte pojmy přesnost a úplnost vyhledávání dokumentů

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)

- Popište rozdíl mezi invertovaným a signaturovým souborem

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.

signaturový soubor: zakódování dokumentu do 0/1, signatura S je d bitový řetězec přiřazený dokumentu resp. dotazu $\mathbf{S_Q}$ **AND** $\mathbf{S_D} = \mathbf{S_Q}$, pak D je částí odpovědi
Vytváření S:

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

ad 2 př.

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

- Popište princip a vlastnosti booleovského modelu textové databáze (vektorového modelu)

Boolovský 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).

př. Oracle SQL Text retrieval

```
SELECT seznam položek
FROM  seznam tabulek
WHERE položka CONTAINS textový výraz
```

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)

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).
Podobnost dokumentu s dotazem nerespektuje závislost vah termů na délce dokumentu
Zdokonalení: Podobnost dokumentu Di s dotazem Q lze vyjádřit kosinovou mírou.
pozn.
- jiné možné míry (Dice) - jsou to heuristiky (kromě kosinové)
- v případě binární reprezentace vah termů Sim = počet shodných termů dokumentu a dotazu
- ladění dotazu = změna vah termů dotazu (interpretace nejasná)
- nelze odlišit konjunktivní a disjunktivní dotaz
- nerealizovatelná operace not

- Jaké vlastnosti termů zohledňují TF IDF váhy termů pro indexaci

Volba termů a jejich vah pro indexaci
TF_{ij} frekvence termu j v dokumentu i
$NTF_{ij} = ((TF_{ij} / \max TF_{ik}) / + 1) / 2$ - normalizovaná frekvence, kde $\max TF_{ik}$ je maximální frekvence termu v řádku i matice D

DF_j počet dokumentů s termem j
$IDF_j = \log (m / DF_j) + 1$ - inverzní frekvence, Kde m je celkový počet dokumentů v kolekci D

Váha termu:
$w_{ij} = TF_{ij} * IDF_j$ nebo
$w_{ij} = NTF_{ij} * IDF_j$

- Co rozumíme pod pojmem kompaktnost a vrstevnatost hypertextového dokumentu

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

Prestiž uzlu i - suma konečných hodnot řádku i mínus 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)

- Web content mining (analýza obsahu) – cíl, využití, techniky

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)

Web Data Mining Query Language (WebML)
SELECT * FROM dokument in "www.zcu.cz" WHERE ONE OF keywords COVERS "database";

- Web usage mining (analýza logů) – cíl, využití, techniky

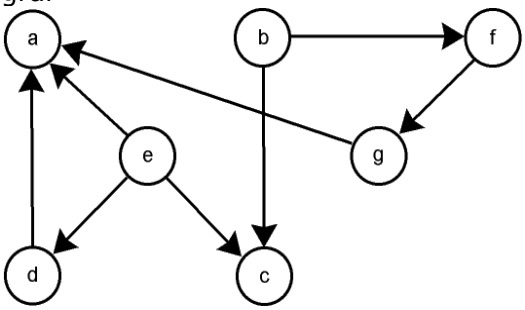
Vyhledání vzorů přístupu uživatelů ke stránkám z Web log záznamů.
Web server zaznamenává: požadované URL, IP adresu odkud pochází žádost o přístup, timestamp.

Využití ke zlepšení návrhu stránek (strukturu, návštěvnost), efektivity zpracování (page caching, swapping),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))

- Web structure mining (analýza topologie web stránek) – cíl, využití, techniky

graf matice sousednosti



matice X (sousednosti vrcholů)

	a	b	c	d	e	f	g
a	0	0	0	0	0	0	0
b	0	0	1	0	0	1	0
c	0	0	0	0	0	0	0
d	1	0	0	0	0	0	0
e	1	0	1	1	0	0	0
f	0	0	0	0	0	0	1

g 1 0 0 0 0 0 0

Hodnoty (p,q) v matici XX^T odpovídají počtu stránek, na které odkazují stránky p a q zároveň. Tato hodnota může sloužit ke stanovení, kolik mají stránky p a q společného.
Hodnoty (p,q) v matici X^TX udávají počet stránek, které odkazují jak na stránku p, tak na q. Tato informace může měřit, kolik autorů považuje tyto stránky za podobné.
Hodnoty diagonály (p,p) matice XX^T reprezentují počty stránek, na které je ze stránky p odkaz (out-degree). V teorii grafů je tento počet nazýván polostupněm výstupu uzlu a je možno ho zjistit i z původní matice sousednosti jako součet hodnot v p-tém řádku.
Hodnoty diagonály (p,p) matice X^TX udávají, z kolika dalších stránek existuje odkaz na stránku p (in-degree). V teorii grafů toto číslo nazýváme polostupněm vstupu uzlu a též je ho možné zjistit z původní matice sousednosti jako součet hodnot v p-tém sloupci matice.

- Popište metodu HITS

Hledání vysoce relevantních stránek (pro dané téma) *AUTHORITIES*
Hledání stránek obsahujících kolekce odkazů na autority *HUBS*

- 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
  $a_p = \Sigma h_q$ (q takové, že $q \rightarrow p$)
  $h_p = \Sigma a_q$ (q takové, že $q \leftarrow p$)

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.

- Popište metodu PageRank (Google)

Důležitá stránka: vede k ní mnoho odkazů, odkazují na ní vysoce ohodnocené stránky.
Nechť
u je webová stránka,
F_u je množina stránek, na které stránka u odkazuje a
B_u je množina stránek, které odkazují na u,
$N_u = |F_u|$ je počet odkazů z u
c je konstanta používaná pro normalizaci, zajišťující konstantní celkové ohodnocení všech stránek

$$R(u) = c \sum_{v \in Bu} \frac{R(v)}{N_v}$$

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