DATABÁZOVÉ TECHNOLOGIE (DB) – PŘÍPRAVA NA SZZ

1. Relační model dat, relační algebra, tvorba dotazů pomocí relační algebry

Existují 2 modely dat k dané struktuře – síťový (založen na principu spojky mezi daty) a relační.

V relačním modelu dat se na data díváme jako na relaci – založen na matematickém principu relace (teorii množin). Sdružuje data do tzv. relací (tabulek), které obsahují n-tice (řádky).

Jedná se o nejrozšířenější způsob uložení dat v databázi v logickém smyslu, nejčastěji využíván v komerčním SŘBD.

Tabulka, tvořící základ relační DB, je struktura záznamů s pevně stanovenými položkami (sloupci-atributy). Každý sloupec má definován jednoznačný název, typ, rozsah (neboli doménu). Relační DB tvoří kolekce tabulek, jejich vztahů, indexů atp. Relační model umožňuje:

- Přirozenou reprezentaci dat
- Možnost snadné definice
- Jednoduché zpracování vazeb
- Klade důraz na zachování integrity dat (stav kdy přečtená data jsou totožná s přečtenými a jsou kompletní)
- Nezávislé uložení dat

V relačním modelu dat se zavádí pojmy referenční integrita, cizí a primární klíč, normální forma aj. S relačními DB je úzce spojen jazyk SQL (strukturovaný dotazovací jazyk)

Relační algebra

Relační algebrou rozumíme dvojici RA = (R,O), kde nosičem R je množina relací a O je množina operací, která zahrnuje:

- Tradiční množinové operace jako sjednocení, průnik, rozdíl, součin
- Speciální relační operace jako projekce, selekce, spojení a dělení.

Operace pro práci s daty

Pro operace s daty byly definovány 3 operace: projekce, selekce, spojení.

- Projekce relace R s atributem A na množině atributů B, kde B je obecnou podmnožinou A – operace vytvoří relaci se schématem B a prvky, které vzniknou z původní relace odstraněním hodnot atributů A\B. Odstraněny jsou i případné duplicitní prvky (řádky).
- Selekce relace s atributy A podle logické podmínky – operace vytvoří relaci s týmž schématem a ponechá ty prvky z původní relace, které splňují podmínku. Formule podmínky je boolovský výraz.
- Spojení relací R a S se schématy (atributy) A, resp. B – operace vytvoří relaci se schématem A v B, jejíž projekce na A je relace R a projekce B je relace S.

Projekce (SELECT a,b …)

- Zúžení relace R na sloupce s atributy (vybírá sloupce)
- Vstupem operace je jedna relační tabulka a seznam atributů.
- Výstupem je tabulka obsahující jen sloupce uvedené v seznamu atributů a jen neopakující se řádky (unikátní) (SQL92 SELECT – operace projekce neodstraňuje duplicitní záznamy a tak může vybrat SELECT sloupce vícekrát. K odstranění duplicitních záznamů použít DISTINCT)

Selekce (SELECT… WHERE…)

- Selekce je zúžení relace R na řádky vyhovující dané podmínce.

- Značí se R(podmínka), př. S = Student(město='Plzeň')

Spojení (… JOIN…)

- Výsledkem spojení obou tabulek je jen jedna tabulka, která obsahuje všechny sloupce ze spojovaných tabulek a počet řádek je roven součinu řádků první a druhé tabulky.
- Značí se R○S, př. spojení R[A ⊗ B]S.

Z kartézského součinu jsou vybrány pouze ty n-tice, které splňují Θ podmínku

- Θ spojení vytvoří poměrně značný počet prvků, což jsou všechny prvky z R a S, pro které platí spojení
  - Speciálním typem Θ spojení přes rovnost
- Přímé spojení přes rovnost, kdy se jeden z těchto dvou atributů vypustí
  - Obvykle, když provádíme spojení, tak relace R a S mají stejné atributy
- Kompozice je spojení, kde se vypustí oba atributy, kterými se srovnává (v SQL: NATURAL JOIN)

Mezi základní typy spojení v SQL patří

- přirozené spojení = *natural join* – při výběru více tabulek je nutné z důvodu jednoznačnosti udávat plně kvalifikované názvy sloupců. Spojení je realizováno přes rovnost primárního klíče (tituly.id) a cizího klíče (pisne.id)
- polospojení = *semi-join* – lze si jej představit jako operaci selekce, kde podmínka selekce je závislá na existenci dat druhé tabulky. Proto je tento typ spojení realizován pomocí operátoru EXISTS, který nabývá hodnoty TRUE, pokud vnořený dotaz poskytne nějaká data. Je možné i NOT EXISTS.
- self-join –specifický typ přirozeného spojení, které spojuje jednu a tu samou tabulku. Toho se nejčastěji využívá při evidenci zaměstnanců, kde každý zaměstnanec má svého nadřízeného zaměstnance.

2. Závislost atributů, primární klíč, 1. NF, 2. NF, 3. NF, bezztrátová dekompozice

Problém: Mám relaci R (přednáška, učitel, místnost, hodina, student, známka)

Relace navržena nevhodně – např. obsahuje duplicity – u předchozí relace tomu tak je, protože přednáška, učitel, místnost, hodina se musí opakovat tolikrát, kolik studentů přednášky navštěvuje.

Řešením tohoto problému se nazývají normální formy.

Závislost atributů

Závislostí atributů se musíme zabývat při zjišťování normálních forem, nebo při převodu na NF.

Funkční závislostí rozumíme vztah mezi atributy jedné relace. Funkční závislosti zjistíme z analýzy úlohy.

- Máme relaci R a v ní dvě množiny atributů A a B, pak existuje funkční závislost B na A, jestliže jedné A-hodnotě se přiřadí nejvýše jedna B-hodnota.
- Máme tabulku, ve které evidujeme rozvrh malé vysoké školy, kde každý předmět je učen právě jedním učitelem a každý učitel je natolik vytížen, že učí jen jeden předmět:

Předmět	Učitel	Kroužek	Místnost	Čas
Matematika	Novák	A4-01	PC406	Út3
Matematika	Novák	A4-01	UL411	Po9
Matematika	Novák	E3-15	PC406	Út3
Matematika	Novák	E3-15	UL411	Po9

Dějepis	SUKOVÁ	S1-07	AM701	St5
Dějepis	SUKOVÁ	A4-01	AM701	St5
Dějepis	SUKOVÁ	S1-87	AM701	St5
Dějepis	SUKOVÁ	K3-05	AM701	St5

- Z ukázky dat (relace) nelze dokázat, že platí nějaká funkční závislost (nemusí být zobrazeny některé možnosti, které potom vedou k tomu, že funkční závislost mezi určitými množinami atributů neexistuje), ale naopak negativní fakta mohou být z dané relace zjistitelná, tvoří totiž protiklad. Kladná fakta víceméně určíme z obecných znalostí a podmínek, které se daného případu týkají.
- Atribut *Učitel* **je** funkčně závislý na množině atributů {*Místnost-Čas*}, protože pro každou hodnotu množiny atributů {*Místnost-Čas*} existuje nejvýše jedna hodnota atributu *Učitel*. To jinými slovy znamená, že v jedné místnosti ve stejnou dobu nemohou učit dva učitelé.
- Atribut *Předmět* **je** funkčně závislý na množině atributů {*Místnost-Čas*}.
- Atribut *Učitel* **není** funkčně závislý na atributu *Čas*.

Pohled ze cvičení DB2: Vybereme 2 podmnožiny atributů A a B. Ty jsou funkčně závislé, pokud když vybereme atributy A, určíme atributy B.

Primární klíč

Pomocí závislostí lze definovat klíč. Na klíči jsou závislé všechny ostatní atributy relace.

Je sloupec, který jednoznačně identifikuje záznam. To znamená, že tento sloupec musí být u každého záznamu tabulky jiný. Primární klíč se využívá k vytváření relací mezi tabulkami.

Normální formy

Používají se pro lepší návrhy databázových systémů. Obecně platí, že čím je tabulka ve vyšší normální formě, tím je kvalitněji navržena.

Existuje i 0. NF, ve které se tabulka nachází, existuje-li v ní alespoň jedno pole, které obsahuje více, než jednu hodnotu (např. sloupec Adresa obsahuje Brno, Hluboká 3). Není- li tabulka v 0. NF, pak je alespoň v 1. NF.

Jmeno	Adresa
Petr Kaláb	Brno, Hluboká 3
Žofie Nová	Brno, Kamenná 15
Patricie Zrzavá	Břeclav, Dolní 48

1. NF

Relace R (tabulka) se nachází v 1. NF, pokud všechny komponenty n-tice jsou atomické (tj. nejsou opět relací). Tzn. do každého pole lze dosadit pouze jednoduchý datový typ. Podmínka 1. NF není splněna například u tabulky, kde je jméno a příjmení (nebo datum a čas) v jednom sloupci a přitom aplikace pracuje s těmito položkami jako samostatnými.

Prijmeni	Jmeno	Město	Ulice
Kaláb	Petr	Brno	Hluboká 3
Nová	Žofie	Brno	Kamenná 15
Zrzavá	Patricie	Břeclav	Dolní 48

2. NF

Relace R je v 2. NF, pokud je v 1. NF a pokud každý neklíčový (který nepatří žádnému klíči) atribut silně závisí na celém primárním klíči relace R. Pokud je klíč jednoduchý atribut, je tabulka automaticky ve 2. NF

Pravidlo NF tedy říká, že všechny objekty obsahující sloupce s duplicitními položkami, které mezi sebou vytvářejí částečné závislosti, je třeba rozdělit do objektů nových, v nichž bude každý záznam uložen pouze jednou.

Př.

ČÍSLO	JMÉNO	PŘÍJMENÍ	ČÍS_PRAC	NÁZEV_PRAC
1	jan	novák	10	studovna
2	petr	nový	15	centrála
3	jan	nováček	10	studovna

Není ve 2. NF, protože zvolíme-li jako primární klíč dvojici ČÍSLO, ČÍS_PRAC, pak atribut NÁZEV_PRAC je závislý jen na části primárního klíče a sice pouze na ČÍS_PRAC. Nutná následující dekompozice:

ČÍSLO	JMÉNO	PŘÍJMENÍ	ČIS_PRAC
1	jan	novák	10
2	petr	nový	15
3	jan	nováček	10

ČÍSLO	NÁZEV
10	studovna
15	centrála

3. NF

Relace je ve 3. Normální formě, pokud je v 2.NF a každý neklíčový atribut není tranzitivně závislý na žádném klíči (neboli mezi neklíčovými atributy nesmí existovat jiná než triviální závislost). Tranzitivní závislost je taková závislost, mezi minimálně 2 atributy a klíčem, kde jeden atribut je funkčně závislý na klíči a druhý atribut je funkčně závislý na prvním.

Tranzitivnost = nezávisí přímo na klíči, ale silně závisí na atributu, který také není klíčem a ten teprve slabě závisí na klíči. To je nutné odstranit.

Jinak řečeno 3. NF pravidlo vyžaduje důsledné odstranění a oddělení veškerých dat, která nejsou v přímém vztahu s daným objektem.

Př. Primární klíč je číslo

ČÍSLO	JMÉNO	PŘÍJMENÍ	FUNKCE	PLAT
1	jan	novák	technik	15000
2	petr	nový	vedoucí	21500
3	jan	nováček	správce	17500

Nesplňuje 3. NF, protože PLAT závisí na FUNKCE, v tomto příkladu to bohužel není moc vidět, ale je to tak. Existuje tedy závislost mezi neklíčovými atributy. Provede se tedy dekompozice.

ČÍSLO	JMÉNO	PŘÍJMENÍ	FUNKCE
1	jan	novák	technik
2	petr	nový	vedoucí
3	jan	nováček	správce

FUNKCE	PLAT
technik	21500
vedoucí	17500
správce	15000

Dalším příkladem by mohla být tabulka :

- *OBJEDNAVKY(ID_Zakaznik, ID_Objednavky, ObjednavkaDatum, DodavatelNazev).*

Ta obsahuje sloupce *DodavatelNazev,* který ale přímo nesouvisí s daným objektem *OBJEDNAVKY* (nepopisuje žádnou jeho vlastnost) a proto jej podle pravidel 3. NF vyčleníme do nového objektu (tabulky). Vytvoříme tabulku DODAVATEL a mezi tabulkami DOADAVATEL a OBJEDNAVKA vytvoříme relaci 1:1

Další normalizační formou je Boyce-Codova NF, 4. NF, 5. NF.

V praxi často stačí ale 3. NF, která na rozdíl od BCNF se už nezabývá závislostmi mezi klíči. Také bychom měli zvážit normalizaci, zda není přehnaná, abychom pak neměli velké množství 2 sloupečkových tabulek). Přechod do 2. Resp. 3. NF znamená v ERA modelu přidat entitní množinu připojenou vazbou 1:N.

Normalizace

Postupná transformace tabulky do vhodnějšího tvaru (postupná dekompozice). Žádoucí vlastnosti takovéto dekompozice jsou pak:

- Bezztrátovost při zpětném spojení
- Zachování závislostí
- Odstranění opakování informace (redundance)

Bezztrátová dekompozice:

- spojení tabulek, které vzniknou dekompozicí, musí dát přesně původní tabulku.

3. Základní principy transakčního zpracování, žurnál, základní způsoby použití žurnálu, konzistence dat

Transakce

Transakce je skupina příkazů (posloupnost operací nad objekty DB), která spolehlivě a nezávisle na jiných transakcích převedou DB z jednoho konzistentního stavu do druhého. Transakce realizuje jednu ucelenou operaci z pohledu uživatele, říkáme, že je logickou jednotkou práce a je také jednotkou zotavení z chyb.

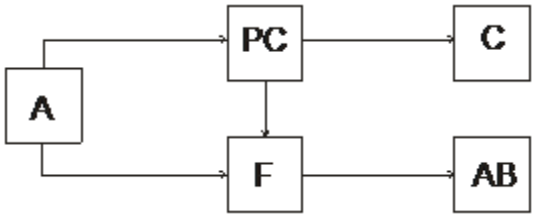
Př. transakce – převod peněz z účtu na účet – 2 zápisy do DB.

Během provádění transakce není DB v konzistentním stavu. Pokud nemůže systém z nějakého důvodu DBzi aktualizovat, vrátí ji transakce do takového stavu, v jakém byla před zahájením transakce. Platí, že se transakce provede celá, nebo se neprovede vůbec. Bod od kterého lze transakci považovat za úspěšnou, nazveme bod potvrzení.

Transakce je možné do sebe vnořovat. Pořizuje se kopie DB v kontrolním bodu (checkpoint). Od kontrolního bodu se zapisují změny do žurnálu.

Transakce se může dostat do jednoho z 5 stavů:

- A = aktivní – vznik transakce, provádění
- F = chybný – nelze dál pokračovat
- PC = částečně potvrzený – provedení poslední elementární transakce – transakce nebyla potvrzena
- C = potvrzení (commit)
- AB = zrušení – objekty se uvedou do stavu před započetím transakce.



Vlastnosti transakcí

Existují tzv. ACID vlastnosti:

- Atomicita – transakce musí proběhnout buď celá, nebo vůbec ne
- Konzistence – transformuje DB z konzistentního stavu do jiného konzistentního stavu
- Nezávislost – dílčí efekty jedné transakce nejsou viditelné jiným transakcím
- Trvanlivost – efekty úspěšně ukončené (potvrzené) jsou trvale uloženy.

Žurnál

Žurnál je log operací. Než něco zapíšu do DB, zapíšu to nejdřív do žurnálu.

Slouží k zaznamenávání realizovaných transakcí, resp. Všech změn prováděných v DB, pro bezproblémové zotavení systému v případě chyby. Do žurnálu se zapisují změny od kontrolního bodu transakce.

Jedná se o tabulku (soubor), v níž se udržují následující data:

- Transakce, objekt transakce
- Kdo spustil transakci
- Kdy spustil transakci
- Nová hodnota objektu
- Stará hodnota objektu
- ID transakce je obdoba ID procesu

Způsoby použití žurnálu

- REDO – Obnovený zápis, po chybě se vezme konzistentní stav DB z checkpointu, beru žurnál krok za krokem a provádím transakce. Řeší se tím globální chyby
- ROLL-BACK, UNDO – zpětné použití žurnálu, vychází ze současného stavu DB. Využívá se žurnálu – zpětným odvíjením se dostanu až do konzistentního stavu. Má význam pouze pro nedokončené transakce.

Metody zpracování žurnálu jsou:

- Dvoufázové potvrzování (odkládaná aktualizace) – tvorba žurnálu s odloženými realizacemi změn.
- Transakce s ukládáním změn

Nevýhoda žurnálu je, že je objekt uzamčen příliš dlouho, nevýhoda pro systém s paralelním zpracováním transakcí. Alternativa žurnálu je:

- Přímý zápis do báze dat (okamžitá aktualizace) – tvorba žurnálu s bezprostředním zapsáním změn. Do žurnálu zapisuje staré hodnoty. Po obnově provádí rollback. Způsoby jsou:
    - Uzamčené objekty po celou dobu transakce – nevyžaduje ukládání starých hodnot
    - Umožňuje otevření po zapsání – lepší pro paralelní běhy – vyžaduje ukládat staré hodnoty – nevýhoda.

Konzistence dat

Konzistence dat je jen jinak řečeno integrita dat. Konzistentním stavem se rozumí klidový stav DB, který je ve shodě s definicí pravidel (např. jedinečnost, vzájemné vztahy v řádku nebo mezi tabulkami, omezující podmínky,…) daných již při návrhu DB v datovém modelu. Na zajištění konzistentního stavu slouží integritní omezení.

Konzistence předpokládá, že data distribuovaná do DB budou distribuovaná ve správném pořadí a jejich správná kopie.

Uložená data nesmí být ve sporu

Konzistentní stav je takový stav DB, kde hodnoty odpovídají nějaké možné variantě v reálném světě. Konzistence je prakticky i vyjmenování, jakých hodnot může položka nabývat.

4. Nedostatky relačního modelu

Je třeba se zamyslet, zda níže uvedené nedostatky jsou opravdu nedostatky a právě ne výhodami, které RDBMS proti ODBMS mají. Vždy záleží na typu aplikace, jakou reprezentaci dat bude vyžadovat.

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 jistého druhu
- Nepodporuje hledisko času a verzí objektu
- Předpokládá se velký počet relativně malých databázových objektů, typicky miliony s málo relacemi
- Vyjadřuje se spíš struktura dat, než jejich sémantika (význam)
- Impedance mismatch (impedanční nesoulad)

Impedance mismatch (impedanční nesoulad)

Jedná se o termín původně z elektrotechniky, který v softwarovém světě označuje rozdíl plynoucí z podstatné odlišnosti mezi relačním a objektovým modelem databáze.

- 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 vs. Relační model

- 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, obsahuje tedy 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 v relačním modelu

- Rozšířená relační technologie
- Vytvoření nového datového modelu
- Rozšíření OO programovacího jazyka o databázové prostředky
- Vytvořit rozšířené DBMS knihovny
- Zahrnutí konstrukcí OODB jazyka do konvenčního jazyka
- Vývoj aplikačně doménových prostředků používajících OODB technologie
- Použití frameworků na straně programovacího jazyka pro vyřešení impedančního nesouladu.

5. Vlastnosti objektového, konceptuálního a fyzického modelu

Konceptuální model

Zachycuje pouze vztahy a struktury dat samotných. Zabývá se modelováním reality, není ovlivněna budoucími prostředky řešení. Popisuje sémantiku (význam) domény (oblast zájmu), pro kterou se model vytváří. Skládá se z entitních tříd a vztahů mezi entitami (ER model, class diagram). Je nezávislý na databázi a na implementaci a definuje omezení kladená na data.

- Je to jednotný centrální popis informačních obsahů, které mohou být v datových základnách.
- Říká, co musí systém dělat, aby zajistil uživatelovy požadavky.
- Vymezuje, co budeme v datové základně sledovat, aniž obsahuje údaje o tom, jak budeme tyto předměty v datové základně realizovat.

Objektový model

Vychází z konceptuálního modelu. Uplatňuje objektové prvky jako např. dědičnost, polymorfismus atd.

Definuje chování systému pomocí metod nad daty. Dále viz otázka č.4.

Objektový přístup k návrhu – objekty stejného typu mají stejné chování a stejnou množinu stavů:

- Metody (funkce, uložené procedury, triggery)
- Složité struktury datové
  - Seznamy, pole, množiny, strukturované hodnoty
  - Hnízděné tabulky
- Abstraktní datové typy

Fyzický model

Představuje fyzické uložení dat pro konkrétní technologii, např. DB schéma pro systém Oracle 10g. Programátor je odstíněn od tohoto návrhu typicky vrstvou SŘBD. Model fyzického uspořádání dat (vyjadřuje jejich uložení na konkrétním médiu).

Dále Logický model

Určuje, jak je konceptuální struktura dat implementována v konkrétním techniko-programovém prostředí.

Logické databázové modely jsou: lineární, hierarchický, síťový, relační, objektově orientovaný

Využití: údaje potřebné pro projektanta při algoritmizaci a programování.

6. CASE systémy – klasifikace, vlastnosti, použití

CASE = Computer aided software (system) Engineering.

Modely softwarových systémů jsou příliš složité a je nutná podpora různých úrovní abstrakce, různých pohledů a nutnost rozdělení mezi jednotlivé vývojáře.

CASE nástroje slouží pro standardizaci používaných postupů. Jsou to nástroje na podporu práce analytiků a programátorů při vývoji informačních systémů, zejména ve fázi analýzy a návrhu – tvorba modelů. CASE nástroje tvoří mezistupeň mezi analýzou problému a programováním.

CASE nástroje je označení pro integrovanou tvorbu programových aplikací pomocí programových prostředků s minimální potřebou manuálního psaní zdrojového kódu programu.

- **Obsah:** databáze (repository, systémová encyklopedie) navrhovaných prvků informačního systému
- **Funkce:** Podpora realizace projektu informačního systému
- **Základ:** metodika = návod na vytváření modelů a určení závislostí mezi nimi.

Klasifikace

CASE systémy se dělí na následující:

- **Nižší CASE – podpora tvorby software**
  - Návrhy obrazovek, formulářů, menu. Jazyková podpora
- **Vyšší CASE – podpora analýzy návrhu**
  - Tvorba diagramů a modelů. Kontrola konzistence modelu. Př. AxiomSys: strukturovaná analýza.
- **Integrované CASE – podpora živ. cyklu**
  - Do analýzy po generování kódu. Round trip engineering. Podpora tvorby dokumentace. Př. Oracle designer
- **Komponentové CASE – otevřenost**
  - Repository s rozhraním (SCM, testování). Integrace nástrojů od různých výrobců. Př. Rational Suite Enterprise.

Dělení CASE dle rozsahu možností

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

Vlastnosti CASE

Kladné

- **Zvýšení produktivity** – automatizace prací – čas na podstatné věci. Lepší přehled o modelu a implementaci. Podpora rozdělení práce. Snazší údržba dokumentace i systému.
- **Zvýšení kvality –** Podpora analýzy – lepší znalost požadavků. Kontroly konzistence a úplnosti. Synchronizace reality a dokumentace. Podpora používání standardů

Záporné

- **Cena –** CASE jsou drahé (řádově 100 000 Kč). Vybírat s rozvahou (reklamní slogany vs. skutečné možnosti vs. Skutečné potřeby)

- **Doba návratnosti investice –** Na počátku potřebné zaškolení a zaučení. Přínosy obvykle až od 2/3 projektu
- **Změna stylu práce –** Nástroj bez používání metodiky je k ničemu. Nutnost pořídit si CASE (techniky, metoda). Podpora managementu nutná.

Použití CASE systémů

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

7. Objektově relační databáze – objektové vlastnosti jazyka SQL99

Za posledních 25 let je mohutný trend přechodu od strukturovaného k objektově orientovanému programování, a to i v oblasti zpracování dat a databází. Objektově orientované DB se objevili v 90. Letech minulého století a kladou si za cíl urychlit a ulehčit práci s daty. Situace ovšem není jednoduchá, protože relační a objektový přístup je od základu rozdílný. Existuje tak mnoho výhod i mnoho nevýhod pro relační i objektové DB. Na současném trhu existují 3 základní typy:

- Relační DB (Relational Database Management System, RDBMS) – výkonné na tradičních datech. Př. Oracle 7.x, DB2.
- Objektově-relační (Object Relational Database Management t System, ORDBMS) – Poskytuje programátorské pohodlí, rychlý a udržitelný vývoj aplikací. Př. Oracle 8.x, 9.x, 10.x.
- Objektové DB (Object Database Management system, ODBMS) – Výkonné na netradičních datech, slabší v databázových rysech. Př. Jasmine, Gemstone, O2.

Objektově relační databáze

Objektově relační DB se snaží přinést objektové rysy do relačních databází. ORDBMS je specifikována v rozšíření SQL standardu – SQL3. Do této kategorie patří např. Informix, IBM, Oracle a Unisys.

Objektově relační mapování přináší vrstvy mezi OO aplikací a SQL databází. Charakteristiky jsou:

Myšlení v objektech, ev. Cache objektů, zodpovědné za persistenci.

Datový model ORDBMS

Rozšiřují datový model tak, že přidávají objektovost do tabulek. Všechny trvalé informace jsou stále v tabulkách, ale některé položky mohou mít bohatší datovou strukturu (porušení 1. NF), nazývanou abstraktní datové typy, tzv. ADT:

- ADT je datový typ, který vznikne kombinací základních datových typů
- Podpora dědičnosti, polymorfismu, referencí a integrace s programovacím jazykem je omezená
- Funkce a operace jsou asociované s novými datovými typy, mohou být použity k indexování, ukládání a získávání záznamů na základě obsahu nového datového typu.

Dotazovací jazyk

ORDBMS podporují rozšířenou verzi SQL – SQL3 (SQL 99), důvodem je podpora objektů (tj. dotazy obsahující atributy objektů). ORDBMS je stále relační, protože data jsou uložena v řádcích a sloupcích tabulek a SQL, včetně zmíněných rozšíření. Typická rozšíření zahrnují dotazy obsahující vnořené objekty, atributy, abstraktní datové typy a použití metod.

Jazyk SQL s rozšířením pro přístup k ADT je stále hlavním rozhraním pro práci s DB.

Objektové vlastnosti SQL99 (SQL3)

Objektové rozšíření standardizované v SQL99 zahrnuje:

- Strukturované uživatelské typy – Oracle od verze 9.i včetně jednoduché dědičnosti. Mohou být organizovány do hierarchie s děděním. Chování uživatelem definovaných typů je realizováno pomocí procedur a funkcí a (metod u ADT). Jedná se o řádkové typy, ADT, odlišující typy.
- Pole s proměnnou délkou (VARRAY) – CREATE TYPE typ AS VARRAY(5) OF VARCHAR(15)
- Hnízděné tabulky – typ TABLE
- Typ REF – odkaz ukazatel – jeho obsahem je OID nějakého záznamu, nelze s ním manipulovat jako s hodnotou, ale jako s odkazem. Zajišťuje objektovou identitu.
    - CREATE TYPE TypHerec AS (jmeno CHAR(30), nejlepsiFilm REF (FilmTyp))
    - Dereference př. SELECT Film->titul FROM hrajev WHERE herec->jmeno='Chaplin';

SQL99 je kompatibilní s existujícími jazyky a další vlastnosti jsou:

- Řádkové typy (jsou typem relace)
    - Vytvoření řádkových typů:
      ```
      CREATE ROW TYPE typadresa (ulice CHAR VARYING(50), mesto CHAR VARYING(20));
      ```
    - Příklad tvorby tabulky s řádkovým typem:
      ```
      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';
      ```
    - Lze definovat i podtypy a podtabulky CREATE TYPE typSekretarka UNDER typ ZAMESTNANEC AS(...
- Abstraktní datové typy (jsou typem atributu relace) – umožňují zapouzdření atributů a operací (na rozdíl od řádkových typů). Hodnoty jejich typů mohou být umístěny do sloupců tabulek.
    - Př.
      ```
      CREATE TYPE typZamestnanec AS (
              c_zam INTEGER,
              METHOD mzda() RETURNS DECIMAL);
      CREATE METHOD mzda ... FOR typZamestnanec BEGIN ... END
      ```
    - Instance vznikají konstruktorem jmenoTypu(), operátorem NEW jmeno hodnota, příkazem INSERT INTO osoby VALUES(...)
    - Funkce a procedury vyjádřeny v SQL/PSM (Persistent stored module), nebo C/C++, Java, ADA.... Jsou svázány s ADT. Metody jsou uložené ve schématu typu definovaného uživatelem. Metody se dědí. Metody i funkce mohou být polymorfní (liší se způsobem výběru). CREATE PROCEDURE zjisti_cenu ...; CALL zjisti_cenu(...);

- Odlišující typy (musí být FINAL) – emulace domén – strong typing
- OID – záznamy mají/mohou mít OID (v relačních DB mohou být použity jako primární klíče), které zajišťují objektovou identitu. V jiných záznamech atribut typu REF – odkaz ukazatel. Zpřístupnění klauzulemi REF IS SYSTEM GENERATED a REF IS USER GENERATED.

8. Dotazovací jazyk SQL – jazyky DDL, DML a DCL, vybrané příkazy těchto jazyků

SQL (Structured Query language) je založeno na relačních a SET operacích s určitými modifikacemi a rozšířeními. Typický SQL dotaz má následující strukturu:

SELECT A1, ..., An

FROM r1, r2, ..., rm

WHERE P

A reprezentuje atributy, r – relace, P je podmínka.

Takový dotaz má ekvivalent výrazu relační algebry. Výsledek SQL dotazu je relace.

SQL dotazy jsou deklarativní – říkají co se má udělat, ne jak se to má udělat. SQL se uplatní jako dotazovací jazyk, databázový programovací jazyk, jazyk pro správu databází, klient/server jazyk.

ANSI SQL 99 – norma definující, co mají umět jazyky pracující s SQL

- DDL – Data definition language
- DCL – Data control language (někdy i TCC – Transaction control commands)
- DML – Data manipulation language

SQL má definován ANSI/ISO standard, je sestavován pomocí ASCII textů v předem definované struktuře, ovšem ne každý SŘBD tento standard dodržují. Často jsou doplněny ještě o další nadstandardní příkazy.

DDL (Data definition language) – vytváříme tabulky, indexy, pohledy

Jazyk DDL definuje entity (množiny dat) = tabulky, jejich atributy = sloupce a vazby mezi nimi. Každý atribut je pojmenován a je mu přiřazen také datový typ (celé číslo, řetězec konstantní délky, desetinné číslo, datum a čas, měna, binární data…). Volitelně lze také u atributů nastavit různá omezení (uživatel nemůže zadat datum menší než je aktuální kalendářní rok. Různé DB systémy realizují DDL různě, aplikace často skrývají uživateli přímou manipulaci s DDL a poskytují mu komfortnější nepřímou práci s DDL pomocí formulářů.

Umožňuje popis nejen množiny tabulek, ale také informaci o jednotlivé tabulce (např. schéma každé tabulky, integritní omezení, fyzickou strukturu uložených dat na disku...)

Příkazy

DDL poskytuje prostředky pro vytváření (CREATE) a rušení (DROP):

- Databází (DATABASE), tabulek (TABLE), indexů (INDEX), vazeb a integritních omezení (CONSTRAINT), pohledů (VIEW).

A modifikaci struktury tabulek (ALTER TABLE).

Pomocí DDL lze tedy vytvářet, upravovat, doplňovat, mazat:

- CREATE – vytváření nových objektů
  CREATE TABLE osoby (os_cislo NUMBER(5) PRIMARY KEY, jmeno VARCHAR2(30) NOT NULL ...)
- ALTER – změny existujících objektů
  ALTER TABLE osoby ADD CONSTRAINT pk_osoby PRIMARY KEY (os_cislo)
- DROP – odstraňování objektů

Př. vytváření tabulek v SQL:

```
CREATE TABLE Ucitel (id integer PRIMARY KEY, jmeno varchar(50), prijmeni
varchar(100) NOT NULL, plat float DEFAULT 10000.00);
```

DCL (Data control language) – nastavujeme přístupová práva a řídíme transakce

Do této skupiny patří příkazy pro nastavování přístupových práv a řízení transakcí. Podmnožinou jsou tzv. TCC – Transaction Control Commands (jazyk pro ovládání transakcí). Příkazy řídící transakce se často oddělují do samostatné skupiny.

Příkazy pro řízení dat

- GRANT – příkaz pro přidělení oprávnění uživateli k určitým objektům
  GRANT SELECT ON potraviny TO max; CREATE ROLE zapi_tab; GRANT INSERT ON tab TO zapis_tab
- REVOKE – příkaz pro odnětí práv uživateli.
  REVOKE zapis_tab FROM jarda
- BEGIN - zahájení transakce.
- COMMIT – potvrzení transakce.
- ROLLBACK – zrušení transakce, návrat do původního stavu.

DML (Data manipulation language) - získáváme data a upravujeme obsah DB

DML je 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 DB do jiného stavu DB.

Další možností jazyků tohoto typu jsou jazyky pro manipulaci schématu DB, tj. definice změn schématu. Podle způsobu zpracování v logické struktuře rozlišujeme DML do dvou skupin:

- Navigační
- Specifikační

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

Akce

Lze uvažovat jednotlivě nebo jejich kombinace.

1. Přiřazení bodu zpracování v logické struktuře (např. Uzlu grafu, řádku tabulky, odpovídající relaci, apod.)
2. Získávání (zpřístupnění) části DB
3. Vložení (přidání) dalších dat do DB
4. Rušení (odebírání) dat z DB
5. Změna dat v DB

Výběry

Výběr části DB můžeme specifikovat následujícími způsoby:

1. Logickou pozicí (např. bodu zpracování) – získáme jak explicitně, tak implicitně
2. Specifikací hodnot dat podmínkou
3. Relatibilitou (vzájemnou souvztažností dat).

Příkazy pro manipulaci s daty

Příkazy pro získání dat z DB a pro jejich úpravy. Označují se zkráceně DML.

- SELECT – vybírá data z DB, umožňuje výběr podmnožiny a řazení dat

SELECT * FROM rozvrh WHERE semestr = ‘ZS‘;

- INSERT – vkládá do DB nová data

  INSERT INTO predmety VALUES (99, 'DS', 'KIV');

- UPDATE – mění data v DB (editace)

  UPDATE predmety SET kredity = 20 WHERE katedra = ‘KIV’ AND zkratka = ‘KP’

- DELETE – odstraňuje data (záznamy) z DB
- EXPLAIN PLAN FOR – speciální příkaz, který zobrazuje postup zpracování SQL příkazu. Pomáhá uživateli optimalizovat příkazy tak, aby byly rychlejší.

9. Integritní omezení, integrita databáze

Integrita databáze

Pod tímto pojmem rozumíme, že data v DB uložená jsou konzistentní (správná, úplná, nepoškozená). Lze zadávat pouze data, která vyhovují předem definovaným kritériím (např. musí respektovat datový typ nastavený pro daný sloupec tabulky). K zajištění integrity slouží integritní omezení. Jedná se o nástroje, které zabrání ztrátě, či poškození stávajících záznamů v průběhu práce s databází. Tímto způsobem je možné zajistit mazání dat, která již ztratila svůj význam. Např. smažeme-li uživatele, odstraní se i zbytek jeho záznamů v ostatních databázových tabulkách.

Integritní omezení

Integritní omezení vymezují hodnoty objektů databáze tak, aby mohly mít v reálném světě smysl (aby odpovídali požadavkům na správnost). Rozlišujeme 3 typy integritních omezení:

- Entitní – co musí splňovat entita. Požadavek na jednoznačnou identifikaci řádky v tabulce. Definice primárního klíče (jedinečný, vyplněn). Vyžadováno přímo databází.
- Doménové – omezení atributů. Omezuje hodnoty položek (intervalem, výčtem). Zajišťuje, aby každá hodnota atributu byla v souladu s množinou přípustných hodnot (datové typy a omezení na nich).
- Referenční – popisuje, jaké vztahy platí mezi tabulkami. Obvykle se zavádí cizí klíč.

Primární klíč

Jednoznačný identifikátor každé entity, nebo jinak řečeno, každého záznamu (řádku). Měl by se skládat z minimálního počtu atributů)sloupců a jeho hodnota musí být v rámci tabulky jedinečná. Primární klíč musí nabývat konkrétní hodnotu, tzn., že nesmí být nikdy NULL. Bez primárního klíče není možné definovat relace mezi tabulkami.

Při výběru atributů, které budou tvořit primární klíč je třeba dbát na to , aby vybrané atributy skutečně plnili úlohu identifikačního klíče, ale na druhé straně, aby jejich použití bylo efektivní i z časového a paměťového hlediska.

Cizí klíč

Atribut, nebo skupina atributů tvořící v jiné relaci primární klíč nemůže nabývat hodnoty, které jsou v rozporu s hodnotami odkazovaného primárního klíče – různé způsoby zajištění.

Jedná se o sloupec, případně kombinaci více sloupců, které jsou propojeny s primárním klíčem v jiné tabulce (odpovídající řádky mají stejnou hodnotu). Zavedením relací mezi tabulkami pomocí cizího klíče se minimalizuje objem údajů, které jsou v DB uloženy, protože namísto kompletních údajů o zákazníkovi je v tabulce objednávek uveden jen klíč do jiné tabulky.

Způsoby zajištění integrity

Způsoby zajištění **Referenční integrity** jsou 3:

- Restriktivní – nedovolí zrušit řádku nadřízené tabulce, pokud k ní existují řádky v podřízené tabulce. Rovněž nepřipustí změnu hodnoty klíče.
- Kaskádovitý – zruší záznam a zruší všechny jeho podřízené záznamy. Povolí přepsat klíč a všude povolí přepsat podřízené klíče.
- Dosazení prázdné hodnoty

Zajištění doménové integrity:

- Lze uskutečnit pomocí napsaných programů
  - Uložená procedura – program uložený na serveru
  - Trigger – program uložený na serveru, který se spouští při určité činnosti
- Klauzule CHECK v SQL (login VARCHAR(30) NOT NULL CHECK (login <> “”))
- Klauzule UNIQUE – vynucení unikátní hodnoty atributu v rámci tabulky.
- Klauzule NOT NULL

Zajištění entitní integrity:

- Primární klíč je implicitně deklarován jako UNIQUE a NOT NULL, tudíž je jeho unikátnost v rámci tabulky zajištěna

Integritní omezení v SQL

- SQL 86 – NOT NULL, UNIQUE
- SQL 89 – PRIMARY KEY, CHECK, REFERENCES a FOREIGN KEY
  CREATE TABLE predmet (... FOREIGN KEY (garant) REFERENCES ucitel(cislo_ucitel));
- SQL 92- ON UPDATE (DELETE) CASCADE (NULL), ALTER TABLE ADD (DROP) CONSTRAINT.

10. Formulace dotazů v SQL, příkaz SELECT, operátory (NOT) IN, (NOT) BETWEEN, (NOT) EXISTS,…

SQL (Structured Query language) je založeno na relačních a SET operacích s určitými modifikacemi a rozšířeními. Typický SQL dotaz má následující strukturu:

```
SELECT A1, ..., An
FROM r1, r2, ..., rm
WHERE P
```

A reprezentuje atributy, r – relace, P je podmínka.
Nejpoužívanější příkaz SELECT:

```
SELECT [DISTINCT | ALL] {* | [sloupcový_výraz [AS nový_název]] [,…]} FROM
název_tabulky [alias] [,…]
[WHERE podmínka]
[GROUP BY seznam_sloupců] [HAVING podmínka]
[ORDER BY seznam_sloupců]
```

Pokud budeme chtít získat obsah celé tabulky, postačí příkaz selekce:

```
SELECT * FROM cd;
```

Ale obvykle si s tímto příkazem nevystačíme, protože nás převážně zajímají jen některá data. Buď jsou to hodnoty některých atributů (sloupců), nebo jen některé záznamy. Nejčastěji se používá kombinace obojího.

Příkaz projekce

Výběr určitých sloupců projekce:

```
SELECT interpret FROM cd;
```

Dotaz vybral některé interprety vícekrát, protože operace projekce neodstraňuje duplicitní záznamy. Toho dosáhneme použitím klíčového slova DISTINCT – výběr s eliminací redundantních řádků.

```
SELECT DISTINCT interpret FROM cd;
```

Výběr s odvozeným sloupcem

```
SELECT pisen, d_min * 60 + d_sec FROM cd;
```

Ve výsledku dotazu má druhý sloupce stejný název, jako aritmetický výraz a to není příliš vhodné a použijeme alias pro pojmenování odvozeného sloupce.

```
SELECT pisen, d_min * 60 + d_sec AS d_sec FROM cd;
```

Výběr záznamů s podmínkou – selekce

Vybíráme pouze ty záznamy, které vyhovují dané podmínce. Následující dotaz kombinuje selekci s projekcí. Pokud bychom chtěli použít jen selekci, nahradíme atributy znakem *.

```
SELECT pisen, d_min FROM cd WHERE d_min >= 5;
```

V podmínce WHERE je možné použít relační operátory a logické spojky: =, <, >, <=, >=, <>, !=, AND, OR, NOT.

Výběr záznamů, kde hodnota atributu patří (nepatří) do intervalu

```
SELECT  pisen, d_min FROM  cd
WHERE  d_min BETWEEN 5 AND 7;  (lze napsat i jako WHERE d:min >= 5 AND d?min <= 7)
```

Nepatří do intervalu: nahradíme BETWEEN -> NOT BETWEEN.

Výběr záznamů, kde hodnota atributu je/není členem výčtu (množiny) hodnot

```
SELECT  interpret, pisen FROM  cd
WHERE  interpret IN ('The Beatles', 'The Cranberies');
```

Není členem výčtu: nahradíme IN -> NOT IN

Výběr záznamů, kde hodnota řetězce ne/odpovídá uvedené masce (operátor LIKE)

```
SELECT  interpret, pisen FROM  cd WHERE  interpret LIKE 'The%';
```

Neodpovídá masce: nahradíme LIKE -> NOT LIKE.

Speciální znaky při vyhledávání řetězců jsou: % (procenta – libovolný počet libovolných znaků včetně 0 znaků) a _ (podtržítko – jeden libovolný znak). Lze si nadefinovat vlastní escape znak pomocí klíčového slova **ESCAPE**

Výběr záznamů, kde hodnota atributu je/není definována (NULL)

```
SELECT  pisen, poznamka FROM  cd WHERE  poznamka IS NULL;
```

Je definována: nahradíme IS NULL -> IS NOT NULL

Řazení výsledků dotazu

V předchozích dotazech byla získaná data neuspořádaná. Ve skutečnosti SŘBD tato data vypisuje chronologicky, tj. data jsou seřazena podle času, kdy byla do DB vložena.

K seřazení slouží klauzule ORDER BY.

```
SELECT  interpret, pisen FROM  cd ORDER  BY interpret DESC;
```

Směr řazení udává klíčové slovo **ASC** (vzestupné řazení - implicitní), **DESC** (sestupné řazení)

Řadit lze i podle odvozených sloupců (pokud nelze odvozenému sloupci přiřadit alias, použijeme pro označení sloupce u řazení jeho pořadové číslo).

Dotazy na větším počtu tabulek

Dosud veškeré dotazy byly realizovány nad jednou tabulkou, avšak většina dotazů získává informace z více tabulek. Problém spočívá v tom, jakým typem spojení tyto tabulky jsou pro daný dotaz. Mezi základní typy spojení patří:

- Přirozené spojení – natural join
- Polospojení – semi-join
- Self-join

Přirozené spojení – při výběru z více tabulek je nutné z důvodu jednoznačnosti udávat plně kvalifikované názvy sloupců. Spojení realizováno přes rovnost primárního klíče.

```
SELECT  t.id, t.titul, p.pisen, p.d_min, p.d_sec FROM  tituly t, pisne p
WHERE  t.id = p.id;
```

Polospojení – lze si představit jako operaci selekce, kde podmínka selekce je závislá na existenci dat druhé tabulky. Proto je tento typ spojení realizován pomocí operátoru **EXISTS** (TRUE, pokud vnořený dotaz poskytne data, FALSE – pokud neposkytne)

```
SELECT p.zkratka, p.nazev FROM  predmety p
WHERE  EXISTS ( SELECT  * FROM  rozvrh r WHERE  p.id = r.id_predm );
```

Podmínku můžeme otočit a chtít nerozvrhované předměty, když použijeme operátor NOT EXISTS.

Self-join – specifický typ přirozeného spojení, které spojuje jednu a tu samou tabulku. Např. když zaměstnanec má svého nadřízeného zaměstnance.

```
SELECT  a.prijmeni, a.jmeno, b.prijmeni, b.jmeno FROM  osoby a, osoby b
WHERE  a.id_nad = b.id_osoby;
```

Velmi často se využívá také **levé (left outer join)**, nebo **pravé (right outer join)** spojení dvou tabulek. Tyto spojení se liší od přirozeného spojení tak, že mimo jiné zahrnuje i výsledné řádky, které obsahují hodnoty NULL atributů tabulky z levé resp. Pravé strany (Ještě existuje **FULL JOIN** – z obou stran). Syntaxe zápisu těchto spojení není sjednocená, standard dodržuje MySQL a PosgreSQL, jinou syntaxi ale používá Oracle. Standard SQL 92 tedy je:

```
SELECT  p.zkratka, r.semestr, r.den FROM  predmety p
LEFT JOIN rozvrh r ON  p.id = r.id_predm;
```

Množinové operace: sjednocení, průnik a rozdíl

Pokud chceme získat souhrnný výsledek z více dotazů.

- Sjednocení – UNION (odstraňuje duplicity). Pokud nechceme odstranit duplicitní záznamy, použijeme UNION ALL
- Průnik – INTERSECT
- Rozdíl – MINUS (odstraňuje duplicity)

Tyto klauzule vložíme mezi dotazy

11. Agregační funkce – definice, použití, klausule GROUP BY, HAVING

Pokud nás nezajímají konkrétní hodnoty jednoho sloupce, ale jen statistický údaj vypočtený z těchto hodnot, poslouží nám Agregační funkce. Pracují se všemi vybranými hodnotami sloupců tabulky a vracejí hodnotu. Standard SQL92 nabízí tyto agregační funkce:

- COUNT() – vrací počet hodnot ve specifikovaném sloupci.
- SUM() – součet hodnot ve specifikovaném sloupci
- AVG() – aritmetický průměr hodnot ve specifikovaném sloupci

- MIN() – minimální hodnotu ze specifikovaného sloupce.
- MAX() – maximální hodnotu.

SELECT COUNT(interpret) **FROM** cd;

Počítá pouze ty hodnoty parametru, které mají vyplněnou hodnotu. Pokud některý záznam nemá vyplněnou hodnotu, do výsledku se nepočítá. Pokud chceme celkový počet všech záznamů bez ohledu na případné NULL hodnoty sloupců, použijeme jako parametr '*'.

SELECT COUNT(*) **FROM** cd;

V prvním dotazu jsme se ptali kolik záznamů má vyplněno atribut *interpret.* Když se ptáme, kolik různých interpretů je v tabulce. Musíme se zeptat takto:

SELECT COUNT(**DISTINCT** interpret) **FROM** cd;

Agregační funkce nelze použít v klauzuli WHERE a nelze je vnořovat do sebe.

GROUP BY

Klausule slouží ke slučování vybraných záznamů do skupin. Hodnoty agregačních funkcí se zpravidla získávají pro definované množiny záznamů, ne pro celou tabulku. Pro definici této množiny se musí v příkazu SELECT použít klauzule GROUP BY. V tomto dotazu tvoří vždy danou množinu ty záznamy z tabulky CD, které mají stejnou hodnotu atributu *interpret.* Aby byl dotaz korektní, musí klausule GROUP BY obsahovat všechny neagregované sloupce, jejichž hodnotu chceme z dotazu znát.

```
SELECT  interpret, COUNT(stopa) AS pocet_stop,  SUM(d_min * 60 + d_sec) AS delka
 FROM  cd  GROUP  BY interpret  ORDER  BY interpret;
```

HAVING

Pokud z předchozího dotazu budeme chtít jen některé výsledné záznamy, musíme dotaz doplnit o klausuli HAVING, ve které je definována požadované podmínka. Podmínky v klauzuli HAVING jsou aplikovány až po zformování skupin (GROUP BY), zatímco podmínky v klauzuli WHERE jsou aplikovány před vytvořením skupin. V klauzuli HAVING nejdou použít nově definované názvy sloupců, a lze v ní použít jen agregační funkce a konstanty.

```
SELECT  interpret, COUNT(stopa) AS pocet_stop,  SUM(d_min * 60 + d_sec) AS delka
 FROM  cd  GROUP  BY interpret
HAVING  SUM(d_min * 60 + d_sec) > 300 AND COUNT(stopa) < 12
 ORDER  BY interpret;
```

Výběr záznamů na základě vnořeného dotazu

Některé SQL dotazy mohou mít v sobě obsaženy další kompletní příkazy SELECT, kterým se říká vnořené dotazy. Ty se mohou vyskytovat v klauzulích WHERE a HAVING vnějšího příkazu SELECT. Vnořené dotazy se mohou vyskytovat i v příkazech INSERT, UPDATE, DELETE.

Př. Získání nejdelší písně:

```
SELECT  pisen, d_min  FROM  cd  WHERE  d_min = ( SELECT MAX(d_min) FROM cd );
```

Vnořený dotaz je vždy uzavřen mezi závorkami a je vždy ve výrazu na pravé straně. Vždy musí (až na pár výjimek) vracet pouze jednu hodnotu. Ve vnořeném dotazu nemá smysl používat klauzuli ORDER BY.

12. Anonymní PL/SQL blok

Hlavním omezením jazyka SQL je skutečnost, že se jedná o neprocedurální jazyk. V praxi to znamená, že příkazy SQL se vykonávají sekvenčně, bez možnosti použití klasických programátorských konstrukcí, jako jsou např. cykly, podmínky, využití procedur a funkcí a podobně. Proto má skoro každá moderní databázová platforma

implementované určité procedurální rozšíření jazyka SQL. Takto rozšířený jazyk SQL se stává mocným nástrojem, který umožňuje naprogramovat i ty nejsložitější algoritmy pro práci s údaji, případně pro jejich zpracování a vyhodnocování.

PL/SQL (Transaction Processing Language) je rozšíření jazyka SQL o procedurální rysy. Je specifické pro produkty firmy Oracle, procedurální rozšíření SQL produktů jiných firem se zpravidla navzájem liší. Základním stavebním kamenem PL/SQL je tzv. PL/SQL blok. Nástroje pro tvorbu a spouštění PL/SQL = SQL*Plus, SQL Worksheet. PL/SQL poskytuje prostředky pro definování a spouštění programových jednotek PL/SQL (procedury, funkce, balíky-package).

PL/SQL blok

Jedná se o základní kódový segment jazyka a může být buď tělem triggeru, procedury a funkce, nebo samostatný. Struktura je následující:

DECLARE

Deklarace konstant a proměnných – deklarační sekce

BEGIN

Výkonné příkazy – výkonná sekce

EXCEPTION

Výkonné příkazy – sekce pro zpracování výjimek

END;

Sekce DECLARE a EXCEPTION jsou nepovinné.

Př.

```
DECLARE
    numerator    NUMBER;
    denominator  NUMBER;
    the_ratio    NUMBER;
    lower_limit  CONSTANT NUMBER := 0.72;
    samp_num     CONSTANT NUMBER := 132;

BEGIN
    SELECT  x, y INTO numerator, denominator
      FROM  result_table
     WHERE  sample_id = samp_num;
    the_ratio := numerator / denominator;
    IF the_ratio > lower_limit THEN
        INSERT INTO ratio VALUES (samp_num, the_ratio);
    ELSE
        INSERT INTO ratio VALUES (samp_num, -1);
    END IF;
    COMMIT;

EXCEPTION
    WHEN ZERO_DIVIDE THEN
        INSERT INTO ratio VALUES (samp_num, 0);
        COMMIT;

    WHEN OTHERS THEN
        ROLLBACK;
END;
```

Anonymní PL/SQL

Anonymní PL/SQL blok ukazuje předchozí příklad.. Při použití SQL konzole je PL/SQL blok spuštěn teprve po zadání lomítka na začátku samostatného řádku.

Jedná se o nepojmenovaný blok. Je použit pouze jednou v místě jeho definice. Tento blok lze uvést do příkazové dávky SQL, definiční části databázového triggeru, do definice prvku aplikačního menu.

Proměnné

Je potřeba před prvním použitím deklarovat. Navíc je možné proměnnou také inicializovat (přiřazení počátečních hodnot nebo pomocí klíčového slova DEFAULT).

13. Kurzory – definice, klasifikace, použití kurzorů

Kurzor je prostředek pro získání informace z DB a předání do programu v jazyce PL/SQL.

Kurzor je abstraktní datový typ umožňující procházet záznamy vybrané dotazem, který je s kurzorem spojen.

Nad kurzorem jsou definovány následující operace:

- OPEN – otevření kurzoru
- FETCH – přečtení dalšího záznamu
- CLOSE – zavření kurzoru

Kromě těchto operací obsahuje kurzor navíc následující stavové proměnné:

- %FOUND – boolean, true - pokud FETCH vybral další záznam, false –pokud FETCH nic nevybral. Před prvním FETCH je NULL
- %NOTFOUND – boolean, opak FOUND
- %ISOPEN – boolean, true – pokud byl kurzor otevřen příkazem OPEN, jinak FALSE
- %ROWCOUNT – INTEGER, počet řádků dosud zpracovaných příkazem FETCH. Před otevřením kurzoru nebo před prvním FETCH je 0.

Typy kurzoru

Explicitní kurzor

Nutno deklarovat, otevřít, načíst data, zavřít.

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

Př.

```
DECLARE
  num INTEGER;
  CURSOR osoba (name IN VARCHAR2) IS
    SELECT  os_cislo
      FROM  osoby
     WHERE  jmeno = name;

BEGIN
  OPEN osoba('Max');

  FETCH osoba INTO num;
  IF osoba%FOUND THEN
    dbms_output.put_line('Osobni cislo : '||num);
  END IF;

  CLOSE osoba;
END;
/
```

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.

Syntaxe implicitního kurzoru:

SELECT <jmeno_sloupce1>, <jmeno_sloupce2>,... INTO <jmeno_promenne1>, <jmeno?promenne2>, ...

FROM ...;

V implicitním kurzoru se musí shodovat datové typy a délka sloupců a proměnných. Navíc se ve výše uvedené syntaxi může objevit i INSERT, UPDATE, DELETE. Implicitní kurzor SELECT musí vracet pouze jeden řádek, jinak je nutné změnit dotaz, nebo použít explicitní kurzor.

Kurzor pro cyklus LOOP

Syntaxe kurzoru:

FOR <jmeno_radku> IN <jmeno_kurzoru> LOOP

<loop-blok>

END LOOP;

Cyklus začne s prvním načteným řádkem a skončí s posledním načteným řádkem. Loop-blok obsahuje opakované příkazy. V cyklu jsou vybrány všechny záznamy kurzoru. Proměnná kurzoru nemusí být deklarována a její struktura odpovídá struktuře řádky vybrané kurzorem.

Př.

```
DECLARE
  CURSOR  plist(num IN INTEGER) IS
    SELECT * FROM osoby
     WHERE os_cislo > num;

BEGIN
  FOR p IN plist(1) LOOP
    dbms_output.put_line(p.jmeno||' '||p.prijmeni);
  END LOOP;
END;
/
```

Použití kurzoru

- V triggerech
- V uložených procedurách
- Kurzor lze předat i jako parametr procedury, nebo funkce.
  - Silně typovaný kurzor – specifikuje i řádkový typ, se kterým bude kurzor pracovat

    ```
    DECLARE
      tmp   osoby%ROWTYPE;
      TYPE  t_crsr IS REF CURSOR RETURN osoby%ROWTYPE;
      plist t_crsr;
    ```

  - Slabě typovaný kurzor – nespecifikuje řádkový typ
    Příklad jako předchozí bez `RETURN osoby%ROWTYPE`

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

Podprogram je pojmenovaný blok, který může být opakovaně volán a může přebírat aktuální parametry. Typy podprogramů jsou funkce a procedury. Procedury a funkce lze sdružovat do logických celků – balíků (package).

Všechny anonymní bloky PL/SQL lze natrvalo uložit do databáze, a to buď ve tvaru procedury, nebo funkce.

Uložené podprogramy – procedury a funkce

Procedury a funkce mohou být trvale uloženy do DB a mohou být použity jakoukoli aplikací, která s databázovým strojem komunikuje. Jednou přeložený a uložený program patří mezi databázové objekty a může být referován libovolným počtem aplikací.

Uložené podprogramy mohou být samostatné, nebo součástí balíku.

Podprogramy lze volat z:

- DB triggerů
- Jiných uložených podprogramů
- Aplikačních programů zapsaných ve vyšším programovacím jazyce, pro nějž existuje předkompilátor (ORACLE Pro)
- Interaktivně (SQL*Plus) : EXECUTE jmeno_procedury(parametry)

Uložené procedury a funkce mohou mít parametry, které mohou být:

- IN - vstupní
- OUT - výstupní
- IN OUT – vstupní i výstupní

Uložená procedura

Procedura se vstupními parametry:

```
CREATE OR REPLACE PROCEDURE nastav_plat (id IN NUMBER, novy_plat IN NUMBER) AS
  zam_plat   NUMBER(5,2);
  nema_plat  EXCEPTION;
BEGIN
  SELECT plat INTO zam_plat FROM osoby WHERE os_cislo = id;
  IF zam_plat IS NULL THEN
    RAISE nema_plat;
  END IF;
EXCEPTION
  WHEN nema_plat THEN
    UPDATE osoby SET plat = novy_plat WHERE os_cislo = id;
    COMMIT;
END;
```

Pro zrušení (smazání) uložené procedury použijeme příkaz:

```
DROP PROCEDURE nastav_plat;
```

Uložená funkce

Způsob vytvoření uložené funkce (s parametry) ukazuje následující příklad:

```
CREATE OR REPLACE FUNCTION secti(a IN INTEGER, b IN INTEGER) RETURN INTEGER AS
BEGIN
  RETURN (a + b);
END;
```

Pro zrušení (smazání) uložené funkce použijeme příkaz:

```
DROP FUNCTION secti;
```

Uložené balíky

Uložené balíky procedur a funkcí slouží ke sdružení logicky spolu souvisejících procedur a funkcí, ale i typů a objektů. Mohou obsahovat i globální proměnné, jejichž platnost je omezena délkou aktuálního spojení s databází.

Definice balíku představuje definici rozhraní balíku (deklarace typů, proměnných, konstant, podmínek definujících nestandardní stavy, kurzorů, podprogramů dostupných zvenčí) a těla balíku (implementuje specifikaci).

Př. rozhraní:

```
CREATE OR REPLACE PACKAGE arithmetic AS
  usage INTEGER;

  FUNCTION  add(a IN INTEGER, b IN INTEGER) RETURN INTEGER;
  FUNCTION  sub(a IN INTEGER, b IN INTEGER) RETURN INTEGER;
  PROCEDURE inc(a IN OUT INTEGER);
END;
/
```

Př. Těla:

```
CREATE OR REPLACE PACKAGE BODY arithmetic AS

  FUNCTION  add(a IN INTEGER, b IN INTEGER) RETURN INTEGER IS
  BEGIN
    usage := usage + 1;
    RETURN (a + b);
  END;

  FUNCTION  sub(a IN INTEGER, b IN INTEGER) RETURN INTEGER IS
  BEGIN
    usage := usage + 1;
    RETURN (a - b);
  END;

  PROCEDURE inc(a IN OUT INTEGER) IS
  BEGIN
    usage := usage + 1;
    a := a + 1;
  END;

BEGIN
  usage := 0;
END;
/
```

Kompilace

- Pokud je volána procedura/funkce ve stavu INVALID, kompiluje se automaticky.
  - Pokud se kompilace nezdaří, dojde k výjimce.
- Ruční kompilace
  - ALTER PROCEDURE[FUNCTION] jmproc COMPILE;
  - Pokud se kompilace nezdaří, dojde k výjimce
- SQL*Plus a chyby kompilace – pomocí něj můžeme zjistit, jaké chyby se vyskytly při kompilaci. Pokud uložení procedury/funkce neproběhlo bez chyb, nelze ji používat a je nutné ji opravit. Pomocí SQL*Plus příkazů:
  - SHOW ERROR – vypíše popis poslední chyby, na kterou při ukládání (kompilaci) narazil
  - SHOW ERR typ jméno – např. SHOW ERR FUNCTION F – pro uvedený objekt
  - Pomocí dotazu na tabulku USER_ERRORS

Spouštění

Proceduru můžeme zavolat (spustit různými způsoby:

- Pomocí direktivy EXEC – EXEC nastav_plat(123, 10000);
- V těle jiného PL/SQL bloku

```
BEGIN
       ...
       Nastav_plat(123, 10000);
       ...
END;
/
```

Funkci můžeme volat také dvěma způsoby:

- V příkazu SELECT – SELECT secti(2, 3) FROM dual;
- V těle jiného PL/SQL bloku

```
DECLARE
  a INTEGER;
  b INTEGER;
BEGIN
  a := 5;
  b := secti(a, 2);
  dbms_output.put_line('Vysledek : '||b);
END;
/
```

Při volání funkcí/procedur z balíků používáme tečkovou notaci (package.funkce()) pro kvalifikaci jejich jména.

Zabalený podprogram lze volat z db triggeru, jiného uloženého programu, aplikace napsané pro některý z předkompilátorů, standardních klientských nástrojů (SQL*Plus)

Standardní balík STANDARD = definuje prostředí PL/SQL.

15. Aktivní databáze – Oracle triggery, klasifikace a spouštění triggerů

V řadě IS, které běží nad nějakou databází, potřebujeme v případě vzniku nějaké události (např. modifikujeme řádek v nějaké tabulce) automaticky spustit příkaz, který provede nějaké operace. K tomuto účelu slouží triggery (spouštěče). Trigger je speciální typ uložené procedury, která se aktivuje při splnění nějaké podmínky na serveru (aktualizace dat, události spojené s DB nebo session).

Triggery jsou vlastně procedury, které automaticky volá (spouští) SŘBD při definované události. Touto událostí může být buď vložení (INSERT), rušení (DELETE), nebo aktualizace (UPDATE) záznamu v tabulce. Triggery bývají obvykle volány buď:

- Před specifikovanou událostí – BEFORE
- Po specifikované události – AFTER

Od uložených procedur a funkcí se odlišují tím, že jsou spuštěny při modifikaci tabulky, definují se pouze pro db tabulky a nepřijímají argumenty a lze je spustit jen při zmiňovaných DML příkazech.

U triggeru lze rovněž specifikovat podmínku, kdy má být vykonáno jeho tělo (PL/SQL blok) a to použitím klauzule WHEN.

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

Trigger lze také

- deaktivovat (zakázat) - ALTER TRIGGER jmeno DISABLE;
- aktivovat – ALTER TRIGGER jmeno ENABLE;
- zrušit – DROP TRIGGER jmeno;

Výhody triggerů jsou:

- nepovolí neplatné datové transakce, zajišťují komplexní bezpečnost, zajišťují referenční integritu přes všechny uzly db, zajišťují audit (sledování), spravují synchronizaci tabulek, zaznamenávají statistiku často modifikovaných tabulek.

Klasifikace triggerů

Tabulkové triggery

Triggery se spustí nad tabulkou bez ohledu na to , kolik tabulka obsahuje řádek. Např. pokud chci logovat změny provedené v DB, která obsahuje tabulky PRACOVISTE, ZAMESTNANCI do tabulky LOGY. Po každé operaci I, U, D nad tabulkami PRAC a ZAM se do tabulky LOGY vloží záznam o modifikaci tabulky a typu modifikace.

```
CREATE TRIGGER tai_pracoviste
AFTER INSERT ON pracoviste
BEGIN
  INSERT INTO logy VALUES ( 'PRACOVISTE', 'I');
END;
/
```

Řádkové triggery – FOR EACH ROW

Trigger se spouští jednou pro každý řádek tabulky. Např. z předchozího příkladu chci mít u každého záznamu informaci, kdy byl zadán a kdo jej zadal.

```
CREATE TRIGGER trbi_pracoviste
BEFORE INSERT ON pracoviste
FOR EACH ROW
BEGIN
  :new.zadal := user;
  :new.datum := sysdate;
END;
/
```

Vlastní obsluhu události lze nadefinovat v PL/SQL bloku. Uvnitř PL/SQL bloku (a také klauzuli WHEN) řádkového triggeru se lze odkazovat na původní a nový záznam pomocí pseudoproměnných **:new** (obsahuje vkládaný záznam) a **:old** (původní záznam). Je zřejmé, že při vkládání nového záznamu není definována :old a při mazání :new. Jejich jména lze předefinovat v klauzuli **REFERENCINGOLD AS.**

Business rules triggery

Triggery jsou také často používány při realizaci tzv. business rules, tj. integritních omezení specifických pro danou oblast použití. Např. použijeme – studenti si mohou zapsat max 20 kreditů za semestr. Pokud při vkládání dat do tabulky ZAPIS překročíme maximální povolenou hodnotu, dostaneme chybové hlášení – a to zařídí business trigger.

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

SQL je velice flexibilní – dvěma i více dotazy je možné obdržet stejná data, ovšem rychlost dotazů nemusí být stejná.

Důvodem optimalizace je minimalizace nákladů na:

- Zdrojový čas
- Kapacitu paměti, či prostoru
- Programátorskou práci

Y malých DB optimalizace nepatrná, projeví se až u objemných x u často navštěvovaných webů může při špatně formulovaných dotazech vzrůst trafik v obou směrech.

SŘBD Oracle již sám používá optimalizačních technik pro vyhodnocení jakéhokoli dotazu. Těmito technikami jsou:

- Rule based optimaliaztion (RBO)
- Cost based optimalization (CBO) – od verze Oracle 9i je preferovaný

Jak zjistit způsob provedení příkazu? Abychom mohli zjistit, jak ve skutečnosti daná optimalizace vyhodnocení dotazu funguje, potřebujeme vytvořit tabulku **PLAN_TABLE** (podle skriptu *utlxplan.sql),* kam optimalizátor ukládá své vítězné plány právě vyhodnoceného dotazu. Pro vysvětlení (tj. zjištění optimálního plánu) vyhodnocení dotazu použijeme příkaz **EXPLAIN_PLAN.**

```
explain plan for select p.NAZEV,m.ZKRATKA, ...
```

Vykonáním tohoto příkazu se vítězný plán uloží do tabulky PLAN_TABLE v podobě několika záznamů. Informace vyčteme s použitím dotazu:

```
select plan_table_output from table(dbms_xplan.display());
```

Existuje několik různých přístupů k tabulkám:

- Plný přístup (Full scan) – prochází se celá tabulka – všechny záznamy, u každé řádky se ověří podmínka. Vhodné, pokud procento vyhovujících řádek bude velké.
- Index-Range-Scan – vyhledání intervalu v indexu. Ověření ostatních podmínek v odkazovaných řádcích
- Unique-Index-Scan – vyhledání jediné možné vyhovující řádky podle unikátního indexu
- ROWID-Scan – vyhledání řádky na základě známé hodnoty jejího fyzického identifikátoru v DB.

RBO

Vyhodnocuje jednotlivé přístupové cesty pomocí předem daného systému pravidel. Jedná se o starší optimalizátor. Plán se sestavuje nezávisle na datech v tabulkách, pouze na základě tvaru dotazu. Používá se pouze pro zpětnou kompatibilitu. Odvozuje plán ze syntaxe příkazu a existence indexů.

Nevýhody

Tento způsob optimalizace je v SŘBD Oracle již označen jako *deprecated (zavrhovaný)*. Z důvodu kompatibility je možné jej však aktivovat odpovídajícím hintem RULE.

Nevýhody RBO jsou:

- Možnost špatného výběru použitého indexu - Pokud existuje více neunikátních indexů na jedné tabulce, nemusí optimalizátor vybrat ten nejlepší. Použití určitého indexu je možné optimalizátoru znemožnit použitím výrazu v dotazu.

CBO

Hledá plán s nejmenšími náklady pomocí statistik. Optimalizace je založena na vyhodnocení kvantitativního využívání zdrojů v průběhu zpracování dotazu (CPU, paměť, I/O,…). Využívá statistiky o tabulkách a datech, které jsou uloženy v Data Dictionary:

- Počet různých hodnot ve sloupci, histogramy rozložení hodnot ve sloupci, počet řádek v tabulce, průměrná délka jedné řádky.
- Některé statistiky jsou přístupné i pro uživatele db pomocí pohledů, či tabulek
- Aktualizují se výpočtem nebo odhadem
- Typy:
  - Údaje o tabulkách – počet řádků, bloků, délka záznamu
  - Údaje o sloupcích – počet unikátních hodnot a NULL, histogram
  - Údaje o indexech – počet listových bloků, clustering

Dokáže rozlišit plány i pro různé typy konstant v dotazu.

Použití CBO je doporučeno firmou Oracle. Vybírá se plán s nejnižší váženou cenou.

Hints

V CBO lze využít pro optimalizaci i nápovědu – Hints. Prostřednictvím ní mohu optimalizátoru vnutit některou operaci, protože si myslím, že její užití přispěje k lepší optimalizaci. Tato nápověda se zapisuje jako komentář specifického tvaru.

```
explain plan for select /*+ INDEX (predmety predmety_index1) */
       p.NAZEV,...
```

Výběr typu optimalizace

Je dalším krokem, kterým můžeme ovlivnit optimalizaci. Jedná se o změnu optimalizačního typu optimalizátoru SŘBD Oracle. Typy optimalizace jsou:

- CHOOSE – výběr podle (ne)přítomnosti statistik nejsou-li k dispozici, potom RBO, jinak CBO.
- ALL_ROWS – vždy CBO, minimalizuje se cena za získání všech řádek odpovědi. Vhodné pro dávkové zpracování.
- FIRST_ROWS – vždy CBO, minimalizuje se cena za získání prvních řádek odpovědi. Vhodné pro interaktivní zpracování.
- RULE – vždy RBO.

Typ optimalizace se vybírá příkazem:

```
ALTER SESSION SET OPTIMIZER_GOAL = ALL_ROWS;
```

Další možnost ladění je změna módu optimalizátoru. Dotazy mohou být optimalizovány na:

- Nejlepší průchodnost – ALL_ROWS
- Nejrychlejší odezvu – FIRST_ROWS_1 – zkrátí čas vyhodnocení dotazu.

Př. nastavení módu:

```
ALTER SESSION SET optimizer_mode = all_rows;
```

Obecná pravidla pro psaní SQL dotazů

Vyplývají z technik optimalizace:

- V selectu nepoužívat v seznamu sloupců *, protože ve většině případů nepracujeme se všemi.
- Používat co nejméně klauzuli LIKE, IN, NOT IN (vhodnější je WHERE a WHERE NOT EXISTS)
- Používat klauzule typu LIMIT
- Používat hinty
- Nastavit indexy

17. NLS – podpora národních jazyků v Oracle, formát datumu a času, konverzní funkce TO_CHAR() a TO_DATE()

Podpora národních jazyků v Oracle zahrnuje řadu rozšíření pro provoz v národním i vícejazyčném prostředí:

- Znakové sady
- Národní datové typy
- Použití speciálních znaků v názvech objektů
- Zobrazování čísel
- Kódy měn
- Kalendář, formát datumu, ...
- Způsob třídění a porovnávání dle pravidel jazyka
- Jazyk komunikace

Národní prostředí databázového serveru disponuje

- Dvojí znakovou sadou databáze –Univerzální (UTF-8, ISO Latin 2, CP 1250,...) a Národní (ISO Latin 2, US ASCII 7bit)
- Dvojími znakovými datovými typy – Dle ANSI SQL (NATIONAL <jmeno_typu>) a proprietární typy (N<jmeno_typu>).

Národní prostředí klienta nastavíme pomocí proprietárního řešení, které používá přepínače NLS_*

- NLS_LANG (základní nastavení NLS prostředí, ostatní nastavení jsou pouze doplňkové pro předefinování detailů), NLS_LANGUAGE (předefinuje nastavení jazyka z NLS_LANG), NLS_TERRITORY(předefinuje nastavení oblasti z NLS_LANG), NLS_SORT (předefinuje nastavení třídění)…

Existují tři úrovně nastavení národního prostředí:

- Pro počítač, resp. Uživatele – proměnné prostředí (UNIX), hodnoty v registru (WIN)
- Pro session – na začátku dle nastavení předchozího bodu. Změna příkazem ALTER SESSION SET ...
- Pro konkrétní příkaz – dodatečný (nepovinný) parametr u některých funkcí.

Formát datumu a času

Formát datumu se nastavuje pomocí přepínače **NLS_CALENDAR**. (Defaultně je nastaven na Gregorian, ale existují i další – Persian, Japanese Imperial, Arabic Hijrah,...)

Přepínač **NLS_DATE_LANGUAGE** – jazyk pro názvosloví dní a měsíců (default odvozen od NLS_LANGUAGE). Je použitelný v konverzní funkci TO_CHAR().

- Formátovací značky – Day (Pondělí, úterý), Dy (Po, út), Month (Leden,únor), Mon (Led,ún)
- Neovlivní spelování a skloňování (12th, ne 12tého)

Přepínač **NLS_DATE_FORMAT** – přepínač pro formát datumu – default je odvozen z NLS_TERRITORY (DD-MON-RR pro America, DD.MM.YYYY pro Czech Republic)

- Použitý jako default v konverzní funkci TO_CHAR(datum), TO_DATE('text').

Pro formát času se využívá datový typ TIMESTAMP, který je obdobný typu DATE a je přesný na tisíciny sekundy. Přepínač **NLS_TIMESTAMP_FORMAT –** přepínač pro formát času.

TO_CHAR()

Jedná se o konverzní funkci, která převede číslo na řetězec podle zadaného formátu, číslem může být i měna.

TO_CHAR(číslo, formát)

Tato funkce bere v úvahu nastavení národního prostředí.

TO_DATE()

Konverzní funkce, která převede textové vyjádření datumu a času na formát pro uložení do DB.

TO_DATE('text')

Tato funkce bere v úvahu nastavení národního prostředí.

18. Distribuované databáze, problémy replikace a fragmentace dat

Distribuovaná DB je množina databází, která je uložena na několika počítačích. Uživateli se může jevit jako jedna velká databáze. V DB neexistuje žádný centrální uzel nebo proces odpovědný za vrcholové řízení funkcí celého systému. Výrazně to zvyšuje odolnost systému proti výpadkům jeho částí. Data i funkce rozděleny mezi více počítačů

Charakteristické vlastnosti

Distribuovaná DB je charakterizována:

- Transparentností – z pohledu klienta se zdá, že data jsou zpracována na jednom serveru v lokální databázi. Uživatel si rozdělení nemusí být vědom.
- Rozšiřitelností – zvýšení výkonu přidáním dalších počítačů.
- Robustností – výpadek jednoho počítače neovlivní funkci ostatních
- Jsou syntakticky shodné příkazy pro lokální i vzdálená data, nespecifikuje se místo uložení dat (řeší to distribuovaný SŘBD)
- Autonomností – s každou lokální bází dat zapojenou do distribuované databáze je možno pracovat nezávisle na ostatních databázích.
- Lokální Db je funkčně samostatná, připojení do jiné části distribuované db se v případě potřeby zřizují dynamicky.
- Nezávislost na počítačové síti – jsou podporovány různé typy architektur lokálních i globálních počítačových sítí (LAN, WAN)
- V distribuované databázi mohou být zapojeny počítače i počítačové sítě různých architektur, pro komunikaci se používá jazyk SQL.

Distribuované databáze jsou výhodné kvůli:

- Lokální autonomie – odpovídají struktuře decentralizovaných organizací. Data jsou uložena v místě nejčastějšího využití a zpracování – zlevnění provozu
- Zvýšení výkonu (rozdělení zátěže na více počítačů)
- Spolehlivosti (replikace dat, degradace služeb při výpadku uzlu, přesunutí na jiný uzel)
- Rozšiřitelnosti
- Schopnosti sdílet informace integrací podnikových zdrojů
- Agregaci informací – z více bází dat lze získat informace nového typu.

Naopak Distribuované Db mohou způsobit i pro ně specifické problémy:

- Složitost – distribuce db, distribuce zpracování dotazu a jeho optimalizace, složité globální transakční zpracování, distribuce katalogu, paralelismus a uvíznutí, složité zotavování z chyb
- Cena (komunikace je navíc)
- Bezpečnost
- Obtížný přechod – neexistuje automatický konverzní prostředek z centralizovaných DB na DDB

Problémy replikace a fragmentace dat

U fragmentace a replikace bychom měli respektovat hlediska:

- Rozdělit relace do lokálních serverů tak, aby aplikace zatěžovali servery stejnoměrně.
- Přístupnost a spolehlivost - Replikací zlepšíme spolehlivost a read-only dostupnost.
- Lokality zpracování (maximalizovat lokální).
- Dostupnosti a ceny paměti v jednotlivých uzlech

Replikace dat

Replikační transparentnost je neobtěžovat uživatele skutečností, že pracuje s daty existujícími ve více kopiích = uživatel neví o replikách.

Při replikaci dat se nachází kopie množiny objektů v každém uzlu, ve kterém je využívaná. 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. 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.)

Fragmentace dat

Fragmentační transparentnost = uživatelův dotaz je specifikován na celou relaci, ale musí být vykonán na jejím fragmentu = uživatel neví o fragmentech.

Fragmentace dat se dělí na:

- Horizontální – dle selekční podmínky rozdělíme tabulku na 2 části horizontálním řezem,tedy např. na knihy s ISBN nižším než 122 a na knihy s ISBN větším nebo rovným 122.
- Odvozená horizontální – dochází k rozdělení tabulky na 2 a více částí horizontálním řezem, v tomto případě však je fragmentace založena na jiné relaci. Např. DODAVATELE a KNIHY. DODAVATELE rozdělíme do fragmentů a na základě těchto fragmentů provedeme fragmentaci v tabulce KNIHY, která je spojena s DODAVATLE relací.
- Vertikální – rozdělení tabulky podle sloupců na 2 a více částí. V jedné skupině jsou jedny sloupce, v druhé jiné.
- Smíšená – tabulku např. rozdělíme dle sloupců a následně v jednotlivých částech provedeme horizontální fragmentaci.

19. Architektura klient – server, ODBC, JDBC

Architektura klient – (databázový server)

V podstatě je založena na počítačové síti, personálních počítačích a databázovém serveru. Na personálních počítačích běží program podporující např. vstup dat, formulaci dotazu atd. Dotaz se dále předává pomocí jazyka SQL na databázový server, který jej vykoná a vrátí výsledky zpět na personální počítač.

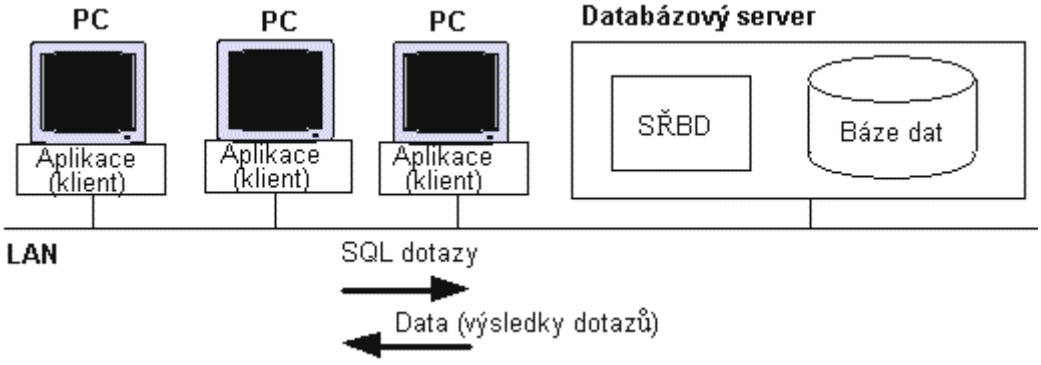
Databázový server je tedy nejvíce zatíženým prvkem systému a musí být tvořen dostatečně výkonným počítačem.

Celá komunikace probíhá tímto způsobem:

1. Uživatel zadává dotaz (v SQL, nebo musí být do tohoto jazyka přeložen)
2. Dotaz odeslán na db server
3. Db server vykoná dotaz
4. Výsledek dotazu je poslán zpět na vysílací počítač, kde je zobrazen

Architektura redukuje přenos dat po síti, protože dotazy jsou prováděny přímo na DB serveru a na personální počítač jsou posílány pouze výsledky. Výhodu to má, že pokud vyhovuje dotazu 100 záznamů z 10 000, tak na personální počítač se posílá jen těch 100 záznamů, kdežto na architektuře file-server by bylo nutné poslat všech 10 000 záznamů na personální počítač, kde by se dotaz teprve zpracoval.

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



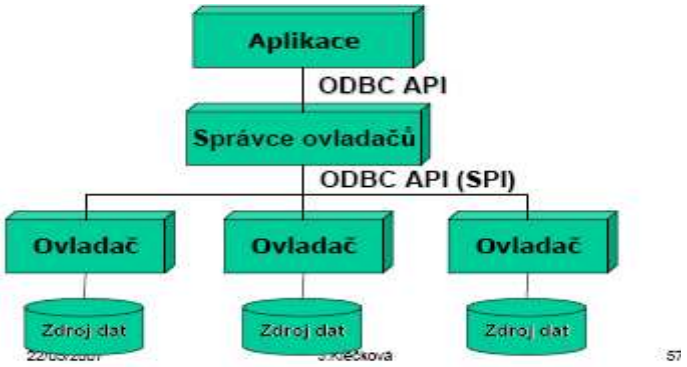
Prvky architektury jsou: Hardwarová platforma, operační systémy, databázový systém, vývojové prostředí, standard architektury. Architektura klient-server znamená dekompozici funkcionality a tedy ideální model, např. pro online zpracování transakcí OLTP v distribuovaném prostředí.

Síla architektury spočívá v:

- Pružnější rozdělení práce, aplikace běží na levnějších zařízeních, klientem mlže být oblíbený prezentační software (Excel, PowerBuilder), standardizovaný přístup pomocí SQL, centralizace dat – lepší a účinnější ochrana dat.

ODBC (Open Database Connectivity)

Standardizované softwarové API pro stejný přístup k různým databázovým systémům (DBMS). Snahou ODBC je poskytovat přístup nezávislý na programovacím jazyku, operačním systému a databázovém systému. Poskytuje ale databázově závislé ovladače, které spravuje Správce ovladačů (Driver manager).



Typy ovladačů

- Ovladače založené na souborech – poskytují přímý přístup k datům, protože ovladač = zdroj dat
- Ovladače založené na SŘBD – dotazy se předávají SŘBD. Transformuje ODBC SQL na konkrétní dialekt SQL.

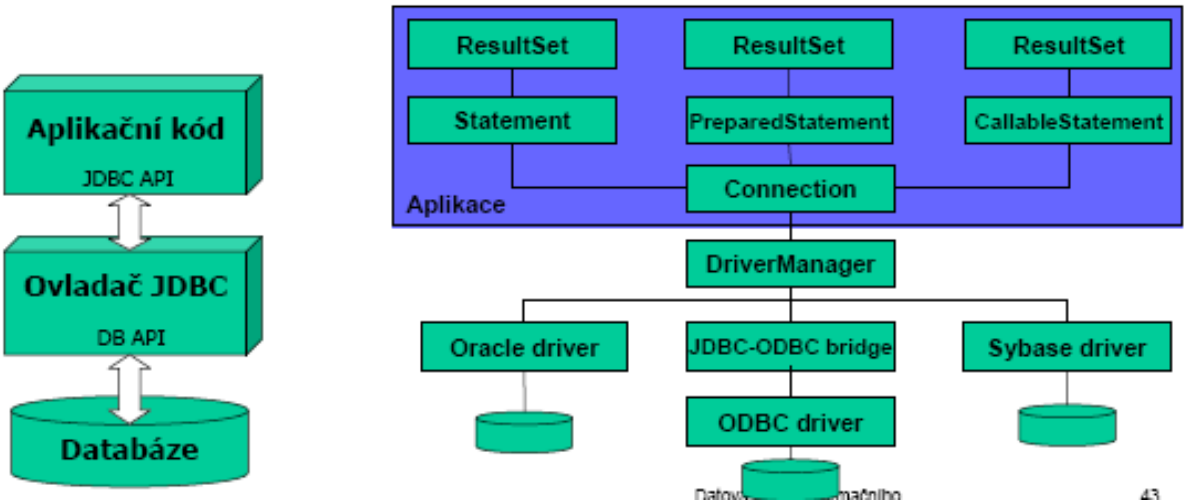
Typy zdrojů dat

- Systémové datové zdroje – Informace jsou uloženy přímo v systému (registry) a jsou identifikovány jménem zdroje.
- Souborové datové zdroje – informace jsou uloženy v samostatném souboru (*.dsn). Možnost sdílení, přenosu na jiné počítače. Jsou identifikovány jménem souboru.

Postup k získání dat je: Připojení k datovému zdroji, inicializace, vytvoření a provedení dotazu, získání výsledku, ukončení transakce, odpojení od datového zdroje.

JDBC (Java Database Connectivity)

Rozhraní pro unifikovaný přístup k datům. Použití je možné i mimo databáze – data ve formě tabulek (CVS, XLS,…). Ovladače jsou k dispozici pro většinu databázových systémů. JDBC je inspirováno rozhraním ODBC (objektové rozhraní, možnost spolupráce s ODBC). Architektura JDBC zahrnuje: Aplikační kód, Ovladač JDBC, databázi.



JDBC ovladač

Zprostředkovává komunikaci aplikace s konkrétním typem databáze. Implementován obvykle výrobcem databáze. Pracuje s dotazovacím jazykem SQL, Je reprezentován specifickou třídou (org.gjt.mm.mysql.Driver, sun.jdbc.odbc.JdbcOdbcDriver).

Typy ovladačů:

- Typ 1 – využívá ODBC (přes JDBC-ODBC bridge). Je obtížně konfigurovatelné.
- Typ 2 – komunikace s nativním ovladačem
- Typ 3 – komunikuje s centrálním serverem síťovým protokolem. Určen pro rozsáhlé heterogenní systémy.
- Typ 4 – založen čistě na jazyce Java. Používá přímý přístup do DB.

Objekty:

- DriverManager – navazuje spojení. Přes něj získáme connection spojení (po zadání hesla, user name, url)
- Statement – reprezentuje SQL příkaz
- ResulSet – reprezentuje výsledek dotazu SELECT.

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

Temporální databáze jsou databáze určitým způsobem podporujícím čas. Čas potřebujeme v databázích např. ve studijním informačním systému, účetních a bankovních systémech, docházkových systémech. Hlavní cíl temporálního DM by měl být zachytit sémantiku dat měnící se v čase. V praxi existuje mnoho nekompatibilních datových modelů s mnoha dotazovacími jazyky.

Porovnání klasických a temporálních DB

Klasický DB systém

Zachycuje stav systému v aktuálním časovém okamžiku. Problém: co dělat se starými daty. V případě, že se systém v čase vyvíjí, změny se v DB projeví přidáním nových informací a mazáním starých. Klasické DB neobsahují informaci o čase. V případě, že požadujeme uchování historie změn, či alespoň předchozího stavu, je nutné do DB doplnit informace o čase. Aktualizaci a operace s časem musí zajistit uživatel, což není triviální.

Temporální DB

Databáze určitým způsobem podporující čas. Poskytuje vhodný dotazovací jazyk zahrnující práci s časem – výhodou jsou jednodušší dotazy, v nichž se vyskytuje čas, což přináší méně chyb v aplikačním kódu a zajišťuje jednodušší udržování aplikací.

Temporální projekce – jako projekce v klasických databázích (z celé relace jsou vybrány hodnoty podle zadaných atributů), navíc bere v úvahu čas. V případě, že dvě n-tice výsledku mají stejné hodnoty všech svých atributů a překrývají se nebo dotýkají se časem, srostou tyto dvě n-tice s časem odpovídajícím sjednocení obou n-tic.

Temporální spojení – stejné jako spojení (JOIN) v klasických DB (zadáme sloupce a podmínku, která říká, kdy jsou dva řádky tabulky spojeny), navíc bere v úvahu čas události. V Temporálních DB nemusíme zadávat sloupce uchovávající čas, pouze podmínku na spojení pro čas.

Modely času

Temporální logika: čas je libovolná množina okamžiků s daným uspořádáním. Modely času se rozlišují podle:

- Uspořádání
  - Lineární – čas roste od minulosti k budoucnosti lineárně
  - Větvený (čas možných budoucností) – lineární minulost až do teď, pak se větví do několika časových linií reprezentujících možný sled událostí. Každá linie se může dále větvit.
  - Cyklický – opakující se procesy. Př. týden, každý den se opakuje po sedmi dnech.
- Hustoty
  - Diskrétní – spolu s lineárním uspořádáním. Každý okamžik má právě jednoho následníka.
  - Hustý – Mezi každými dvěma okamžiky existuje nějaký další
  - Spojitý – každé reálné číslo odpovídá bodu v čase.
- Omezenost času – Omezený – nutnost zejména kvůli reprezentaci v počítači, Neomezený.
- Absolutní /relativní čas – Absolutní se vyjádří hodnotou, také ale potřebuje počátek. Relativní vyžaduje nějaký počátek, čas se pak vyjádří jako vzdálenost a směr od počátku.

Datové typy pro čas jsou:

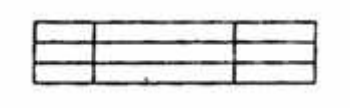
- Časový okamžik (instant) – DATE, TIME, TIMESTAMP
- Časový úsek (time period) – doba mezi dvěma časovými okamžiky (15:30-16:00)
- Časový interval (interval) – doba o specifikované délce, ale bez konkrétních krajních bodů (30 minut)
- Množina časových okamžiků (instant set)
- Množina časových úseků (temporal elements)

Vztah událostí a času (snapshot)

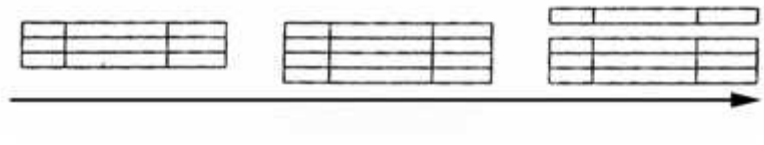
- Čas platnosti (valid time)
  - Čas, kdy byla událost pravdivá v reálném světě. Může být v minulosti přítomnosti i budoucnosti.
  - Reprezentace času platnosti – časový okamžik, doba, časový úsek, množina okamžiků
  - Čas platnosti může být přidružen k atributům, množině atributů, celé n-tici nebo objektu
- Transakční čas (transaction time)
  - Čas, kdy byl fakt reprezentován v DB. Nabývá pouze aktuální hodnoty. Monotónně roste.
  - Reprezentace transakčního času – časový okamžik (nová n-tice se stejným klíčem – logické odstranění původní), časový úsek (teď, dokud nezměněno), tři časové okamžiky (čas zaznamenání začátku v reálném světě, konce události v reálném světě, logického odstranění události z db), množina časových úseků

Snapshot

Datový model nepodporující čas platnosti ani transakční čas. Je to klasický relační model. Každá n-tice je fakt platný v reálném světě. Při změně reálného světa jsou do relace prvky přidávány nebo z ní odebírány.



Další datové modely mohou být valid-time relace (podporuje čas platnosti, umožňuje klást dotazy o faktech z minulosti i budoucnosti), transaction-time relace (podporuje pouze transakční čas, umožňuje získat informaci ze stavu db v nějakém okamžiku v minulosti), bitemporální, temporální.



Obr – transaction-time

Temporální SQL

Temporální datový model obsahuje objekty s přesně danou strukturou, omezení pro dané objekty a operace na daných objektech (temporální dotazovací jazyky). Temporálních dotazovacích jazyků je velké množství. Nejčastěji jsou založené na SQL. Typy jsou:

- Relační – HQL, HSQL, TDM, TQuel, TSQL, TSQL2
- Objektově orientované – Matisse, OSQL, OQL, TMQL

TSQL2

Temporal SQL 2 – měl sjednotit přístupy k temporálním datovým modelům. Je to nadmnožina SQL92.

V TSQL2 je časová osa na obou koncích omezena, ale dostatečně daleko (18 miliard let). U časových údajů jsou možné různé granularity. Časové typy: DATE, TIME, TIMESTAMP, INTERVAL, PERIOD.

Datový model je bitemporální. Řádek je orazítkován množinou bitemporálních chrononů. Bitemporální chronon je dvojice (chronon transakčního času, chronon času platnosti). Př. Relace ZAMESTNANEC – umístění lidí v odděleních určitého podniku. Schéma (Jméno, Oddělení) + časové razítko.

Př. SELECT – komu byl předepsán nějaký lék – výsledek bez podpory času

```
SELECT SNAPSHOT Jmeno FROM Predpis
```

21. Uživatelské rozšíření databázových systémů – data cartridge, příklady použití

Oracle Extensibility Framework – cartridge – způsob uživatelského rozšíření databáze Oracle. Implementuje se pomocí sady rozhraní pro jednotlivé přístupy (k datům, indexům, ...).

Data cartridge

Data cartridge je způsob uživatelského rozšíření databáze Oracle (program, knihovna, ...). Přitom obdobné možnosti mají i servery ostatních výrobců – Data blade (IBM Informix Server), Component Integration Layer (Sybase). Data cartridge implementujeme pokud je nutné zpracování komplexních dat (neodpovídají standardním relačním informacím) a nutná snadná manipulace s takovými daty.

Data cartridge je rozšíření serveru. Může definovat:

- Nové datové typy a jejich funkčnost – zaměřené obvykle na konkrétní oblast (obraz, zvuk, otisky,…)
- funkce

- Nové typy indexů (prostorové, full-textové,…)
- Nové definice výpočtu ceny přístupu k datům

Příklady data cartridge:

- Multi-oborové – datové řady, statistické výpočty, prostorové databáze, multimédia
- Specializované – finančnictví, právní systémy

Rozšiřitelnost = možnost přidávání nových datových typů a programů (funkcí) zabalených do speciálního modulu. Rozšíření se integruje prostřednictvím sady rozhraní pro přístup – k datovým typům, vykonáváním příkazů, vykonáváním dotazů, indexování,...

Příklady použití

Na serveru je možné rozšířit např. datové typy – LOB (Large objects).

Large objects

Součást rozšíření Oraclu – LOBy. Jsou to standardní typy pro ukládání objemných dat na serveru, Až 4GB dat. Jsou dva podtypy:

- Externí (pouze pro čtení)
  - BFILE- samostatný binární soubor uložený vně db v OS.
- Interní
  - CLOB – znakový typ v univerzální znakové sadě serveru
  - NCLOB – znakový typ v národní znakové sadě serveru
  - BLOB – binární typ

Ve sloupci tabulky je uložen pouze deskriptor odkazující na samotná data. Hodnoty pro xLOB sloupce – NULL, EMTY_CLOB(), EMPTY_NCLOB(), EMPTY_BLOB() – deskriptor odkazující na prázdná data.

Manipulace – Oracle nabízí pro manipulaci balík BDBMS_LOB)vytváření, mazání, kopírování). Manipulace s daty xLOBU se provádí po částech prostřednictvím bufferu.

Indexace – Oracle nenabízí standardní možnost indexovat LOBy, nabízí však možnost využití poskytovaného rozhraní pro implementaci vlastních indexů.

Př. deklarace BLOB:

```
CREATE TABLE Album(id NUMBER PRIMARY KEY, popis VARCHAR2(100), foto BLOB);
```

Extensible Server Execution Environment

Server dovoluje psát implementace procedur a metod v řadě jazyků.

- Nativním PL/SQL – ne moc efektivní pro vědecké výpočty
- Jazyku Java – běží v interním JVM
- V čemkoli s konvencí volání dle jazyka C a zkompilovaném do DLL.

Java kód na serveru:

- CREATE JAVA
- CREATE PROCEDURE/FUNCTION
- Externí program Loadjava

Příklad standardní cartridge Oracle Text

Rozšíření pro vyhledávání v textových sloupcích.Podpora indexování a vyhledávání v textech. Sloupce libovolného textového typu včetně CLOB a NCLOB. Veškeré podpůrné balíky a objekty definované ve schématu CTXSYS.

Index lze vytvořit příkazem:

```
CREATE INDEX FULLTEXTINX ON documents (TEXTCOLUMN) INDEXTYPE IS
CTXSYS.CONTEXT
```

Index je využíván v operátoru CONTAINS (na základě indexu se najdou všechny dokumenty obsahující obě slova):

```
SELECT id, title, score(1) FROM documents WHERE CONTAINS(
       TEXTCOLUMN, 'text AND retrieval', 1) > 0;
```

22. Data Warehousing, modely datových skladů

Základní problémy u běžných transakčních databázových systémů (OLTP):

- Nedosažitelnost dat skrytých v transakčních systémech
- Dlouhá odezva při plnění komplikovaných dotazů
- Složitá, uživatelsky nepříjemná rozhraní k databázovému softwaru
- Cena v administrativě a složitost v podpoře vzdálených uživatelů
- Soutěžení o počítačové zdroje mezi transakčními systémy a systémy podporujícími rozhodování.

Cestou k řešení těchto problémů je použití datových skladů, tzv. Data warehouse – DW

Data warehouse

Označují DB architekturu používanou pro údržbu historických dat, která jsou získána z jedné nebo více operativních DB. Typicky tato data jsou vyčištěna a restrukturována pro podporu dotazů, agregací a analýz. Data z DW jsou aktualizována v delších časových intervalech, jsou vyjádřena v jednoduchých uživatelských pojmech, jsou sumarizována pro rychlou analýzu.

DW je obrovská databáze obsahující data za dlouhé časové období. Objemově zabírá stovky GB až několik TB. Nejsou tu kladeny nijak důrazné požadavky na správnost a úplnost dat.

Klíčovým prvkem DW je integrace vlastních + externích dat.

Cílem je poskytnout ne operativní data, ale tato data přeměněná ve strategické informace.

Architektura Data Warehouse

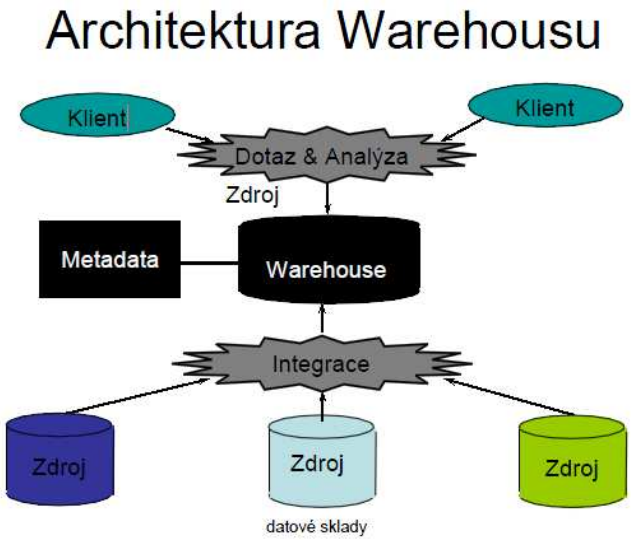
DW v informačním prostředí organizace:

Analogie s průmyslovou výrobou:

- Výroba dat (OLTP – selektivní dotazy)
- Skladování dat (DW, datová tržiště – dotazy intenzivní na data)
- Prodej dat (OLAP – Online analythical processing)

Datová tržiště jsou menší warehousy rozsahem pokrývající pouze část organizace (např. marketing) a nevyžaduje celopodnikové informace.

Architektura samotného Warehouse je pak:



Warehouse je tedy kolekce dat separovaná od operační databáze – určená pro manažery a exekutivu, integrovaná, stálá. Warehouse je kolekce prostředků – sdružování dat, čistění, integrování, data mining, dotazování, výpisy, analýzy, monitorování, administraci warehousu.

Dva přístupy:

- Query-driven (Eager = dychtivé)
- Warehouse (Lazy = zpožděné)

Výhody Warehousingu

- Vysoký výkon dotazování
- Nepřístupnost z vně warehousu
- Lokální zpracování bez ovlivnění zdrojů
- Může operovat při nepřístupných zdrojích
- Dotazování na data neukládaná v DBMS

Modely datových skladů

- Relační
- Hvězdice a vločky
  - Hvězdice – tabulka faktu, tabulka dimenzí, měřítka (Kč, cm)
  - Hierarchie dimenzí – vločkové schéma, konstelace
- Krychle – pohled tabulky faktů, multi-dimenzionální krychle

Operátory (typické OLAP operace)) warehousu jsou pak:

- Slice and dice – řez a výřez (selekce a projekce) – redukce dimenzionality dat
- Roll-up, drill-down – zvýšení/snížení stupně agregace (srolování, zavrtání)
  - Roll-up – od prodeje podle města k prodeji podle obvodu
  - Drill down – od prodeje podle obvodu k prodeji podle města
- Pivoting (rotating) – přeorientování vícerozměrného pohledu na data – změna vizualizace dat kostky
- Další

MOLAP – Multidimenzionální OLAP

- Datová krychle (obsahuje fakta)
- Hierarchické dimenze (částečné, či totální uspořádávání)

- Vločkové schéma – hlavní tabulka faktů je v relaci s dimenzionálními tabulkami, přes cizí klíče, dimenzionální tabulky mohou být také v relaci s dalšími subdimenzionálními tabulkami podobně jako hlavní tabulka faktů. Vytváří hierarchie dimenzí.
- Hvězdové schéma – je speciální případ vločkového, dimenzionální tabulky již nejsou v relaci s dalšími subdimenzionálními tabulkami, žádné hierarchie, jednodušší.

ROLAP – Relační OLAP

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

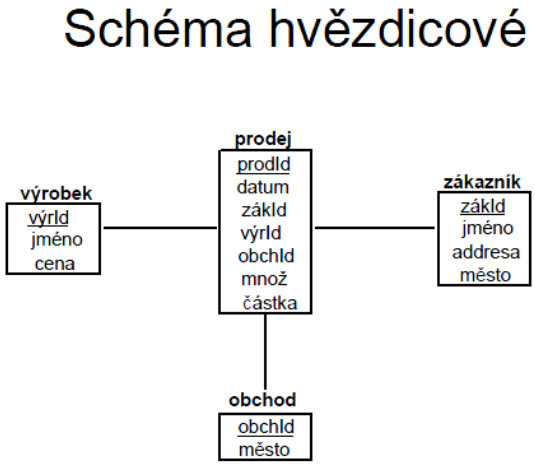
Užívá relační, nebo rozšířený relační DBMS, např. server METACUBE (Informix), pracuje s relačními tabulkami uspořádanými do hvězdy/vločky, adresuje pomocí klíče, data jsou neagregovaná.

Hvězdicové schéma

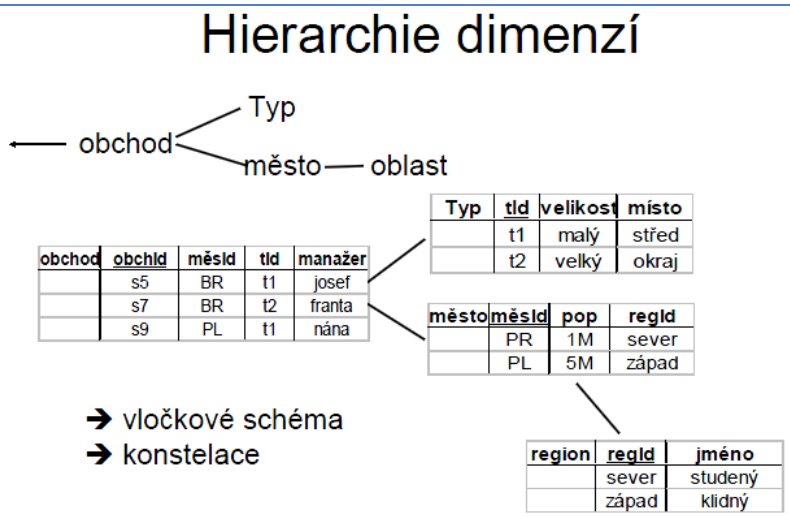
Na základě dotazu se pospojují potřebná data do vícerozměrné tabulky (nebo více tabulek), do kterých lze klást SQL dotazy. Pro častější dotazy si uchovávají předem připravené vícerozměrné tabulky. Data bývají modelována vícerozměrně (rozměry mohou být např. čas prodeje, místo prodeje, prodavač, výrobek, atd.)

Obsahuje:

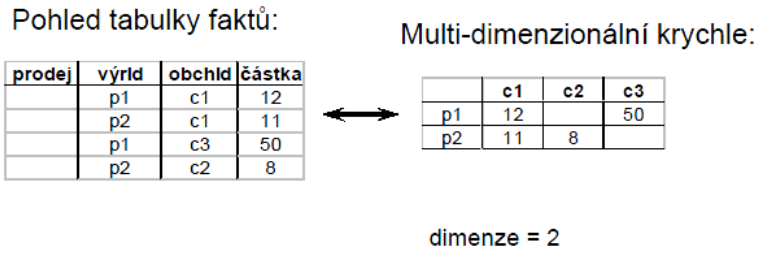
- tabulku dimenzí – položky sledované oblasti
- tabulku faktů – struktura klíčů hodnot



Hierarchie dimenzí



Krychle



23. Srovnání OLAP a OLTP, výhody a nevýhody, vhodnost užití.

OLAP je hlavní komponentou data Warehouse. OLTP pak představuje transakční procesing – operativní DB.

OLAP databázový systém (Online Analytical processing)

Je orientovaný na trh (prodej dat). Popisuje zpracování dat ve warehousu.

Modelování DW – normalizované tabulky optimalizované pro INSERT, UPDATE.

Oproti OLTP umožňuje rychlé získání výsledků na analytické dotazy. OLAP databáze jsou určeny pro historická data.

Schéma hvězdičky a vločky.

Agregovaná data (nenormalizovaná = redundantní)

Výhody:

- vysoký výkon dotazování
- poskytuje modifikované,agregovaná, sumarizovaná data a historické informace
- podporuje komplexní dotazy složité dotazy pro analytické zpracování

Nevýhody – v operativních DB by složité OLAP dotazy vyústili do nepřijatelné odezvy. Klade také specifické požadavky na DB systémy, které mohou být pak velmi drahé, nebo to neumějí a musí se rozšiřovat pomocí speciálních modulů.

Použití: byznys reporty o prodeji, marketing, management reporty, rozpočty, finanční předpovědi a reporty. Prostě různé analytické a statistické data.

Typické OLAP operace jsou zmíněny v otázce č. 22.

Data bývají modelována vícerozměrně – v obchodním data warehouse mohou být těmito rozměry např. čas prodeje, místo prodeje, prodavač, výrobek, ... Rozměry mohou být navíc hierarchické (den-měsíc-rok). Podporuje spojování více tabulek pomocí odkazu na řádky tabulek. Používají speciální organizaci dat, přístupové a implementační metody, jež nejsou obecně v komerčních Db systémech určených pro OLTP podporovány.

Na DB stroje jsou kladeny speciální požadavky:

- objem zpracovávaných dat
- rychlost odezvy analytického systému je důležitá
- počet uživatelů současně pracujících s DB není zajímavý (počet pracovníků vyššího managementu je omezen a pro pracovníky nižších stupňů bývají údaje z DW převedeny do menších specializovaných DB)

S těmito omezeními se vyrovnává OLAP uzpůsobením stávajících systémů pro práci s vícerozměrnými daty (přidáním modulu, který to zajišťuje a prostředků pro jeho ovládání), nebo vytvořením speciálního systému správy dat, určeného pouze pro OLAP.

OLTP databázový systém (Online transaction Processing)

Je zákaznicky orientovaný (na výrobu dat). Popisuje zpracování v operační databázi.

Modelování DW – odvozené tabulky, redundantní data, optimalizace pro dotazy, procesní logika ve schématech.

ER schéma.

Aktuální data – lze považovat i za slabinu, při výpadku vznikají ztráty pro byznys.

Výhodou OLTP je, že je jednoduché a efektivní.

Nevýhody:

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

Příkladem je bankomat

Vhodnost užití

- OLAP
  - Většinou čte
  - Dlouhé komplexní dotazy
  - GB-TB dat
  - Sumarizovaná konsolidovaná data
  - Vedoucí pracovník, analytik jako uživatel.
- OLTP
  - Většinou updatuje
  - Více malých transakcí
  - MB-TB dat
  - Prvotní data
  - Čerstvá data
  - Konsistence, obnova je kritická
  - Administrativní uživatel

24. Metody dolování znalostí – datamining

„Kde jsou ztraceny informace? – v datech. Kde jsou ztracena data? – v databázích“

Velká množství dat a složitost dotazů tlačí na limity dnešních Warehousů. Potřebné jsou lepší systémy: snáze použitelné a poskytující kvalitní informace. Řešením je datamining – dolování znalostí.

Datamining je prostředek pro dotazy ve warehousu. Dolováním dat nazýváme proces netriviálního získávání dříve neznámé a potencionálně užitečné informace z dat.

Cyklus procesu je následující: Formulace problému, datová a problémová analýza, výběr relevantních dat, předzpracování (integrace do jednotného formátu, transformace, odvozování), dolování nových hypotéz, interpretace výsledků, využití a zhodnocení celého procesu.

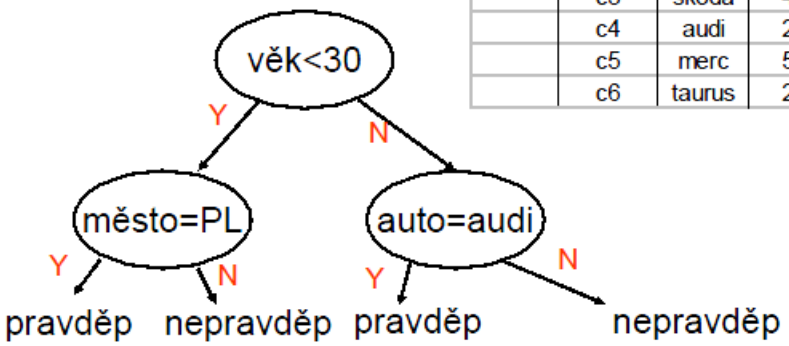
Metody dolování znalostí

Rozhodovací stromy

Jsou to množiny objektů zadaných atributy a doménami, rozdělených do klasifikačních tříd. Třídy jsou charakterizovány hodnotami jednoho, nebo více atributů. Hledá se při jakých hodnotách atributů předpovídajících či vstupních připadnou objekty do těchto tříd. Konstrukce stromu je založena na informačním zisku při rozšíření stromu o další uzly.

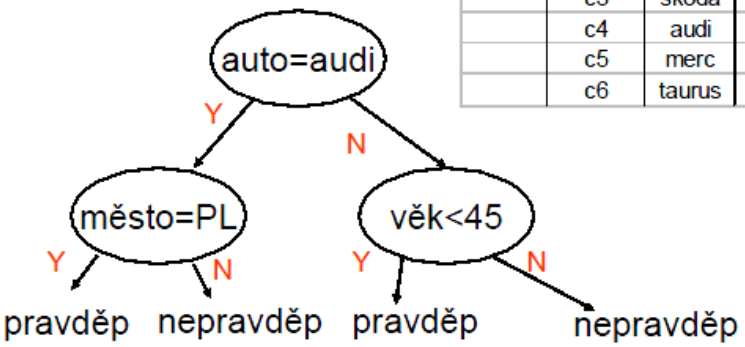
Př. Zjištění údajů jací zákazníci se zajímají o nové modely aut. Najdeme trénovací množinu – tabulka s údaji.

prodej	zákld	auto	věk	město	novéauto
	c1	audi	27	BR	ano
	c2	škoda	35	PL	ano
	c3	škoda	40	BR	ano
	c4	audi	22	BR	ano
	c5	merc	50	PL	ne
	c6	taurus	25	PL	ne



Jiná možnost stromu může být:

prodej	zákld	auto	věk	město	novéauto
	c1	audi	27	BR	ano
	c2	škoda	35	PL	ano
	c3	škoda	40	BR	ano
	c4	audi	22	BR	ano
	c5	merc	50	PL	ne
	c6	taurus	25	PL	ne



Z toho vyplývá, že strom nemůže být příliš hluboký – jinak by neměl statisticky významné hodnoty pro rozhodování v nižších úrovních. Je třeba vybrat strom, který co nejspolehlivěji predikuje výsledky.

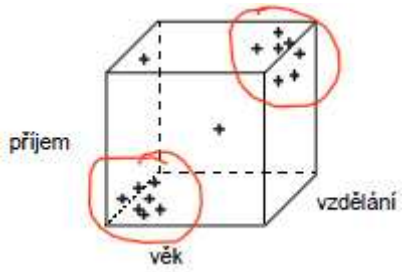
Shlukování (clustering)

Analyzuje, zda se množina subjektů přirozeně rozpadá na výrazné podmnožiny objektů podobných a přitom nepodobných objektům podmnožin ostatních. Tvoří ji řada metod, např. hierarchické/nehierarchické, disjunktní/překrývající se,... Soustředí se na výběr charakterizujících atributů, podobnost a vzdálenost objektů, počet shluků rozkladu.

Shluky tedy obsahují podobná data, která jsou užitečná pro klasifikaci a vyhledávání.

Problémy:

- Je zadán dostačující počet shluků?
- Nalezení nejlepších shluků
- Jsou shluky sémanticky smysluplné? Např. Shluk sázkařů?



Asociační pravidla

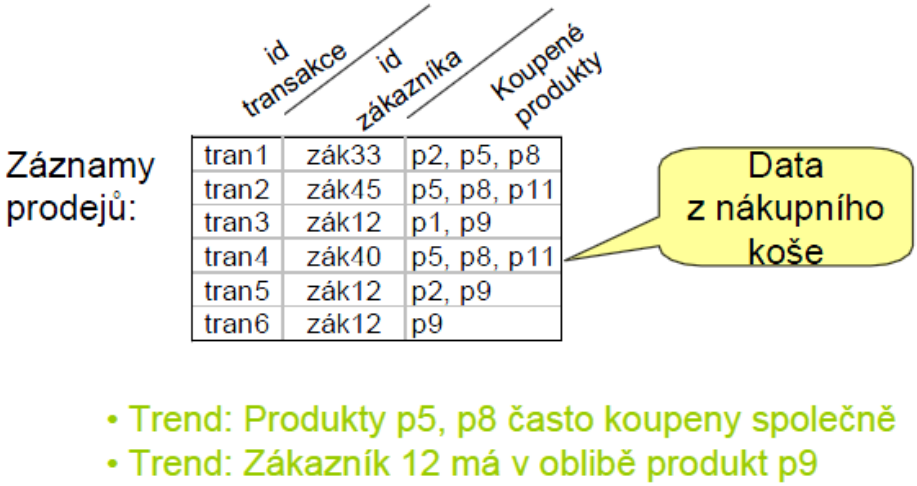
Nacházení typických vzorů v datech, tj. určení, které kombinace atributů se spolu vyskytují nejčastěji. Vhodné především pro analýzy nákupních košíků, nebo web-mining.

Asociací nazýváme vztahy mezi atributy:

- Klasické - mezi dvěma podmnožinami atributů
- Transakční – v rámci dané množiny atributů
- Agregované – mezi charakteristika podmnožina atributů

Nejčastějším použitím analýzy asociací, a zároveň jejím ilustrativním příkladem je tzv. analýza nákupního košíku. Ta se zabývá hledáním kombinací produktů, které se ve vstupních datech (nákupním koši spotřebitelů) vyskytují významně častěji spolu. Cílem je odhalit pravidla typu: Při nákupu zboží A a C spotřebitele výrazněji častěji nakupují zboží D a B. Odhalení takovýchto kombinací pomáhá marketingovým odborníkům v organizování nabídky, či společných balíčků produktů.

U této metody je problém nalezení všech častých množin.



Další metody

- Neuronové sítě – zjednodušený model nervové soustavy, skládá se z vrstev neuronů, které si pomocí synapsí vyměňují informace.
- Metoda nejbližších sousedů – model srovnává zpracovávaný případ se známými případy, které se novému nejvíce podobají ve vstupních proměnných.
- Lineární regrese – klasický statistický model
- Genetické algoritmy – řídí proces učení modelu, vychází ze vzoru biologické evoluce, kdy následující generace modelu je dokonalejší, než jeho rodič :-).

Modely vytvořené jednak na základě různých algoritmů a jednak naučených na různých datech je možné sdružovat a kombinovat, což se osvědčilo i v praxi. Pokud se výsledky několika modelů odlišují, modely pak mezi sebou hlasují o nejlepší predikci.

Jinou možností je sériové spojení modelů. Výstup (predikce) jednoho modelu se stává vstupem do dalšího

25. Dokumentografické systémy, kritéria vyhodnocení dotazu

Dokumentografické systémy (DIS) jsou určeny pro zpracování informací v podobě textu v přirozeném jazyce bez pevné vnitřní struktury. Vznikly postupnou automatizací postupů používaných v knihovnictví.

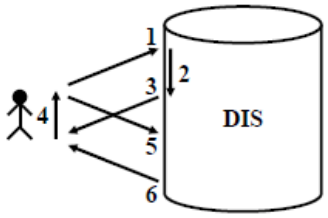
Texty se zde nazývají dokumenty a jako příklad DIS si můžeme uvést knihovnu – jednotlivé dokumenty představují knihy, časopisy, sborníky a další tiskoviny, které knihovny obvykle poskytují.

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 jsou boolský model a vektorový model.

Práce s DIS

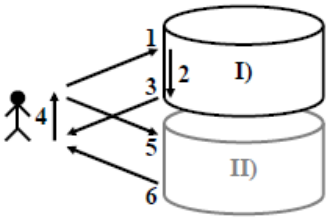
1. Zadání dotazu
2. Porovnání
3. Získání seznamu odpovídajících dokumentů
4. Ladění dotazu
5. Vyžádání dokumentu
6. Obdržení textu



Struktura DIS

I) Systém zpřístupnění dokumentů
- Vrací sekundární informace
  - Autor
  - Název
  - ...

II) Systém dodání dokumentů
- Někdy není řešen pomocí SW



Vyhodnocení dotazu

- Přímé porovnání – náročné na čas.
- Nutné vytvoření modelu dokumentu – ztrátový proces obvykle založený na identifikaci slov v dokumentech. Výsledkem strukturovaná data vhodná pro porovnávání. Dotaz se upraví do odpovídající podoby a následně se porovná s modelem dokumentů.

Problémem v DIS je porozumění textu:

- Jedno slovo může používat stejný tvar pro různé pády a další gramatické jevy, jeden tvar slova může mít různý význam (pět – číslovka vs. sloveso)
- Jednotlivá přiřazení jsou závislá na subjektu, který dokument píše, nebo čte – dva lidé mohou jednomu slovu přikládat zcela nebo částečně jiný význam. Výsledkem je situace, kdy dva různí čtenáři nemusí přečtením získat stejnou informaci, jako autor, ani navzájem.
- Tyto problémy ještě narůstají při přechodu slov k větám.

Částečným řešením problému porozumění textu je předzpracování textu:

- Zahrnutí lingvistické analýzy – určení správného významu slov ve větě
- Lemmatizace – přiřazení správného lemmatu jednotlivým slovům (slovní druh, osoba, číslo,...)

- Označování víceslovných spojení

Výsledkem nejednoznačností žádný existující DIS nedává ideální výsledky. Proto měříme kvalitu výsledné množiny dokumentů na základě čísel:

- Přesnost (Precision) – Pravděpodobnost, že dokument zařazený v odpovědi je skutečně relevantní.
- Úplnost (Recall) – Pravděpodobnost, že skutečně relevantní dokument je zařazený v odpovědi

Kritéria vyhodnocení dotazu

- **Kritérium predikce –** Při formulaci dotazů je třeba uhádnout, které termy (slova) byly v dokumentu autorem použity pro vyjádření dané myšlenky (problémy způsobují synonyma, překrývající se významy slov, opisy jedné situace jinými slovy). Částečným řešením je zařazení tezauru, který obsahuje – hierarchie slov a jejich významů, synonyma slov, asociace mezi slovy. Tazatel může tezaurus využít při formulaci svých dotazů. Vhodné je uživateli pomoci s odstraněním nevhodných částí dotazu, které nepopisují relevantní dokumenty a naopak s přidáváním formulací, které je popisují.
- **Kritérium maxima –** Tazatel obvykle není schopen (ochoten) procházet příliš mnoho dokumentů do té míry, aby se rozhodl, zda jsou pro něj relevantní, nebo ne. Obvykle 20-50 podle velikosti. Potřeba nejen dokumenty rozlišovat na odpovídající/neodpovídající dotazu, ale řadit je na výstupu podle míry předpokládané relevance. V důsledku kritéria maxima se při ladění dotazu uživatel obvykle snaží zvýšit přesnost – malé množství dokumentů v odpovědi, obsahující co největší poměr relevantních dokumentů. Některé oblasti použití vyžadují co nejvyšší přesnost i úplnost (Právnictví).

26. Modely dokumentografických systémů

Úrovně modelů:

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

Boolský model DIS

Automatizace postupů používaných v knihovnictví.

Dokument je reprezentován množinou termů (slov) (obsažených v dokumentu, popisujících dokument). Dotaz je vyjádřený a vyhodnocuje logický (boolovský) výraz (AND – v dokumentu se vyskytují oba termy, OR – alespoň jeden term, NOT – daný term se nevyskytuje).

Tvary dotazů mohou být:

- S boolovskými operátory (AND, OR, AND NOT), Lze využít ve výrazu i víceslovné termy a údaje jako atributy (‘databaze’ AND (autor=’Salton’)). Lze využít zástupné znaky v termech (*, ?)
- S proximitními operátory – proximitní omezení – nahrazení operátoru konjunkcí, vyhodnocení dotazu, ověření v primárním textu (delší čas pro vyhodnocení), doplnění indexu o pozice termů v dokumentech.
- S metasymboly
- V přirozeném jazyce

Indexace Boolského DIS

V základní podobě používají index v podobě lineárního seznamu termů. Přiřazení množiny termů, které jej popisují ke každému dokumentu:

- Ruční – nekonzistence
- Automatická – konzistentní, ale bez porozumění textu
- Řízená – předem daná množina termů

- Neřízená – množina termů se mění s přibývajícími dokumenty

V dokonalejší podobě - Dále se indexuje pomocí Tezauru – vnitřně strukturovaná množina termů (mohou obsahovat specifikace - synonyma, příbuzné termy, hierarchie užších/širších termů, preferované termy, nejširší termy) a Stop-listu – obsahuje nevýznamová slova.

Při indexaci příliš obecná slova nejsou pro identifikaci vhodná, příliš specifická také ne

Nevýhody Boolského DIS

- Formulace dotazů je spíše uměním, než vědou – nedokonalá formulovatelnost dotazu
- Nemožnost ohodnotit vhodnost vystupujících dokumentů
- Všechny termy (slova) v dotazu i v identifikaci dokumentu jsou chápány jako stejně důležité
- Nemožnost řízení velikosti výstupu
- Některé výsledky neodpovídají intuitivní představě.

Zdokonalení nedostatků – **rozšířený bool. Model** – zavádí váhy termů.

Vektorový model DIS

Je cca o 20 let mladší než Boolský DIS. Patří do 70. Let 20. Století.

Snaží se minimalizovat, nebo odstranit nevýhody Boolských DIS.

Dokumenty jsou reprezentovány pomocí vektoru vah termů. Pro podobnost mezi vektorem dotazu a vektorem dokumentu se používá podobnostní funkce (výslednou podobnost ovlivňuje velikost dotazu i velikost vektorů jednotlivých dokumentů, delší vektory jsou zvýhodněny) – vhodné vektory normalizovat, aby neovlivnili podobnost. Eliminuje se tak tím vliv délky vektoru na podobnost.

Normalizace vektorů se provádí během indexace (nezatěžuje vyhledávání), nebo během vyhledávání (součást definice podobnostní funkce, zpomaluje odezvu).

Dokumenty na výstupu jsou řazeny sestupně podle podobnosti. Lze omezit počet vrácených dokumentů a zadat požadavek na minimální nutnou podobnost.

```
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
```

Distribuované DIS

Viz také otázka č. 18.

Data a funkce jsou rozdělena mezi více počítačů – to přináší lepší rozšiřitelnost (zvýšení výkonu přidáním nového počítače) a robustnost (výpadek jednoho počítače neovlivní funkci ostatních) a transparentnost (uživatel si rozdělení nemusí být vědom).

Ukládaná data – primární data (dokumenty), sekundární data (autor, název, rok vydání,...), index – lze rozdělit na různé stroje podle toho, které procesy na nich běží a která data obsahují. Na jednom stroji může běžet i více procesů. Procesy DIS jsou:

- Klient – uživatelské rozhraní
- Dokumentový server – subsystém dodání dokumentu s primárními daty. Např. nezávislý web server

- Server indexu – subsystém zpřístupnění dokumentu s indexem a sek. Daty.
- Integrační uzel – specifický proces zajišťující koordinaci mezi ostatními uzly. Přebírá dotaz od uživatele a určuje strategii vyhodnocení v distribuovaném prostředí na základě znalosti umístění jednotlivých zdrojů. Rozesílá parciální dotazy na jednotlivé index servery a sestavuje výslednou odpověď z odpovědí na parciální dotazy.