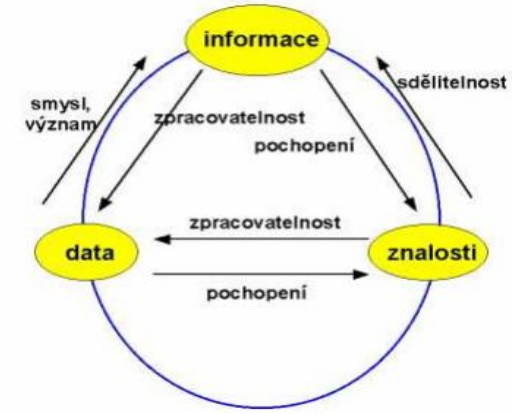
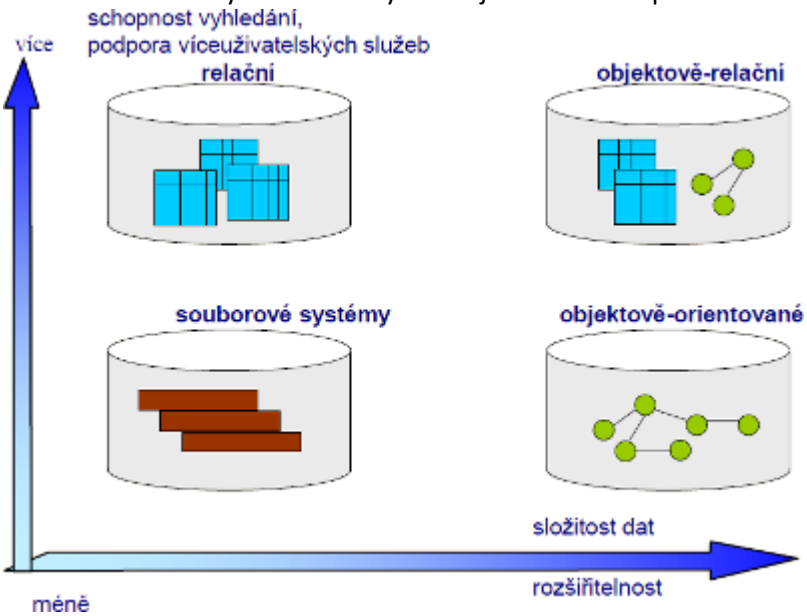
DB2 – otázky ke zkoušce

1. Standardizace vývoje

- **Otázka: Standardizace vývoje databázové technologie, norma ISO/IEC 9075 – rozšíření možností relačního modelu a trend vývoje databázové technologie (No SQL, New SQL)**



-
- Informace + ontologie = znalost (znalost je proces! Viz DBM2)
- Historie
  - Uvedení relačního modelu dat RMD E.F.Codd
    - Trvalo 10 let, něž relační databázová technologie z hlediska výkonu DBS v reálném prostředí srovnatelná s tehdejšími síťovými a hierarchickými databázemi
  - první implementace
    - IBM ze 70. let Systém R (předchůdce)
- Současný stav
  - V současnosti existuje okolo 20 OOSŘBD
  - Počet nasazení OOSŘBD neplnil předpoklady
  - Funkce a výkon těchto systémů jsou dosud na poměrně nízké úrovni



-
- Provoz a údržba
  - Firmy se starají o podnikové aplikace
  - Stávajících systémů málo → špatná databázová schémata, úpadek databází
  - Často mizí role centrálního správce, decentralizovaný přístup s více skupinami DBA
  - Databáze jsou většinou relační
  - Bylo zavedeno několik databázových modelů
    - OO = objektově-orientovaný
    - OR = objektově-relační

        - XML
        - RDF
    - Ale OR + OO SŘBD (systém řízení báze dat) nejsou konkurenceschopné – nedostatek teoretických základů
- Vývoj databází v 90. letech
    - Snaha integrovat heterogenní data v podniku a rozšiřovat možnosti SŘBD o další datové typy – ukládat do DB vše
    - Možnost, jak technicky řešit = rozšířit relační tabulky SQL o objekty (netriviální rozsáhlé objekty typu text, audio, video,…)
    - Nutno vyvíjet nové dotazovací jazyky, které umožnily nejen nové typy dotazů
    - Přehodnocení dotazování
- Vývoj databází v 90. letech
    - Vznik tzv. „univerzálních serverů“ s ad hoc přidávanými novými datovými typy
    - Značné úsilí věnováno řešení sémantických konfliktů mezi daty několika databází a transakcím nad více databázemi
    - Neúspěšnost těchto architektů v rámci IS podniku nakonec vedla k vývoji datových skladů
        - Potřebná data se „vypumpují“ z operačních databází, vyčistí a uloží do databáze speciální
        - Datová pumpa – proces ETL (extrakce, transformace, loading) – nezbytný prostředek i mimo DW
- Integrace heterogenních dat
    - Problém integrace heterogenních dat se řeší v databázové historii neustále
    - 80.+90. léta nabídla řadu technik
    - Možná řešení
        - Přístup přes globální schéma
        - Federativní databáze
            - Více zdrojů, jedno rozhraní
        - Multidatabáze
    - Problém, že řešení jsou statická – potřebujeme dynamická
    - Problém, že předem nevíme, kterou databázi budeme potřebovat
    - Nejméně statické řešení je **architektura federace**
    - Problém, že potřeba integrovat i nestrukturovaná či semistrukturovaná data
        - Tzn. Např. Jmenuji se Jméno příjmení namísto Jméno:…. Příjmení:…
        - Řešení = nový datový model a jazyk XML pro popis semistrukturovaných dat
- Začátek dalšího tisíciletí
    - Přelom tisíciletí je i v databázové technologii ve znamení internetu a webu
    - Web = jednoduchý, univerzální standard pro výměnu informací
    - Po r. 2000 – intenzivní vývoj aktivit v technologii XML
    - Dále pak rozvoj NoSQL databází (kvůli Big Data)
- XML databáze
    - Dvě základní architektury pro ukládání XML dat do databáze
        - Databáze zpřístupňující XML data uložená např. v relačních SŘBD (24 komerčních produktů) – např. medicínská data
        - Nativní XML databáze (39 komerčních produktů)
    - Existují různé standardy, mění se
- Konkrétní problémy
    - Velikost databází
        - Petabajt, exabajt, zettabajt (10^21)
        - Vertikální škálování = změna vlastností stávajících prvků systému (kvalitativní změna)
            - Přidání paměti, procesoru, disku…
            - Vs. Horizontální škálovatelnost = kvantitativní = přidání dalšího počítače, serveru
                - Není hardwareově omezeno
    - Heterogennost databází
    - NoSQL databáze (Not Only SQL)
        - Třída nerelačních datových úložišť

        - Key-value
            - Lehký zápis i dotazování
            - Snadné přidávání atributů (třeba jen dočasných)
            - Ale redundance!!
        - Rozdělování databáze mezi více (levných) strojů přidávaných dynamicky = „horizontální škálování“
            - Efektivnější, levnější
        - Využití v sociálních sítích – horizontální škálování zahrnuje tisíce uzlů
        - Nejvlivnější NoSQL databáze jsou od Googlu a Amazonu
        - Vhodné pro určité aplikace využívající velká data
        - Vývoj speciálních SŘBD – Big Data Management Systems
- Příklady nových databázových architektur
    - ASTERIX 17
        - Využívá fuzzy spojení, nový dotazovací jazyk, práce s grafy
        - AsterixQL, Piglet..
    - ORACLE
        - Oracle Big Data Appliance – kombinace SQL Hadoop a NoSQL
- NewSQL databáze
    - Vysoce škálovatelné a elastické relační SŘBD
        - Je relační – převážně vychází z SQL, ale využívá některé výhody **NoSQL(viz otázka 9)**
    - Navržené pro horizontální škálování
    - Garantují ACID vlastnosti
    - Aplikace na nich interagují s databází primárně přes SQL (včetně spojení)
    - Používají pro řízení souběžného zpracování protokol bez zamykání
    - Poskytují vyšší výkon než tradiční relační
- **Historie SQL + ISO/IEC 9075**
    - SQL - standardním neprocedurálním relačním dotazovacím jazykem, podporovaným většinou dostupných databázových systémů.
    - První návrh 1974 - SEQUEL (Structured English Query Language). Později byl zkrácen na SQL (Structured Query Language)
    - V roce 1986 byl jazyk standardizován americkou ANSI, a o rok později byl následován Mezinárodní organizací pro normalizaci (ISO).
        - Poslední normou pro jazyk SQL je z roku 2003 (ISO/IEC 9075:2003).
        - Součástí – prostředky pro definici + aktualizaci dat
        - Využití v rámci programovacích jazyků = hostitelská verze (příkazy odděleny)
        - databáze ORACLE 9i má své specifické řešení a tím je jazyk PL/SQL = rozšířením standardního jazyka o některé možnosti vyšších programovacích jazyků: větvení, skoky, cykly, vytváření procedur, funkcí a knihoven.
        - SQL obsahuje příkazy pro:
            - 1. Definici dat
            - 2. Editaci dat
            - 3. Výběr dat
            - 4. Definici přístupových práv
            - 5. Transakce Databáze (SAVEPOINT, COMMIT, ROLLBACK)

2. Vnořené SQL (Embedded)

- **Otázka: „Vnitřní“ programovací konstrukce (Embedded SQL) - procedurální prostředky v rámci jazyka SQL.**
- Relační databáze
    - Objektově relační databáze
    - →NO SQL
        - Distribuce, CAP? – nezaručují konzistenci
        - Big Data, XML DB, RDF

    - → New SQL
        - Už jsou ACID (zaručují konzistenci)
        - Umožňují distribuci
- ↑nad tím vším možnost analýzy a dolování dat (DW + Data Mining)
    - Ale pro Data Warehouse složité, opakované dotazy
- Vstup (data) → program → výstup (data)
    - Ale co když chci na ta data pustit více programů?
- Účel použití vnořeného SQL
    - Způsob použití SQL v programovacích jazycích
    - SQL vepsané ve zdrojovém kódu jiného programovacího jazyka
        - Java použije SQL, kompilátor překompiluje, pak už se zpracovává normálním překladačem
    - Všechny příkazy interaktivního SQL jdou použít ve vnořeném SQL
    - Databázový systém musí mít podporu vnořeného SQL v použitém programovacím jazyce
    - Které DBMS to umí?
        - Pozn. DBMS = Database management systém = SŘBD = systém řízení báze dat
        - IBM DB2
            - C/C++, COBOL, FORTRAN, REXX
        - Oracle
            - Java, Ada, C/C++, COBOL, Fortran, Pascal, PL/1
        - Microsoft SQL Server
            - Od roku 2008 ukončena oficiální podpora
        - MySQL
            - Nepodporováno
    - Příklad vnořeného SQL (Oracle a C)

```
#include <stdio.h>
#include <sqlca.h>
int main(void)
{
  EXEC SQL INSERT INTO osoby (prijmeni) VALUES
  ('Sequens');
  return 0;
}
```

- Jak vnořené SQL používáme?
    - Vložíme hlavičkový soubor struktury SQL Communication Area (SQLCA)
    - Příkazy vnořeného SQL začínáme direktivou EXEC SQL a ukončujeme středníkem
    - Oracle prostředí pro SQL vnořené v C/C++ se nazývá Pro*C/C++
    - Zdrojové soubory s příponou *.pc
    - Prekompilátor Pro*C/C++ přeloží *.pc na čistý C/C++ kód
        - Nahrazení konstrukcí vnořeného SQL voláními standardní run-time knihovny
    - C/C++ kód je standardně zkompilován
- Hostitelské proměnné
    - Předávání dat mezi SQL a C/C++
    - Ve vnořeném SQL označujeme dvojtečkou
    - Vstupní
        - EXEC SQL INSERT INTO osoby (prijmeni) VALUES (:prijmeni_osoby);
    - Výstupní
        - EXEC SQL SELECT prijmeni INTO :prijmeni_osoby FROM osoby;
- Indikátorové proměnné ~ hostitelské
    - Motivace: Co uložit do proměnné, když SELECT vrátí NULL?
    - Řešení: Použijeme druhou, „spřátelenou“ proměnnou, která bude indikovat, jak a zda je původní proměnná naplněna
    - Indikátorová proměnná se zapisuje bezprostředně za původní proměnnou, oddělená dvojtečkou

- U výstupních proměnných

Indikátorová proměnná	Hostitelská proměnná
-2	Oříznutá hodnota z databáze, hodnota se do hostitelské proměnné nevejde a její velikost nemůže být určena.
-1	Nedefinovaná hodnota, v databázi byla NULL.
0	Hodnota z databáze, není to NULL.
>0	Oříznutá hodnota z databáze, hodnota se do hostitelské proměnné nevejde. Indikátorová proměnná obsahuje velikost hodnoty v databázi.

  - -2 je největší problém

```
EXEC SQL SELECT pocet_deti INTO :pocet:ind_pocet
  FROM osoby WHERE prijmeni = 'Sequens';
if (ind_pocet == -1) /* NULL v databázi */
  pocet = 0;
```

    - Problém, když počet dětí není udán
- U vstupních proměnných
  - Obráceně → chci něco zapsat z programu do databáze
  - Používám příkazy INSERT nebo UPDATE

Indikátorová proměnná	Zápis do databáze
-1	Zapíše se NULL (hodnota hostitelské proměnné se bude ignorovat).
>= 0	Zapíše se hodnota hostitelské proměnné.

  - INSERT
    - Stejné použití jako při interaktivním SQL
    - Skutečný zápis proveden až po zapsání změn příkazem COMMIT

```
EXEC SQL INSERT INTO osoby (jmeno, prijmeni)
VALUES (:jmeno_osoby, :prijmeni_osoby);
```

  - UPDATE
    - Stejné použití jako při interaktivním SQL
    - Skutečný zápis proveden až po zapsání změn příkazem COMMIT

```
EXEC SQL UPDATE osoby
SET jmeno = :jmeno_osoby
WHERE prijmeni = :prijmeni_osoby;
```

  - DELETE
    - Stejné použití jako při interaktivním SQL
    - Skutečné smazání provedeno až po zapsání změn příkazem COMMIT

```
EXEC SQL DELETE FROM osoby
WHERE prijmeni = :prijmeni_osoby;
```

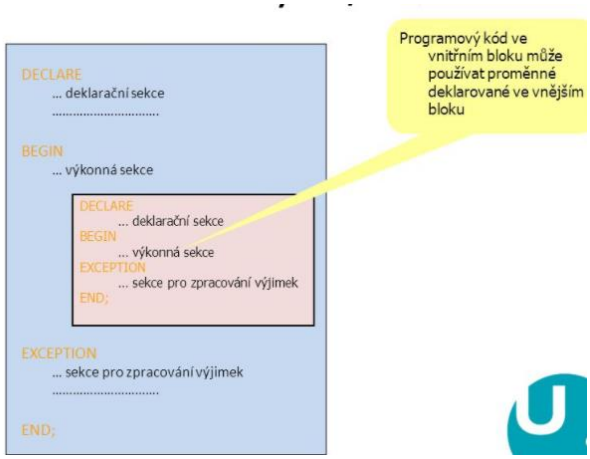
  - SELECT
    - V zásadě stejné použití jako u interaktivního SQL
    - Je třeba vyřešit, kam a jak uložíme to, co SELECT vrátí
    - Víme-li, že náš SELECT vrátí nejvýše jeden řádek, je situace jednoduchá

```
EXEC SQL SELECT jmeno, prijmeni
INTO :jmeno_osoby, :prijmeni_osoby
FROM osoby;
```

- Vrací-li SELECT více než jeden řádek, je třeba použít ke zpracování výsledku tzv. **kurzor**

Jazyk PL/SQL – procedurální jazyk

- Oproti datům vkládaným do programu musíme řešit integritní omezení
  - Některé hodnoty nemohou být vkládány (např. známka 1-5, nemohu vložit 6)
- Deklarativní vs. Procedurální programování
  - Deklarativní = co se má udělat (ale už ne způsob jak)
    - + jeden příkaz
    - - žádná kontrola na procesem získávání dat
  - Procedurální (imperativní) programování
    - Jakým způsobem získat sekvenci příkazů
    - + kontrola nad prováděním úlohy
    - - více příkazů
- Přehled SQL
  - Hl. omezení – je neprocedurální
    - Ideální příklad deklarativního jazyka
    - Vychází z potřeb, pro které byl vytvořen
    - Zajišťuje bezproblémovou manipulaci s daty
    - Sekvenční provádění
    - Bez možnosti klasických programátorských konstrukcí (cykly, podmínky, procedury, funkce, objektové programování)
    - Jazyk nebyl navržen pro návrh aplikační logiky
  - PL/SQL
    - = řešení → rozšíření, které umožní naprogramování složitějších algoritmů pro práci s daty
- Motivační příklad
  - Business pravidlo – aplikace musí umožnit přidání nových uživatelů
    - Má uživatel právo přidávat uživatele?
    - Existuje zadávaný uživatel už v DB? Ověřit (duplicity)
    - Souhlasí i opakované heslo → pravidlo kvality hesel
- PL/SQL
  - Procedural Language/Structured Query Language
  - Rozšíření SQL o konstrukce procedurálního programování
  - Kombinace deklarativního a imperativního programování
  - Možnosti
    - Umožňuje deklarovat konstanty, proměnné, kurzory
    - Možnost rozhodovacích struktur (IF/THEN)
    - Podpora dynamické deklarace
    - Podpora transakčního zpracování
    - Chybové stavy procesu je možné ošetřit pomocí výjimek
    - Podpora modularity (vkládání modulů i do sebe)
    - Podporuje dědičnost
  - Vlastnosti
    - Lze využívat SQL DML příkazy = podjazyk SQL (insert, update, delete…), transakční příkazy SQL funkce, operátory, plná podpora SQL datových typů
    - Vysoký výkon
      - Do DB může být odeslán jeden PL/SQL blok obsahující více jednotlivých SQL příkazů
      - Předkompilování uložených bloků PL/SQL → efektivnější volání příkazů
    - Bezpečnost
      - Přesun kódu z aplikace do DB – možnost skrýt interní detaily

        - Možnost tvorby vlastních funkcí, procedur, triggerů, datových typů…
        - Přesun aplikační logiky ze strany klienta na stranu serveru
            - Zapouzdření business pravidel a jiné komplikované logiky do DB
            - Modularita (=bloky) a abstrakce
        - Zajištění komplexních integritních omezení
        - Podpora výjimek
            - Předdefinované i vlastní výjimky
        - Zajištění zabezpečení dat
            - Audit
            - Uživatelská práva (omezení, přidání)
        - Kontrola syntaxe a platnosti objektů
        - Platformě nezávislý
            - Oracle je implementován v rámci mnoha hardwarových platforem – PL/SQL je ale stejné na všech těchto platformách
    - Struktura jazyka PL/SQL
        - Základní jednotkou PL/SQL je blok, spojující související deklarace a výrazy
        - DECLARE - Deklarační část
        - BEGIN – výkonná část
        - EXCEPTION – spravování výjimek
        - END
        - Identifikátory
            - V PL/SQL slouží k pojmenování konstant, proměnných, výjimek, kurzorů
            - Minimální délka jeden znak – 30 znaků
            - První znak identifikátoru musí být písmeno, následovat cokoliv
        - Proměnné
            - Typový jazyk → před prvním použitím vždy deklarovat
            - Možné během deklarace omezit
- Použití SQL dotazů v PL/SQL
    - Standardně lze používat jen DML operace
        - DDL (data definition language) lze využít jen ve formě tzv. dynamických dotazů – CREATE TABLE
    - Práce s daty v tabulkách v PL/SQL
        - Definice datového typu proměnné pomocí atributu %TYPE
            - Umí datový typ odvodit dle jiné proměnné ve stejném sloupci (např.)
            - Odvození pouze datového typu, nikoliv omezení
- Vnořené bloky PL/SQL
    - 

- Ošetření chyb
    - V zásadě se mohou v PL/SQL vyskytnout 2 druhy chyb:
        - **Syntaktické** – projeví se ještě v procesu kompilace (upozorní nás na ně překladač)
        - **Run-time** – projeví se až za běhu programu
    - Nejčastěji se vyskytují následující výjimky:

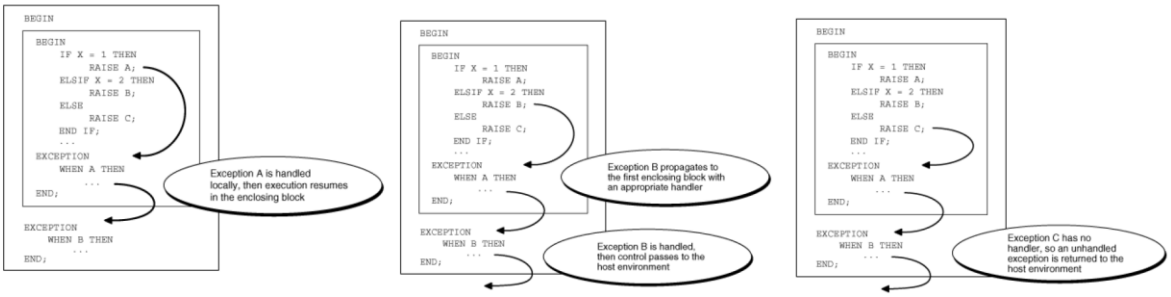
        - **DUP_VAL_ON_INDEX** výskyt duplicitní hodnoty ve sloupci, který připouští jen jedinečné hodnoty
        - **INVALID_NUMBER** neplatné číslo nebo data nemohou být převedena na číslo
        - **NO_DATA_FOUND** nebyly nalezeny žádné záznamy
        - **TOO_MANY_ROWS** dotaz vrátil více než jeden záznam
        - **VALUE_ERROR** problém s matematickou funkcí
        - **ZERO_DIVIDE** dělení nulou
    - Výjimky RAISE
    - EXCEPTION – END
    - RAISE NO_DATA_FOUND
        - V EXCEPTION je nutné nadefinovat, co se má udělat, když nebyla nalezena data
        - **EXCEPTION**
            - **WHEN <název výjimky> THEN <příkazy>;**
            - **OTHERS THEN <příkazy>**
        - **END**

    - Pro zachycení výjimky je nutné definovat její obsluhu (exception handler).
    - Každý handler obsahuje WHEN klauzuli, ve které specifikuje typ výjimky, kterou „zachytává“.
    - Speciální typem handleru je WHEN OTHERS THEN
        - Tímto handlerem jsou zachyceny všechny výjimky, které nebyly zachyceny handlery před.
        - Použití WHEN OTHERS THEN garantuje, že žádná z výjimek nezůstane neobsloužena.
        - Ošetření OTHERS musí být uvedeno jako poslední ošetření výjimek v bloku.
    - POZOR! Když si chybu už nadeklaruju, k tomu bloku EXCEPTION se ani nedostanu
    - Aktuální kód chyby vrací systémová funkce **SQLCODE** a její textový popis systémová funkce **SQLERRM**, takže při zpracování výjimky máme k dispozici tyto údaje.

    - Příklad
        - **DECLARE**
        - **v_vysledek NUMBER(9,2);**
        - **BEGIN**
            - **v_vysledek := 5/0;**
            - **EXCEPTION**
            - **WHEN OTHERS THEN**
                - **DBMS_OUTPUT.PUT_LINE(' Chyba ');**
                - **DBMS_OUTPUT.PUT_LINE('Kód chyby:' || SQLCODE);**
                - **DBMS_OUTPUT.PUT_LINE('Popis chyby:' || SQLERRM);**
        - **END;**
- Definování vlastních výjimek
    - V DECLARE dám **<název výjimky> EXCEPTION**
    - Příklad

```
DECLARE
    PRILIS_MNOHO_TRPASLIKU EXCEPTION;
   v_pocet_trpasliku NUMBER;

BEGIN
    v_pocet_trpasliku:=7;
    IF v_pocet_trpasliku > 7 THEN
                RAISE PRILIS_MNOHO_TRPASLIKU;
   END IF;

EXCEPTION

     WHEN PRILIS_MNOHO_TRPASLIKU
      THEN DBMS_OUTPUT.PUT_LINE('Trpaslíků může být maximálně sedm');

END;
```

- Propagace výjimek
    - Nejprve v daném bloku, když nenajde, tak v bloku vně (pokud je vnořený)



- Nezachycené výjimky
  - Jsou „vypropagovány“ až do hostitelského prostředí databáze.
  - To pak rozhodne, jak naloží S výsledkem běhu PL/SQL bloku.
  - V případě nezachycené vlastní výjimky skončí běh s chybovým hlášením:
    - ORA-06510: PL/SQL: unhandled user-defined exception.

3. Kurzory

- **Otázka: Kurzory - definice, klasifikace, použití kurzorů. Uložené procedury a funkce, balíky, kompilace, spouštění. (Standard *SQL/PSM).***
- Kurzory
  - Privátní pracovní oblasti, jsou databázovým serverem vytvořeny pro každý příkaz SQL
  - Ukazatele do paměti, ne přímo do dat – odkud to načítám, zavírám,…
  - Výsledné záznamy dotazu do paměti ukládány ve chvíli otevření kurzoru
  - Nástroj rozšíření sady výsledků
  - v PLSQL jinak vždy jedna hodnota, jeden řádek, ale co když jich je víc a ty chci procházet?
  - Implicitní
    - Není nutné otevírat, zavírat, deklarovat, načítat data, vytvářeny automaticky databázovým serverem
    - Obecně efektivnější, výkonnější než explicitní
    - Úspornější z hlediska kódu
  - Explicitní
    - Deklarované programátorem
    - Výhodou je flexibilita – mohu použít, jak chci
    - Základní kroky pro práci
      - CURSOR <název kurzoru> IS <příkaz SELECT>; deklarace kurzoru
      - OPEN <název kurzoru>; Otevření kurzoru
      - FETCH <název kurzoru> INTO <seznam proměnných>; Výběr dat prostřednictvím kurzoru
      - CLOSE <název kurzoru>; Uzavření kurzoru
    - Musím testovat stav – existují následující atributy
      - %ROWCOUNT Pořadové číslo aktuálního záznamu, nebyl-li vybrán, hodnota=0
      - %FOUND
        - Pokud příkaz FETCH načetl nějaký záznam, atribut má hodnotu TRUE
        - Používám pro zjišťování konce cyklu
      - %NOTFOUND Používám pro zjišťování konce cyklu
      - %ISOPEN Je-li kurzor ještě otevřen, má hodnotu TRUE
    - Práce s explicitními kurzory
      - DECLARE
        - v_jmeno ucitel.jmeno%TYPE;
        - v_Id ucitel.Id%TYPE;
      - CURSOR k1 IS
        - SELECT jmeno, Id FROM ucitel;
      - BEGIN

            - OPEN k1;
            - LOOP
                - FETCH k1 INTO v_jmeno, v_Id;
                - EXIT WHEN k1%NOTFOUND;
                - DBMS_OUTPUT.PUT_LINE('Jméno ' || v_jmeno || ', Id ' || v_Id);
            - END LOOP;
            - CLOSE k1;
        - END;
    - Práce s kurzory a záznamy
        - S využitím záznamů můžeme s kurzory pracovat mnohem efektivněji
        - Cyklus FOR s explicitním kurzorem – není nutno uzavírat/otevírat
        - Příklad
            - DECLARE
                - rec_ucitel ucitel%ROWTYPE;
                - CURSOR k1 IS
                - SELECT jmeno, Id FROM ucitel;
            - BEGIN
                - FOR rec_ucitel IN k1 LOOP
                    - DBMS_OUTPUT.PUT_LINE('Jméno ' || rec_ucitel .jmeno || ', Id ' || rec_ucitel.Id);
                - END LOOP;
            - END;
- Práce s implicitními kurzory
    - Příkaz SELECT … INTO … FROM …
        - Vrátit přesně jeden řádek, jinak chyba
        - počet sloupců musí odpovídat počtu proměnných uvedených za klauzulí INTO včetně použitelnosti datových typů.

- Kurzory s parametry
    - Parametry budou dosazeny do dotazu až během otevření kurzoru
    - CURSOR <název kurzoru> [(<název parametru> <datový typ>, …)] IS <příkaz SELECT>;
    - Příklad
        - DECLARE
            - rec_ucitel ucitel%ROWTYPE;
            - CURSOR k1 (v_jmeno VARCHAR2) IS
            - SELECT jmeno, Id FROM ucitel WHERE jmeno LIKE (v_jmeno || '%');
            -
        - BEGIN
            - FOR rec_ucitel IN k1 (‘Za’)
            - LOOP
                - DBMS_OUTPUT.PUT_LINE('Jméno ' || rec_ucitel .jmeno || ', Id ' || rec_ucitel.Id);
            - END LOOP;
        - Hledám učitele, jehož jméno začíná nebo obsahuje „Za“
    - Explicitní vs. Implicitní kurzory
        - **Implicitní kurzory** jsou obecně výkonnější.
            - Implicitní kurzory jsou z pohledu kódu úspornější a automatizují operace **OPEN, CLOSE, FETCH** a **EXIT WHEN .. %NOTFOUND** explicitního kurzoru.
            - Nehrozí opomenutí EXIT WHEN, což má za následek uváznutí v nekonečné smyčce.
        - **Explicitní kurzory** lze předávat parametrem mezi jednotlivými procedurami/funkcemi.
            - Využívá se při tom datový typ SYS_REFCURSOR.
            - Větší flexibilita než implicitní kurzory.

- Záznamy
    - Tato struktura zapouzdřuje více položek i rozdílných datových typů
    - DECLARE

        - TYPE <název proměnné typu záznam> IS RECORD
            - (<název atributu> <datový typ>);
- Procedury a funkce
    - Bloky příkazů jazyka PL/SQL lze pojmenovat a uložit ve spustitelné formě do databáze. Těmto blokům říkáme procedury, resp. funkce.
    - Vlastnosti procedur a funkcí:
        - Jsou uloženy ve zkompilovaném tvaru v databázi.
        - Mohou volat další procedury či funkce, či samy sebe.
        - Lze je volat ze všech prostředí klienta.
    - Funkce vrací jedinou hodnotu (procedura může vracet hodnot více, resp. žádnou).
    - Zrušení vždy pomocí DROP PROCEDURE/FUNCTION jméno;
    - Procedury
        - Procedura je posloupnost příkazů, které se provedou v okamžiku spuštění procedury.
        - Na základě vstupních parametrů jsou vráceny výsledky v podobě výstupních parametrů.
        - **CREATE [OR REPLACE] PROCEDURE <název> (<seznam parametrů) AS BEGIN … END**
            - Není tam DECLARE
        - Může mít navíc COMMIT
        - Formální parametry procedury
            - Jméno_parametru [IN OUT IN OUT] typ_parametru [:= hodnota]
            - Mohou být vstupní, výstupní a vstupně-výstupní
            - Buď klauzule DEFAULT, nebo přiřadit pomocí :=
        - proceduru vykonáme EXEC Procedura(1,4);
    - Funkce
        - **CREATE FUNCTION jméno_funkce(formální_parametry) RETURN typ_návratové_proměnné AS**
        - Spuštění
            - **SELECT vyber_dat_funkce('druhy') FROM dual;**

4. Aktivní databáze

- **Otázka: Dynamické a statické SQL, aktivní databáze. – charakteristika, význam, příklady využití, porovnání s jinými možnostmi ovládání integrity a konzistence databáze.**
- Aktivní databáze rozšíření databázových systémů o aktivní pravidla. Aktivní pravidla = triggery
- uživatelsky definovaný blok PL/SQL sdružený s určitou tabulkou. Je implicitně spuštěn (proveden), jestliže je nad tabulkou prováděn aktualizační příkaz
- trigger ~ spoušť, kohoutek (rychlé a nebezpečné)
- aktivní pravidla (active rules, triggers)
    - pro vyhodnocení složitých podmínek kladených na data
    - kontrola na databázové úrovni
    - usnadnění práce
- Historický vývoj
    - Konec 80. let
        - první snahy o formální definici
    - SQL92
        - triggery neobsahuje
        - nedostatky ve standardizačních dokumentech
    - SQL1999
        - triggery již obsahuje
    - Starbust
        - IBM, Almaden Research Center
        - Starburst Active Rule Systém
        - Získalo popularitu
        - Jednoduchá syntaxe a sémantika

        - Množinově orientovaná
        - Pravidla založena na ECA-paradigmatu (Event-Condition-Action)
- spustí se při výskytu nějaké sledované události
- ECA-paradigma
    - Událost (Event)
        - SQL-příkazy pro manipulaci s daty (INSERT, DELETE, UPDATE)
        - Není k SELECTU
    - Podmínka (condition)
        - Booleovský predikát nad stavem databáze, vyjádřen pomocí SQL
    - Akce (Action)
        - Provádí libovolné SQL dotazy (Například SELECT, INSERT, DELETE, UPDATE)
        - Navíc mohou obsahovat příkazy pro manipulaci s aktivními pravidly a transakční instrukci ROLLBACK WORK
- Říkáme, že pravidlo je:
    - spuštěno (triggered) – pokud nastane příslušná Událost
    - vyhodnoceno (considered) – po vyhodnocení dané Podmínky
    - vykonáno (executed) – po provedení jeho Akce
- Základní vlastnosti triggerů
    - Přidané do schématu databáze a jsou sdílené všemi aplikacemi
        - Rozšíření integritních omezení
    - Mohou být dynamicky aktivovány a deaktivovány každou transakcí
    - Každé pravidlo má ve Starbustu jedinečné jméno a je spojeno s jednou určitou tabulkou, zvanou rule's target
    - Každé aktivní pravidlo může sledovat více událostí, tzv. rule's triggering operations
    - Jeden SQL příkaz může být sledován více pravidly
        - Pořadí pravidel je určeno na základně jejich částečného uspořádání
- Problémy s triggery
    - proprietární řešení výrobců DB systémů
        - rozdíly v syntaxi i sémantice
        - vazba aplikace na konkrétního výrobce
    - technické problémy
        - nekonečné vzájemné volání triggerů (retriggering)
        - několik možných řešení
        - používají se všechna
- Triggery
    - Jedná se o PL/SQL objekty spouštěné vyvoláním příslušné události v DB
    - Vyvolání může způsobit DML(INSERT, DELETE, UPDATE) událost, DDL operace nebo speciální DB událost
    - Vyvolat trigger můžeme před nebo po provedení operace
    - Existuje také možnost vyvolání místo příslušné operace
    - Je možné omezit vyvolání podmínkou
- Typy triggerů (spouštění)
    - Triggery pro DML příkazy nad tabulkami
        - DML aktivované triggery:
            - při rušení řádků (DELETE)
            - při vkládání řádků (INSERT)
            - při modifikaci určitých sloupců (UPDATE OF)
            - Způsob vyvolání triggeru:
                - jednou při celé operaci
                - pro každý řádek (FOR EACH ROW) vstupující do zpracování operace
                - Možnost kombinací operací (slučování OR)
    - Syntaxe
        - **CREATE [OR REPLACE] TRIGGER <název triggeru>**
        - **{ BEFORE | AFTER } [INSTEAD OF]**

        - **{ INSERT | UPDATE | DELETE } ON <název tabulky>**
        - **[FOR EACH ROW [WHEN <podmínka provedení triggeru>]]**
        - **BEGIN…**
        - **END**;
        - Způsob odkazování
            - Definuje, jak budou přístupné původní a nové záznamy (vstupující do DML operací)
            - Implicitně :new, :old, :parent
            - Existuje klauzule
                - **REFERENCING OLD/NEW/PARENT AS jméno**
        - Způsob vyvolávání triggerů
            - Pro BEFORE a AFTER je trigger chápán jako tzv. statement trigger a vyvolán je pouze jedenkrát (není-li klauzulí FOR EACH ROW explicitně stanoveno jinak)
            - V případě INSTEAD OF triggeru je trigger implicitně chápán jako řádkový trigger, protože zde statement trigger nemá prakticky žádný význam

    - DDL triggery
        - Jsou vyvolány po provedení DDL příkazu
        - Mohou být BEFORE, AFTER, mohou být omezeny podmínkou (WHEN)
        - Definují se dvěma způsoby:
            - jméno události ON DATABASE
            - jméno události ON jméno schématu
            - Existuje řada definovaných událostí, např. CREATE, ALTER, DROP, RENAME, GRANT, COMMENT, AUDIT, DDL
    - Triggery pro databázové a klientské události
        - jiná množina povolených událostí, specifické, jinak podobné
        - Typicky se jedná o události zásadních událostí v celé databázové instanci, např. STARTUP, SHUTDOWN, LOGON, LOGOFF, SERVERERROR, SUSPEND apod.
- Omezení triggerů
    - BEFORE a AFTER triggery nelze specifikovat nad pohledy
    - V BEFORE triggerech není možné zapisovat do :old záznamů
    - V AFTER triggerech nelze zapisovat ani do :old, ani do :new záznamů
    - INSTEAD OF triggery pracují **jen s pohledy**, mohou číst :old i :new, ale nemohou zapisovat ani do jednoho
    - Nelze kombinovat INSTEAD OF a UPDATE
- Nachází se v systémovém katalogu
    - Na úrovni tabulek, sekvencí, pohledů…
- Použití
    - Nejčastější použití = kontrola zadaných dat + zajištění referenční integrity
        - Používat, zda není možné použít následující integritní omezení:
            - NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DELETE CASCADE, DELETE SET NULL
        - Nebo když tabulky nejsou v jedné databázi
    - Zamezení invalidním transakcím
    - Automatické generování odvozené hodnoty hodnot sloupců
    - Zajištění komplexní bezpečnostní autorizace
    - Implementace business pravidel – např. míra slevy podle platební historie zákazníka
    - Zajištění logování událostí
    - Zajištění synchronní replikace tabulek
    - Vyvolání chyby (Př. „nemůžete přidat 8. trpaslíka“)
    - Musím určit:
        - Akci, která trigger aktivuje
        - Kdy se trigger aktivuje (před/po)
            - AFTER a BEFORE nelze použít pro pohledy (old a new)

        - Zda se trigger spouští pro každý zpracovávaný řádek FOR EACH ROW nebo jen jednou = typy aktivačních událostí
            - **Aktivační událost na úrovni řádků -** Pro každý řádek, který je ovlivněn příkazem DML se trigger spustí jednou.
            - **Aktivační událost na úrovni příkazu -** V rámci jednoho DML příkazu se spustí trigger jen jednou, bez ohledu na to, kolik řádků příkaz ovlivní.
        - Více spouštěcích příkazů lze oddělit klauzulí OR (INSERT OR DELETE)
- Pozor při použití
    - Přílišné použití může způsobit nepřehlednost aplikace
    - Nepoužívat rekursivní triggery
    - Nepoužívejte triggery tam, kde si můžete pomoci například omezeními (CHECK na vkládané hodnoty)
    - Nedělat kód triggerů delší než 32kB – použít procedury/funkce
    - Není možné použít příkazy COMMIT, ROLLBACK nebo SAVEPOINT v těle triggeru
        - Výjimka jsou systémové triggery – jazyk DDL příkazy CREATE, ALTER, DROP TABLE, ALTER… COMPILE
- Možnost pozastavení triggeru
    - **ALTER TRIGGER <jméno triggeru> DISABLE/ENABLE;**
    - Když se jedná o všechny triggery svázané s konkrétní tabulkou
        - **ALTER TABLE <jméno tabulky> DISABLE/ENABLE ALL TRIGGERS;**
- Triggery – auditování
    - Bezpečnostní audit – příkaz AUDIT
    - Finanční audit – přes triggery
- Udržování integrity
    - integritní omezení (constraints) jsou zadána predikáty označované jako integritní pravidla
    - hlídají platnost podmínky vyjádřené predikátem
    - statické omezení je predikát nad stavem databáze
    - dynamické omezení je predikát nad přechodem porovnávající stavy způsobené transakcí
    - vestavěné omezení je speciální konstrukce jazyka
        - např. v SQL92 klíče, unikátní atributy, NOT NULL, referenční integrita
    - obecné omezení je specifikováno libovolným predikátem/dotazem
        - není podporováno ve všech DB

5. Objektové vlastnosti jazyka SQL99

- **Otázka: Objektové vlastnosti jazyka SQL99 rozšíření datových typů. Vlastnosti objektově orientovaného datového modelu, možnosti použití. (Standard SQL/OLB).**
- Objektově orientované + objektově relační databáze
    - Motivace
        - Požadavky nových aplikací:
            - nové typy objektů a funkcí
            - OO analýza a návrh vs. relační db
        - Cíl: integrace a správa dat v jednom systému

"Relační databáze je podobná garáži, která vás nutí rozmontovat vaše auto a uložit díly do malých zásuvek..."

Objektově orientované databáze

- objektový datový model:
    - je v souladu s viděním světa (entita => objekt)
    - definice složitých objektů a jejich manipulace
- RSŘBD (relační)
    - Výkonné OLTP
    - Dostupnost dat
    - Utajení
    - Prostředky pro správu dat
    - Standardní jazykové rozhraní
    - Řízení paměti
    - Souběžné zpracování dat
    - Integrita

- OOSŘBD
  - Operace na složitých objektech
  - Rekurzivní struktury
  - Abstraktní datové typy
  - Rozhraní k OO jazyku
  - Složité transakce
- 1993: konsorcium vůdčích výrobců OOSŘBD => návrh standardu ODMG-93.
- nadmnožina obecnějšího modelu Common Object Model (COM) vytvořeného skupinou OMG. Převzat byl jeho definiční jazyk IDL.
- dotazovací část Object Query Language (OQL), která souvisí s koncepcí dotazovací části standardu SQL92.
- rozhraní k OO PJ C++, Smalltalk (k Java: nahrazeno Java Data Objects (JDO))
- 2001: skupina rozpuštěna (verze ODMG 3.0)
- *OOPJ + SŘBD = OOSŘBD*
- Základní koncepty ODMG-93
  - třída (nebo typ), instance (nebo objekt), atribut, metoda a integritní omezení
    - třída - šablona pro instance (objekty), které mohou sdílet atributy a metody.
      - doména atributů: primitivní typ dat, abstraktní typ dat (ADT), nebo odkaz na třídu.
      - metoda je funkce aplikovatelná na instance třídy
  - identifikátor objektu (OID)
    - každý objekt má jednoznačný identifikátor, prostřednictvím kterého lze z databáze získat odpovídající objekt.
  - Zapouzdření
    - data jsou "zabalena" spolu s metodami. Jednotkou zapouzdření je objekt. Metody jsou platné pouze na příslušných objektech, se kterými jsou zapouzdřeny.
  - hierarchie tříd, dědění
    - podtřída => hierarchie
    - dědění je proces znamenající pro podtřídu osvojení všech atributů a metod z nadtřídy.
    - vícenásobné dědění (=> problémy např. řešení konfliktů stejných jmen zděděných atributů a metod).
- Nástup OO DB technologie
  - Zdroje: OO programování, OO analýza a návrh, relační SŘBD
    - Objektově relační mapování (ORM)
      - Př.: Hibernate (rozhraní: Session, Transaction, Query)
      - TopLink (ORM aplikace vlastněná Oracle, Inc.)
      - Přístup k objektům: např. Hibernate Query Language => SQL;
      - programovací jazyk + metody realizující vstup do SQL databáze
    - problém: méně sémantiky v relacích
  - 2. pol. 80. let - OO databáze
    - Př.: O2 (→ Unidata → Informix → IBM)
    - ObjectStore (dnes Execlon)
    - Versant Object Database, Objectivity/DB, GemStone,
    - GemSTone/J, DB4Objects, Jasmine,
    - Dotazování: OQL – neúplný, jinak: např. Objectivity/SQL++
  - Dnes: Cache, ObjectStore, Versant Object Database
  - částečný neúspěch: nenabídly pružnost a výkonnost relačních SŘBD
- Shrnutí relační model
  - Relační model - jednoduchý a elegantní, naprosto rozdílný od objektového modelu.
  - Relační db - nenavrhovány pro ukládání objektů, naprogramování složité
  - Relační databázové systémy - řízení velkého množství dat, vyhledávání dat, nízká podpor pro manipulaci
  - Dvourozměrné tabulky + vztahy mezi daty porovnáváním hodnot
  - Jazyky jako SQL umožňují tabulky propojit za běhu – aktuální vyjádření vztahu mezi daty.
- Shrnutí - objektově orientovaný model
  - Objektově orientovaný model je založen na objektech – strukturách kombinujících daný kód a data.
  - Objektové databázové systémy – umožnění využití hostitelského objektového jazyka (C++, Java)
    - Přímo na objekty v databázi – vnořené SQL
  - ODBMS jsou výborné pro manipulaci s daty.

- Objektově relační databáze
  - "Rozšířená relační" (="objektově-relační") - snaha sjednotit rysy jak relačních, tak objektových databází.
  - ORDBMS je specifikována v rozšíření SQL standardu — SQL3.
  - Příklady: Informix, IBM, Oracle, Unisys, ……
  - Datový model
    - ORDBMS využívají datový model tak, že "přidávají objektovost do tabulek".
    - informace v tabulkách + některé bohatší datovou strukturu = abstraktní datové typy (ADT).
    - ADT - k indexování, ukládání, získávání záznamů na základě obsahu nového datového typu.
    - ORDBMS jsou nadmnožinou RDBMS, bez objektového rozšíření jsou ekvivalentní SQL2.
    - ORDBMS - omezená podpora dědičnosti, polymorfismu, referencí a integrace s programovacím jazykem.
  - Nástup OR db technologie
    - Důvody:
      - obdržet maximum z rozsáhlých investic do relační technologie (data, nabyté zkušenosti),
      - využít výhody v pružnosti, produktivitě a provozních přínosů OO modelování,
      - integrovat databázové služby do systémů výroby a dalších aplikací.
    - 90. léta - OR přístup – ORSŘBD
    - kombinace OO a relačních SŘBD
    - Př.: 1992: UniSQL/X, dále: HP - OpenODB (později Odapter) 1993: Montage Systems (později Illustra) - komerční verze Postgres
    - dnes: DB/2, INFORMIX, ORACLE, Sybase Adaptive Server+Java, OSMOS, Unidata
    - počátky: univerzální servery
      - rozšiřitelnost relačního přístupu
      - Příklady:
        - Informix (Universal Server),
        - ORACLE 7.3
        - DB2 Common Server (1995!), pozdější DB2 Universal Database
      - Použití: pro již existující data v relační DB
  - Dva přístupy:
    - univerzální paměť, kdy všechny druhy dat jsou řízeny SŘBD), jde o integraci (různými způsoby!)
      - univerzální servery
    - univerzální přístup, kdy všechna data jsou ve svých původních (autonomních) zdrojích
      - Technika: middleware
        - brány (min. dva nezávislé servery)
        - zobrazení schémat, transformace dotazů
        - objektové obálky: Ontologic, HP, Next, ... (problémy: výkon)
        - DB založené na Web
  - Rozšiřitelnost, uživatelsky definované typy a funkce
    - Požadavek: manipulace BLOB = Binary large object (v RSŘBD atomický)
    - Rozšiřitelnost: možnost přidávání nových datových typů + programů (funkce) „zabalených“ do speciálního modulu
      - UDT (uživatelsky definované typy)
      - UDF (uživatelsky definované funkce)
    - Př.: DB/2 v r. 2006:
      - MapInfo
      - NetOwl (přirozený jazyk v bussiness intelligence)
      - EcoWin (časové řady, makroekonomická časové řady, …)
      - GIS a prostorové objekty
      - SQL expander (matematické, finanční, konverzní, … funkce)
      - VideoCharger (audio a video objekty v reálném čase)
      - text, XML, audio, video, obrázky
      - FormidaFire (integrace heterogennich dat)
    - Př.: Informix v r. 2006:
      - C-ISAM, Excallibur Text Search, Geodetic, Image Foundations, Spatial, TimeSeries, Video Foundation, Web
    - Standardizace: SQL/MM

        - (Př.: Full-Text - řeší ADT + odpovídající funkce)
        - Implementace: technologie „plug in“ pomocí různých technik:
    - Př.: DataBlades - přímý přístup k databázovému jádru ORACLE 7.3 - architektura více serverů a API
- „Opravdové“ ORSŘBD
    - Won Kim: „rozšiřitelnost je pouze druhotný, i když užitečný rys, jde o důsledek OO přístupu“
    - Stonebraker: „rozšiřitelnost typu “plug-in” (např. ORACLE) jako sice vhodnou pro konektivitu aplikaceaplikace, nicméně nemá nic do činění s databázovou “plug-in”. Jde o pouhý middleware, který nezakládá OR technologii“.
    - Požadavky:
        - datový model s hlavními rysy ODMG-93
        - odpovídající objektový jazyk vyšší úrovně
    - Řešení: OO rozšíření SQL naplňuje (přibl.) tyto požadavky
    - Dnes: standard SQL:1999, SQL:2003 + další vývoj
- Objektově relační modelování
    - Rozšíření relačního modelu o objekty a konstrukty pro manipulaci nových datových typů,
    - atributy n-tic jsou složité typy, včetně hnízděných relací,
    - zachovány jsou relační základy včetně deklarativního přístupu k datům,
    - kompatibilita s existujícími relačními jazyky (tvoří podmnožinu).

SQL:1999

- Pět části:
    - SQL/Framework
    - SQL/Foundations
    - SQL/CLI (Call Level Interface - alternativa k volání SQL z aplikačních programů (implementace: ODBC))
    - SQL/PSM (Persistent Store Modules - procedurální jazyk pro psaní transakcí)
    - SQL/Bindings 250 str. (SQL Embedded, Dynamic SQL, Direct invocation)
- Vlastnosti
    - podpora objektů
    - uložené procedury
    - triggery
    - rekurzivní dotazy
    - rozšíření pro OLAP
    - procedurální konstrukty
    - výrazy za ORDER BY
    - savepoints
    - update prostřednictvím sjednocení a spojení
- Objekty: od SQL3 k SQL:1999
    - SQL3 pro podporu objektů používá:
        - uživatelem definované typy (ADT, pojmenované typy řádků a odlišující typy),
        - konstruktory typů pro typy řádků a typy odkazů,
        - konstruktory typů pro typy kolekcí (množiny, seznamy a multimnožiny),
        - uživatelem definované funkce (UDF) a procedury (UDP),
        - velké objekty (Large Objects neboli LOB).
    - Standard SQL:1999 - podmnožina celkové koncepce
- UDT – USER-DEFINED-TYPES
    - **Objektově relační DB systémy** nejsou plně objektové (nemodelují objekty a vztahy mezi nimi přímo).
    - V nejnižší vrstvě jsou relační a objekty nad nimi jsou simulovány pomocí SŘBD.
    - Čistě objektové systémy zatím nebyly standardizovány. Neměly standardní jazyk.
    - Přínosy
        - Standardizace (SQL 1999) a podpora výrobců relačních systémů
        - Uživatelské datové typy
        - Data reprezentována objekty (které mají OO vlastnosti)
        - Jednoznačná identifikace objektu — OID
        - OID umožňuje tvorbu trvalých vztahů a zjednodušuje indexaci
        - Vnořené struktury (např. adresa) a tabulky
        - Kolekce (řetězec, posloupnost, seznam, soubor) a kurzor

    - SQL 1999 zavedlo podporu pro objektově relační databáze
        - Objektové rysy
            - Uživatelské datové typy (UDT) — definice struktur a jednoduchá dědičnost, vzácné typy (vycházejí z vestavěných) – objekt, kolekce
            - Typované tabulky — tabulka je typu UDT (tedy je to třída a řádky jsou objekty)
        - Nové relační rysy
            - Nové datové typy — LOB (BLOB, CLOB), BOOLEAN, ARRAY, ROW (složený sloupec)
            - Regulární výrazy — WHERE x SIMILAR TO regexp (řetězec popisující množinu řetězců)
            - Databázové triggery — podpora aktivních prvků databáze
            - OID — objekty lze reprezentovat jako řádky tabulek (tříd) a mít reference typu REF (identifikuje řádek v typované tabulce)
            - Přístup k hodnotám struktur přes operátor
            - dědičnost, polymorfismus
- ADT – abstraktní datový typ
    - výraz pro typy dat, které jsou nezávislé na vlastní implementaci.
    - implementačně nezávislá specifikace struktury dat s operacemi povolenými na této struktuře.
    - Základní ADT jsou například:
        - asociativní pole
        - zásobník
        - seznam
        - fronta
        - množina
        - zobrazení
        - textový řetězec
        - strom
        - halda - prioritní fronta
        - hašovací tabulka
- Další typy v SQL1999
    - Konstruované atomické typy:
        - Reference
    - Konstruované kompozitní typy:
        - array /* podtyp collection */
            - uspořádaný seznam dané maximální délky
            - nejsou povoleny žádná pole polí nebo vícedimensionální
    - pole
        - row
    - UDT
        - odlišující typy (jsou zatím budované pouze na předdefinovaných typech)
        - strukturované typy (mohou být definované s více atributy, které jsou předefinovaných typů, typu ARRAY, nebo dalšího strukturovaného typu)
        - chování je realizováno pomocí funkcí, procedur a metod
        - mohou být organizovány do hierarchií s děděním
        - jde o abstraktní datový typ (ADT)
- Uživatelsky definované procedury a funkce
    - programy vyvolatelné v SQL: procedury a funkce
        - procedury mají parametry typu IN, OUT, INOUT
        - funkce mají parametry jen typu IN, vracejí hodnotu
    - konstrukce programů:
        - hlava i tělo v SQL (buď 1 SQL příkaz nebo BEGIN…END)
        - hlava v SQL, tělo externě definované
- uživatelsky definované metody
    - SQL:1999 přidává metody
    - Rozdíly metod a funkcí:
        - metody jsou vždy svázány s typem, funkce nikoliv,
        - daný datový typ je vždy typem prvního (nedeklarovaného) argumentu metody,

        - metody jsou uloženy vždy ve stejném schématu, ve kterém je uložen typ, ke kterému mají nejblíže. Funkce nejsou omezeny na specifické schéma.
        - funkce i metody mohou být polymorfické, liší se v mechanismu volby konkrétní metody v run time,
        - signatura a tělo metody jsou specifikovány odděleně,
- Podtypy
    - CREATE TYPE úředník_t UNDER zaměstnanec_t …
    - strukturované typy mohou být podtypem dalšího UDT
    - UDT dědí strukturu (atributy) a chování (metody) ze svých nadtypů
    - povolena je jednoduchá dědičnost (vícenásobná odložena, nevyskytuje se ani v SQL:2003)
- Podtabulky
    - aparát závislý na aparátu typů
        - **CREATE TABLE osoby OF osoby_t**
        - **CREATE TABLE zaměstnanci OF zaměstnanec_t**
        - **UNDER osoby;**
            - Zaměstnanec_t musí být podtypem osoby_t
    - dědí sloupce, IO, triggery, … dané nadtabulky
    - Požadavky konsistence pro podtabulky a nadtabulky
        - každá n-tice v nadtabulce (např. osoby) může korespondovat nejvýše k jedné n-tici v podtabulkách (např. zaměstnanci a OON)
        - tj. každá entita musí být nespecifičtější typ
    - Výběr omezený na tabulku X pomocí
        - FROM ONLY (X)
    - – jinak i z podtabulek X.
- Problémy s OO v SQL
    - tabulky jsou jediné pojmenované entity
    - typ REF aplikovatelný pouze na objekty dané řádkem – reference na něco
    - UDT je prvním krokem k OO
        - umožnit perzistenci, musí být objekt v tabulce
        - individuální instanci nelze přiřadit jméno
        - nelze použít dotazy na všechny instance ADT
- Objektové vlastnosti SQL99
    - Kompabilita s existujícím jazyky
    - OID - identifikátor objektů
    - Hnízděné tabulky - odpovídá virtuální databázové tabulce, práce s ní je sekvenční (tabulka jako sloupec v tabulce)
    - rozšiřitelnost - nové datové typy UDT
        - abstraktní datové typy
        - řádkové typy
        - reference - ukazatel např. v tabulce účet je ref(klient) . dám tam ukazatel
    - možnost uživatelem definované funkce či metody
- Rozšíření datových typů
    - standardizace typů pro objemná data
    - typ BLOB (Binary Large Objecs)
    - typ CLOB (Character Large Objects)
        - možnosti intepretace jejich vnitřního obsahu – texty, obrázky, zvuky, videa
        - možnost doplňování dalších formátu a manipulace s nimi
    - boolean - nahrazení bit

6. Zpracování full -textových dat.

- **Otázka: Zpracování full -textových dat. XML databáze – charakteristické vlastnosti, výhody a nevýhody. (Standard *SQL/XML).***

XML databáze

- Extensible markup language
    - Prezentace pomocí kaskádových stylů
- Základní možnosti uložení

- V systému souborů
  - Uložení XML v textové formě do souboru na disku
  - Při nahrávání se konstruují DOM stromy (Document Object Model)
  - Nevýhody
    - Nutnost držet celý dokument v paměti v průběhu zpracování
    - Nutnost analyzovat celý text
  - Vylepšení: index pozic značek
    - Stačí analyzovat jen tu část dokumentu, kterou se prochází při vyhodnocení
    - Není třeba držet v paměti vše
    - Faktická nemožnost aktualizace
    - Zbytečně složité
- V relační databázi
  - Dnes spíše využití relačních databází pro uložení XML
  - dotazování bývá řešeno SQL nebo překladem dotazů do SQL
  - Obvyklé je také ukládání XML jako BLOB objektů

```
<katedra id_katedry="11">
  <student id_studenta="1">
    <jméno>Novák</jméno>
    <kurz>AIL010</kurz>
    <kurz>AIL020</kurz>
  </student>
  <student id_studenta="2">
    <jméno>Dvořák</jméno>
  </student>
```

  - Uložení hran stromu
    - Id uzlu získané preorder průchodem
    - Hrana: pětice
      - Výchozí ID, cílové ID, značka, pořadí, vložený obsah
  - Nula ve sloupci cílového ID značí, že uzel je atribut
  - Doporučuje se index: (značka, data), (výchozí ID, ordinál), (ID cíle)

Výchozí ID	Značka	Pořadí	Cílové ID	Data
1	Katedra	1	2	
2	id_katedry	0	0	„11“
2	Student	1	3	
2	Student	2	4	
3	id_studenta	0	0	„1“
3	Jméno	1	0	
3	Kurz	2	0	„AIL010“
3	Kurz	3	0	„AIL020“
4	id_studenta	0	0	„2“
4	Jméno	1	0	

    - Uložení pomocí hran – ale poněkud neúsporné
    - Možnost zrušení sloupce se značkou a rozdělení do více relačních tabulek
      - Zbavíme se tím ale možnosti ukládat libovolná XML data
      - Se stovkami relačních tabulek se RDBMS hůře vypořádá
      - Tabulky se generují od kořene
      - Každé n-tici je přiřazen unikátní identifikátor a vyplněn sloupec identifikující rodičovský element
      - Elementy bez vícenásobného výskytu jsou vloženy přímo jako další sloupce do tabulky rodičovského elementu

Katedra		
ID rodiče	ID	id_katedry
1	2	„11“

Kurz		
ID rodiče	ID	DATA
3	5	„AIL010“
3	6	„AIL020“

Student			
ID rodiče	ID	id_studenta	Jméno
2	3	„1“	„Novák“
2	4	„2“	„Dvořák“

-
    - Problém: zkoumání vztah předek-potomek
        - Mnoho operací join
        - Problém s dokumenty s cyklickým DTD – hluboké hnízdění SQL dotazů, případně potřeba nějakého (např. procedurálního) rozšíření dotazovacího jazyka
    - Netriviální převod XQuery -> SQL
- V objektově-relačním databázovém systému
    - Využívá ADT, abstraktní datové typy
    - Obvykle je vyžadováno DTD (document type definition – jazyk pro popis struktury XML)
    - Podobné strukturálnímu uložení do RDBMS
    - Problém s rekurzivními strukturami, smíšenými elementy, problematické složitější modelové skupiny DTD
    -

id_katedry	Student		
	id_studenta	Jméno	kurz
	„1“	„Novák“	„AIL010“
„11“	„1“	„Novák“	„AIL020“
	„1“	„Dvořák“	

- Nativní XML úložiště
    - Zcela přizpůsobeno stromové struktuře XML dat
    - Výhody
        - Rychlost, flexibilita, snadné a efektivní XML dotazování
    - Nevýhody
        - Nutnost vybudování zcela nových datových struktur a metod pro vyhodnocení dotazů

- Vložení XML dokumentu do databáze
    - Existuje několik způsobů:
        - A. s dodatečnými oprávněními:
            - 1. vytvoříme/vyhradíme adresář, v databázi zaregistrovat příkazem CREATE DIRECTORY
            - 2. vyžaduje dodatečná oprávnění, která běžný uživatel nemá přidělena a mít nebude
            - 3. Do zmíněného adresáře umístíme XML soubory, které chceme do databáze uložit
            - 4. Jejich uložení provést příkazem INSERT
        - B. bez potřeby výše zmiňovaných oprávnění
    - **CREATE TABLE xml_soubory(**
        - **Nazev VARCHAR2(256),**
        - **Obsah XMLType**
        - **);**
- XMLQuery vs. XMLTable
    - Table – vrátí hodnoty v jediném sloupci
    - Query – více detailní

Zpracování full-textových dat

- SFQL 1991-1992
    - = Structured Full-text query language
    - Rozšíření SQL pro práci s full-textovými daty
    - Konflikt klíčových slov (CONTAINS) - Možný konflikt klíčových slov je třeba řešit

- Full-text
    - Dokumenty obsahují plný text
    - Potřebují speciální vyhledávání
    - Způsob indexování
        - Index slov v dokumentu
        - Index vzájemných vzdáleností slov a frází
    - Vyhledávací metody
        - CONTAINS – ano/ne
        - RANK
    - Konverze do běžných SQL znak. Řetězců
        - FullText_to_Character
- Vyhledávání
    - Vzorek (+wildcards - *)
    - Odvozená slova (stemmed)
    - Podobně znějící slova (sounds like)
    - Slova s podobným/stejným významem (thesaurus)
    - Dle pozice v textu (`NEAR`, …)
    - Dle konceptu textu (`IS ABOUT`)

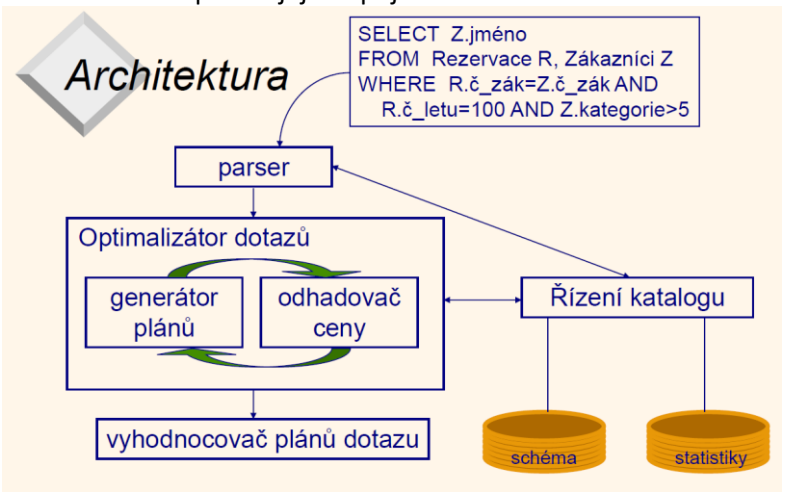
- SQL/MM počítá především s podporou jazyků, kde je snadné výpočetně rozeznat jednotlivé tokeny.

7. Optimalizace

- **Otázka: Optimalizace - důvody optimalizace, prostředky optimalizace, význam indexů a způsobu zápisu dotazu.**
- Motivace
    - dvěma či různými dotazy je možné získat stejná data
    - rychlost dotazů může být ovšem rozdílná
    - cíl – nalézt nejefektivnější plány, které vedou ke stejnému výsledku
        - selekce by se měly vyhodnocovat dřív
        - zavedení indexů
    - přínos optimalizace – minimalizace nákladů
        - na zdrojový čas
        - kapacitu paměti
        - programátorskou práci
- **podílí se -** návrhář DB, vývojář, správce DB, uživatel

- **postup zpracování SQL dotazu**
    - uživatel zadá SQL příkaz – SELECT * FROM LIDE
    - příkaz se zpracuje v parseru
    - potom se použije nastavený optimalizátor
    - generátor řádkových zdrojů
    - vlastní provádění SQL
- plán vyhodnocení – strom dotazu + algoritmus pro každou operaci
- myšlenky- jaké plány se uvažují pro dotaz a jak se odhadne cena plánu
- podstata – jednotlivé akce si optimalizátor ohodnotí nějakou cenou a pak pro daný plán určí celkovou cenu
- **Pravidla pro vytváření dotazů**
    - Používat select jenom na sloupce, které nás zajímají
        - většinou pracujeme stejně jenom s některými při více podmínkách začínat tou nejobecnější
        - není vhodné využívat SELECT * FROM…
    - Výběr vhodného pořadí spojení
        - 1) vyhnout se plnému prohledávání tabulky (pokud možno využít index)
        - 2) efektivně vybírat takové indexy, které načtou z tabulky co nejméně záznamů
        - 3) vybrat takové pořadí spojení ze všech možných pořadí, aby bylo spojeno co nejméně položek
    - Používat hinty - podnět, kterým optimalizátoru určíme, jaký má použít plán vykonávání dotazu
    - Používání indexů - **B-Tree, Btimapové indexy**
        - Při prohledávání nejede podle toho, jak řádky za sebou, ale podle indexů

- Používat LIMIT
  - Když chci nejstaršího člověka, dám seřadit podle věku, ale když dám limit 1, mám rovnou toho nejstaršího
- Používat minimálně IN, NOT IN , LIKE
  - Like – větší textová pole mohou obsahovat až několik GB dat
  - NOT a NOT IN je lepší WHERE EXISTS a NOT EXISTS
    - Blbě: **WHERE Doprava IN (‚Ford‘,‘Octavia‘,‘Seat‘,‘Peugeot‘);**
    - Správně: **WHERE typ_dopravy = ‚Automobil‘**
- Obecnější podmínky používat na začátku dotazu
  - Snažíme se, aby systém vyřadil na začátku co nejvíce řádků; ty se pak již při další podmínce nezkoumají… (věk, pak až pohlaví → vyřadí víc)

- **Optimalizátor**
  - **Jádro** celého zpracování analyzuje sémantiku dotazu
  - Hledá optimální způsob jeho provádění v ORACLE CBO a RBO
  - Generátor plánů a odhadovač ceny, který pro každý vygenerovaný plán odhadne cenu
  - Výstup
    - plán vykonávání, který určuje
      - přístupové cesty k jednotlivým tabulkám používaným dotazem
      - pořadí jejich spojování
  - 

- Přístupy optimalizace
  - 1) Rule-Based Optimizer – vyhodnocuje jednotlivé přístupové cesty pomocí předem daného systému pravidel
  - 2) Cost-Based Optimizer – používá slovník se statistikami, hledá plán s nejmenšími náklady doporučené ORACLE
    - pro jednotlivé plány se počítá cena provedení v řadě hledisek
    - množství IO operací, řádek, byte
    - cena prováděných třídících operací
    - cena za HASH operace – funkce, která přiřadí index
- statistiky – řada údajů o DB objektech (tables, indexes..)
  - přístupné i z vnějšku – uložené v tabulkách, pohledech
  - výpočtem nebo odhadem se aktualizují
  - typy
    - údaje o tabulkách – počet řádků, bloků, nevyužitých bloků, prům. délka záznamu
    - údaje o sloupcích – počet unikátních / NULL hodnot, histogram dat
    - údaje o indexech – počet listových bloků, počet úrovní, cluster faktor
- Využití indexů
  - Rychlejší přístup k datům – nalezení pouze relevantních záznamů
  - Zásadní ovlivnění výkonu u velkých tabulek
  - Nutná správa indexu při vkládání/mazání → pro malé tabulky zbytečné, nebo pro velké, ze kterých se ale jen málokdy čte (logy = rozpoznávání kdy a kým aplikace/služba využívána, zaznamenání událostí v systému)
  - Primární klíč – speciální typ indexu
    - **CREATE [UNIQUE] INDEX název ON tabulka (sloupec, ...)**

- Indexy nad více sloupci (např. když dělám matici – souřadnice řádku a sloupce)
- Obvyklá implementace pomocí B stromů O(logN)

8. Temporální databáze

- **Temporální databáze, porovnání klasických a temporálních databází, modely času, vztah událostí a času (snapshot), temporální SQL.**
- Motivace
  - Potřebujeme v databázích čas?
  - Studijní informační systém
  - Skladová evidence
    - zboží které je aktuálně na skladě, jeho množství
    - zajímavá je rovněž historie pro konkrétní druhy zboží (kolik se ho za jakou domu prodalo)
  - Účetní a bankovní systémy
  - Docházkové systémy
    - kdo kdy přišel, zda byl v danou hodinu v budově
- Klasické vs. temporální databáze
  - Klasický databázový systém
    - Zachycen stav systému v aktuálním časovém okamžiku – přidání nových, mazání starých
    - Problém: co dělat se starými daty
    - Neobsahují informaci o čase
    - Chceme-li uchovávání historie změn, nutné do databáze doplnit informaci o čase. Aktualizaci a operace s časem musí zajistit uživatel - netriviální
  - Temporální databázový systém
    - Databáze určitým způsobem podporující čas
    - Vhodný dotazovací jazyk zahrnující práci s časem
    - Výhodou jsou jednodušší dotazy v nichž se vyskytuje čas → méně chyb v aplikačním kódu
- Příklad – chceme znát zaměstnancův aktuální plat
    - Zaměstnanec(Jméno, Plat, Funkce,
      Datum_narození,
      Platí_od **DATE**, Platí_do **DATE**)
    - Jaký je Pepův aktuální plat?
    - **SELECT** Plat
      FROM Zaměstnanec
      WHERE Jméno = 'Pepa'
      AND Platí_od <= **CURRENT_DATE AND CURRENT_DATE** <= Platí_do
  - Chceme-li jen aktuální stav, nevyžaduje temporální databázi, pokud bychom dali datum narození, stále nevyžaduje (Ukázka omezené podpory práce s časem v SQL-92)
  - Temporální databáze – klasicky přidáme dva sloupce s daty od do → evidence historie změn
- Temporální projekce
  - Jako projekce v klasických databázích = z celé relace jsou vybrány hodnoty zadaných atributů.
  - Navíc ale každá n-tice obsahuje čas.
  - Realizace temporální projekce v klasických databázích představuje netriviální problém a zesložiťuje údržbu aplikace,
  - podpora času v klasických databázových systémech je nedostatečná pro jednoduché dotazování.
  - Nedostatečná podpora v klasických databázových systémech
  - Příklad temporální projekce

Jméno	Plat	Funkce	Datum narození	Platí_od	Platí_do
Pepa	60000	Vrátný	1945-04-09	1995-01-01	1995-06-01
Pepa	70000	Vrátný	1945-04-09	1995-06-01	1995-10-01
Pepa	70000	Vrchní vrátný	1945-04-09	1995-10-01	1995-02-01
Pepa	70000	Ředitel bezpečnosti	1945-04-09	1996-02-01	1997-01-01

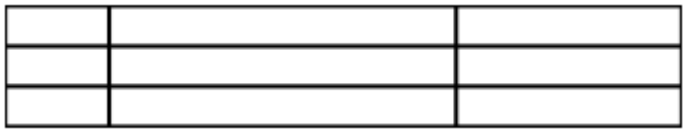
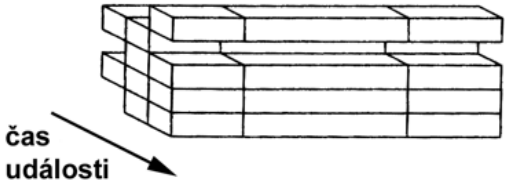
Jméno	Plat	Platí_od	Platí_do
Pepa	60000	1995-01-01	1995-06-01
Pepa	70000	1995-06-01	1997-01-01

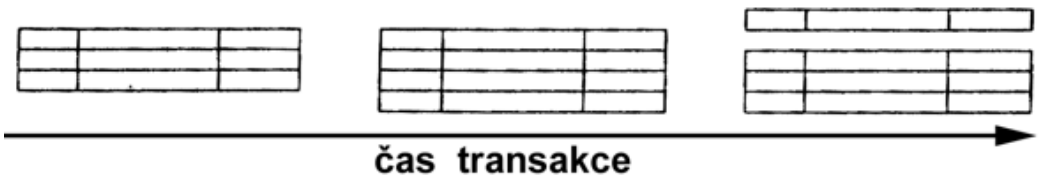
- Temporální spojení
  - Stejné jako spojení (join) v klasických databázích

- Sloupce + podmínka, kdy řádky spojeny - v temporálních db ale nemusíme zadávat sloupce uchovávající čas, pouze podmínku na spojení pro čas.
- Navíc bere v úvahu čas události

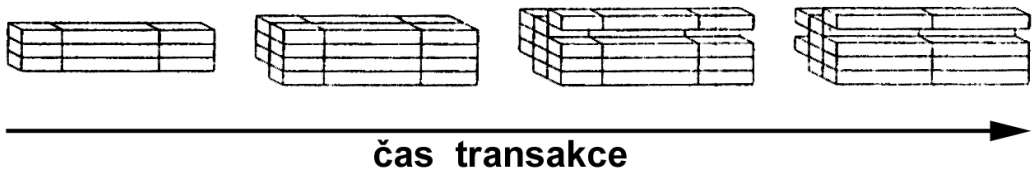
- Modely času
  - Temporální logika: čas je libovolná množina okamžiků s daným uspořádáním
  - Modely času podle uspořádání
    - **Lineární**
      - Čas roste od minulosti k budoucnosti lineárně
    - **Větvený (čas možných budoucností)**
      - Lineární minulost až do teď, pak se větví do několika časových linií reprezentujících možný sled událostí
      - Každá linie se může dále větvit
    - **Cyklický**
      - Opakující se procesy
      - Příklad: týden, každý den se opakuje po sedmi dnech
  - Modely času podle hustoty
    - **Diskrétní**
      - Spolu s lineárním uspořádáním
      - Každý okamžik má právě jednoho následníka
      - Každé přirozené číslo odpovídá nerozložitelné jednotce času (**chronon**).
      - Chronon je nejmenší časová jednotka reprezentovatelná v diskrétním modelu. Není to **okamžik**, ale **doba**.
    - **Hustý**
      - Isomorfní s racionálními nebo reálnými čísly
      - Mezi každými dvěma okamžiky existuje nějaký další
    - **Spojitý**
      - Isomorfní s reálnými čísly
      - Na rozdíl od racionálních čísel, neobsahuje „mezery“
      - Každé reálné číslo odpovídá bodu v čase (okamžiku)
  - Omezenost času
    - Omezený – nutnost zejména kvůli reprezentaci v počítači
    - Neomezený
  - Absolutní / relativní čas
    - Absolutní čas – vyjádří se hodnotou. Také ale potřebuje počátek.
    - Relativní čas – vyžaduje nějaký počátek, čas se pak vyjádří jako vzdálenost a směr od počátku.
- Datové typy pro čas
  - Časový okamžik (instant) (SQL-92)
    - DATE
      - Nejmenší jednotka je **den** v roce 1-9999
    - TIME
      - Nejmenší jednotka je **sekunda** v rámci 24 hodin
    - TIMESTAMP
      - Nejmenší jednotka je zlomek sekundy, standardně **mikrosekunda** v rámci 24 hodin
  - Časový úsek (time period)
    - Doba mezi dvěma časovými okamžiky
    - 15:30 – 15:50
  - Časový interval (interval)
    - Doba o specifikované délce, ale bez konkrétních krajních bodů
    - 30 minut
  - Množina časových okamžiků (instant set)
  - Množina časových úseků (temporal elements)
    - Konečná množina **time periods**
  - Datové typy reprezentující čas musí být reprezentovatelné v počítači.
  - Nejjednodušším způsobem omezená diskrétní reprezentace celočíselným počtem časových okamžiků od pevného počátku.

- Hustá reprezentace - poměrně jednoduchá na implementaci, protože racionální čísla jsou hustá a každé racionální číslo lze vyjádřit jako podíl dvou celočíselných.
- Spojitá reprezentace je nejsložitější.

- Vztah událostí a času
  - Čas platnosti (valid time)
    - Čas, kdy byla událost pravdivá v reálném světě
    - Může být v minulosti, přítomnosti i budoucnosti
    - Nezávislý na zaznamenávání události do databáze
  - Transakční čas, čas transakce (transaction time)
    - Čas, kdy byl fakt reprezentován v databázi
    - Nabývá **pouze** aktuální hodnoty
    - Monotónně roste
    - Identifikuje transakci, která vložila data do db a transakci, která informaci z db odstranila
  - Čas platnosti a transakční čas jsou ortogonální, v některých případech mohou být korelované
  - snapshot
    - Datový model nepodporující čas platnosti ani transakční čas
    - Snapshot relace
      - Klasický relační model
      - Každá n-tice je fakt platný v reálném světě
      - Při změně reálného světa jsou do relace prvky přidávány nebo z ní odebírány
      - 
  - valid-time
    - Datový model podporující pouze čas platnosti
    - Valid-time relace
      - Cokoliv v relaci může být upraveno
      - Hodnoty n-tic
      - Čas události (začátek i konec)
      - Umožňuje klást dotazy o faktech platných v minulosti i budoucnosti
      - Nelze zjistit **stav databáze** v konkrétním časovém okamžiku (transaction time)
      - 

  - transaction-time
    - Datový model podporující pouze transakční čas
    - Transaction-time relace
      - Posloupnost snapshot-ů indexované transakčním časem
      - Umožňuje získat informaci ze stavu databáze v nějakém okamžiku minulosti
      - Je možné uvažovat i větvení
      - 

      - Na začátku byla relace prázdná
      - První instrukce do relace vložila tři fakty (n-tice) => což znamená přidání nového snapshotu s časem **commit**u do posloupnosti
      - Další transakce vložila do relace jednu n-tici => další stav byl přidán do posloupnosti

                - Poslední transakce odstranila z relace první n-tici a jinou vložila => další stav vložen do posloupnosti
                - **Poznámky**
                    - Na rozdíl od snapshot relací, transakce nemodifikují existující data v transaction-time relacích.
                    - Změna je provedena ve snapshotu aktuálního stavu a nově vzniklý stav je přidán do relace.
                    - Transaction-time relace jsou read-only.
    - bitemporální
        - Datový model podporující čas platnosti i transakční čas
        - Bitemporální relace
            - append-only
            - Na začátku byla relace prázdná
            - První instrukce do relace vložila tři fakty (n-tice)
            - Další transakce vložila do relace jednu n-tici
            - Poslední transakce ukončila jednu událost (logicky odstranila) z relace první n-tici a jinou vložila
            - Byl změněn čas platnosti (počátek na pozdější čas) události, která byla vložena v první transakci
            - 

    - temporální
        - Datový model podporující čas platnosti nebo transakční čas
    - Obvykle založeno na relačním datovém modelu nebo objektově orientovaném datovém modelu.
    - Reprezentace času platnosti
        - Reprezentace času platnosti
            - časový okamžik
            - doba
            - časový úsek
            - množina okamžiků
        - Čas platnosti může být přidružen k
            - atributu
            - množině atributů
            - celé n-tici nebo objektu
        - Rozšíření operací relační algebry
    - Reprezentace transakčního času
        - 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 události v reálném světě
                - čas zaznamenání konce události v reálném světě
                - čas logického odstranění události z databáze
            - Množina časových úseků
        - Někdy podporován i versioning
- Příklady datových modelů
    - Segevův datový model
        - Podporuje pouze čas platnosti

        - Razítko označuje, kdy fakt začal platit
        - Razítkují se řádky relace. Razítko je jeden chronon.
        - Erik začal pracovat v Obuvi 1.6., pak se přesunul do Knih 6.6. a vrátil se zpátky do Obuvi 13.6.
        - Skončil 13.6., což vyžaduje zvláštní řádek s hodnotami Null pro všechny neklíčové atributy.

Jméno	*Oddělení*	*Čas*
Erik	Obuv	1
Erik	Knihy	6
Erik	Obuv	11
Erik	Null	13

    - Sardův datový model
        - Podporuje jen čas platnosti
        - Razítko má podobu intervalu
        - Razítkují se řádky relace. Razítko je dvojice chrononů.
        - Na rozdíl od předchozího případu(Segev) nepotřebujeme zvláštní řádek na vyjádření toho, že Erik v práci skončil.

Jméno	*Oddělení*	*Čas*
Erik	Obuv	[1-5]
Erik	Knihy	[6-10]
Erik	Obuv	[11-12]

    - Datový model HRDM
        - Podporuje jen čas platnosti
        - Razítkují se hodnoty atributů
        - Hodnoty atributů jsou funkce z časové do hodnotové domény.
        - Celá Erikova „zaměstnanecká“ historie je v jediném řádku relace.

Jméno	*Oddělení*
1 → Erik … 12 → Erik	1 → Obuv … 5 → Obuv 6 → Knihy … 10 → Knihy 11 → Obuv 12 → Obuv

    - Extensionální datový model
        - Razítkuje se každý řádek jedním chrononem času platnosti a jedním chrononem transakčního času.

Jméno	*Odd*	*VT*	*TT*
Erik	Obuv	1	1
Erik	Obuv	2	1
…	…	…	…
Erik	Obuv	1	2
Erik	Obuv	2	2
…	…	…	…
Erik	Obuv	1	8
…	…	…	…
Erik	Obuv	5	8
Erik	Knihy	6	8
…	…	…	…
Erik	Obuv	1	11
…	…	…	…
Erik	Obuv	5	11

Jméno	*Odd*	*VT*	*TT*
Erik	Knihy	6	11
…	…	…	…
Erik	Knihy	10	11
Erik	Obuv	11	11
…	…	…	…
Erik	Obuv	1	13
…	…	…	…
Erik	Obuv	5	13
Erik	Knihy	6	13
…	…	…	…
Erik	Knihy	10	13
Erik	Obuv	11	13
Erik	Obuv	12	13

        -

    - Bhargavův datový model

Jméno	Oddělení
[1, 12] x [1, ∝] Erik [13, →] x [1, 12] Erik	[1, 7] x [1, ∝] Obuv [8, 10] x [1, 5] Obuv [8, 10] x [6, ∝] Knihy [11, 12] x [1, 5] Obuv [11, 12] x [6, 10] Knihy [11, 12] x [11, ∝] Obuv [13, →] x [1, 5] Obuv [13, →] x [6, 10] Knihy [13, →] x [11, 12] Obuv

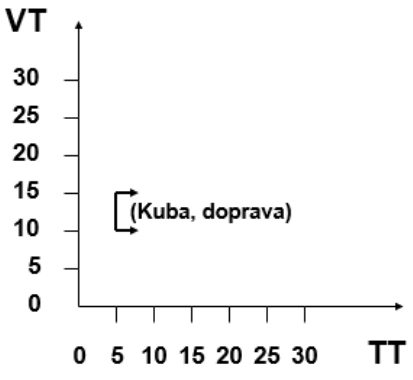
        -

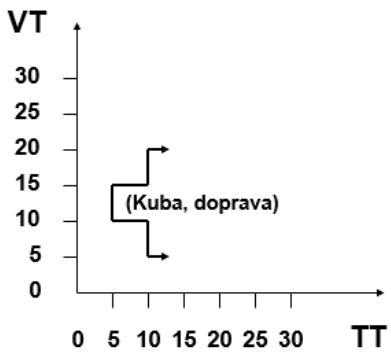
- Temporální dotazovací jazyky
    - Temporální datový model
        - Objekty se přesně danou strukturou
        - Omezení pro dané objekty
        - Operace na daných objektech
            - Temporální dotazovací jazyky

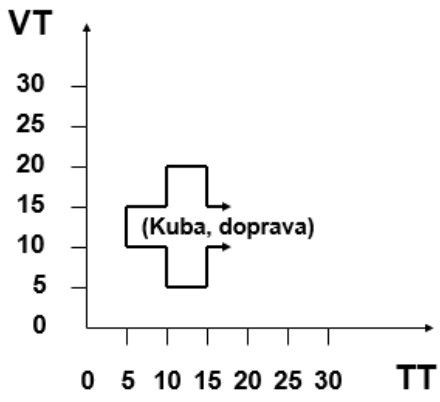
    - Velké množství
    - Nejčastěji založeny na SQL
    - Typy
        - Relační - HQL, HSQL, TDM, TQuel, TSQL, TSQL2
        - Objektově orientované - MATISSE, OSQL, OQL, TMQL
- Shrnutí
    - Ideální temporální datový model
        - Minimální rozšíření existujícího DM
        - Souvislá prezentace chování měnícího se v čase
        - Snadná implementace
        - Vysoký výkon
    - Dosažení ideálu je prakticky nemožné
    - Hlavní cíl temporálního DM by měl být zachytit sémantiku dat měnící se v čase
        - Ale často se dostává do pozadí
    - Mnoho nekompatibilních datových modelů s mnoha dotazovacími jazyky

TSQL2

- Temporal Structrured Query Language 2
- Měl sjednotit přístupy k temporálním datovým modelům
- Nadmnožina SQL-92
- Pojetí času
    - Časová osa TSQL2 je na obou koncích omezena, ale dostatečně daleko (18 miliard let)
    - U časových údajů jsou možné různé granularity
    - Časové typy
        - DATE, TIME, TIMESTAMP, INTERVAL, PERIOD
    - Po sobě jdoucí chronony mohou být seskupovány do granulí, různým seskupováním dosáhneme různých granularit.
    - Všechny uvedené časové typy kromě PERIOD byly už v SQL-92.
- Datový model
    - Je použit BCDM (Bitemporal Conceptual Data Model)
    - Řádek relace je orazítkován množinou bitemporálních chrononů = dvojice (chronon transakčního času, chronon času platnosti)
    - Každý bitemporální chronon odpovídá malému obdelníčku ve dvourozměrném prostoru (transakční čas x čas platnosti).
    - Protože nejsou povoleny duplicitní řádky, tak celá historie jednoho faktu je na jednom řádku. Říkáme, že BCDM je slévaný datový model.
- Příklad bitemporální relace
    - Relace Zaměstnanec – umístění lidí v odděleních určitého podniku
    - Schéma (Jméno, Oddělení) + časové razítko
    - Předpokládejme granularitu 1 den u času platnosti i u transakčního času
    -
    - Časový úsek našeho zájmu je daný měsíc v daném roce, tedy 15 znamená např. 15. 6. 2005.







- Zaměstnanec Kuba byl najmut jako dočasná výpomoc v oddělení dopravy na dobu od 10. do 15.
- Šipky směřující doprava značí, že řádek nebyl logicky smazán; jeho platnost pokračuje až do času *until changed (U.C.).*
- Personální oddělení zjistilo, že Kuba byl ve skutečnosti najmut na dobu od 5. do 20., opraveno
- Všechny zmíněné změny se objeví v databázi 25.

Jméno	*Oddělení*	*Časové razítko*
Kuba	Doprava	{(5, 10),…,(5,15),…(9, 10),…,(9, 15), (10,5),…,(10,20),…,(14,5),…,(14,20), (15,10),…,(15,15),…,(24,10),…,(24,15)}
Kuba	Nakládání	{(U.C., 10), …, (U.C., 15)}
Katka	Doprava	{(U.C., 25), …, (U.C., 30)}

-
- „Negrafická“ reprezentace posledního obrázku

- Definice schématu
  - Čas platnosti má granularitu 1 den
  - Transakční čas má granularitu 1 ms nebo menší
  - Čas platnosti specifikuje úsek, kdy měl pacient lék předepsaný.
  - Transakční čas určuje, kdy se tato informace nalézala v databázi.
  - Řádky, které nebyly updatovány nebo smazány, budou mít transakční čas obsahující nynější okamžik.
  - Příkaz SELECT
    - Kdo bral (bere) lék Proventil?
      - **SELECT SNAPSHOT** Jméno
      - **FROM** Předpis
      - **WHERE** Lék = ‘Proventil’
    - Komu byl předepsán nějaký lék a kdy?
      - **SELECT** Jméno
      - **FROM** Předpis
    - TSQL2 provede automaticky slévání, a tak výsledkem je množina řádků, každý asociovaný s jednou nebo více maximálních period, během kterých pacient bral aspoň jeden lék.
  - Restrukturalizace
    - Na úrovni klauzule FROM se provede projekce a slévání
    - Kdo měl (má) brát nějaký lék déle než 6 měsíců (v souhrnu)?
      - **SELECT** Jméno, Lék
      - **FROM** Předpis(Jméno, Lék) **AS** P
      - **WHERE CAST**(**VALID**(P) AS **INTERVAL MONTH**) > **INTERVAL** ‘6’ **MONTH**
  - Parcializace
    - Kdo měl předepsaný jeden lék déle než 6 měsíců nepřerušovaně?
      - **SELECT SNAPSHOT** Jméno, Lék, **VALID**(P)
      - **FROM** Předpis(Jméno, Lék)(**PERIOD**) **AS** P
      - **WHERE CAST**(**VALID**(P) **AS INTERVAL MONTH**) > **INTERVAL** ‘6’ **MONTH**;
    - Někdy potřebujeme potřebujeme zjistit maximalní souvislé období.
    - Použijeme restrukturalizaci v klauzuli FROM a navíc přidáme klíčové slovo PERIOD. To zajistí, že ke každé dvojici Jméno, Lék není množina intervalů, ale jednotlivé maximální souvislé časové intervaly.
  - Klauzule VALID
    - Jaké léky měla Marie předepsané v roce 1996?
      - **SELECT** Lék
      - **VALID INTERSECT** (**VALID**(Předpis), **PERIOD** ‘[1996]’ **DAY**)
      - **FROM** Předpis
      - **WHERE** Jméno=‘Marie’

        - Někdy potřebujeme omezit čas platnosti pouze na určité období.
        - Použijeme k tomu klauzuli VALID a operátor průniku INTERSECT
        - Výsledkem je seznam léků a ke každému léku je přiřazena množina intervalů z roku 1996 během nichž Marie užívala tento lék.
    - Rising
        - Jaké bylo nejdelší období, kdy dávka Proventilu pro Marie rostla?
        - **SELECT SNAPSHOT RISING**(Dávka)
        - **FROM** Předpis
        - **WHERE** Jméno = 'Marie' **AND**
        - Lék = 'Proventil'
        - RISING nová agregační funkce. Vrací nejdelší interval, po který daný atribut monotóně roste.
- Shrnutí
    - Výsledek bez podpory času **SELECT SNAPSHOT**
    - **Restrukturalizace** – provede projekci na vybrané sloupce a provede slití času platnosti ekvivalentních řádků
    - **Partitioning** – vybere nejdelší interval(y) času platnosti pro každou řádku

9. No SQL databáze

- **Otázka: No SQL databáze – charakteristika, porovnání ACID a BASE, CAP teorém.**
- Charakteristika
    - databáze, které nejsou založené na datovém modelu – nejsou relační
    - dovedou zpracovávat obrovské množství dat v reálném čase
    - neřeší se transakční zpracování (ACID), data jsou snadno dostupná vždy i v částečně konzistentním stavu
    - podporují nerelační datový model
    - podporují distribuovanou architekturu – lze použít jako centrální db, ale síla v distribuční.
    - většina DB NoSQL je open source
    - řeší většinou CAP omezením konzistence dat – BASE
- kategorizace
    - klíč-hodnota (key-value) – do DB se ukládá klíč a hodnota
        - podle klíče jsme schopni získat hodnotu
        - zástupci: Redis, Riak
    - sloupcové (big table) – ke každému klíči je možné uložit více hodnot odpovídající příslušnému sloupci
        - každý klíč může mít vyplněné hodnoty jiných sloupců – heterogenní záznamy
        - zástupci: Hadoop, Cassandra
        - dokumentové (document) – do DB se ukládají dokumenty ve formátech JSON a BSON
        - každý ukládaný dokument může mít jinou strukturu
        - zástupci: MongoDB , Elasticsearch
    - grafové (graph) – do DB se ukládají uzly, jejich vlastnosti a hrany mezi nima
        - hlavním přínosem jsou implementované algoritmy pro prohledávání grafů a vyhledávání příslušných uzlů
        - neporovnatelně rychlejší než běžná DB
        - zástupci: Neo4j, HyperGraphDB

ACID vs BASE

- ACID
    - Koncept ACID je základem transakčního zpracování nejvíce rozšířených a používaných relačních db.
        - Transakce = více operací najednou (např. poslání peněz z účtu na účet = odepsání z jednoho, připsání na druhý, ale dohromady – když se nepovede jedna operace, ta druhá se vrátí do původního stavu)
    - A – atomičnost transakcí, žádný rozpracovaný stav a to i ve vztahu k možné chybě OS nebo HW (celá transakce proběhne nebo nic)
    - C – konzistence, v DB jsou pouze platná data podle daných pravidel. izolovaná transakce zachovává konzistenci DB
    - I – nezávislost, souběžné transakce se neovlivňují – serializace, pořadí není zajištěno
    - D – trvanlivost, uskutečněná transakce nebude ztracena, a to ani po pádu SW nebo HW
    - nevýhody – netriviální, omezuje změny dat (zamykání) a přístup k datům (rychlost)

- teorém CAP u sdílených distr. Systémů
  - CAP teorém byl prezentován Eric A. Brewerem v roce 2000
  - Dokázán S. Gilbertem a N. Lynch
  - Přijat v NoSQL komunitě - ovlivňuje návrh nerelačních databází.
  - pro distribuovaný systém (konkrétně web) je nemožné současně garantovat následující vlastnosti:
  - Konzistence - consistency – každý uzel/klient vidí ve stejnou chvíli stejná data
    - konzistentní nezávisle na běžících operacích či jejich umístění
    - všechny operace čtení vrací verzi dat vloženou poslední operací zápisu do sdíleného úložiště.
  - Dostupnost - Availability – každý požadavek obsloužen úspěšně či neúspěšně
    - nepřetržitý provoz, vždy možnost zapsat a číst data
  - Tolerance k rozdělení - Partition - funkční navzdory chybám sítě nebo výpadkům uzlů
    - možnost výpadku části infrastruktury (odstávka pro údržbu)
  - Teorém: U sdílených systémů je možné uspokojit maximálně 2 ze 3 požadavků
    - → CA/CP/AP
      - Consistence + Availability
        - = 2fázový commit, protokoly pro (in)validaci cache (např. Cluster databases, LDAP, xFS file system)
        - Příklady
          - RDBMSs - Postgres, MySQL,…
      - Consistency + Partition tolerance
        - = agresivní zamykání, ustojí malé výpadky (např. distribuované databáze a zamykání, protokol pro většinovou shodu)
        - Příklady
          - BigTable (column-oriented/tabular), MongoDB (document-oriented),
      - Availability + Partition tolerance
        - = střídání uzlů, řešení konfliktů, optimistická strategie
        - Příklady
          - Voldemort (key-value), Cassandra (column-oriented/tabular),…
- požadavky na moderní DB
  - cloud, distribuovaná DB
    - decentralizace úložiště dat, redundance pro odolnost proti výpadkům a rychlost, velké objemy dat a velké množství operací
  - problematické datové typy
    - údaje klíč-hodnota, objekty, nestrukturované dokumenty...
  - iterativní vývoj
    - časté změny schématu DB, nebo dokonce žádné schéma a vysoké požadavky na škálovatelnost
    - mobilní zařízení jako klienti
    - nerovnoměrné zatížení prostorové i časové
    - spec. požadavky na dostupnost
  - relační db přestává odpovídat relačnímu konceptu
    - nejedná se o matematické relace, ale spíše kolekce/množiny/ grafy nestr. Dat
    - dodržování ACID omezuje práci s DB
  - úmyslné zanedbání/odpuštění A , C, I nebo D – pro vznik rychlosti a dostupnosti dat
- ACID a BASE
  - Internetové aplikace, jako jsou sociální sítě, blogy, znalostní databáze a další, vytvářejí obrovské množství dat, která musí být dále zpracovávána.
  - Společnosti provozující takové aplikace si musí stanovit preference týkající se výkonnosti, spolehlivosti, dostupnosti a konzistence. Jak již bylo zmíněno výše, CAP teorém – 2/3
  - Pro narůstající počet aplikací je dostupnost a tolerance k rozdělení důležitější než konzistence.
  - Požadavek na internetové aplikace - musí být především výkonné a spolehlivé.
  - Vysokého výkonu lze dosáhnout horizontálním škálováním.
  - Spolehlivosti vyšší lze dosáhnout vyšší redundancí.
  - Velmi obtížné při plném dodržení vlastností ACID.
  - Proto je aplikován přístup BASE.
- BASE
  - řeší CAP omezením konzistence dat
  - Basically Available - aplikace pracuje víceméně pořád

    - Soft state - nemusí být stále konzistentní
    - Eventual consistency - nakonec se do konzistentního stavu dostanou
        - nekonzistence řešeny při čtení (verzování, nevalidní cache), zápisu (distribuce změn) nebo asynchroně (replikace dat)
    - nemá silnou konzistenci = při čtení není zajištěno, že každý, kdo čte, dostane aktualizovaná data
        - ošetření chyb ne na datové, ale v aplikační vrstvě
- ACID vs BASE
    - silná konzistence X slabá konzistence,
    - izolovanost X dostupnost na prvním místě
    - orientace na commit X přibližné odpovědi jsou ok
    - vnořené transakce X jednodušší, rychlejší
    - dostupnost? X dodávka jak to jen půjde,
    - konzervativní (pesim.) X agresivní (optimistické)
    - složitá evoluce (schématu..) X jednodušší evoluce

10. Kategorie NoSQL databází

- **Otázka: Kategorie No SQL databází na základě datového modely, příklady architektur.**
- SQL a NoSQL databáze
    - Pod pojmem SQL databáze – především relační a objektově-relační databáze používající jazyk SQL.
    - Dlouhý vývoj - na velmi vysoké úrovni poskytovaných služeb i spolehlivosti.
    - Transakční zpracování - silná konzistence dat.
    - Je-li silná konzistence podmínkou zpracování – SQL databáze je správná volba, není důvod hledat jiné řešení.
- Problém zpracování velkého objemu dat
    - V případě velmi velkého množství dat nebude pravděpodobně možné dosáhnout potřebné kapacity a výkonu škálováním vertikálním - bude nutné použít škálování horizontální.
    - Při horizontálním škálování je udržování silné konzistence databáze se vzrůstající zátěží stále náročnější a vede ke snižování výkonu i dostupnosti.
    - Pro řadu aplikací nepřípustné:
        - Amazon uvádí, že prodloužení odezvy o 0,1s znamená snížení prodeje o 1%.
        - Google uvádí, že prodloužení odezvy o 0,5s znamená propad v provozu až 20% a s tím spojené finanční ztráty.
        - Pro mnoho aplikací je přijatelnější nemít vždy databázi zcela konzistentní, ale zato neustále dostupnou s vysokou rychlostí odezvy uživateli.
- Řešení
    - NoSQL DB vyvinuty podle požadavků odlišných od požadavků na relační (SQL) DB v době jejich vzniku.
    - Jejich použití je také jiné a nemá smysl dělat přímá srovnání.
    - Oba přístupy jsou opodstatněné v určitých případech použití.
    - Mnoho společností proto používá současně databáze relační (SQL) a NoSQL.

Kategorie NOSQL databází

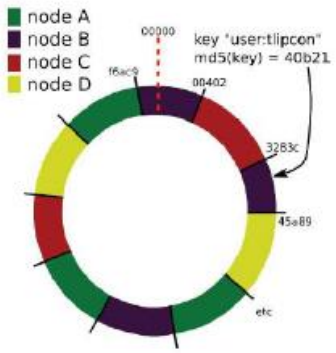
- Možnosti kategorizace
    - Velmi široká škála produktů, jsou často od sebe navzájem odvozovány
    - Řada společných vlastností
    - Rozdělení na základě datového modelu
        - Klíč-hodnota (Key-value)
            - Inspirováno Amazons Dynamo
            - Datový model: kolekce klíč-hodnota (keyvalue)
            - Společné charakteristiky (věk-pohlaví)
            - Příklad:
                - Dynomite, Voldemort, Tokyo,
        - Sloupcové databáze (Column-oriented)
            - Inspirováno Google Bigtable

        - Datový model: rodiny sloupců (Column family)
        - Například adresa (ulice-město-země)
        - Příklad:
            - Hbase, Hypertable
    - Dokumentové databáze (Document databases)
        - Inspirováno Lotus Notes
        - Datový model: kolekce semistrukturovaných dokumentů
        - Obecně textové dokumenty, data v XML,…
        - Příklad:
            - MongoDB, CouchDB,
    - Grafové databáze (Graph databases)
        - Inspirováno teorií grafů
        - Datový model: vrcholy, hrany, key-value u obou
        - Příklad:
            - Neo4j, VertexDB, DEX

Architektura NOSQL databází

- Existující projekty a řešení
    - Jednotlivé implementace NoSQL databázových systémů se od sebe hodně liší,
    - lze však vysledovat některá opakující se architektonická řešení,
    - u jednotlivých implementací se potom vyskytují určité rozdíly.
- Partitioning
    - 1. Virtuální nódy/uzly
        - Infrastruktura, na které jsou provozovány NoSQL databázové systémy, jako je například Bigtable, MongoDB nebo Cassandra, je tvořena velkým počtem serverů, které se od sebe liší svojí konfigurací.
        - Z důvodu zohlednění a využití rozdílného výkonu serverů se používají v konfiguracích databázových systémů takzvané virtuální nódy.
        - Na jednom fyzickém serveru může běžet několik virtuálních nodů, v závislosti na jeho konfiguraci.
        - Virtuální nódy používá MongoDB.
    - 2. Dělení (Sharding)
        - Sharding je metoda, která rozděluje velké soubory dat rozdělit na několik serverů.
        - Metodu používá například MongoDB nebo Redis.
        - Shard v pojetí MongoDB je skupina serverů, které udržují identické kopie určené části dat.
    - 3. Hashing
        - Nejjednodušší způsob rozdělení dat spočívá v rozdělení rozsahu primárního klíče
        - pomocí hash funkce na stejné části, kdy každá část leží na jednom serveru. Klient pomocí hash funkce určí, na kterém serveru leží požadovaná data.
        - Tento přístup využívá například systém Memcached.
        - Nevýhoda: V případě výpadku jednoho ze serverů nebo při přidání dalšího serveru, musí být rozdělení dat provedeno znovu a data podle nového rozdělení opět rozdistribuována.
        - Poznámka:
            - Systém Memcached, který má relativně malý objem dat uložen pouze v operační paměti a redistribuce je snadno provedena přirozeným během databáze – důvod, proč se nevýhoda neprojeví.
            - U databází, které mají data uložena na lokálním disku, je nutno kopírování velkých objemů dat mezi servery.
    - 4. Consistent Hashing
        - Některé NoSQL databázové systémy, jako Cassandra nebo MongoDB, používají Consistent hashing.
        - V systému Consistent hashing leží celý rozsah primárního klíče na pomyslném kruhu.
        - Každá hodnota primárního klíče má svého správce, kterým je první nod na pomyslné kružnici ve směru hodinových ručiček.
        - Výhoda řešení:
            - přidání, odebrání nebo výpadek nodu se dotkne pouze nodů s ním sousedících.

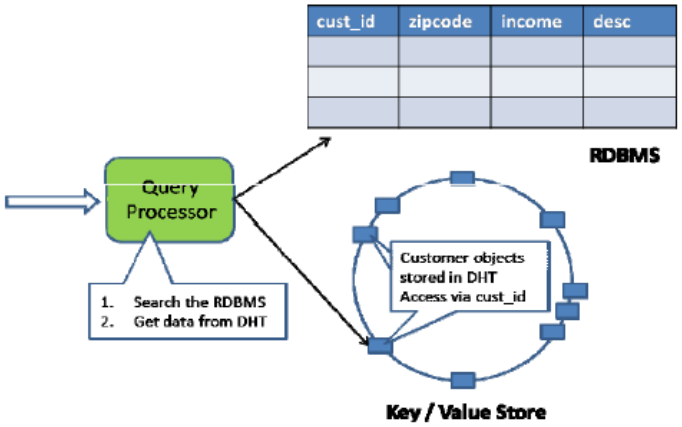
- Redistribuce dat je nutná pouze mezi těmito sousedícími nody a ostatních nodů se nijak nedotkne.

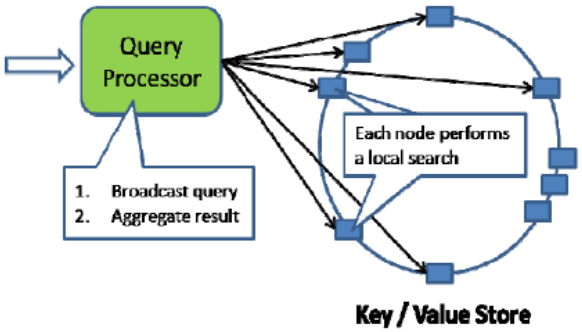


- 5. Replikace
  - K zajištění odolnosti proti výpadkům serverů (při vysokých počtech v infrastrukturách horizontálně škálovaných databází velmi časté) a k zajištění vysoké dostupnosti jsou data udržována ve více kopiích pomocí replikace.
  - Replikace současně umožňuje rozkládání zátěže.
- 6. Správa členských serverů (Membership)
  - Přidávání i odebírání nodů musí probíhat bez dopadu na dostupnost databáze.
  - Přidání nového nodu:
    - Nod náhodně vybere svoji pozici na pomyslné kružnici hodnot primárního klíče.
    - Oba nejbližší sousedi předají části svých rozsahů a provedou změny na replikách.
    - Nový nod si zkopíruje data od svých sousedů a informuje o nastalých změnách všechny zbývající nody.
    - V případě odebrání nebo havárie nodu nod přestane rozesílat a odpovídat na stavové zprávy z ostatních nodů.
    - Data zůstávají dostupná na jeho replikách.
    - Sousední nody si opět upraví své rozsahy, včetně všech replik.
- 7. Gossip
  - Pro šíření stavových informací a detekci havarovaných nodů je používán protokol Gossip.
  - Gossip nepoužívá broadcasty - každý nod posílá zprávy jen omezenému počtu dalších nodů.
  - Zpráva se postupným předáváním dostane ke všem nodům.

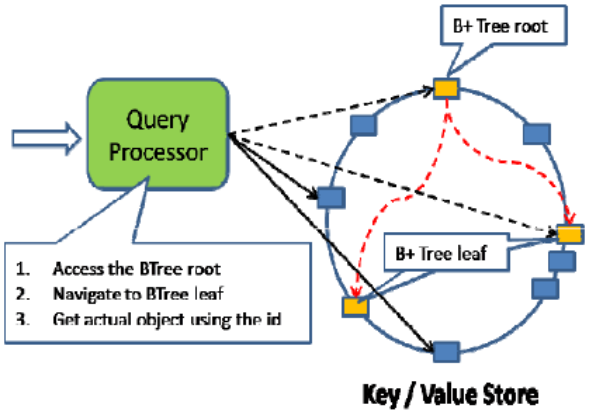
- Konzistence dat
  - Podle CAP teorému nemůže být dosaženo striktní konzistence společně s vysokou dostupností a tolerancí k rozdělení (partition-tolerance).
  - **Striktní konzistence** je zaměněna za přístup **Eventually consistent** (tímto způsobem je dosaženo preferované vysoké dostupnosti a tolerance k rozdělení.
  - Následně je někdy potřeba rozhodnout, ke které z konkurenčních aktualizací dat se přiklonit.
  - A. Časová značka (Timestamps)
    - Závisí na synchronizovaných hodinách na všech serverech a nezachycuje kauzalitu aktualizací
  - B. Verzování (Multiversion)
    - Udržování více verzí dat jednotlivých řádků databáze, u každé verze je uložen časový údaj.
    - Časový údaj nemusí odpovídat skutečnému času - může to být jakákoli hodnota umožňující určit pořadí aktualizací.
    - Na dotaz klienta vrací databáze nejaktuálnější hodnotu.
    - Také může vracet nejaktuálnější hodnotu před zadaným časem.
  - C. Vektorové hodiny
    - přístup, který zachycuje pořadí aktualizací a umožňuje udělat rozhodnutí v případě konfliktů.
    - Vektorové hodiny jsou definovány jako N-tice stavu hodin na každém nodu: Vx{tnode1, tnode2, … ,tnodeN}
    - Aktualizace dat mezi nody (například pomocí protokolu Gossip) obsahuje společně s hodnotou dat i časový vektor, který je složen z časových hodnot na všech serverech.
    - Hodnota hodin nemusí vycházet ze skutečného času (jakákoli hodnota, která umožní určit pořadí).
    - Pořadí aktualizací je určováno porovnáním vektorových hodin.
    - Vektorové hodiny jsou používány k zabezpečení konzistence dat mezi více replikami.

        - Mohou být použity i v komunikaci klienta s databází:
            - Klient současně s požadavkem na server pošle poslední hodnotu vektorových hodin, kterou zná.
            - Server zaručí, že data, která mu pošle zpět, budou novější.
        - Výhoda použití vektorových hodin:
            - oproti předchozím systémům spočívá v nezávislosti na synchronizovaných hodinách.
            - není potřeba mít k dispozici dlouhou historii aktualizací ani není potřeba udržovat několik verzí dat na všech nodech.

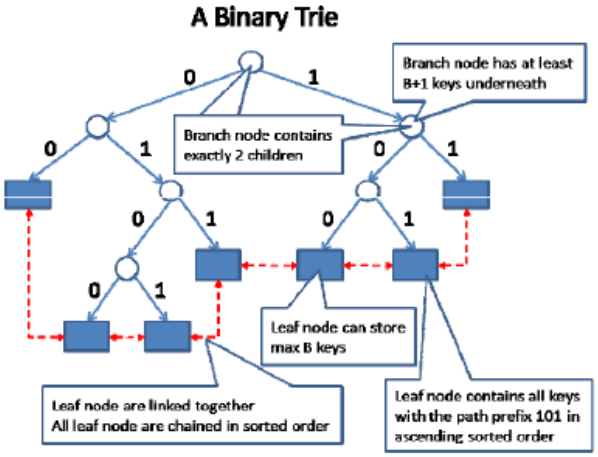
- Vyhledávání
    - Mezi je různými NoSQL databázemi je velký rozdíl v možnostech vyhledávání.
    - Databáze skupiny klíč-hodnota (key-value)
        - nabízejí většinou jen možnost vyhledávání podle primárního klíče a veškeré další operace jsou prováděny na straně klientské aplikace.
    - Některé dokumentové a grafové databáze mají bohatší dotazovací jazyk pro provádění komplexnějších dotazů.
    - 1. Companion SQL databáze
        - a. Databáze, do které se zkopírují atributy podle kterých je potřeba vyhledávat.
        - b. Schopnosti vyhledávání SQL jsou použity k vyhledání primárních klíčů, podle kterých se potom k datům přistupuje v NoSQL databázi.
        - 

    - 2. Scatter-Gather lokální vyhledávání
        - No SQL databáze umožňuje vyhledávání a indexování na jednotlivých nodech.
        - Dotaz od klienta je nejdříve připraven dotazovacím zpracovatelem (query processor),
        - Dotaz rozešle na všechny nody databáze, kde je provedeno lokální vyhledávání.
        - Každý z nodů pošle výsledek lokálního vyhledávání zpět zpracovateli, který výsledky agreguje a pošle klientovi.
        - 

    - 3. Distributed B-tree
        - Lze udělat hash atributu, podle kterého se bude vyhledávat a najít kořenový nod distribuovaného B-tree indexu.
        - Ten obsahuje hodnotu, která je identifikátorem jeho potomka.
        - Klient potom vyhledá nod potomka.
        - Opakováním tohoto procesu je dosaženo nodu, kde leží vyhledávaná data.



- 4. Prefix Hash Table (Distributed Trie)
  - Trie je stromová datová struktura, kde každý uzel obsahuje všechny podřetězce, kterými může pokračovat řetězec v dosud prohledané cestě.
  - Všichni následníci mají společný prefix, který je shodný s řetězcem přiřazeným k danému uzlu.
  - Kořen je asociovaný s prázdným řetězcem.
  - Hodnoty nejsou obvykle asociovány se všemi uzly, ale jen s listy a některými vnitřními uzly, které odpovídají klíčům.



11. Big Data

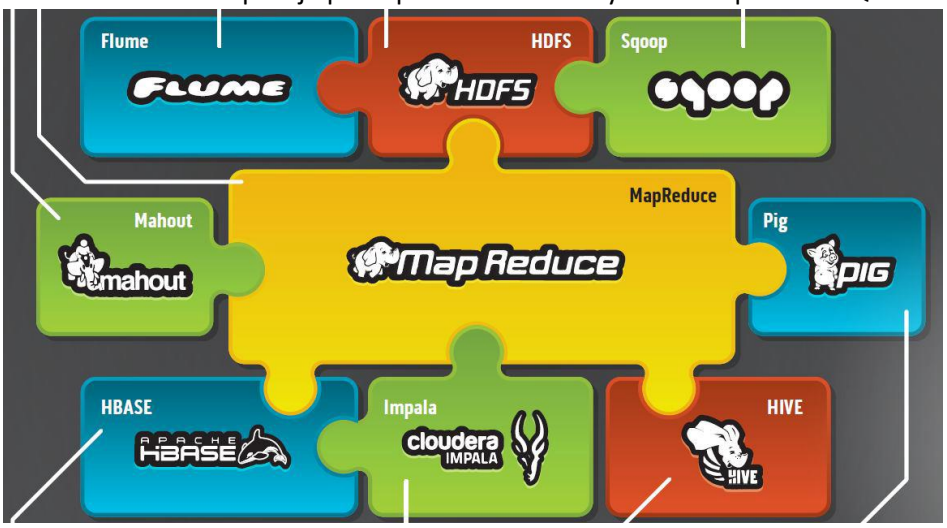
- **Otázka: Problém zpracování velkých dat, Big Data – charakteristické vlastnosti (3V), příklady uvedeného typu dat, alternativy zpracování, princip Map Reduce, Hadoop.**
- Aktuální problém – Big Data
  - Big data
    - 3V – objem (Volume)
      - velikost
    - Rychlost (Velocity)
      - Batch = data, která se stahují např. jednou denně
      - Near-time = stahuje se dle události
      - Real-time
      - Stream = stahuje se pořád
    - Různorodost (Variety)
      - Strukturované
      - Semi-strukturované
      - Nestrukturované
      - kombinace
    - + věrohodnost (Veracity)
      - Kvalita dat
  - Zpracování Big Data

        - Efektivní využívání systémů vyžaduje odpovídající nástroje pro jejich ukládání na nízké úrovni a analytické nástroje ve vyšších úrovních
        - Z pohledu uživatele je nejdůležitějším aspektem zpracování velkých objemů dat na počítači jejich analýza – Big analytics
    - Exponenciální tendence růstu dat (v ZB)
    - Big analytics
        - Data v různých formátech – relační tabulky, XML data, textová data, multimediální data nebo RDF trojice
        - Problém při zpracování při data miningu
        - Řešení škálování – selhání tradičních databázových architektur
        - Rozšíření metod používaných v datových skladech
        - Analytické zpracování vyžaduje nejen nové databázové architektury, ale i metody analýzy dat
    - Možná řešení problému Big Data
        - Lze volit tradiční SŘBD, paralelní DBS, distribuované souborové systémy (např. HDFS), datová úložiště typu klíč-hodnota (NoSQL = Not Only SQL databáze) a nové databázové architektury (NewSQL databáze)
        - Pro volbu technologie – rozhodující aplikace, které transakční i analytické
        - Hardware – konsolidovaná integrovaná řešení s důrazem na výkonnost (storage, server, …)
        - Distribuce dat – např. Hadoop
        - Data Management – např. NoSQL
        - Analýza a vizualizace – trend: in-memory

Apache Hadoop

- = programové prostředí, které umožňuje paralelní běh big data aplikací v rámci výpočetního clusteru
- Zahrnuje sadu nástrojů pro distribuované pořizování, ukládání a zpracování velkých dat
- Open-source systém volně dostupný i pro komerční použití (podobně jako webový server Apache)
    - Existují i komerční distribuce, součástí jejichž licence je i provozní podpora (např. Hortonworks, Cloudera, MapR)
- Framework – sada open-source komponent určených pro zpracování velkého množství nestrukturovaných a distribuovaných dat HDFS (Hadoop Distributed File Systém)
- Verze 2012 – až 4000 uzlů
- Programový model: map-reduce
- Kdy použít Hadoop?
    - Ke zpracování je potřeba velký výpočetní výkon, data mining, statistické metody
    - Data přibývají velmi rychle (desetitisíce zpráv/s), analýzy dat v reálném čase
    - Celkový objem dat k uložení je velký (desítky TB)
        - ~ spolehlivé, dostupné úložiště
- Hadoop vs. Databáze
    - Tabulky vs. Soubory
        - Možnost uložení dat ve zdrojové struktuře i formátu
        - Systém HDFS zajistí bezpečné a efektivní uložení souborů libovolné velikosti
    - Pevné schéma vs. Volná struktura
        - Uložení souborů, struktura se definuje až při použití
    - Transakčnost vs. Dostupnost a škálovatelnost
        - Hadoop upřednostňuje rychlost odpovědi před úplností,
        - Spolehlivost je zajištěna redundancí – každý uzel clusteru je kdykoliv nahraditelný
- Komponenty Hadoopu
    - Map Reduce
        - Prostředí pro vývoj JAVA aplikací pro řešení distribuovaných úloh metodou lokálního zpracování (Map) a skládání výsledků (Reduce)
        - Distribuované – rovnoměrné rozložení úlohy do jednotlivých výpočetních uzlů s ohledem na jejich výkon
        - Škálovatelné – snadné doplnění dalších uzlů do clusteru

        - Robustní – automatická obnova úlohy při lokálním výpadku
    - Pig
        - Programové prostředí a skriptovací jazyk pro snazší programování MapReduce (vyvinul Yahoo!)
    - Flume
        - Zpracování objemných proudů dat v reálném čase
    - HDFS
        - Distribuovaný souborový systém, který umožňuje bezpečně a efektivně ukládat a zpracovávat velké soubory v rámci Hadoop clusteru
        - Úsporný – není nutný specializovaný HW, poběží na obvyklých serverech
        - Spolehlivý – obsahuje vlastní řešení redundance dat (bez nutnosti RAID) a obnovy dat při výpadku části clusteru
        - Rozšiřitelný – automaticky přenáší data na nové uzly při zvětšení clusteru
    - Sqoop
        - Dávkové přenosy dat z relačních databází
    - Mahout
        - Knihovna pro data mining a strojové učení
    - HBASE
        - Nerelační databáze v HDFS
        - Umožňuje asociativní uložení řídkých dat (po vzoru Bigtable od Google) a velmi rychlé distribuované prohledávání, čtení i zápis
        - Rychlost odezvy je důležitější než časová konzistence a úplnost
    - Impala
        - In-memory databáze pro přístup do HDFS a HBASE v reálném čase (vyvíjí Cloudera)
    - HIVE
        - Prostředí datového skladu nad HDFS (vyvinul Facebook)
        - Umožňuje vytvářet tabulky, schémata a reporty přímo nad daty v HDFS
        - Podporuje přístup ke strukturovaným datům pomocí HQL dotazů (obdoba SQL)



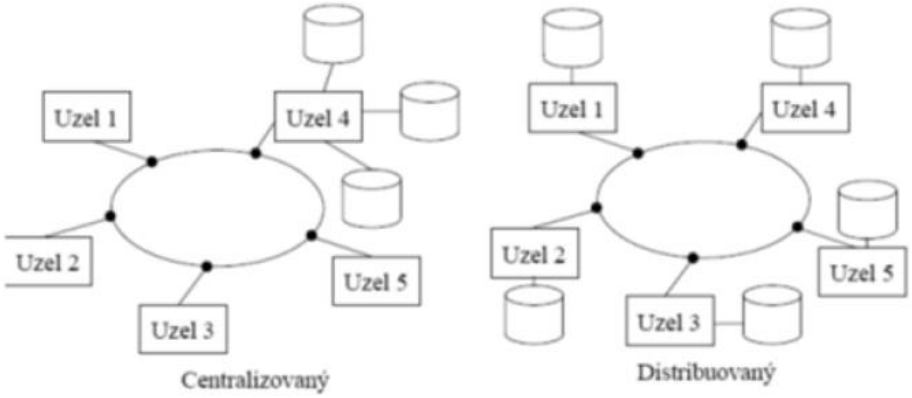
- Největší uživatelé Hadoop technologií
    - Facebook, Yahoo, eBay, last.fm
- Další možnosti
    - HANA – high-performance analytic appliance (od SAPu)
        - Podporuje SQL
        - Real-time analýzy a aplikace
    - MongoDB
        - Dokumentově orientovaná NoSQL databáze
        - Vysoký výkon, snadná škálovatelnost = rozšiřitelnost (množství funkcí, sdílení uživatelů,..při větším zatížní), dostupnost
        - Vlastní dotazovací jazyk podobný SQL
    - Apache Cassandra

        - Apache Cassandra je masivně škálovatelná NoSQL databáze.
        - Je navržena pro práci s velkým objemem real-time dat na několika datových centrech s žádným kritickým bodem pro selhání celého systému.
        - Poskytuje vysoký databázový výkon a neustálou dostupnost.
        - Cassandra na rozdíl od Hadoop nepracuje na bázi key – value (klíč – hodnota) úložiště, nýbrž jako column – family (sloupec – rodina) úložiště, kde se ukládaný objekt skládá z trojice: název, klíč, časový otisk.
        - V praxi to znamená upřednostnění využití projektu Cassandra na úkor Hadoop při zpracování dat z naměřených časově podložených údajů nebo bezpečnostních logů, kde čas hraje významnou roli.
- Hype křivka – přijetí nové technologie
    - 

12. Distribuované databáze

- **Otázka: Distribuované databáze - koncepce distribuovaného databázového systému, replikace a fragmentace dat, distribuovaná správa transakcí. Distribuované systémy v souvislosti s databázemi No SQL.**
- Definice
    - Distribuovaný systém je takový systém propojení množiny nezávislých uzlů, který poskytuje uživateli dojem jednotného systému
    - Jednotný obraz systému, virtuální uniprocesor
    - Hardwarová nezávislost
        - uzly jsou tvořeny nezávislými “počítači” s vlastním procesorem a pamětí
        - jedinou možností komunikace přes komunikační (síťové) rozhraní
    - Dojem jednotného systému
        - uživatel (člověk i software) komunikuje se systémem jako s celkem
        - nemusí se starat o počet uzlů, topologii, komunikaci apod.
    - distribuovaný systém - celá škála různých řešení
        - multiprocesory na jedné základové desce
        - celosvětový systém pro miliony uživatelů
    - Uvažované termíny v dalším výkladu: uzel = (‘běžný’) počítač, softwarový systém
        - principy, algoritmy
    - V databázi neexistuje žádný centrální uzel nebo proces odpovědný za vrcholové řízení funkcí celého systému
    - výrazně to zvyšuje odolnost systému proti výpadkům jeho částí
- = databázové systémy + počítačové sítě
    - DBS = integrace
    - PS = decentralizace
    - DDBS= integrace dat bez centralizace (je variantou DVS)
- Požadavky na moderní databáze
    - cloud, distribuované databáze (decentralizace úložiště dat, úmyslná redundance pro odolnost proti výpadkům a rychlost, velké objemy dat a velké množství operací /big data/, atd.)
    - problematické datové typy (údaje klíč-hodnota, objekty, nestrukturované dokumenty, RDF grafy, atd.)
    - iterativní vývoj (časté změny schématu databáze nebo dokonce žádné schéma, různé/nejasné způsoby použití databáze, atd.)

    - vysoké požadavky na škálovatelnost (mobilní zařízení jako klienti i úložiště/poskytovatelé dat, nerovnoměrné rozložení zátěže prostorově i časově, specifické požadavky na dostupnost, předem neznámé dotazy - nelze optimalizovat indexy, atd.)
- Koncepce distribuovaného DBS



-
    - Množina uzlů propojených komunikační sítí
        - Každý uzel je sám o sobě DBS
        - Z každého uzlu lze transparentně zpřístupnit data kdekoliv v síti
- Teorém CAP u sdílených/distribuovaných systémů
    - Consistency
        - každý uzel/klient vidí ve stejný čas stejná data, (data konzistentní nezávisle na běžících operacích či jejich umístění)
    - Availability
        - každý požadavek obsloužen, úspěšně nebo neúspěšně, (nepřetržitý provoz, vždy možnost zapsat a číst data)
    - Partition Tolerance
        - funkční navzdory chybám sítě nebo výpadkům uzlů. (možnost výpadku části infrastruktury, např. odstávka pro údržbu)
    - Teorém: U sdílených systémů je možné uspokojit maximálně 2 ze 3 požadavků. - Eric Brewer (+ N. Lynch), 2000
        - → CA/CP/AP
            - Consistence + Availability
                - = 2fázový commit, protokoly pro (in)validaci cache (např. Cluster databases, LDAP, xFS file system)
            - Consistency + Partition tolerance
                - = agresivní zamykání, ustojí malé výpadky (např. distribuované databáze a zamykání, protokol pro většinovou shodu)
            - Availability + Partition tolerance
                - = střídání uzlů, řešení konfliktů, optimistická strategie
- Požadavky na ideální DDBS
    - Uživateli se jeví distribuovaný systém přesně jako nedistribuovaný
    - Lokální autonomie
    - Nespoléhání se na centrální uzel
    - Nepřetržitá činnost
    - Nezávislost na umístění dat (transparentnost)
    - Nezávislost na fragmentaci
    - Nezávislost na replikaci dat
    - Distribuované zpracování dotazu
    - Distribuovaná správa transakcí
    - Nezávislost na HW, OS, SŘBD, komunikační síti
- Některé problémy
    - Přenos v síti je pomalý → minimalizace poštu a objemu zpráv
    - Zpracování dotazů

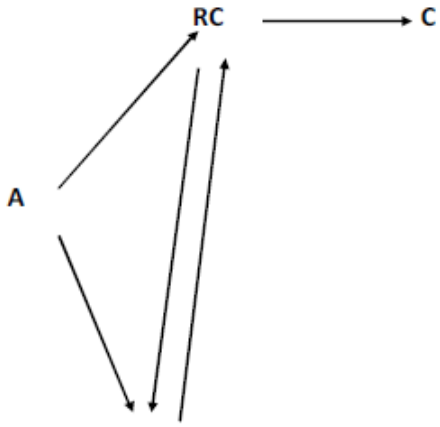
        - Globální/lkální optimalizace
    - Správa systémového katalogu
    - Šíření aktualizace při replikaci
        - Schéma s primární kopií
    - Zotavení po chybách a poruchách
        - Dvoufázový potvrzovací protokol (2PC)
    - Souběžnost
        - Schéma s primární kopií
        - Nebezpečí globálního uváznutí
- Taxonomie
    - Autonomie: těsná integrace, poloautonomie, úplná izolace
        - těsně integrované (uživatel vidí data centralizovaná v jediné databázi). DDB je vybudována nad lokálními DB. Každé místo má úplnou znalost o datech v celém DDBS a může zpracovávat požadavky používající data z různých míst.
        - semiautonomní systémy (lokální DBMS pracují nezávisle a sdílejí svoje lokální data v celé federaci). Jen část jejich dat je sdílena.
        - zcela autonomní = izolované. Lokální DBMS pracují nezávisle a neví o ostatních DBMS. Pro vzájemnou komunikaci potřebují
    - Distribuce: jeden uzel, distribuovaný
    - Heterogenita: homogenní, heterogenní
- Distribuovaný výpočetní systém
    - Je rozložen do uzlů sítě spojených komunikačními kanály
    - V uzlech je možné autonomně ukládat a zpracovávat data
    - Prostředky v uzlech mohou být nehomogenní
    - Uživatel nemusí vědět o existenci ostatních uzlů (tzv. transparentnost)
- Motivace pro distribuované databáze
    - Výhody – proč ano
        - lokální autonomie (odpovídají struktuře decentralizovaných organizací. Data uložena v místě nejčastějšího využití a zpracování - zlevnění provozu). V centralizované DB je nutné připojovat se ke vzdálené databázi = přídavná režie, cena komunikace, zatížení sítě, rychlejší
        - zvýšení výkonu (inherentní paralelismus rozdělením zátěže na více počítačů)
        - spolehlivost (replikace dat, degradace služeb při výpadku uzlu, přesunutí výpočtů na jiný uzel)
        - lepší rozšiřitelnost konfigurace (přidání procesorů, uzlů)
        - větší schopnost sdílet informace integrací podnikových zdrojů
        - uzly mohou zachovat autonomní zpracování a současně virtuálně zabezpečovat globální zpracování
        - agregace informací (z více bází dat lze získat informace nového typu)
    - Nevýhody – proč ne
        - složitost (distribuce databáze, distrib. zpracování dotazu a jeho optimalizace, složité globální transakční zpracování, distribuce katalogu, paralelismus a uvíznutí, případná integrace heterogenních dat do odpovídajících schémat, složité zotavování z chyb)
        - cena (komunikace je navíc)
        - bezpečnost
        - obtížný přechod (neexistence spolehlivého automatického konverzního prostředku z centralizovaných DB na DDB)
- Požadavky DDBS
    - Transparentnost distribuce (míra viditelnosti distribuce dat pro uživatele
    - Autonomie (distribuce řízení)
    - Heterogennost
    - Výkon (vysoká průchodnost krátká odezva)
    - Požadavky DSŘBS
        - Dle formulace (autor Date):
            - lokální autonomie – každé místo má lokální SŘBD
            - vše je distribuované – ve všech službách se nespoléhá na žádné centrální místo
            - kontinuálnost – akce v jednom místě (např. odstranění tabulky) by neměla příliš narušovat provoz DDBS jako celku
            - nezávislost na místě – uživatel nemusí vědět, kde jsou uložena potřebná data

- nezávislost na fragmentaci – nemusí vědět, kde jsou fragmenty
- nezávislost na replikaci
- možnost distribuovaného zpracování dotazu – nemělo by být nutné přesouvat data ke zpracování dotazu do jednoho místa
- možnost distribuovaného zpracování transakcí – může dojít ke konfliktu s 1. Pro zajištění korektnosti se používá 2 fázový potvrzovací protokol
- nezávislost na hardware
- nezávislost na OS
- nezávislost na síti
- nezávislost na DBMS

  - Implementace realizují jen část ideí Date. (Ingres Star, IBM DRDA, Oracle 7 a vyšší)

- Formy transparentnosti
  - Datová nezávislost = imunita uživatelské aplikace ke změnám v definici a organizaci dat. Je požadavkem i pro centralizované DB
    - Logická nezávislost (logická struktura databáze)
    - Fyzická nezávislost (konkrétní způsob uložení dat)
  - Síťová transparentnost = ukrytí síťových detailů = uživatel neví o síti
  - Replikační transparentnost = neobtěžovat uživatele skutečností, že pracuje s daty existujícími ve více kopiích = uživatel neví o replikách
  - Fragmentační transparentnost = uživatelův dotaz je specifikován na celou relaci, ale musí být vykonán na jejím fragmentu = uživatel neví o fragmentech
- Distribuované zpracování transakcí
  - Transakce = posloupnost operací, které se provedou buď všechny nebo žádná –zajišťuje je transakční monitor (Atomicity, Konsistence, Isolation, Durability)
  - Transakce v DDBS rozděleny na podtransakce (prováděny na různých místech).
  - Transakční zpracování v jednotlivých uzlech nestačí.

- **Podtransakce má stavy:**
- **A (Active)**
- **C (Commited)**
- **AB (ABorted)**
- **RC (Ready to Commit)**
- **F (Failed)**
- **Transakční provoz zajišťují zprávy:**
- **PREPARE to COMMIT**
- **READY to COMMIT**
- **COMMIT**
- **COMMIT SUCCESFULL**
- **ABORT**
- **ABORT COMPLETED**
- **Provedení READ/WRITE na datech**



Obr. Stavy podtransakcí

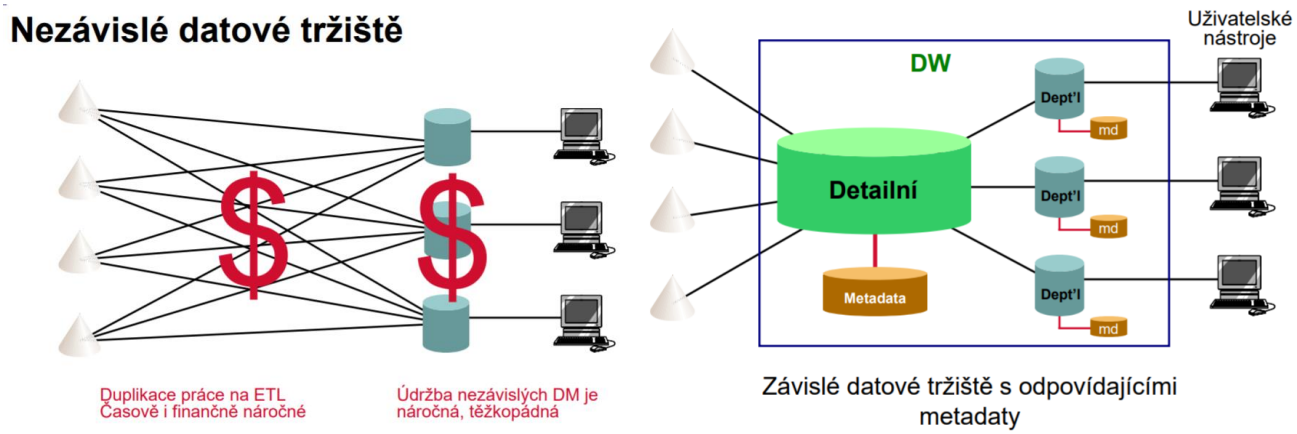
  -
  - Dvoufázový potvrzovací protokol
    - F1. Koordinátor zašle všem místům (kde se provádí podtransakce dané transakce) zprávu s požadavkem na připravenost (PREPARE to COMMIT). Hlásí-li některé místo nepřipravenost, provede koordinátor ROLLBACK transakce ve svém místě a pošle zprávu všem místům na ABORT podtransakcí.
    - F2. Ohlásí-li všechna místa úspěšnost (READY to COMMIT), tak koordinátor potvrdí transakci a její lokální části ve svém místě. Pak odešle zprávy s požadavkem na potvrzení (COMMIT) všem participantům. Tyto zprávy budou dříve nebo později doručeny.
  - Jiný způsob
    - Pravidlo o potvrzení lze formulovat také:
      - Koordinátor zruší transakci právě když alespoň jeden participant hlásí zrušení transakce
      - Koordinátor potvrdí transakci právě když všichni participanti hlásí, že jsou připraveni ji potvrdit
    - Poznámka: Nevrací-li se odpověď, je po uplynutí time-out transakce zrušena.
- Postup návrhu DDBS

    - Top down:
        - Návrh GCS
        - Návrh fragmentace
        - Návrh alokace
        - Návrh fyzické struktury
        - Zelená louka
    - Bottom up:
        - Výběr společného DB modelu pro popis globálního schéma
        - Překlad každého lokálního schéma do společného DB modelu
        - Integrace lokálních schémat do společného globálního schéma
- Distribuce globálních relací (techniky)
    - S replikami
    - Ve fragmentech
    - S replikami i ve fragmentech
- Fragmentace
    - vhodné, pokud se v každém uzlu systému pracuje s částí tabulky
    - rozdělení relace na několik subrelací, které se umístí do jednotlivých DB
    - data se umístí tam, kde se nejčastěji používají – ostatní v ostatních DB
    - omezení síťových přenosů
    - vyšší bezpečnost – data pouze tam, kde jsou třeba
    - musí být – úplná, disjunktní, rekonstruovatelná
    - typy
        - horizontální(selekce) – podmnožina řádků
        - vertikální (projekce)– podmnožina sloupců
        - dokonce smíšená – kombinací
- replikace
    - udržování kopií tabulek nebo fragmentů ve více DB
    - omezení síťových přenosů
    - zvýšení dostupnosti dat při výpadku části DS
    - náročná údržba kopií
    - vhodné pro často používaná data, která se málo aktualizují
    - typy
        - synchronní – aktualizace na všech uzlech, součástí transakce o degraduje výkon X spolehlivý
        - asynchronní – na vyžádání nebo v pravidelných intervalech, nemusí být vždy aktuální
    - typy podle způsobu provádění repliky
        - pouze změněné řádky – nutný log změněných řádků
        - úplná
        - DML příkazy…
    - typy podle způsobu zacházení s replikami
        - pouze pro čtení (master-slave)
            - změny možné provádět jen u mastera
            - repliky jsou ReadOnly
        - pro transakční zpracování
            - změny možné provádět i v replikách – musí se synchronizovat obousměrně
            - přináší řadu problémů – konflikty při změně

	master	kdekoliv
synch	+ změny nemusí být koordinovány + žádné nekonzistence - dlouhá odezva - v hodné pro málo změn - lokální kopie mohou být pouze čteny	+ žádné nekonzistence + elegantní (symetrické) řešení - dlouhá odezva - změny musí být koordinovány
asynch	+ koordinace není nutná + krátký čas odezvy - lokální kopie nejsou up-do-date - nekonzistence	+ žádná centrální koordinace + nejkratší čas odezvy - změny mohou být ztraceny (urovnávání rozdílů) - nekonzistence

13. Datový sklad

- **Otázka: Datový sklad, multidimenzionální databáze, možnosti využití.**
- Data Warehouse (datový sklad)
  - Centrální úložiště různých dat firmy
  - Integrace firemních datových zdrojů
  - Historie v datech
  - Centrální místo pro podporu informačních potřeb uživatelů
  - Velké objemy dat – lze v rozumném intervalu
  - Navrženo pro podporu analýzy – speciální struktury uložení dat
  - Pro analytickou podporu rozhodování
  - Nejen data v databázi, ale i nástroje pro extrakci dat, pro reporting, analýzu dat, data mining¨
  - Prezentace dat uživatelsky příjemným způsobem – není potřeba IT (stř.+vyšší management)
  - Přínosy řešení datového skladu
    - Integrace a čistota dat, integrační efekty
    - Podpora úloh analytického charakteru a jejich ekonomické a mimoekonomické efekty
      - Vyšší flexibilita řízení a realizace změn
      - Zpětná vazba
      - Nezávislost vedoucích pracovníků na oddělení IT
      - Kvalifikační efekty – v IT i v ekonomice řízení (práce s informacemi, uvažování v kontextu)
    - Automatizace rutinních procesů (tvorba výkazů, zpráv)
    - Kontrola plnění plánů a finanční analýza
    - Data jsou „čistější“, vhodnější k interpretaci,..
    - Podpora analýzy dat
      - Trendy, sledování a analýza časových řad
      - Poměrové ukazatele
      - Identifikace odchylek
      - Dril-down, dril-up, dril-across, slice-dice
- Cíle datového skladu
  - Zajistit dostupnost firemních informací
  - Zajistit konzistenci firemních informací
  - Vytvořit adaptivní a pružný zdroj informací
  - Zabezpečit ochranu firemních informací
  - Vytvořit základnu pro firemní podporu rozhodování (analytické centrum)



-
- Datové tržiště = podmnožina datového skladu
  - Část řešení datového skladu určená pro podporu specifické analýzy (účetnictví nebo oddělení firmy (marketing, prodej)

Vlastnost	Data Warehouse	Data Mart
Rozsah	enterprise	oddělení
Obsah	Více oblastí	Jedna oblast
Zdroje	hodně	málo
Velikost (typicky)	100 GB až 1TB	<100GB
implementace	Měsíce až roky	měsíce

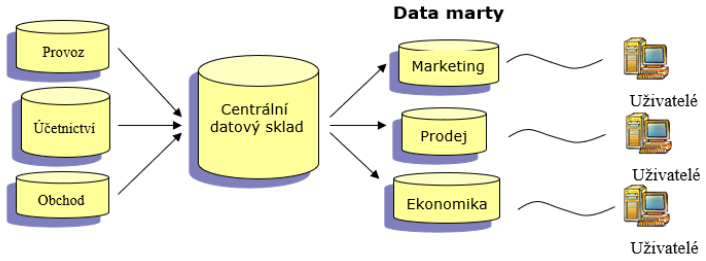
- ETL
  - Extraction – transformation – loading
  - Kompletní proces načtení dat do datového skladu
  - Zahrnuje mnoho subprocesů:
    - Extrakce – výběr dat
    - Transformace – ověření, čištění, integrace dat
    - Loading – načtení dat do DW
    - + = doporučované, ne nutné
    - +kontrola kvality
    - +auditovaní
    - +bezpečnost
    - +zálohování a obnova

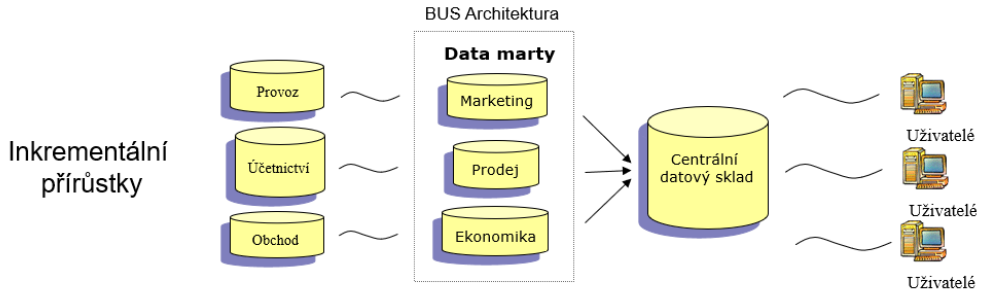
	OLTP	OLAP
Charakteristika	Provozní zpracování	Informační zpracování
Orientace	Transakční	Analytická
Uživatel	Úředník, DB administrátor	Manažer, analytik
Funkce	Každodenní operace	Dlouhodobé info, podpora v rozhodování
data	Současná, zaručeně aktuální	Historická
Zaměření	Zakládání dat	Získávání informací
Počet záznamů	Desítky	Miliony
Velikost DB	100 MB až GB	100 GB až TB

Přístupy k vytváření datových skladů a datových tržišť

- Dva základní způsoby budování
  - Metoda velkého třesku
    - Mám velký datový sklad, kam vše uložím, teprve poté pro ty jednotlivé oblasti čerpám ty metody
    - Provoz, účetnictví, obchod → CDS ← marketing, prodej, ekonomika
    - To berou uživatelé
    - Systémovější, než druhá metoda, rychle se zváží pro a proti, nedojde k duplicitám
  - Přírůstková metoda
    - Inkrementální přírůstky:

        - Provoz + účetnictví + obchod → BUS architektura (data marty = marketing + prodej + ekonomika) → centrální datový sklad
    - Může dojít k duplicitám, každá skupina potřebuje externí data → teprve poté budeme čistit
- ↑ Pozn. – investiční náklady jsou poměrně veliké, ale výhody po implementaci rovněž



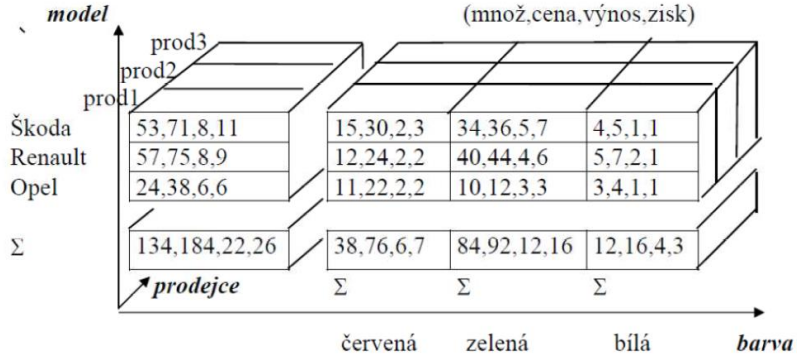


- Nezávislé datové tržiště
- OLTP – datové tržiště (na těch vazbách od OLTP do DT je ETL)
    - ETL – musím určit, která ta data do toho budu dávat (extrakce)
    - ETL – převedení na stejný formát = míle, kilometry, různé měny,…) (transformace)
- Když mám tři datová tržiště – stejné zdroje, stejná tržiště, vazby mezi všemi – ten proces opakuji (tolikrát, kolik datových tržišť) => velmi problematické
    - Zdržení kvůli duplikaci práce na ETL
    - Finančně i časově náročné
    - Musím udržovat → velké zatížení
- Způsoby budování DW
- Preferovaná je přírůstková metoda
    - Zaručuje:
        - Projektovou zvládnutelnost řešení
        - Reálné časové horizonty jednotlivých etap (2-4 měsíce)
        - Řešení aktuálních uživatelských (obchodních) potřeb
        - Zpětnou vazbu uživatelů
- Vs. Metoda velkého třesku je projektově nezvládnutelná
- DW procesy
    - Hlavní proces při tvorbě datového skladu
    - Podprocesy = práce s daty
        - Abych mohl dělat kvalitní rozhodnutí, musím mít kvalitní data
        - Extrakce → označit, co je pro mě důležité
            - Pro DW: musím vědět, co chci = znát trendy, jaká hodnocení, regrese, získávat znalosti?
            - Pro databázi: prioritní úloha programátora – ten rozhoduje o většině procesů
        - Transformace
            - Čištění dat
            - Výběr dat
            - Integrace
            - Umělé klíče
            - Agregace
        - Načtení (Loading) a tvorba indexů
        - Data Quality Assurance
    - Další procesy v DW
        - Publikace dat (prezentační server)
        - Update dat → nemohu to měnit, pokud to tam jednou je

        - Update míněn v tom, že např. některá data nepotřebuji, nebo se objeví nové věci, vazby,..
    - Dotazování
    - Zpětná vazba (čistá dat do OLTP, data z DM do DW)
        - Může se ukázat, že zdrojová data jsou špatně (např. formáty)
    - Audit dat
        - Kontrola
    - Bezpečnost (např. otázky dopravy)
    - Zálohování a obnova
- Nástroje pro BI
    - Microsoft (MS SQL Server, Analysis Services, Reporting Services), Oracle, Sybase IQ, IBM DB2, DB2 OLAP Server, Microstrategy, SPSS, SAS, SAP – Business Warehouse
- Klientské nástroje
    - ProClarity, Oracle Discoverer, MS Excel 2000, Business Objects, Cognos: PowerPlay, Impromptu, Brio: Brio Query, Quadbase – EspressReport
- Přenos dat = budování/vytvoření DW
    - Iniciální nahrávání (velký třesk) = všechna data do čistého skladu
    - Inkrementální nahrávání – promítnutí změn v DB do DW
    - Přepis dat – kompletní smazání obsahu skladu a nahrání aktuálních dat
    - Nahrání (loading), přidání (append), destruktivní sloučení, konstruktivní sloučení

Logický a fyzický model datového skladu

- DW – multidimenzionální DB
- logická úroveň = hyperkostka
- fyzická úroveň = multikostka
- multidimenzionální databáze (hyperkostka) je m-rozměrný prostor dimenzí, nad nímž existuje n-rozměrný prostor faktů. Nad nimi mohou existovat k-rozměrné prostory agregovaných dimenzí



-
- Hyperkostka
    - Každá kombinace dimenzí nemusí být v kostce zastoupena
    - Stanovení řídkosti by mělo být součástí dimenzionálního modelování
    - Z důvodu úspory (paměti) se chápe jako logický model
    - Implementují se jen „obsazené“ části
- Multikostka
    - Podprostor hyperkostky, ve kterém nejsou neexistující kombinace dimenzí
    - Hyperkostka může být sjednocení multikostek (obal)
    - Hyperkostka je logicky i implementačně jednodušší, multikostka je implementačně efektivnější
        - **blokové** multikostky – fakty i dimenze včetně času tvoří rozměr kostky,
        - **sériové** multikostky – pro každý fakt existuje samostatná kostka se všemi dimenzemi, fakty tak tvoří dimenzionální řady (pro dimenzi časovou - časové řady).
- Při implementaci DW se používají hyperkostky i multikostky
- ROLAP (relační OLAP) = hvězd, souhvězdí, sněhové vločky
    - MOLAP = sumarizovaná data
    - ROLAP = SQL engine + granulární data
- Hvězda + souhvězdí

- Sněhová vločka + sněžení
- Konceptální modelování
    - obsah DS – entity, atributy, ERD pro základní typy entit a typy vztahů
    - rozhodnutí o centrálním DW, tržištích a komunikaci mezi nimi
    - přenos ze zdrojů, filtrace, transformace, odvození
- Dimenzionální modelování
    - Rozdělení atributů na dimenze, fakta, atributy
    - Definování typů vztahů tvořících základ tabulek faktů (ERD)
    - Definování hierarchie dimenzí
    - Určení aditiv faktů, definice omezení dotazů

14. SQL/MM – multimediální databáze

- **Otázka: SQL/MM – multimediální databáze – (základní rámec normy ISO/IEC 13249), příklady dat–problémy dotazování – relevance odpovědi, přesnost a úplnost.**

ISO/IEC 13249

- Mezinárodní norma
- připravena technickou komisí ISO/IEC JTC 1 Informační technologie, subkomisí SC 32, Počítačové aplikace.
- sestává z následujících částí pod obecným názvem *Informační technologie - Databázové jazyky - Knihovny SQL pro multimédia a další použití:*
    - *Část 1: Základní rámec*
    - *Část 2: Plnotextová data*
    - *Část 3: Prostorová data*
    - *Část 5: Statický obraz*
- Ostatní části této normy specifikují požadavky a všechny jsou závislé na různých částech ISO/IEC 9075 a rovněž na této části ISO/IEC 13249.

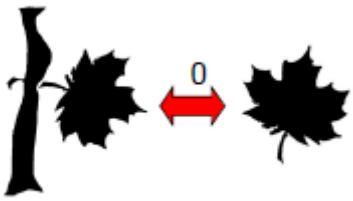
Multimediální databáze

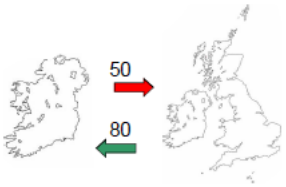
- Motivace
    - klasické (relační, objektové) databáze
        - pevně daná struktura i sémantika (schéma databáze, tj. typované atributy, tabulky, integritní omezení, funkční závislosti, dědičnost, atd.)
        - „umělá“ povaha dat (člověkem vytvářené atributy a jednoznačně interpretovatelné atributy)
        - víme co hledáme = stačí dotazy na úplnou shodu
    - multimediální databáze
        - kolekce obrázků, audia, videa, časových řad, textů, XML, atd.
        - obecně kolekce nestrukturovaných dat (dokument)
        - vnitřní struktura i sémantika je skrytá a nejednoznačná
            - závislá na aplikaci, datech, i subjektivitě uživatele
        - „analogová“ povaha dat (digitalizace signálů/senzorových dat)
        - nevíme pořádně co hledáme ani jak se ptát = nestačí dotazy na úplnou shodu
- Příklady multimediálních dat
    - obrazové databáze
        - biometrické databáze (otisky prstů, oční duhovky, obličejové rysy)
        - medicínské snímky (rentgen, tomografie, ultrazvuk, atd.)
        - satelitní snímky, meteorologický radar
        - snímky materiálových řezů
        - heterogenní kolekce (web)
    - video kolekce
        - TV zpravodajství
        - filmové kolekce, domácí video
        - záznamy z bezpečnostních kamer (letiště, supermarkety, centra měst, atd.)
        - „netradiční“ sekvence (medicínské, průmyslové, atd.)
    - geometrické kolekce
        - CAD modely
        - opět biometrické databáze
        - geografická, kartografická a GIS data

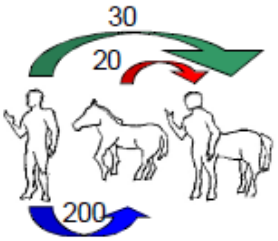
- časové řady, audio, (obecně diskrétní signály)
  - vývoj kurzů akcií, měn, atd.
  - medicínská data - EEG, EKG, atd.
  - řeč (obecně zvuk)
- biologické databáze
  - chemické látky (molekuly, sloučeniny, atd.)
  - sekvence DNA, bílkovin
- melodie
  - notové partitury
  - MIDI soubory
- text, hyper-text
  - digitální knihovny, archivy, e-mail
  - web
- „document-centric“ XML data, semi-strukturovaná data

- Dotazy
  - klasické (relační, objektové) databáze
    - dotaz lze jednoduše formulovat, např. pomocí SQL
    - dotaz na úplnou shodu přesně určuje jak vypadá plně relevantní a plně nerelevantní možný výstup
    - výsledek dotazu není dále strukturován (všechno je stejně relevantní)
    - propracované přístupové metody = rychlé vykonávání dotazu
  - multimediální databáze
    - jak vůbec formulovat dotaz?
    - jak dopředu kvantifikovat co pro mně (ještě) je a co (už) není relevantní?
    - co je to vlastně relevance dokumentu k dotazu?
    - jak dotaz provést efektivně (rychle)?
- Metody řešení
  - První část (obecné aspekty, architektury, modelování):
    - struktura MDB systémů, modality vyhledávání, dotazy na podobnost
    - extrakce vlastností, míry podobnosti, kvalita a rychlost vyhledávání
    - mapování a redukce dimenze
    - aplikace, ukázky existujících systémů
    - (důraz na kvalitu vyhledávání)
  - Druhá část (implementace, indexování):
    - metrické přístupové metody (MAM) vs. Prostorové přístupové metody (SAM)
    - principy indexování pomocí MAM
    - statické MAM, dynamické MAM
    - přibližné a pravděpodobnostní vyhledávání
    - ostatní...
- typy MDB systémů
  - text-based retrieval systémy
    - vyhledávání pouze podle textové anotace (meta-informace)
      - automatické anotování (např. images.google.com využívá textu na stránce, kde je na obrázek odkaz, případně název souboru obrázku)
      - ruční anotace – většinou kvalitnější, anotuje expert, který ví, jak anotovat
    - dotazy podobně jako u fulltextových vyhledávačů, tj. množina klíčových slov
    - výhoda – využití stávající implementace fulltextových vyhledávačů
    - nevýhody
      - nelze aplikovat na neanotované kolekce, ruční anotování je drahé
      - anotace je vždy nějak nepřesná (subjektivní, neúplná, zavádějící, atd.)
      - získané dokumenty můžou být úplně irelevantní
      - nezískali jsme dokumenty, které jsou relevantní - „netrefili“ jsme se do anotace
  - content-based retrieval systémy
    - vyhledávání pouze podle obsahu
      - různé metody popisu obsahu
    - výhody

                - vyhledávání podle skutečného obsahu
                - nezávislost na anotaci,
            - nevýhody – mnoho různých metod modelování struktury a sémantiky obsahu, kterou vybrat?
        - hybridní systémy
            - kombinují výše zmíněné dva
- Modality vyhledávání
    - dotazování (querying)
        - dotaz v kontextu dokumentu
            - dokument chápán jako databáze, kde hledáme dílčí fragment
            - rozpoznávání/analýza obrazu, vyhledávání v DNA sekvencích, řetězcích, apod.
        - dotaz v kontextu kolekce
            - celý dokument představuje sémantickou jednotku
            - databázový přístup
    - prohlížení (browsing)
        - navigace v celé kolekci
            - hierarchická struktura kolekce
            - okolí (web, ontologie)
        - vhodné pro interaktivní hledání formou zpřesňování
- Dotazování podle podobnosti
    - query-by-example typy dotazů
        - ptáme se přímo nějakým dokumentem (ať dokumentem z databáze ve které hledáme, nebo z jiným)
        - navíc specifikujeme rozsah dotazu nebo výsledku
            - bodový dotaz
            - rozsahový dotaz – práh r
            - k nejbližších sousedů – k reverzních k nejbližších sousedů – k
- Kritika metrických vlastností
    - reflexivita: objekt nemusí být sám sobě podobný
    - pozitivita: objekt je maximálně podobný (totožný) jinému objektu
    - symetrie: objekt 1 je podobný objektu 2 jinak, než je tomu naopak (záleží na směru porovnávání)
    - trojúhelníková nerovnost - obecně neplatí tranzitivita







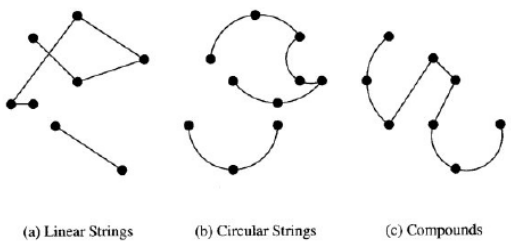
    -
- Kvalita vyhledávání vs. Efektivita vyhledávání
    - kvalita vyhledávání (retrieval effectiveness) je úspěšnost vyhledání dokumentů vzhledem k očekávání uživatele
        - vždy subjektivní, nelze dosáhnout dokonalosti
        - měření na základě subjektivně ohodnocené kolekce
        - nejčastěji přesnost P = |RelOdp|/|Odp| a úplnost R = |RelOdp|/|Rel|
    - rychlost vyhledávání (retrieval efficiency) ovlivňuje reálnou použitelnost a škálovatelnost
        - I/O operace, množství výpočtů podobností/vzdáleností, ostatní CPU náklady
        - potřeba speciálních přístupových metod, resp. indexování, sekvenční průchod je u velkých databází nereálný
- SQL/MM (1999-2002)
    - Jde o řadu UDT a UDF dle SQL:1999
    - Okruhy působnosti
        - Full-textová data
        - Prostorová data
        - Obrázky (statické i videa)

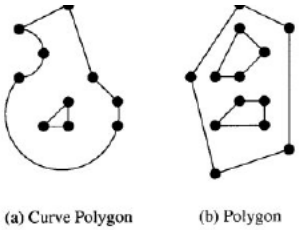
15. Prostorové databáze

- **Otázka: 15. Prostorové databáze, modelování prostorových dat, možnosti dotazování, oblasti využití.**
- Databáze, ve kterých jsou kombinována formátovaná data a prostorová data
- Příklady
    - Geografické informační systémy (GIS)
        - Plánování výstavby měst
        - Řízení dopravy
        - Mapovaní nalezišť nerostných surovin
        - Distribuční sítě
        - Bankovnictví
        - Pojišťovnictví
    - CAD/CAM (Computer-aided design and manufacturing)
        - Návrh integrovaných obvodů
        - Návrh mostu
        - Návrh motoru
- Potřebujeme vhodný dotazovací jazyk
    - Použití standardních dotazovacích jazyků je nevhodné
        - Neposkytují žádnou podporu pro dotazování nad prostorovými daty
- Jazyk
    - Nevyvíjet zcela nový dotazovací jazyk, ale využít jazyka SQL
    - SQL je databázový standard
    - V praxi se často vyskytují prostorová data společně s lexikálními daty
    - Požadavky
        - Musí umožnit formulování dotazů následujících typů
            - Dotazy výhradně na prostorové vlastnosti (Najdi všechna města rozdělená řekou)
            - Dotazy pouze na neprostorové vlastnosti (Kolik lidí žije v Ostravě?)
            - Kombinace předchozích typů (Vypiš jména všech sousedů parcely číslo 15 v … ulici)
- Historie
    - Prostorová data jsou uložena v tabulkách v 1NF
    - Prostorová data jsou uložena jako nestrukturované rozsáhlé binární objekty (BLOB) či jako “opaque types”
    - Prostorová data jsou uložena v relační databázi rozšířené o typy prostorových dat
    - Prostorová data jsou implementovaná pomocí uživatelsky definovaných typů

SQL/MM Spatial

- OpenGIS konsorcium: OpenGIS Simple Features Specification (geometrický model)
    - Geometrický model – polygon, multipolygon, multikřivky,…
- Standard SQL:1999 (objektová orientace)
- Definováno pomocí prostorových typů (Spatial Types)
    - Body = lavička, lampa
    - Křivky = koryto řeky
    - Polygony = půdorys budov
- SQL geometrický model
    - Hierarchický model, zachycuje vztahy mezi prostorovými datovými typy (dědičnost)
    - Prostorové datové typy reprezentují 0, 1 a 2 dimenzionální geometrické útvary
        - Prostorové datové typy jsou založeny na 2D geometrii s lineární interpolací mezi vrcholy
            - 0-dimenzionální objekty = bod, kolekce bodů (souřadnice)
            - 1-dimenzionální objekty = křivky
            - 2-dimenzionální objekty = povrchy
    - Prostorové systémy souřadnic
        - Více druhů
        - Některé berou v potaz zakřivení Země
    - Relace
        - Sousedí (park sousedí se školou)





        - Obsahuje (park obsahuje třešeň)
- Typické dotazy
    - Najdi všechna města vzdálená od Londýna nejvíce 50 km
    - Najdi 5 nejbližších měst k Londýnu
    - Najdi všechny dvojice měst, která jsou do sebe vzdálena nejvýše 200 km
    - Najdi mi všechny budovy, které sousedí s vlakovým nádražím
- Metody definované na prostorových datových typech
    - Rozděleny do čtyř základních kategorií:
        - Konverze prostorových objektů do/z externích datových formátů
            - SQL/MM standard definuje tři implementačně nezávislé externí datové formáty pro reprezentaci prostorových objektů:
                - Textová reprezentace
                - Point(10 10)
                - Well-known text representation (WKT)
                - Binární reprezentace
                - 1000111000
                - Well-known binary representation (WKB)
                - Geography markup language (GML)
        - 

        - Práce s atributy prostorových objektů
            - Všechny prostorové datové typy mají nějaké společné atributy (př. dimenze)
            - Každý podtyp v SQL geometrickém modelu si přidává další, jemu specifické, atributy
            - Př. Polygon – plocha, kterou zabírá
            - Příklady metod
                - ST_Boundary - Vrací hranici prostorového objektu
                - ST_IsEmpty
                - ST_X - Vrací X-ovou souřadnici bodu
                - ST_Length - Vrací délku křivky
        - Porovnávání prostorových objektů
            - ST_Equals - Test na prostorovou rovnost dvou objektů
            - ST_Intersect, ST_Crosses, ST_Overlaps - Test na prostorový průnik dvou objektů
            - ST_Touches - Test, jestli se objekty dotýkají hranicemi, ale neprotínají se
            - ST_Within, ST_Contains - Test, jestli jeden objekt obsahuje jiný
            - Metody pro porovnávání objektů jsou založeny na binárních prostorových predikátech
            - Všechny výše uvedené metody vracejí 1 pokud je příslušná relace splněna (TRUE), jinak vrací 0
        - Generování nových prostorových objektů
            - Nové objekty mohou být výsledkem provedení nějaké operace na množině stávajících objektů
            - ST_Intersection
            - ST_Union
            - ST_Difference
            - Mohou být získány pomocí nějakého algoritmu provedeného na jeden objekt
                - ST_Buffer („nárazníková“ zóna)
                - ST_ConvexHull (konvexní obal)
                - ST_Envelope
                - Vrací minimální ohraničující obdélník (MOO) k danému objektu