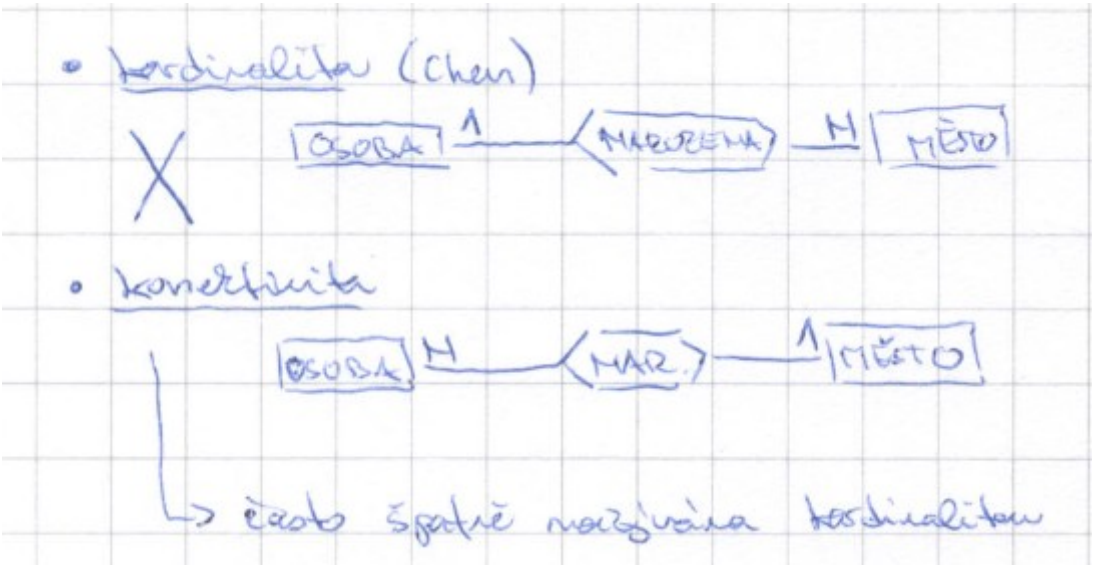
Softwarové systémy - otázka č.06 - A4B33DS - Ondřej Smrž přepsáno od Václav Burda

ZÁKLADY MODELOVÁNÍ DAT

- konceptuální model -> logický model:
  - entitní typ -> tabulka (atribut entitního typu -> sloupec tabulky)
  - vztah 1:1, 1:N -> vztah cizí klíč je na straně N (atribut vztahu na straně N)
  - vztah N:M -> vazební tabulka (attribut vztahu sloupec vazební tabulky)
- kardinalita vs konektivita



- databázový model
  - konceptuální - nezávisí na použité DB technologii
  - logický - závisí na technologii, ale ne na typu DB
  - fyzický - závisí na konkrétní databázi

  Pozn: vzhledem k povaze dnešních relačních databází se příliš nerozlišuje mezi konceptuálním a logickým modelem
- E-R diagram = abstraktní a konceptuální zobrazení dat
  - notace Crows-foot
  - tvoří relační model databáze - tabulky a sloupce o n řádcích
- Normálové formy = omezení duplikování stejných dat (redudance)
  - 1.NF

    - atomické atributy(dále nedělitelné)
    - přístup k řádku podle obsahu (klíčových) atributů
    - jedinečné řádky
  - 2.NF
    - je v 1.NF
    - žádný z neklíčových atributů není parciálně funkčně závislý na klíčovém atributu

    Příklad: relace {idStudent, idPredmet, jmenoStudent, semestr} musíme rozdělit do 3 tabulek

    {idStudent, idPredmet}{idStudent,jmenoStudent}{idPredmet, semestr}
  - 3.NF
    - je v 2.NF
    - žádný neklíčový atribut není tranzitivně závislý na klíčovém atributu dané relace

    Příklad: relace {idStudent, jmenoStudent, fakulta, děkan} se musí rozdělit na 2 tabulky, protože děkan je závislý na fakultě a ta je závislá na idStudenta

    tzn. {idStudent, jmenoStudent, fakulta},{fakulta, děkan}
  - jsou i další NF, ale méně používané

- Integritní omezení
  - definují omezení, které musí vkládané nebo updatované řádky splnit (not null, primary key,...)
  - stanovení defaultní hodnoty
  - používáme jen pokud nutné (dráhé zachovávání), standartně definujeme při návrhu databáze

- Referenční omezení
  - co se děje při mazání s cizími klíči (např. ON DELETE CASCADE)
  - cizí klíč nesmí odkazovat na neexistující řádek

ZÁKLADY JAZYKA SQL + DOTAZY

- standart SQL92 + novější revize
- datové typy: integer, small int, numeric(p,s) - celkem p cifer z toho s za desetinnou čárkou, real, double precision, float, char(x) - x znaků, varchar - řetězec o proměné délce, date<yyyy-mm-dd>, time<hh:mm:ss>, timestamp<date time>

- příkazy:
  - CREATE TABLE Zbozi (packId CHAR(4), jmeno CHAR(20) NOT NULL, cena DECIMAL(10,2));
  - DROP TABLE Zbozi;
  - INSERT INTO Zbozi VALUES ('5dfs', 'Pleny', 500.20); INSERT INTO Zbozi (jmeno, cena) VALUES ('Plenky', 300.10);
  - SELECT sloupce FROM tabulky WHERE podminky GROUP BY seskupeni HAVING podminkaSeskupeni ORDER BY sloupec ASC DESC;
    - WHERE cena BETWEEN 200 and 400; WHERE cena >200 AND cena < 400;
    - WHERE cena LIKE '%foo%'; //obsahuje ...foo... i prázdné
    - WHERE cena IS NULL - vrací true když není definován
    - WHERE typ IN ('pivo', 'vino'); WHERE typ='pivo' OR typ='vino';
  - AGREGAČNÍ FUNKCE - slouží k získání "hromadných" údajů o tabulce podle podmínek
    - COUNT(sloupec/*) - vrací počet řádků podle podmínek
    - COUNT(DISTINCT sloupec) - vrací počet různých záznamů
    - SUM(sloupec), AVG(sloupec), MAX(sloupec), MIN(sloupec),...
    - GROUP BY - spojení řádků se stejnými hodnotami : SELECT typ, AVG(cena) FROM tabulka GROUP BY typ;

    - WHERE vs HAVING - WHERE se aplikuje pro výběr řádků, HAVING může být použito pouze s GROUP BY a je aplikováno až po WHERE. Pravidlo: pokud se podmínka vztahuje a agregovanému atributu použijte HAVING
    - JOIN - pro výběr dat z více tabulek. Typicky se spojují přes id. ... WHERE t1.id = t2.id..
      - INNER JOIN - hledá podmínky které jsou splněny. Nutnost stejného pojmenování sloupců SELECT * FROM t1 JOIN t2 USING(id);
      - OUTER JOIN - (LEFT, RIGHT, FULL) - do výsledku se promítne i když neexistuje záznam v protější tabulce (první, druhé, obou)

        - JOIN ON - pokud nemají stejně pojmenované sloupce SELECT * FROM t1 JOIN t2 ON t1.id=t2.idecko
    - UNION - pro sjednocení více příkazu SELECT do jedné tabulky
    - INTERSECT - průnik dotazů. Např. nalézt společné autory knih
    - EXCEPT - rozdíl

TRANSAKCE

- Nutnost zachování konzistence a správnosti databáze. Při chybě vykonávání příkazu musí být dosud provedená část vrácena do původních hodnot, aby se zachovala konzistence dat.
- vlastnosti ACID:
  - Atomicity - transakce je atomická = buď se provede celá nebo vůbec
  - Consistenci - konkrétní transformace stavu, zároveň integritní omezení
  - Isolation - když jsou transakce vykonávány zároveň, výsledek musí být stejný jako kdyby se vykonávaly za sebou
  - Durability - po úspěšném ukončen transakce musí být změny trvalé, i v případě poruchy a zotavování z chyb
- zamykání
  - SLOCK - (shared) - více transakcí si může objekt zamknout pro čtení(před read)
  - XLOCK - (exclusive) - zamyká 1 transakce před writem pro zápis. Nelze zamknout SLOCK.

    při uváznutí je nutné použít ROLLBACK = vrátit systém do původního stavu před nedokončenými transakcemi
- stupně izolace
  - pěkne popsáno zde: stupně izolace
- Transakce je serializovatelná(s vyjímkou fantomů) když:
  - je dobře formulovaná(akce pokryty zámky) a zamyká všechna data která jsou modifikována
  - je dvoufázová = neměla by uvolňovat zámky dříve než se všechny aplikují
  - drží všechny zámky až do COMMIT/ROLLBACK
- Uváznutí transakcí
  - například vzájemné čekání na zámek -> nutné nějak odstranit

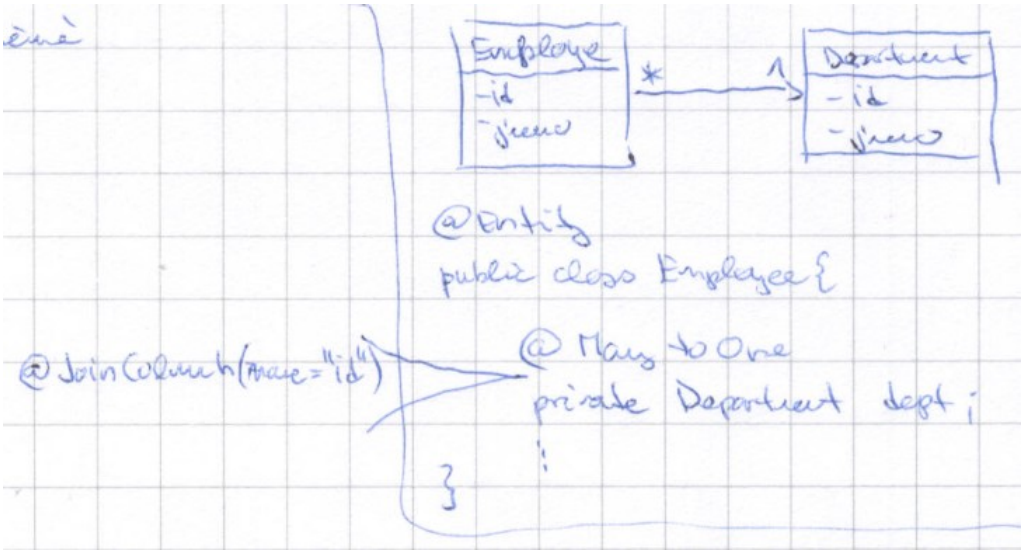
  - v grafu transakcí se projevuje cyklem => odstranění cyklů
    1. přerušovat nejmladší tranksakci (a ovlivnit tak co nejméně dalších)
    2. přerušit transakci s největším počtem zámků
    3. nepřerušovat transakci, která již byla přerušena
    4. přerušit transakci, která se účastní více cyklů

- prevence před fantomy
  - predikátové zámky - náročné, i výrobci DB se jim vyhýbají
  - časové značky
  - MVCC - multiversion concurrency control
    * ze začátku transakce udělá snapshot databáze
    * po skončení se provede commit jen tehdy když updady nejsou u konfliktů s commit, které byly provedeny po snapshotu

OBJEKTOVĚ REALAČNÍ MAPOVÁNÍ, JPA

= převedení objektové struktury dat do relační databáze
Table ~~ Class, row ~~ object, column ~~ property of object

- JPA nezávislé na konretním SQL, používá Factory pattern pro konkrétní převod do databáze
  - vytvoří si entity manager, který umožňuje vytvářet dotazy
    * JPQL(string) - jako klasické SQL
    * Criteria API - hůře čitelné, ale kontroluje správnost dotazu
  - anotace pomocí @:
    * fyzické schéma - @Table, @Column, @JoinTable,...
    * logické schéma - @Entity, @OneToMany, @ManyToMany, id -> @Id


Employee
-id
-jpeno
*
1
Department
-id
-jpeno
@Entity
public class Employee{
@ Many to One
private Department dept;
@JoinColumn(name="id")
}