1. Důvody vzniku DBS, výhody relačních DBS, definice a architektura relačních DBS

Důvody vzniku

- Potřeba uchování dat
- Organizace dat
- Rychlý a snadný přístup k datům (např. kartotéku musel obstarávat člověk)

Definice a architektura relačních DBS

- Model založen na pevném matematickém základu
    - Koncept relací, teorie množin, predikátové logiky
    - Relační algebra
- Uložení objektů/entit a vztahů mezi nimi ve formě relací/tabulek
- Základní konstrukty
    - Relace – pojmenovaná tabulka s řádky a sloupci, podmnožina kartézského součinu
    - Atribut – pojmenovaný sloupec
    - Doména – množina všech přípustných hodnot pro atribut
    - Stupeň n – počet atributů relace
- Řádky tabulky reprezentují vztahy mezi daty
- Integritní omezení
    - Entitní integrita – doména atributu, povinný/nepovinný, jedinečnost, všechny atributy PK musí být povinné (nesmí být NULL)
    - Referenční integrita – vztah mezi FK jedné relace a PK druhé, pokud relace obsahuje FK, musí se jeho hodnota rovnat hodnotě PK jedné z n-tic rodičovské relace nebo být NULL

Výhody relačních DBS

- Existence kvalitních implementací
- Jednoduchost modelu
- Formální základ
- Dotazování jazyky vyšší úrovně – jednoduché, ale dostatečně expresivní
- Konzistence dat – více instancí databáze má celou dobu stejná data
- Jednoduché a rychlé spravování memes kolekce

2. Postup návrhu datové vrstvy shora/zdola, charakteristiky konceptuálního modelu, ER diagram, rozšířený ER diagram, UML diagram, popis konstruktů

Postup návrhu datové vrstvy

- Shora – začínáme nejdříve obecným návrhem a postupně přidáváme další specifika
    - Specifikace požadavků
    - Konceptuální model – ER diagram, nezávislý na typu databáze
    - Relační model – jak bude konceptuální model reprezentován ve zvoleném typu DB, nezávislý na konkrétní implementaci a fyzickém uložení dat, tabulky
    - Fyzický model – jak bude model implementován v konkrétní implementaci
- Zdola – opačný postup než při návrhu shora

Charakteristiky konceptuálního modelu

- Zachycuje požadavky klienta
- Zachycuje doménu problému ve formě entit, vztahů mezi nimi a kladených omezení
- Plně nezávislý na implementačních detailech (DBMS, hardware, programovací jazyky, DB model – relační/hierarchický/síťový/objektový)
- Vhodné využít existující konvence

ER diagram

- Entita
    - Něco, o čem je potřeba uchovávat data
    - Má nezávislou existenci – abstrakce, název pro množinu podobných věcí
    - Instance entity – konkrétní výskyt entity, jednoznačně odlišitelná
- Atribut
    - Vlastnost entity, která nás zajímá a jejíž hodnotu chceme mít v DB uloženou
    - Má datový typ
    - Je proměnlivý (věk) nebo vhodněji stálý (datum narození)
    - Atomický nebo složený (např. adresa = psč, město, ulice)
    - Jednohodnotový, vícehodnotový (telefonní číslo) nebo odvozený (věk)
- Vztah
    - Vazba, asociace mezi entitami (např. Student se hlásí na školu)
    - Stupeň vztahu = počet entit ve vztahu (binární, ternární, rekurzivní)
    - Atribut vztahu (M:N atd.)
    - Instance vztahu (např. Alice se hlásí na TUL)
- Entitní IO
    - Doména atributu, povinnost atributu, jedinečnost, identifikační klíč, primární klíč, složený klíč
- Vztahové IO
    - Kardinalita – počet možných relací pro každou relaci ve vztahu (1:1, 1:M, M:N)
    - Parcialita – popisuje, jestli se všechny entity musí nacházet ve vztahu (totální, parciální)
- Silná/slabá entita
- Silná entita – identifikačně a existenčně nezávislá
- Slabá entita – identifikačně a existenčně závislá na jiné entitě

- Rozšířený ER model – vhodný pro zachycení společných vlastností instancí entit
    - Podtyp obsažen v nadtypu – dědí jeho vlastnosti (vztahy a atributy) + může mít další vlastní

3. Charakteristiky relačního modelu, popis konstruktů, transformace konceptuálního modelu na relační, speciální případy (ternární vztah, rekurze)

Charakteristiky relačního modelu

- Viz otázka č. 1 – definice a architektura relačních DBS

Transformace konceptuálního modelu na relační

- Přizpůsobení
    - Odstranit M:N vztahy
    - Odstranit vztahy a atributy (pomocí vazební entity obsahující atribut)
    - Odstranit vícehodnotové atributy
    - Odstranit n-ární vazby
    - Odstranit redundantní vztahy
- Transformace
    - Silná entita se transformuje na relaci, která obsahuje všechny jednoduché atributy, složené atributy rozložíme na jednoduché složky
    - Slabá entita se na transformuje na relaci, která obsahuje všechny jednoduché atributy entity + FK (PK klíč identifikačního vlastníka/ů)
    - Vztah 1:M – entita na straně „1“ je rodič a entita na straně „M“ je dítě, dítě doplníme o FK
    - Vztah 1:1 – entita, která má nepovinné členství je rodič a entita, která má povinné členství je dítě, dítě doplníme o FK (pokud je u obou členství povinné/nepovinné – můžeme si role vybrat)
    - Rekurzivní vztah – doplníme FK
    - V rozšířeném ERD – podtyp je dítě, nadtyp je rodič
    - Definovat integritní omezení
- Zabezpečení referenční integrity
    - Může být cizí klíč NULL? (povinné členství -> not NULL)
    - Strategie když existuje dítě jehož cizí klíč odkazuje na primární klíč, který má být změněn
        - NO ACTION – nelze smazat, existují na něj odkazy
        - CASCADE – vymaž i záznamy, které na něj odkazují
        - SET NULL – vymaž a odkaz nastav na NULL
        - SET DEFAULT – vymaž a odkaz nastav na defaultní hodnotu
        - NO CHECK – bez kontroly

4. K Čemu slouží funkční závislosti, normalizace, 1.NF, 2.NF, 3.NF

Funkční závislost

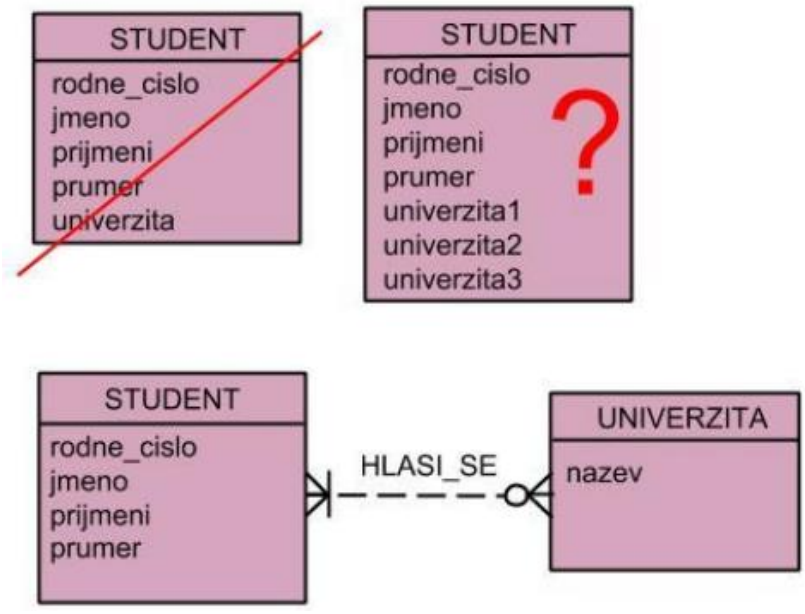
- Popisuje vztah mezi atributy v relaci
- Vychází ze specifikace požadavků
- Musí platit pro všechny n-tice v relaci
- Pravidla:
  - Rozdělení: $A \rightarrow [B_1, \dots, B_n] \Rightarrow A \rightarrow B_1, \dots, A \rightarrow B_n$
  - Spojení: $A \rightarrow B_1, \dots, A \rightarrow B_n \Rightarrow A \rightarrow [B_1, \dots, B_n]$
  - Reflexivnost: $B \subseteq A \Rightarrow A \rightarrow B$
  - Rozšíření: $A \rightarrow B \Rightarrow AC \rightarrow BC$
  - Tranzitivnost: $A \rightarrow B,\ B \rightarrow C \Rightarrow A \rightarrow C$
  - Pokud: $A \rightarrow$ *všechny atributy v relaci* $\Rightarrow A$ *je kandidátní klíč relace*
- Uzávěr atributů A^+ = všechny atributy relace funkčně závislé na A

Normalizace

- Technika dekompozice na základě vlastností dat
- Získáme menší relace, které splňují normální formy
  - Omezují redundanci a nekonzistenci dat
  - Zachycují stejné informace
- Normalizované schéma je snadno udržovatelné, zabírá méně místa

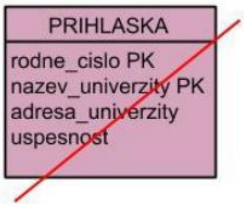
1.NF

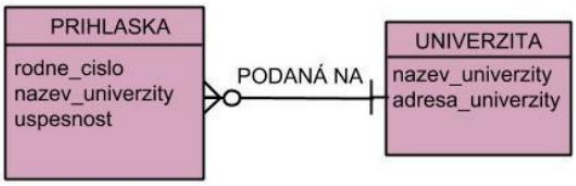
- Každý atribut obsahuje pouze atomické hodnoty
- Relace neobsahuje vícehodnotové atributy
- Všechny n-tice mají stejný počet atributů



2.NF

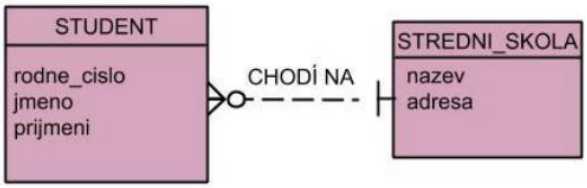
- Relace je v 1. NF a zároveň každý neklíčový atribut je plně závislý na primárním klíči

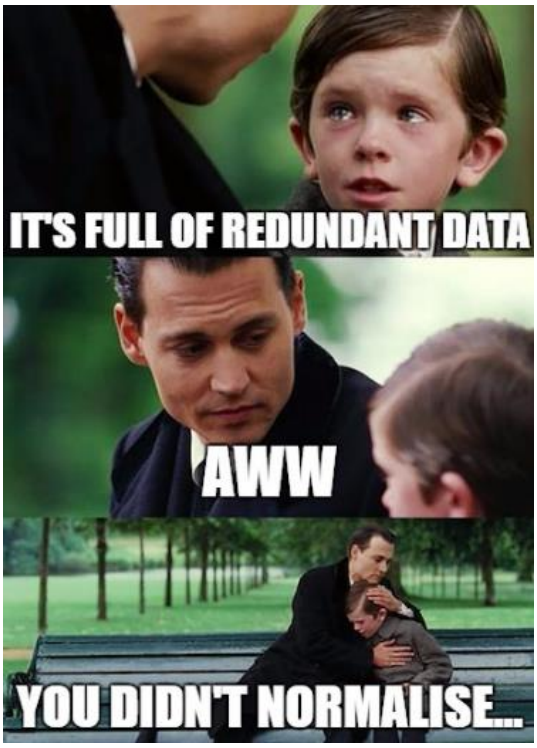




3.NF

- Relace je ve 2. NF a zároveň všechny neklíčové atributy jsou nezávislé, resp. neexistuje tranzitivní funkční závislost na primární klíč





5. Relační alge-bruh, základní operace, odvozené operace relační algebry

Relační algebra

- Formalismus pro dotazování
- Pracuje se s celými relacemi
- Aplikuje se na relace a výsledkem je opět relace
- Výsledná relace může být vstupem do další operace (vnořování)

Základní operace

- Selekce: $\sigma_{podmínka}\ (R)$
  - Výsledek je podmnožina n-tic z R, které splňují danou podmínku
  - Výběr řádků
- Projekce: $\Pi_{seznam\ atributů}(R)$
  - Výsledek je podmnožina z R, která obsahuje pouze atributy ze seznamu a eliminuje duplicitní hodnoty
  - Výběr sloupců
- Kartézský součin: R x S
  - Výsledek je kombinace každé n-tice z R s každou n-ticí z S
  - Obsahuje všechny atributy z R i S, tj. m + n
  - n*s n-tic
- Sjednocení: $R\ \cup S$
  - Výsledek je relace, kde k n-ticím z R jsou přidány n-tice z S a duplicity se eliminují
- Rozdíl: R – S
  - Výsledek je relace, obsahuje jen ty n-tice z R, které se nenacházejí v S

Odvozené operace z RA

- Průnik $R \cap S - (R - S)$
  - Relace musí mít stejný počet sloupců a stejné domény atributů
  - Výsledek je relace obsahující jen ty n-tice, co se nacházejí v R i S
- Theta spojení: $R \bowtie_{podmínka} S = \sigma_{podmínka}(R \times S)$
  - Vrátí podmnožinu n-tic kartézského součinu R x S, které splňují podmínku
- Přirozené spojení: $R \bowtie S = \Pi_{A \cup B}(\sigma_{rovnost\ společných\ atributů}(R \times S))$
  - Spojení přes největší množinu atributů, které se vyskytují v obou relacích
  - Společný atribut se ve výsledné relaci vyskytuje pouze jednou
- Vnější spojení: $R_{outer} \bowtie S$
  - Do výsledné relace se přidají i n-tice jedné z relací, které nemají stejné hodnoty ve společných atributech
  - Levé vnější, pravé vnější
- Přejmenování:
  - $\rho_{R1}(R)$ – přejmenování relace
  - $\rho_{a1,\ a2,\ldots}(R)$ – přejmenované atributy
  - Sjednocení schéma před aplikováním binárních operací, spojení se sebou samým
- Dělení: $R\ \div S$
  - Výsledek je relace obsahující nespolečné atributy C relací R a S a n-tice z R, které odpovídají každé n-tici v S

6. SQL - DDL, referenční integrita

SQL

- Standardizovaný jazyk pro přístup k relačním databázím
- Jazyk pro definici dat DDL (data definition language) a jazyk pro manipulaci s data DML (data manipulation language)
- Neprocedurální jazyk – popisuje, co od databáze chceme, ne jak to provést
- Není case sensitive

SQL – DDL

- CREATE TABLE
  - Vytvoří schéma tabulky
  - Definice tabulky – název, definovaná sloupce, integritní omezení nad sloupcem a tabulkou

```
CREATE TABLE Teachers
(
        id_teacher Integer Identity NOT NULL,
        id_office Integer NOT NULL,
        name Nvarchar(30) NOT NULL,
        surname Nvarchar(30) NOT NULL,
        title_before Nvarchar(10) NULL,
        title_after Nvarchar(10) NULL,
        email Nvarchar(30) NOT NULL UNIQUE,
        dob Datetime NOT NULL,
PRIMARY KEY (id_teacher)
CONSTRAINT fk_TeachOffice FOREIGN KEY (id_office)        // pojmenování IO
REFERENCES Offices(id_office)
CONSTRAINT uc_NameSurnameDob UNIQUE (name,surname,dob) //přes více sloupců
)
```

- SQL identifikátory (názvy tabulek, atributů)
  - Nesmí být delší než 128 znaků (nebo méně)
  - Musí začínat písmenem
  - Nesmí obsahovat mezery
- Datové typy závislé na DBMS (integer, float, character, date, boolean …)
- Integritní omezení sloupce
  - NOT NULL – nesmí být NULL
  - UNIQUE – musí být unikátní
  - PRIMARY KEY – sloupec je PK
  - CHECK – omezení zadané podmínkou
  - DEFAULT – určení implicitní hodnoty sloupce
- ALTER TABLE
  - Modifikace již vytvořené tabulky
  - Přidání, odebrání, změna sloupce
  - Přidání, odebrání IO
- DROP TABLE
  - Vymaže obsah a schéma tabulky z databáze
- DROP BEAT
  - Záleží na konkrétní implementaci (obvykle spustí Darude - Sandstorm)

7. SQL - DML, typy SELECT

- Aktualizační příkazy
    - UPDATE
    - DELETE
    - INSERT
- Příkazy pro výběr dat
    - SELECT

INSERT

- Vložení dat do DB
- `INSERT INTO jméno_tabulky [(jmeno_sloupce)] (VALUES (hodnota) | select výraz)`

UPDATE

- Modifikace dat v DB
- `UPDATE jméno_tabulky SET (jmeno_sloupce = hodnota) [WHERE podmínka]`

DELETE

- Vymazání dat z DB
- DELETE FROM jméno_tabulky [WHERE podmínka]

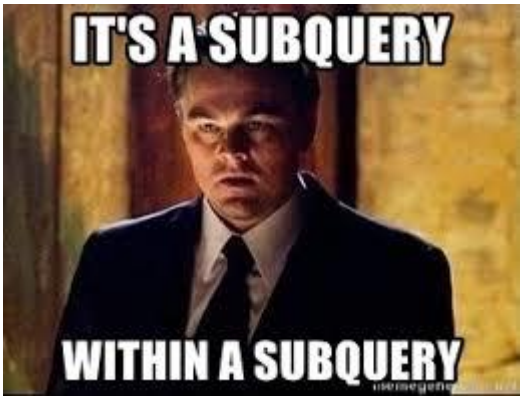
SELECT

- Výběr dat z DB
- Dovoluje duplikáty a NULL

- 3. SELECT $a_1, \ldots, a_n$
- 1. FROM $R_1, \ldots, R_m$
- 2. WHERE podmínka

- DISTINCT – bez duplicit
- ORDER BY – třídění, DESC = sestupně (defaultně ASC = vzestupně)
- LIKE – porovnání řetězců
    - % - libovolný podřetězec
    - _ - libovolný znak
- AS – Matematické operace, vytvoření nového sloupce

atd… zkrátka další selecty vnořený kam se jim zamane



8. K čemu slouží triggery, typy triggerů, porovnání s uloženou procedurou

TRIGGERY

- Procedura, automaticky spuštěná jako reakce na specifickou akci
- Použití: složitější integritní omezení, posun programovací logiky do DB
- Syntaxe silně závislá na DBMS

```
CREATE TRIGGER jméno_triggra
BEFORE | AFTER | INSTEAD OF událost ON jméno_tabulky
[REFERENCING referenční proměnné AS jména]
[FOR EACH ROW]
WHEN (podmínka)
Akce
```

- Granularita
  - `FOR EACH STATEMENT` – trigger je aktivován jednou pro celý příkaz
  - `FOR EACH ROW` – aktivován pro každý řádek, vhodný když je závislost na hodnotách modifikovaných řádků
- Referenční proměnné
  - `old row, new row` – odkazují na modifikovaný řádek
  - `old table, new table` – odkazují na modifikovanou tabulku
- INSTEAD OF trigger
  - Aktivuje se místo nějakého jiného příkazu (např. místo insertu do tabulky T)
- BEFORE trigger
  - Pracuje se stavem DB před provedením dotazu, který trigger spustil
  - Vlastní dotaz bude proveden až po vykonání triggeru
  - Použití: validace dat, automatické generování a doplnění dat pro nové řádky
- AFTER trigger
  - Trigger se aktivuje po vykonání vlastního dotazu

Uložená procedura

- Definice procedury:

```
CREATE [OR REPLACE] PROCEDURE jméno_procedury
[(parameter1 [typ] datový_typ, parameter2 [typ] datový_typ], ...)]
[IS|AS deklarace proměnných]
BEGIN
akce
[EXCEPTION výjimky]
END;
```

- Volání procedury:

```
EXECUTE jméno_procedury(param1, param2, ...)
```

- Odstranění procedury:

```
DROP PROCEDURE jméno_procedury;
```

- Na rozdíl od triggerů, procedury můžeme spouštět nezávisle na jiném příkazu

9. K čemu slouží transakce, vlastnosti transakcí, používané principy

- Transakce – posloupnost SQL příkazů, s kterou se zachází jako s celkem

K čemu slouží

- Nezávislý přístup více uživatelů do DB
- Odolnost DB vůči systémovým poruchám
- Zachovává konzistenci DB, i když může být v průběhu provádění transakce dočasně narušena

Vlastnosti

- Atomicity (atomicita) – transakce se bere jako celek, buď se provede celá nebo vůbec
- Consistency (konzistence) – transakce transformuje DB z jednoho konzistentního tvaru do jiného konzistentního tvaru
- Isolation (izolovanost/nezávislost) – dílčí efekty transakcí nejsou viditelné jiným transakcím – jako by transakce běžela izolovaně
- Durability (trvanlivost) – efekty úspěšné transakce jsou uloženy do DB, logování

Používané principy

- To jsem nehledal, smůla :)

10. K čemu slouží pohledy, problematika modifikace pohledů - v čem je problém a možnosti řešení, k čemu slouží indexy, výhody /nevýhody použití, rozdíl mezi klastrovaným a neklastrovaným indexem

Pohled

- Virtuální relace (tabulka) přizpůsobená specifickým potřebám uživatele
- Důvody použití:
    - Skrytí části dat před uživateli
    - Bezpečnost – definice práv
    - Modularita přístupu k databázi
    - Jednodušší a rychlejší dotazování
- Vytvoření pohledu:
  ```
  CREATE VIEW jmeno_pohledu
  AS vnořený dotaz
  ```
    - Jméno pohledu pak můžeme využívat v dotazech jako každou jinou relaci
- V DB není skutečně uložen výsledek, ale jen předpis

Modifikace pohledů

- Není vždy možná
- Vhodné definovat, co přesně se má stát s relací při modifikacích pohledu – INSTEAD OF trigger
- Nejlépe omezit modifikace pohledů – SQL standard

Index

- Základní mechanismus umožňující zvýšení výkonnosti
- Index obsahuje klíč, složený z jednoho nebo více atributů tabulky a směrník na místo, kde jsou uložena primární data pro danou hodnotu klíče
- Když se má vyhledat nějaký záznam, neprohledávají se sekvenčně všechny záznamy, ale jen datová struktura indexu, která vrátí konkrétní místo na disku, kde je hledaný záznam uložen

Nevýhody indexů

- Vyžadují další místo na disku
- Režie při vytváření indexu
- Údržba indexů, pro změně dat i změně indexů

Klastrovaný index nad atributem A

- Soubor obsahující primární data (relaci), je fyzicky setříděný podle atributu A
- Náročnější údržba (při zápisu setřídit)
- Primární data jsou součástí datové struktury indexu
- Pro jednu tabulku může být definován jen jeden (říkáme mu také **monogamní index**)

Neklastrovaný index nad atributem B

- Soubor obsahující primární data není setříděný podle atributu B. Může být setříděný podle klastrovaného indexu, nebo není setříděný vůbec (halda)
- Primární data nejsou součástí datové struktury indexu, jen směrníky, kde ty data jsou
- Pro jednu tabulku jich může být definovaných více (**polygamní index**)

lol