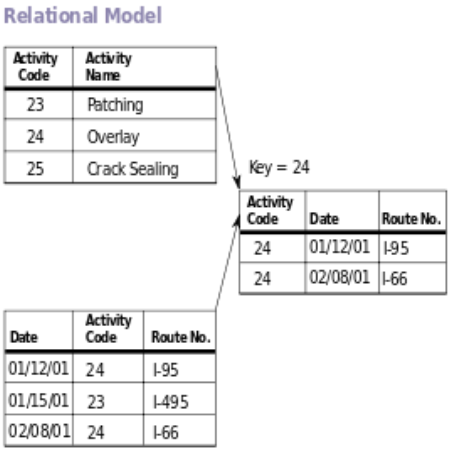
1) Důvod vzniku DBS, výhody relačních DBS, definice a architektura relačních DBS

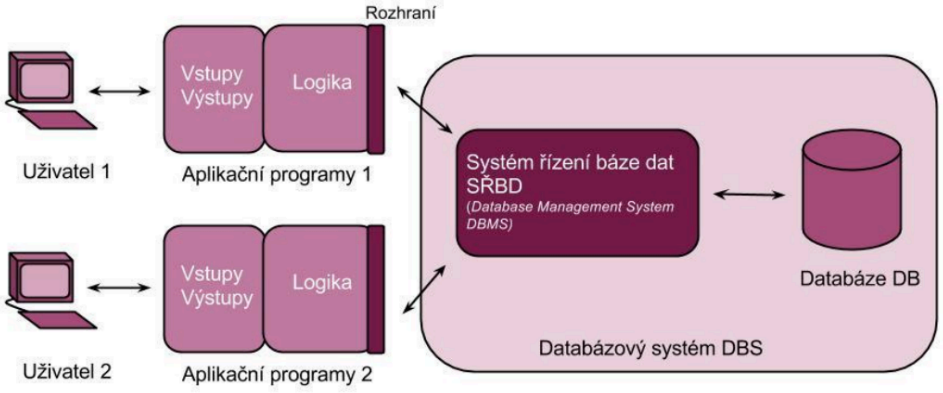
- Databázový systém umožňuje transformovat data na informace
  - Centralizované úložiště
    - řeší izolaci dat a získávání informací
    - definice dat už není součástí aplikace
  - Vytvoření mezivrstvy
    - řeší nezávislost mezi aplikací a daty
    - aplikace a pracuje s mezivrstvou, nezajímá ji jak konkrétně jsou pak data fyzicky uložena
  - Kontrola přístupu mimo aplikaci
    - jednotné zabezpečení integrity dat (integritní omezení)
    - kontrolovaný přístup více uživatelů (transakce)
    - přístup k libovolné podmnožině dat (přístupová práva, pohledy)



- Výhody relačních DBS:
  - Strukturovanost dat
  - Jednoduché vyhledávání (SQL) - dotazování jazyky vyšší úrovně
  - Integrita dat
  - Normalizace (odstranění duplikace a redundance)

- Definice relačních DBS
  - Spravuje data ve formě tabulek s řádky a sloupci
  - Relace (tabulka) - reprezentace vztahu mezi sadou entit (entita,objekt=řádek v tabulce basically, třeba zákazník, produkt, zaměstnanec, objednávka)

- Architektura relačních DBS



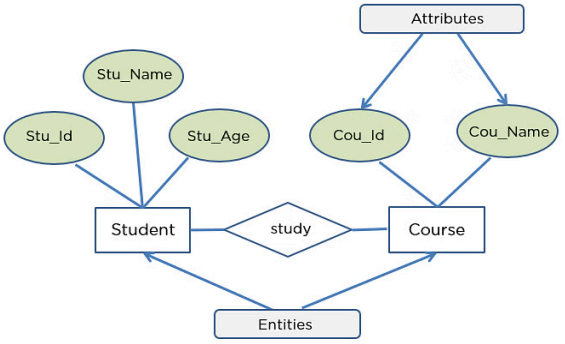
- Databáze DB
  - sdílená kolekce dat popisující aktivity jedné nebo více organizací
- Systém řízení báze dat SŘBD (DBMS)
  - softwarový systém, který umožňuje uživatelům definovat, vytvářet a udržovat databázi a poskytuje k ní kontrolovaný přístup
- Nerelační databáze jsou třeba: JSON, XML

2) Postup návrhu datové vrstvy shora-dolů, charakteristiky konceptuálního modelu, ER diagram, popis konstruktů

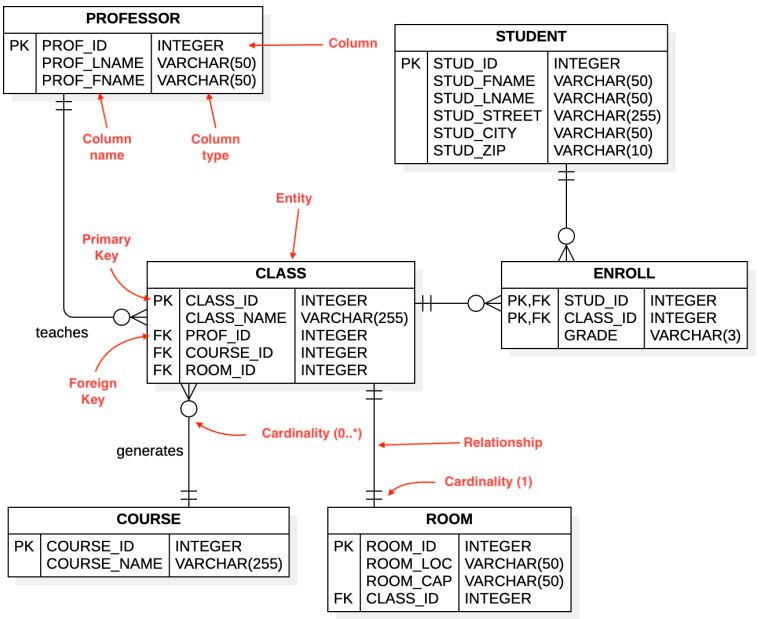
- Postup návrhu datové vrstvy shora-dolů
  - 1) Specifikace požadavků - (potřeby uživatele, požadavky na data)
  - 2) Konceptuální model - (klíčové entity, vztahy mezi nimi) (ER diagram / UML model)
    - ER diagram - (vysvětlen níže)
    - UML model - standardizován, vlastně to co jsme kreslili na zápočtu (jsou tam i podrobnosti)
  - 4) Relační model
    - Nezávislý na konkrétní implementace (Oracle, MS SQL, SQLite)
    - Nezávislý na fyzickém uložení dat
  - 5) Fyzický model
    - Převod do reálné databáze
    - Děláme například indexy, přístupové práva etc.

- Charakteristiky konceptuálního modelu
  - Abstrakce - nejsou tam technické detaily, zaměřuje se jen na klíčové entity a vztahy
  - Nezávislost na technologii

- ER (entity-relationship model) diagram
  - nejvíc basic diagram, může ale bejt i složitější, záleží na autorovi
  - nejjednodušší: jen názvy tabulek(entity), atributy(jméno,cena,věk) a vztahy
  - Identifikace entit, atributů, vztahů mezi entitami
  - Použití klíčů (primárních, cizích) pro propojení tabulek
  - viz. obrázek

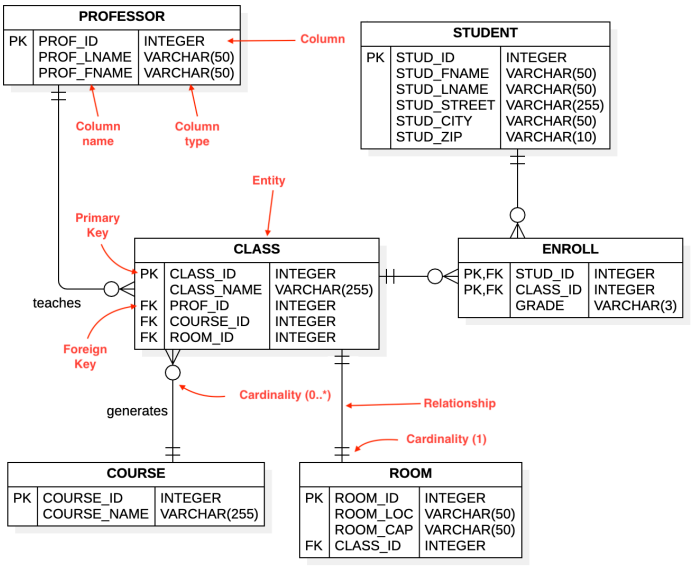


- Popis konstruktů
  - Entita = tabulky
  - Vztah = propojuje entity (určuje kardinalitu, parcialitu)
  - Atribut = vlastnosti entit (sloupečky)
  - Klíč = jedinečně identifikuje záznamy v entitě



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

- Charasterika relačního modelu
  - Tabulky (řádek reprezentuje jeden záznam, sloupec odpovídá atributu)
  - Relace (relace mezi tabulkami je založena na společných hodnotách v klíčích)
  - Normalizace (normalizační pravidla pro odstranění redundance (opakování/přebytečnost informací) a konzistence dat)
  - SQL pro manipulaci a získávání dat



- Konstrukty relačního modelu
  - Relace R = pojmenovaná tabulka s řádky a sloupci
  - Integritní omezení
  - Doména D = množina všech přípustných hodnot pro atribut
  - Atribut A = pojmenovaný sloupec
  - Stupeň n (arita) = počet atributů relace

- Transformace konceptuálního modelu na relační
  - Z entit se stávají tabulky a atributy entit jsou sloupce tabulek
  - Normalizace - rozdělení tabulek a omezení redundance
  - Relační klíče - propojení unikátních klíčů tabulek
  - Vztahy - občas se tabulky rozdělí na několik tabulek

- Speciální případy
  - Ternární vztah
    - Vzácný vztah mezi třemi tabulkami
    - Vytvoříme novou tabulku s cizími klíči a propojíme je, aby tam nebyl ten trojtej vztah
    - Příklad: doktor předepisuje (prescription) několik léků (drug), několika pacientům (pacients) (M:N:M)
  - Rekurze
    - Vztah jedné tabulky k sobě samé
    - Doplnění o cizí klíč (primární klíč jiné entity)
    - Třeba zaměstnanec-nadřízený, entita “zamestnanci” v sobě má “id_nadrizeny”

- Zabezpečení referenční integrity - typy změna / typy DELETE (bylo ve zkoušce)
  - NO ACTION – nemůžeš smazat, pokud 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 závislosti
    - Určují vztahy mezi atributy v tabulkách

- Normalizace
    - Organizace dat, aby byla minimalizována redundance + udržena integrita dat
    - Příklad: rozdělíme tabulky na dvě a více tabulek, minimalizace opakování informací

- 1. NF (1. normální forma)
    - Každá tabulková buňka obsahuje pouze jednu hodnotu (žádné opakování skupiny hodnot v jedné buňce) - neuloží se třeba celá ulice do sloupce “ulice” (např. “Průhonice, Sadová 577” tam nebude), ale bude tam několik sloupečků pro ulici, město, č. p. atd…
    - Všechna data jsou atomická (nedělitelné)
    - Řešení: rozdělení atributu na větší množství

- 2. NF (2. normální forma)
    - Podmínka: Tabulka je v 1. NF
    - Data závisí na celém klíči a nejen na nějaké jeho podmnožině
    - Pokud třeba máme tabulku s výrobci a výrobky tak to rozdělíme na dvě tabulky
    - Data spolu musí souviset
        - Například nevložíme ELO hráčů do tabulky “inventář”

- 3. NF (3. normální forma)
    - Podmínka: Tabulka je v 2. NF
    - Všechny neklíčové atributy jsou navzájem nezávislé (nesmí existovat závislost atributu na neklíčové hodnotě)
    - Rozklad na větší množství relací
    - Nebo například když počítáme “Rank” z “ELO” tak to nebudeme mít v jedné tabulce, ale dáme to dvou (aby to bylo závislé na klíči, nikoli na třeba tom “Rank”u)

5) SQL - DDL, referenční integrita

- SQL - Structured Query Language
- Hromada standardizací
- Komerční systémy provádějí různé implementace - např. nemají všechno, doplňují nestandardní funkce
- Rozdělení SQL
    - jazyk pro definici dat DDL (Data Definition Language)
        - definování DBS schéma - datové typy, struktury
        - vytváření, odstraňování, modifikace definic relací a pohledů (vrituální relace)
        - definování integritních omezení
    - jazyk pro manipulaci s daty DML (Data Manipulation Language)
        - dotazování SELECT, INSERT, UPDATE, DELETE
    - jazyk pro definici přístupových práv DCL (Data Control Language)
        - GRANT, REVOKE
    - jazyk pro řízení transakcí TCL (Transaction Control Language)
        - COMMIT, ROLLBACK
    - jazyk pro spuštění a vložení a dynamické SQL

- SQL DDL
    - CREATE TABLE - vytvoření schématu relace (tabulky)
        - název tabulky, definované sloupce, integritní omezení nad sloupcem, integritní omezení nad tabulkou
        - CREATE TABLE Teachers(
            id_teacher Integer PRIMARY KEY (id_teacher),
            name Nvarchar(30) NOT NULL,
                CONSTRAINT ucNameSurnameDob UNIQUE
            (name,surname,dob) // přes více sloupců
            CONSTRAINT fk_TeachOffice FOREIGN KEY (id_office) REFERENCES Offices(id_office)
        );
    - ALTER TABLE - modifikace již vytvořeného schématu tabulky (ALTER/DROP COLUMN)
    - DROP TABLE - vymazání schématu i dat tabulky
    - TRUNK TABLE - vymazání dat tabulky
    - RENAME - přejmenování již vytvořené tabulky

- IO (integritní omezení / constraints) sloupce
    - NOT NULL - žádná hodnota v daném sloupci nesmí být NULL
    - UNIQUE - všechny hodnoty ve sloupci musí být unikátní
        - je přípustná jedna hodnota NULL, kombinace více sloupců může být UNIQUE
    - PRIMARY KEY - sloupec je primárním klíčem
    - REFERENCES - sloupec je cizím klíčem, definuje referenční integritu vzhledem k jiné tabulce
    - CHECK - IO zadané logickým výrazem
    - DEFAULT - slouží k určení implicitní hodnoty sloupce (NULL nebo hodnota)

- Datové typy
    - ISO: INTEGER,SMALLINT,DECIMAL,FLOAT,DATE,TIME,INTERVAL
    - Nestandardní: MONEY, GEOMETRIC SHAPE

6) SQL - DML, základní a vnořené dotazy. Určení výsledného počtu entic

- DML - Data Manipulation Language

- SELECT
  - Výběr z dat (dotazování)
  - Množinové operace - sjednocení, průnik, rozdíl
  - Vnořené dotazy - WHERE atribut IN / NOT IN (vnořený SELECT atribut…)
  - Agregační funkce - GROUP BY, HAVING
  - Spojení - kartézský součin, JOIN - natural, inner, outer, left, right

- INSERT
  - Vložení dat do DB
  - INSERT INTO jméno_tabulky (jmeno_sloupce) VALUES (hodnota/y)
  - Podporuje to i vložení více řádku naráz (více VALUES oddělené čárkou)

- DELETE
  - Smazání dat z DB
  - DELETE FROM jmeno_tabulky WHERE podminka

- UPDATE
  - modifikace dat v DB
  - UPDATE jmeno_tabulky SET (jmeno_sloupce = hodnota) WHERE podminka
  - UPDATE Studenti SET mesto = ‘Reichenberg’ WHERE mesto = ‘Liberec’

- Určení výsledného počtu entic
  - COUNT() - SELECT COUNT(column) FROM table WHERE condition;
  - COUNT(*) - SELECT COUNT(*) FROM table GROUP BY column;
    - Počítání podle skupiny

7) SQL dotazy - množiny, agregace, spojení. Určení výsledného počtu entic

- Množiny
  - Základní
    - SELECT column FROM table WHERE podmínka;
  - UNION (Sjednocení)
    - SELECT column FROM table1 UNION SELECT column FROM table2;
  - INTERSECT (Průnik)
    - SELECT column FROM table1 INTERSECT SELECT column FROM table2;
  - EXCEPT (Rozdíl)
    - SELECT column FROM table1 EXCEPT SELECT column FROM table2;

- Agregace
  - GROUP BY
    - SELECT column1, COUNT(*) FROM table GROUP BY column1;
  - HAVING (Podmínka pro agregaci)
    - SELECT column1, COUNT(*) FROM table GROUP BY column1 HAVING COUNT(*) > 1;
  - Agregační funkce
    - SELECT AVG(column1), MAX(column2), MIN(column3) FROM table;

- Spojení
  - INNER JOIN (jen společné prvky)
    - SELECT table1.column1, table2.column2 FROM table1 INNER JOIN table2 ON table1.id = table2.id;
  - LEFT JOIN (včetně všeho z levé tabulky)
    - SELECT table1.column1, table2.column2 FROM table1 LEFT JOIN table2 ON table1.id = table2.id;
  - RIGHT JOIN (včetně všeho z pravé tabulky)
    - SELECT table1.column1, table2.column2 FROM table1 RIGHT JOIN table2 ON table1.id = table2.id;
  - FULL JOIN (včetně všeho z obou tabulek)
    - SELECT table1.column1, table2.column2 FROM table1 FULL JOIN table2 ON table1.id = table2.id;

- Určení výsledného počtu entic
  - COUNT
    - SELECT COUNT(*) FROM table;
  - COUNT & GROUP BY
    - SELECT column1, COUNT(*) FROM table GROUP BY column1;
  - Určení unikátních hodnot DISTINCT
    - SELECT COUNT(DISTINCT column1) FROM table;

8) K čemu slouží triggry, typy triggrů, porovnání s uloženou procedurou

- Trigger
  - Procedura uložená na DB serveru, která je automaticky spuštěná jako reakce na specifickou akci v databázi
  - Když nastane “událost” => kontrola podmínky, pokud platí => provedení akce
  - Použití - komplexnější IO, opravy IO
  - BEFORE trigger
    - Před provedením operace
  - INSTEAD OF trigger
    - Nahrazují standardní provedení operace
  - AFTER trigger
    - Po provedení operace

- Událost
  - Změna v DB, třeba INSERT, DELETE, UPDATE OF [seznam_sloupců]

- Podmínka
  - Dotaz, který je proveden, když je trigger spuštěn
  - Když výsledek dotazu je NOT NULL, podmínka platí

- Akce
  - Procedura, která je provedena pokud podmínka platí

- CREATE TRIGGER jméno_triggru BEFORE/AFTER/INSTEAD OF událost ON jméno_tabulky [REFERENCING referenční_proměnné AS jména] [FRO EACH ROW / FOR EACH STATEMENT] WHEN (podmínka) [AS] akce

- Porovnání s uloženou procedurou

Triggery	**Uložené procedury**
○ Automatické spuštění	○ Explicitní volání
○ Sledování změn / ovládání změn / audit / validace dat	○ Manuální komplexní operace
○ Mohou ovlivnit výkon	○ Jednou za čas to asi neuškodí xd

9) K čemu slouží transakce, vlastnosti transakcí, používané principy. (slides)

- Poskytují:
  - nezávislý přístup více uživatelům naráz do DB
  - rezistence DB vůbec systémovým poruchám
- Komponenty
  - souběžné zpracování (každej uživatel vidí konzistentní stavy databáze)
  - zotavení z chyb (error recovery) - systémové poruchy neporuší stav

- Transakce = posloupnost jedné, nebo více SQL operací, se kterou se zachází jako s celkem
- Např. vymazání studenta => vymažem z databáze Studenti + Prihlasky
- Transakční mód = BEGIN TRANSACTION - COMMIT/ROLLBACK
  - můžem se podívat jak se nám příkaz zachová a podle toho vrátit stav nebo to tak nechat

- Vlastností transakcí (ACID)
  - Atomicity
    - Atomicita, transakce se tváří jako celek, provede se celá nebo vůbec
    - Pokud například nastane chyba tak se změny vrátí zpět
  - Consistency
    - Konzistence, transakce transformuje DB z jednoho konz. stavu do dalšího konz. stavu
  - Isolation
    - Izolovanost, nezávislost, efekty nejsou viditelné jiným transakcím, jakoby transakce probíhala izolovaně
    - Zabezpečeno pomocí uzamykacího mechanismu
    - Transakce má trvat rychle a nečekat např. na data od uživatele
  - Durability
    - Trvanlivost, efekty úspěšné transakce jsou uloženy do databáze (logování)
    - Pokud nastane porucha po úspěšné transakci, je garance, že změny vykonané transakcí jsou uložené
    - zabezpečeno pomocí logovacího mechanismu, transakčního žurnálu
- Úrovně izolace
  - Nižší úrovně
    - Lepší víceuživatelský přístup, nižší režie, horší garance konzistence
      - READ UNCOMMITTED (nejhorší izolace, transakce si čtou i uncommited úpravy provedené transakcemi)
        - Vidí a může použít data, které mění jiné transakce před COMMIT
        - Vhodné pouze pro read-only transakce
      - READ COMMITED (transakce si čtou jen commited úpravy, taky nic moc)
        - Nemůže použít data, které mění jiné transakce před COMMIT
      - REPEATEDLY READ (transakce vidí jen jeden OG stav před svým počátkem, už izolace)
        - Jako read committed + nemůžeme použít data, které mění hodnotu, když je použijeme vícekrát, může vzniknout phantom 😱
      - SERIALIZABLE (transakce musí probíhat sériově, nikoli paralelně, tím pádem je to pomalý, ale nemusíme se bát, že si budou navzájem hrabat do dat)
  - Příklady použití
    - SET TRANSACTION READ-ONLY
    - SET TRANSACTION ISOLATION LEVEL -II-

10) K čemu slouží indexy, výhody / nevýhody použití, rozdíl mezi klastrovaným a neklastrovaným indexem. (slides)

- Index
  - Obsahuje klíč vytvořen z jednoho nebo více atributů tabulky (pohledu) + směrník na místo, kde jsou uložena primární data pro danou hodnotu klíče
  - Nemusí se sekvenčně prohledávat všechny záznamy, ale jen datová struktura indexu => ta vrátí konkrétní místo na disku
- Datové struktury pro indexy
  - Vyvážené stromové struktury (B(inary) tree, B+ tree)
  - Hash funkce

- Výhody indexování
  - Zrychlení vyhledávání

- Nevýhody indexování
  - Vyžaduje další úložný prostor na disku
  - Režie při vytváření indexu
  - Údržba indexů - změna dat / změna indexů

- Musíme balancovat mezi rychlostí dotazů a cenou aktualizací indexů
- Chybějící indexy / přeindexovanost => zhoršení výkonnosti

- Jak indexovat
  - Malé množství dat => není potřeba indexovat
  - Často upravované tabulky => miň indexů, indexy nad méně atributy (úzké indexy)
  - Málo upravované a často čtené tabulky => indexy nad více atributy (široké indexy)
  - Často upravované & často čtené => musíš to zkoušet lol a najít balanc

- Typy indexů
  - Úzký - definovaný nad jedním atributem
  - Široký - definovaný nad více atributy
  - Hustý - odkazuje na konkrétní záznam (řádek)
  - Řídký - odkazuje na stránku, na které je záznam
  - Primární - definovaný nad atributem/atributy (včetně primárního klíče)
  - Sekundární - neobsahuje primární index
  - Jedinečný - definovaný and atributem UNIQUE

- Typy indexů
  - Klastrovaný index (clustered) nad atributem A
    - Data jsou fyzicky uspořádána podle klíče indexu
    - vhodné pro rozsahové dotazy
    - při zápisu dat se musí setřídit (náročnější údržba)

  - Neklastrovaný index (unclustered) nad atributem B
    - Data zůstávají nesetříděná nebo může bejt setříděnej klastrovaným indexem (halda)
    - Primární data nejsou součástí datové struktury indexu, jenom směrníky, kde ty data jsou
    - Pro jednu tabulku můžeme mít víc neklastrovaných indexů