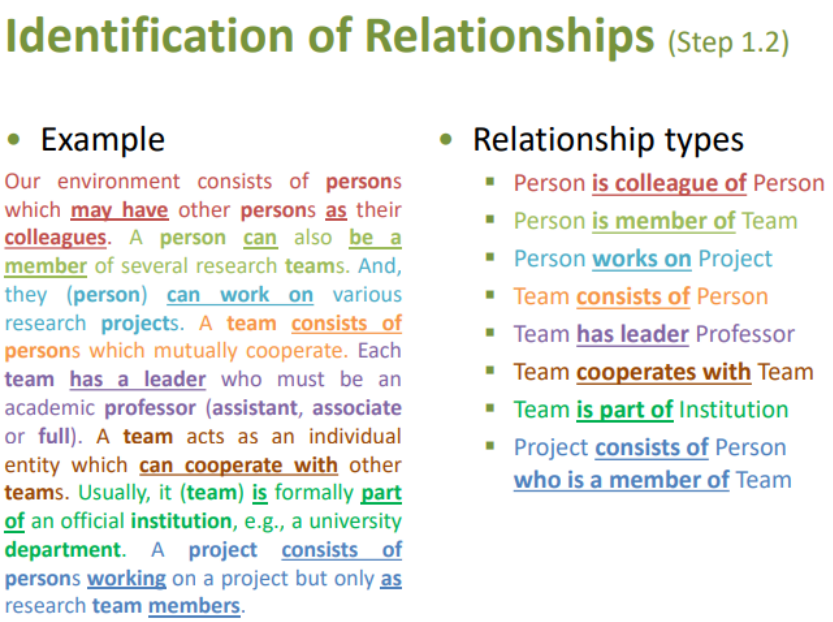
DBS

Konceptuální modelování. ER (entitní typy, atributy, identifikátory, vztahové typy, n-ární, rekurzivní, slabé entitní typy, ISA hierarchie). Logické modely (tabulkový, objektový, stromový, grafový). Relační model (definice, klíč, nadklíč, cizí klíč), transformace ER schématu do relačního schématu.

Konceptuální modelování.

Konceptuální model - mapování skutečnosti na databázovou reprezentaci, používá se UML a ER diagramů
Proces:

1. Analýza požadavků stakeholderů - typ entit, typ vztahů, atributy



2. Modelování - volba jazyka (UML, ER, Object Constraints Language (OCL), Object-Role Model (ORM), Web Ontology Language (OWL), predikátová logika), konceptuální schéma, diagram
3. Iterativní adaptování

ER (entitní typy, atributy, identifikátory, vztahové typy, n-ární, rekurzivní, slabé entitní typy, ISA hierarchie).

ER – Entity-Relationship Model
Nemá standardy -> různé notace
Méně využíván než UML

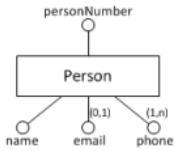
Entitní typy

- Entita obecně osoba, projekt atd.,
  =class v UML
- Graficky obdélník v diagramu
- Rozdělení:
  - Silný entitní typ - má alespoň jeden plný identifikátor
  - Slabý entitní typ - nemá plný identifikátor, má jeden nebo více částečných id. -> závislý na jiné entitě

Atributy

- Vlastnost entity

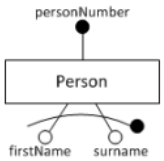
- Mají název a kardinalitu



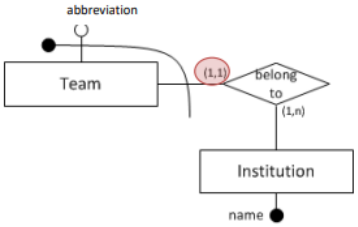
- Složené atributy - atribut má subatributy (např. adresa se skládá ze subatributů ulice, města a země)
- Datové typy

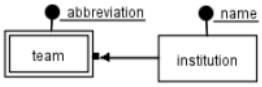
Identifikátory

- Atribut nebo skupina atributů, které jednoznačně určují entitu
- Rozdělení
  - Plné - atribut nebo skupina atributů jedné entity



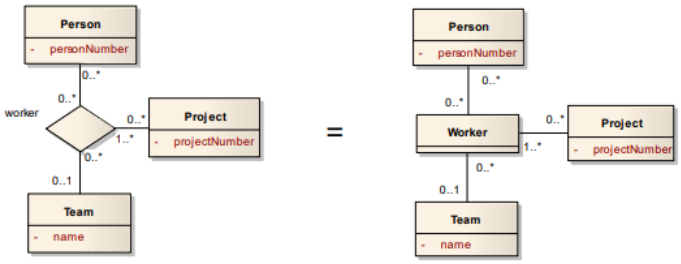
  - Částečné - pomocí vazby na jinou entitu, dovoluje pouze kardinalitu (1,1)



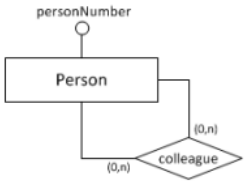


Vztahové typy

- Mají název, entity a kardinalitu
- Binární - 2 entity
- N-ární - více než 2 entity, mohou být nahrazeny třídou, která má binární vztahy se všemi entitami ve vztahu

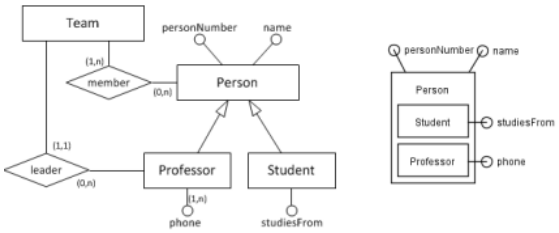


- Rekurzivní - typ vztahu mezi entitami stejného entitního typu

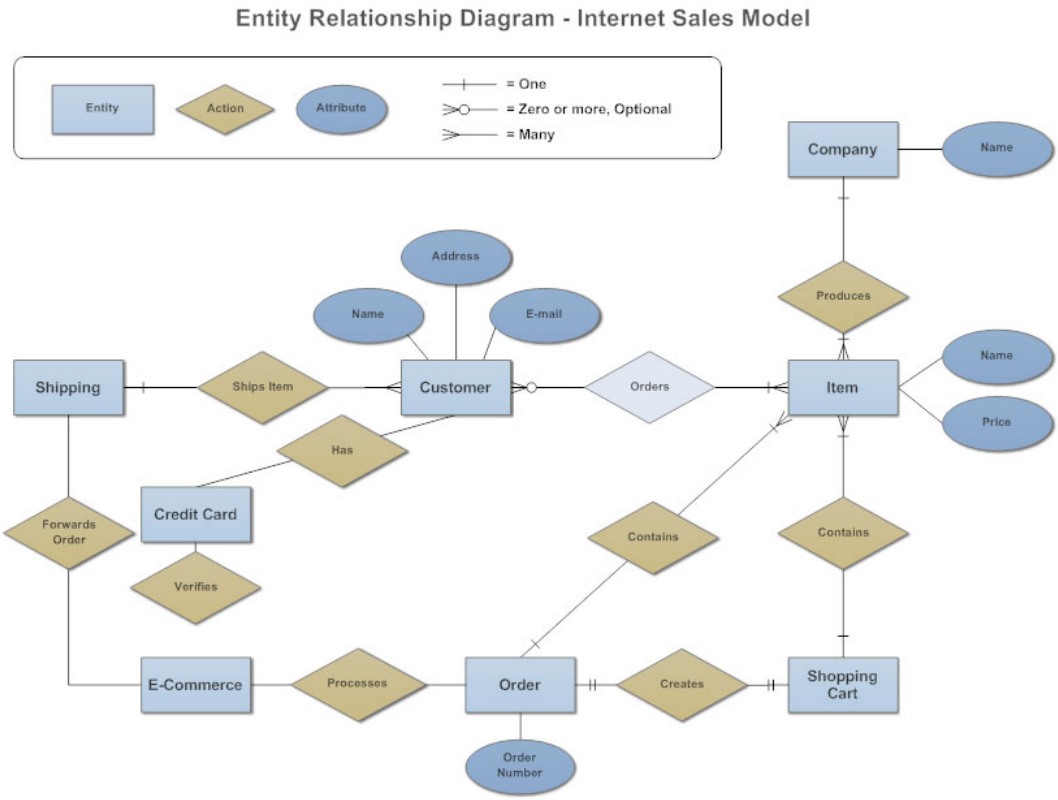


ISA hierarchie

- Specifický vztah bez názvu a kardinalit
- Generalizace / specializace (parent - child)
- Typ entity, která je specializací jiné entity - např. entita student je specializace entity osoba a dědí její atributy plus má svoje vlastní atributy
- Child může mít max jednoho parenta



- Možná omezení:
- Covering constraint (complete/partial) - entita person musí být jedna nebo více ze specifických entit
- Disjointness constraint (exclusive/overlapping) - každá entita může být jen jedním specifickým typem

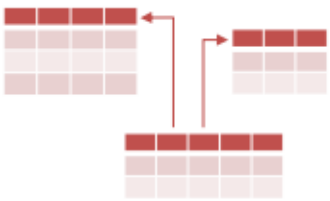


Logické modely (tabulkový, objektový, stromový, grafový).

logický model - specifikuje jak bude konceptuální model vypadat v databázi

Tabulkový

- Struktura - řádky pro entity a sloupce pro atributy
- Operace - selection, projection, join
- Např. relační model, SQL, Oracle, MySQL



Objektový

- Struktura - objekty s atributy, ukazatele mezi obejtky
- Operace - navigation

- Object-oriented programming, zapouzdření, dědičnost

Stromový

- Struktura - vrcholy s atributy, hrany mezi vrcholy
- Hierarchie, kategorizace, semi-strukturovaná data
- Např. hierarchický model, XML dokumenty, JSON dokumenty

Grafový

- Struktura - vrcholy, hrany, atributy
- Operace - Obcházení, porovnávání vzorů, grafové algoritmy
- Např. síťový model, Resource Description Framework (RDF), Neo4j, InfiniteGraph, OrientDB, FlockDB, …

Proces logického modelování

1. Výběr modelu podle charakteristik dat, možností dotazování, použití, požadavky
2. Tvorba logického schématu - transformace z konceptuálního schématu do logického

Relační model (definice, klíč, nadklíč, cizí klíč)

- Tabulkový logický model
- Skládá se z relačního schématu a atributů
- Relační schéma - struktura bez dat, jména atributů a jejich typy (záhlaví tabulky)
- Relace = tabulky - dvourozměrné struktury tvořené záhlavím a tělem, **množina**
- Položky tabulky jsou n-tice (řádky v tabulce)
- Relace vs tabulka - tabulka není množina, proto může obsahovat duplicitní řádky a řádky mohou být seřazeny

Klíče

- **nadklíč :** množina jednoho nebo více atributů, jejichž hodnoty jednoznačně určují entitu (tedy klíč je podmnožina atributů – např. všechny atributy).
- **kandidátní klíč :** nadklíč s minimálním počtem atributů - žádný atribut nemůže být odstraněn bez porušení možnosti identifikace
  - může se stát primárním klíčem, pokud ne, je označován jako alternativní klíč
  - jednoduchý
  - složený
- **primární klíč** - každá tabulka může mít pouze jeden, nesmí se zde vyskytovat hodnota NULL, hlavní a jednoznačný způsob identifikace
- **cizí klíč :** je atribut, který koresponduje s primárním klíčem v jiné relaci (tabulce). Hodnotami cizího klíče v referencující (odkazující) relaci smí být jen ty hodnoty, které se vyskytují jako primární klíč v relaci referencované (odkazované).

Objektově-relační mapování (ORM)

Objektově-relační mapování je programovací technika v softwarovém inženýrství, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem.
Hlavním cílem ORM je synchronizace mezi používanými objekty v aplikaci a jejich reprezentací v databázovém systému tak, aby byla zajištěna persistence dat.
Řada implementací ORM se snaží v co největší míře odstínit vývojáře od nutnosti psaní SQL dotazů a pro selekci objektů z databáze používá raději objektový přístup. Takovýto postup však zpravidla umožňuje vyhledávat objekty jen podle databázového primárního klíče, což zpravidla nestačí. Proto některé implementace ORM využívají pro selekci objektů objektový dotazovací jazyk. Jedna z výhod odstínění od práce s SQL může být i určitá nezávislost aplikace na konkrétním databázovém systému, resp. možnost zvolit databázový systém či jiné datové úložiště tak, aby vyhovovalo konkrétním podmínkám a požadavkům. Nezávislost na konkrétním databázovém systému a skrývání SQL dotazů jsou však již jen příjemné důsledky použití ORM, není to ale primárním cílem.
Pesistence objektů

Java Persistence API (JPA) je framework programovacího jazyka Java, který umožňuje objektově relační mapování (ORM). To usnadňuje práci s ukládáním objektů do databáze a naopak. Je určen jak pro Java SE, tak pro Java EE.

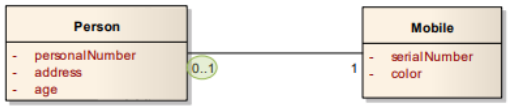
Transformace ER schématu do relačního schématu.

- Převedení entitních typů s atributy do relačních schémat (relací s atributy)
  - přidání umělých numerických identifikátorů ve formě ID - automaticky generováno
- Vícehodnotový atribut (entita ho může mít 1,N) - vytvoření separátní tabulky
  - Např osoba může mít více telefonních čísel - vytvoření tabulky s telefonním číslem a cizím klíčem na personalNumber

    Person(personalNumber)
    Phone(personalNumber, phone)
    Phone.personalNumber ⊆ Person.personalNumber
- Složené atributy - vytvoření separátní tabulky, nebo v případě (1,1) kardinality možné ponechat sub-atributy jako atributy dané entity
- Binární vztahy
  - (1,1):(1,1)
    - 3 tabulky - 2 pro entity a jedna pro vztah (např. ownership) - základní přístup

      Person(personalNumber, address, age)
      Mobile(serialNumber, color)
      Ownership(personalNumber, serialNumber)
      Ownership.personalNumber ⊆ Person.personalNumber
      Ownership.serialNumber ⊆ Mobile.serialNumber
    - 1 tabulka se všemi atributy obou entit

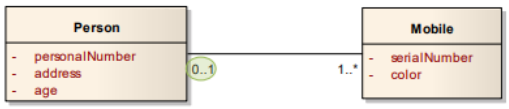
      Person(personalNumber, address, age, serialNumber, color)
  - (1,1):(0,1)
    - 2 tabulky - 1 tabulka nelze, protože jedna z entit nemusí mít vztah s druhou

      - Person(personalNumber, address, age, serialNumber)
        Person.serialNumber ⊆ Mobile.serialNumber
        Mobile(serialNumber, color)
  - (0,1):(0,1)
    - 3 tabulky - 2 pro entity a jedna pro vztah
  - (1,n)/(0,n):(1,1)
    - 2 tabulky

      Person(personalNumber, address, age)
      Mobile(serialNumber, color, personalNumber)
      Mobile.personalNumber ⊆ Person.personalNumber
      Why a personal number is not a key in the Mobile table?
      - Because a person can own more mobile phones
  - (1,n)/(0,n):(0,1)
    - 3 tabulky



- Person(personalNumber, address, age)
  Mobile(serialNumber, color)
  Ownership(personalNumber, serialNumber)
  Ownership.personalNumber ⊆ Person.personalNumber
  Ownership.serialNumber ⊆ Mobile.serialNumber
- Why a personal number is not a key in the Ownership table?
  - Because a person can own more mobile phones

- (1,n)/(0,n):(1,n)/(0,n)
  - 3 tabulky

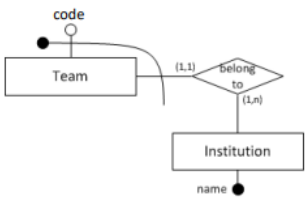
- Person(personalNumber, address, age)
  Mobile(serialNumber, color)
  Ownership(personalNumber, serialNumber)
  Ownership.personalNumber ⊆ Person.personalNumber
  Ownership.serialNumber ⊆ Mobile.serialNumber
- Note that there is a composite key in the Ownership table

- Atributy vztahů -> atributy v tabulkách daného vztahu
- N-ární vztahy - N tabulek pro entity + 1 tabulka vztahu
- ISA hierarchie -
  - tabulka pro každou specializaci pouze s jejími specifickými atributy + tabulka pro generalizaci

    Person(personalNumber, name)
    Professor(personalNumber, phone)
    Student(personalNumber, studiesFrom)
    Professor.personalNumber ⊆ Person.personalNumber
    Student.personalNumber ⊆ Person.personalNumber
  - tabulky pouze pro každou specializaci

    Professor(personalNumber, name, phone)
    Student(personalNumber, name, studiesFrom)
  - pouze jedna tabulka, ve které jsou specializace odlišeny umělým atributem (ne vždy vhodné)

    Person(personalNumber, name, phone, studiesFrom, type)
- Slabé entitní typy



Institution(name)
Team(code, name)
Team.name ⊆ Institution.name

Relační databáze (datový model, NULL hodnoty, tříhodnotová logika), integritní omezení (NOT NULL, UNIQUE, PRIMARY KEY, CHECK, FOREIGN KEY a referenční akce), umělé identifikátory.

- Databáze - Logicky uspořádaný soubor souvisejících dat, obsahuje data, metadata, schéma, omezení integrity atd.
- Database management system (DBMS) - Softwarový systém umožňující přístup do databáze, poskytuje mechanismy pro zajištění bezpečnosti a spolehlivosti, souběžnost, integritu uložených dat, ...
- Databázový systém - informační systém obsahující databázi, DBMS, hardware, procesy, osoby...
- Relační databáze - množina relací

Relační databáze (datový model, NULL hodnoty, tříhodnotová logika)

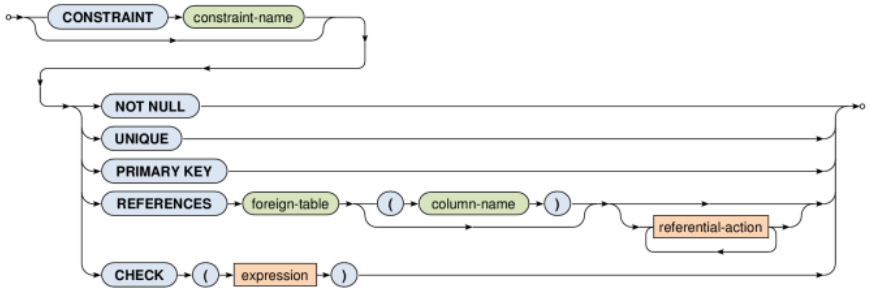
NULL

- Pro zpracování chybějících informací
- Jak systém NULL zpracovává
  - 3 + NULL = NULL
  - 3 < NULL -> UNKNOWN
    -> tříhodnotová logika: TRUE, FALSE, UNKNOWN

p	q	p AND q	p OR q	NOT q
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	UNKNOWN	UNKNOWN	TRUE	UNKNOWN
FALSE	TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE	FALSE	
FALSE	UNKNOWN	FALSE	UNKNOWN	
UNKNOWN	TRUE	UNKNOWN	TRUE	
UNKNOWN	FALSE	FALSE	UNKNOWN	
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	

Integritní omezení (NOT NULL, UNIQUE, PRIMARY KEY, CHECK, FOREIGN KEY a referenční akce),

- **Entitní** – každý záznam v tabulce má platný a jedinečný primární klíč.
- **Doménová** – zajišťují dodržování datových typů/domén definovaných u sloupců databázové tabulky
- **Referenční** – zabývají se vztahy dvou tabulek, kde jejich relace je určena vazbou primárního a cizího klíče



- NOT NULL - hodnoty nesmí být null
- UNIQUE - hodnoty musí být unikátní
- PRIMARY KEY = NOT NULL + UNIQUE
- FOREIGN KEY - hodnoty klíče v odkazující tabulce musí být v odkazované tabulce
- CHECK - definuje se podmínka, která musí být splněna

Referenční akce
Pokud nejsou nastavené referenční akce a zároveň by operace v odkazované tabulce způsobila porušení cizího klíče v odkazující tabulce, tak by operace byla blokována a generoval by se error message.

- Referenční akce

- Trigger - různé případy změn v tabulce
  - ON UPDATE
  - ON DELETE
- Akce - co se provede po změně v tabulce
  - CASCADE - hodnota je také změněna / odstraněna
  - SET NULL - hodnota nastavena na NULL
  - SET DEFAULT - hodnota nastavena na výchozí
  - NO ACTION - zabraňuje provedení operace
  - RESTRICT - podobný no action

Umělé identifikátory

- Zcela automatizované
- Obvykle se jedná o celočíselné řady a každý novější záznam dostává číslo vždy o jednotku vyšší než je číslo u posledního vloženého záznamu (číselné označení záznamů s časem stoupá).

Jazyk SQL. Definice schématu. Manipulace s daty (SELECT, INSERT, UPDATE, DELETE). Spojování tabulek, predikáty, řazení, seskupování a agregace, množinové operace, vnořené dotazy. Pohledy (aktualizovatelnost, CHECK OPTION). Procedurální rozšíření (funkce, kurzory, triggery)

Jazyk SQL.

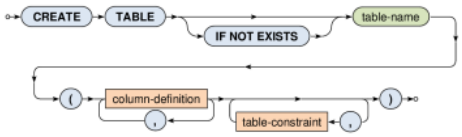
Structured Query Language (SQL) je jazyk pro kladení dotazů do databáze. Obsahuje jak příkazy DML (Data manipulation Language), tak i DDL příkazy (Data Definition Language). SQL je case insensitive (nerozlišuje mezi velkými a malými písmeny).

- Transaction management
- Administrace Databáze

Definice schématu.

Tvorba tabulky

- CREATE TABLE
  - Název tabulky
  - Definování sloupců (atributů) - názvy, datové typy, výchozí hodnota
  - Definování integritních omezení v rozsahu tabulky




```
CREATE TABLE Product (
   Id INTEGER,
   Name VARCHAR(128),
   Price DECIMAL(6,2),
   Produced DATE,
   Available BOOLEAN DEFAULT TRUE,
   Weight FLOAT
);
```

- Datové typy - integer, decimal, boolean, float, real, double, char (daná délka), varchar (variabilní délka), date, time, timestep

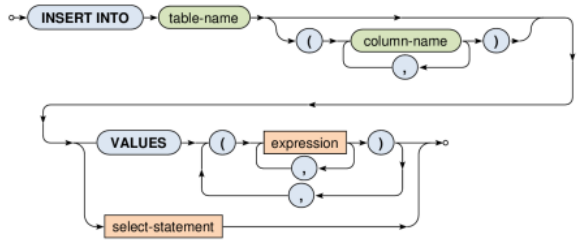
Úprava schématu

- ALTER TABLE – změny existujících objektů.
  - RENAME TO
  - ADD COLUMN
  - ALTER COLUMN
  - DROP COLUMN

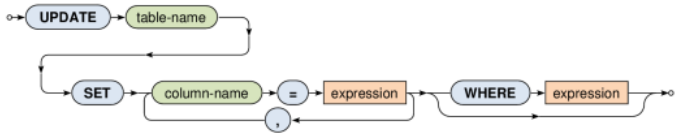
    - ADD (constraint definition)
    - DROP CONSTRAINT
- DROP TABLE

Manipulace s daty (SELECT, INSERT, UPDATE, DELETE).

- Příkazy pro získání dat z databáze a pro jejich úpravy
- DML – Data Manipulation Language
- INSERT INTO – vkládá do databáze nové řádky (záznamy)



- UPDATE – mění data v databázi (editace záznamu)

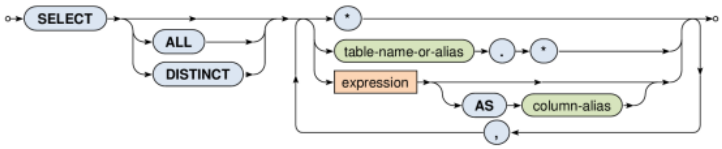


- MERGE – kombinace INSERT a UPDATE – data buď vloží (pokud neexistuje odpovídající klíč), pokud existuje, pak je upraví ve stylu UPDATE.
- DELETE FROM – odstraňuje data (záznamy) z databáze

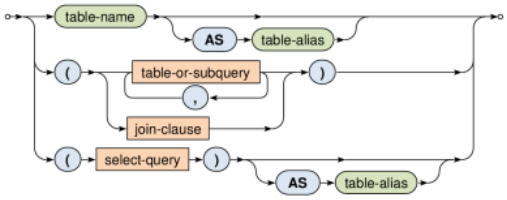
- EXPLAIN – speciální příkaz, který zobrazuje postup zpracování SQL příkazu. Pomáhá uživateli optimalizovat příkazy tak, aby byly rychlejší.
- SHOW - méně častý příkaz, umožňující zobrazit databáze, tabulky nebo jejich definice

SELECT

- Vybírá data z databáze, umožňuje výběr podmnožiny a řazení dat
- SELECT + názvy sloupců, nebo * , nebo definování nových a agregovaných sloupců
    - nové pojmenování sloupců pomocí AS
    - ALL - vše
    - DISTINCT - odstranění duplicit

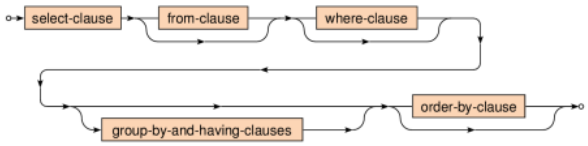


- FROM - z které tabulky, poddotazu nebo pohledu



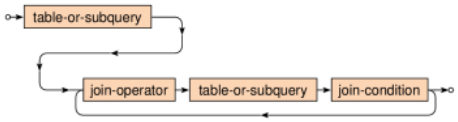
- WHERE - podmínka
    - AND, OR, NOT
    - predikáty
- GROUP BY - volba atributů pro agregaci
- HAVING - podmínka pro agregovaná data

- ORDER BY - řazení

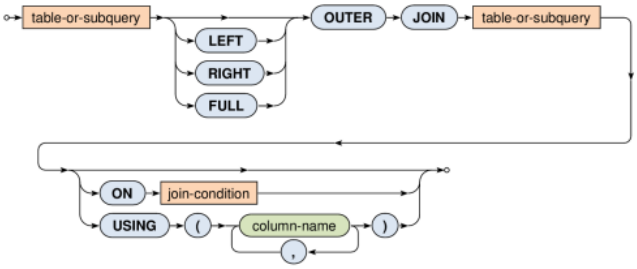


Spojování tabulek

JOIN



- CROSS JOIN - kartézský součin všech řádků z obou tabulek (všechny kombinace)
- NATURAL JOIN - výsledkem pouze řádky, které mají stejné hodnoty atributů
- INNER JOIN - defaultní JOIN
  - podmínka ON
  - USING atribut - odpovídá natural join
  - pokud bez ON nebo USING - odpovídá cross join
- OUTER JOIN - řádky z inner joinu a řádky, které nemohou být spojeny (hodnoty doplněny NULL)
  - LEFT / RIGHT - řádky z levé / pravé tabulky
  - FULL - z obou

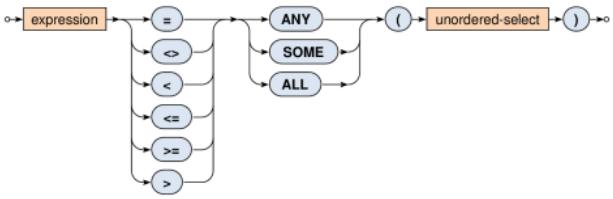


- UNION JOIN - řádky jsou integrovány do jedné tabulky, žádné řádky nejsou spojeny

Predikáty

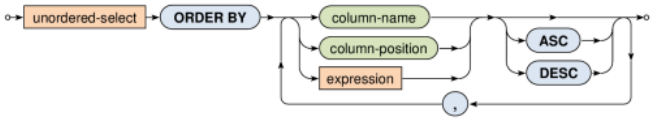
- Srovnávací predikáty =, <>, <, <=, >=, >
- Intervalový predikát BETWEEN ... AND ...

- Predikát shody řetězce LIKE
- Predikát detekce hodnot NULL - IS / IS NOT NULL
- Predikát příslušnosti k množině IN
  - pokud prázdná množina = FALSE
  - pokud množina pouze s hodnotami NULL = UNKNOWN
- Predikát existenčního kvantifikátoru EXISTS - zda je množina prázdná
  - pokud prázdná množina = FALSE
  - pokud množina pouze s hodnotami NULL = TRUE
- Predikáty porovnávání množin
  - ALL - všechny řádky vnořeného dotazu musí splňovat srovnávací predikát
    - pokud prázdná množina = TRUE
    - pokud množina pouze s hodnotami NULL = UNKNOWN
  - ANY = SOME - alespoň jeden řádek musí splňovat srovnávací predikát
    - pokud prázdná množina = FALSE
    - pokud množina pouze s hodnotami NULL = UNKNOWN



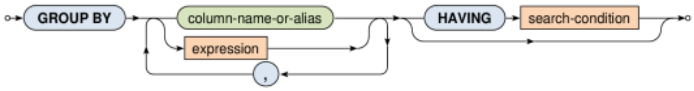
Řazení

- Bez ORDER BY nemají řádky definované pořadí
- NULL v pořadí jako první
- Směry - ASC vzestupně, DESC sestupně



Seskupování a agregace

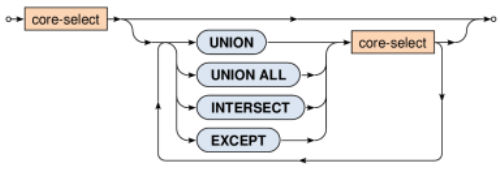
- GROUP BY - řádky jsou rozděleny do skupin podle stejných hodnot určených atributů
- HAVING - podmínka pro agregovaná data



- Agregační funkce jsou v SQL statistické funkce, pomocí kterých systém řízení báze dat umožňuje seskupit vybrané řádky dotazu (získané příkazem SELECT) a spočítat nad nimi výsledek určité aritmetické nebo statistické funkce. Agregační funkce se v SQL používají s konstrukcí GROUP BY. Agregační funkce pracují s kolekcí hodnot a vrací jedinou výslednou hodnotu.
  - **avg :** průměrná hodnota
  - **min :** minimum
  - **max :** maximum
  - **sum :** součet hodnot
  - **count :** počet hodnot
- NULL jsou ignorovány
  - COUNT(ø) = 0
  - SUM(ø) = NULL
  - AVG(ø) = NULL
  - MIN(ø) = NULL
  - MAX(ø) = NULL

Množinové operace

- UNION - sjednocení dvou tabulek bez duplicit
- UNION ALL - sjednocení tabulek s duplicitami
- INTERSECT - průnik
- EXCEPT - rozdíl



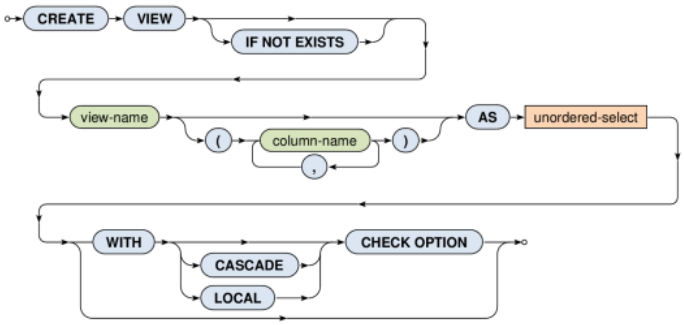
Vnořené dotazy.

- Poddotaz je takový dotaz na databázi, který je umístěn uvnitř jiného „vnějšího“ dotazu a výsledky z něj se používají v nějaké podmínce ve vnějším dotazu.
- Použití:
  - tam, kde není vhodné nebo možné použít agregační funkce nebo (pro dodržení kompatibility) uložené procedury.
  - Pro predikáty ANY, SOME, ALL, IN, EXISTS

- pro definování tabulek do FROM
- Příklady:
  SELECT *FROM Flights WHERE (Passengers > (SELECT AVG(Passengers) FROM Flights))*
  SELECT Flights. , (SELECT COUNT(*) FROM Aircrafts AS A WHERE (A.Company = F.Company) AND (A.Capacity >= F.Passengers)) AS Aircrafts FROM Flights AS F

Pohledy (aktualizovatelnost, CHECK OPTION).

- Pohledy jsou pojmenované SELECT dotazy
- Mohou být používané podobně jako tabulky - ve FROM
- Jsou vyhodnocovány dynamicky
- Využití - Vytvoření virtuálních tabulek, bezpečnostní důvody (skrytí tabulek a jejich obsahu před konkrétními uživateli), opakované použití stejných složitých příkazů, ...
- CREATE VIEW



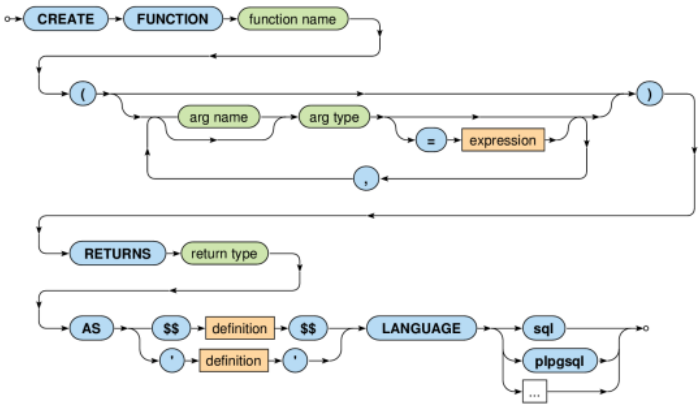
Aktualizovatelnost

- Možnost přidat nebo upravit řádky
- Lze, pokud:
  - je pohled založen na jednoduchém dotazu SELECT (bez agregace, poddotazů, ...) pouze s projekcemi (bez odvozených hodnot) a nad jednou tabulkou (bez joinu)
  - je definována WITH CHECK OPTION
    - LOCAL - nové/upravené řádky jsou viditelné v daném pohledu
    - CASCADE (výchozí) - nové/upravené řádky jsou viditelné v daném pohledu a v pohledech, od kterých je daný pohled odvozen

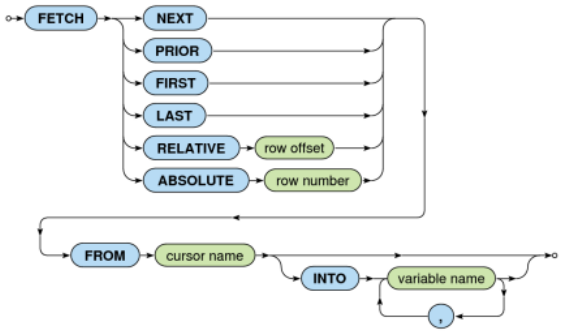
Procedurální rozšíření (funkce, kurzory, triggery)

Funkce

- CREATE FUNCTION

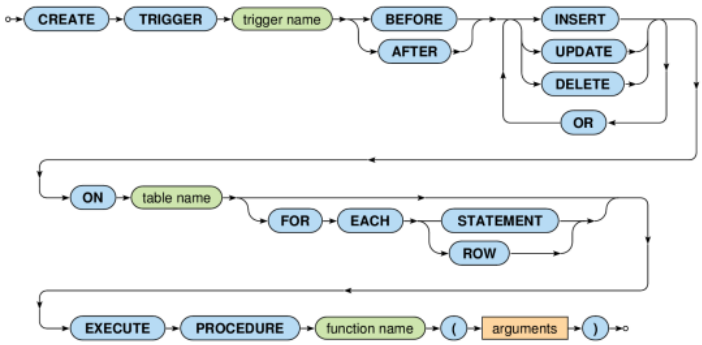


- **Kurzory**
- Databázový kurzor je řídicí struktura, která nám umožňuje přejít řádky vybrané tabulky.
- SCROLL option - zpětné stažení (backward fetch) může být specifikováno, v opačném případě je povoleno pouze dopředné načítání (forward fetch)
- FETCH - načítání dat
  - INTO - místní proměnné, do kterých bude uložen daný řádek, hodnoty NULL jsou vráceny, pokud neexistuje žádný další řádek



Triggery

- Procedura, která se provádí automaticky jako reakce na určité události (INSERT, UPDATE, DELETE).
- Používá se pro udržování složitých integritních omezení
- CREATE TRIGGER



// je pouze v nadpisu otázky, ale ne ve specifikacích

noSQL databáze

NoSQL je databázový koncept, ve kterém datové úložiště i zpracování dat používají jiné prostředky než tabulková schémata tradiční relační databáze. Motivací k tomuto přístupu mohou být jednoduchost designu, horizontální i vertikální škálovatelnost a jemnější kontrola dostupnosti. Databáze bez SQL jsou často vysoce optimalizovaná úložiště typu klíč-hodnota (ne vždy). Díky odlišné struktuře ukládání dat (např. stromová, grafová) oproti relačním databázím, je i algoritmická složitost pro různé operace odlišná. Obecně se vhodnost aplikace daného typu databáze liší podle řešeného problému. Bariéry k rozsáhlejšímu nasazení těchto úložišť do praxe jsou např. nepřítomnost plnohodnotné podpory transakčního modelu ACID, použití (různých) nízkoúrovňových dotazovacích jazyků, nedostatečná standardizace rozhraní a vysoké realizované investice podniků do SQL v minulosti.

Fyzická vrstva

Fyzická vrstva (bloky, sloty, buffer management). Organizace záznamů (halda, seřazený soubor, hašovaný soubor, složitost operací), indexové struktury (B$^+$-stromy, hašované indexy, bitmapové indexy).

Charakteristika

- Specifikuje jak jsou databázové struktury implementovány ve specifickém prostředí
- Zahrnuje soubory, indexovací struktury a tak dále

Stránky, sloty a záznamy

- Záznamy jsou uloženy na disku v *blocích* (angl. *page*) fixní velikosti
- Hardware přistupuje pouze k celým stránkám uloženým na disku (čtení i zápis)
- Čas, který trvá jedna I/O operace se na rotačním pevném disku spočítá jako součet doby hledání, rotačního zpoždění a přenosu dat
- stránky bývají rozděleny na *sloty* ve kterých jsou uložené *záznamy*, které jsou identifikované číslem stránky + číslem záznamu
- Záznamy mohou mít fixní nebo variabilní velikost
- Každá stránka má ještě hlavičku, která obsahuje informace o využití slotů, případně jejich velikosti pokud stránka obsahuje záznamy variabilní velikosti

Buffer management

- *Buffer* je část hlavní paměti, kde jsou dočasně uloženy celé stránky z disku, stránky jsou mapovány na paměťové rámce

- Každý rámec si drží počet referencí na stránku v rámci a *dirty flag*, který označuje změněný záznam, který nebyl zapsán na disk

Datové soubory

- *Halda* je druh souboru, kde záznamy nejsou v blocích řazené a záznamy tedy mohou být vyhledávány pouze sekvenčně, vkládat nové záznamy je snadné, ale při mazání může vznikat nevyužité místo
  - Prázdné stránky jsou udržovány pomocí spojového seznamu nebo tabulky stránek
  - Všechny operace mají lineární složitost, až na vkládání, které je konstantní
- *Řazený soubor* je druh souboru, kde jsou záznamy řazeny na základě nějakého klíče
  - Rychlé vyhledávání (logaritmická složitost)
  - Pro vkládání má každá stránka prázdný overhead, pokud je zaplněn jsou použity nové stránky
  - Ve všem kromě vyhledávání je pomalejší než halda

Hashované soubory

- Soubor je organizovaný do tzv. *kapes* (anglicky *buckets*)
- To v jaké kapse je záznam uložený je rozhodnuto na základě hashovací funkce f
- Pokud je kapsa plná, je alokována nová stránka a propojena s kapsou formou spojového seznamu
- Výhodou je rychlé vyhledávání, nevýhodou je ale paměťová náročnost (řešeno dynamickým hashováním)
- Většina operací má přibližně složitost N/K, kde K je počet kapes

Indexové struktury

- *Index* je pomocná struktura, která poskytuje rychlé vyhledávání na základě hledaných klíčů
- Obsahuje typicky jenom klíče a odkazy k záznamům, vyžaduje proto výrazně méně místa než datové soubory
- Index může obsahovat celý záznam, pár klíče a odkazu na záznam, nebo pár klíče a seznamu odkazů na záznam
- Řazení indexovaných předmětů může být zachováno (*clustered indexing*) nebo nemusí (*unclustered indexing*)

B$^+$-stromy

- Rozšířený B-strom
- Logaritmická složitost pro vkládání, vyhledávání podle rovnosti a mazání podle rovnosti
- Garantuje využití alespoň 50% stránek
  - Oproti B-stromu jsou všechny klíče v listech a stránky příslušné listům jsou provázány pro vykonávání dotazů založených na intervalech

Hašovaný index

- Stejné jako hašovaný soubor, ale obsahuje pouze odkazy na záznamy

Bitmapy

- Vhodné pro atributy, které mohou nabývat malého množství hodnot
- Každý atribut je realizován jako vektor bitů, kde i-tý bit popisuje jestli atribut nabývá i-té hodnoty nebo ne
- Dotazy lze pak provádět pomocí bitových masek a bitwise operací
- Výhody - efektivní, kompaktní, rychlé, snadno paralelizovatelné
- Nevýhody - Pouze pro atributy s doménou malé kardinality, pomalé dotazy na základě rozsahu (lineární v šířce rozsahu)

Funkční závislosti

Funkční závislosti (definice, Armstrongovy axiomy), úpravy závislostí (funkční uzávěr, pokrytí, kanonické pokrytí, redundantní závislost, neredundantní pokrytí, atributový uzávěr, redukovaná závislost, minimální pokrytí). Hledání klíčů (nalezení prvního klíče, Lucchesi-Osborn). Normální formy (první, druhá, třetí, Boyceho-Coddova).

Definice

- *Funkční závislost* $X \to Y$ nad schématem $R(A)$ je mapování $f_i : X_i \to Y_i$, kde $X_i, Y_i \subseteq A$
- U $X \to Y$ říkáme, že hodnoty v X určují hodnoty v Y
- Pokud platí $X \leftarrow Y$ a $Y \leftarrow X$, pak řekneme, že $X \leftrightarrow Y$ jsou *funkčně ekvivalentní*
- Pokud je na pravé straně funkční závislosti jediný atribut a, mluvíme o *elementární funkční závislosti*

Armstrongovy axiomy

1. Triviální funkční závislost: $Y \subseteq X \Rightarrow X \to Y$
2. Tranzitivita: $(X \to Y \wedge Y \to Z) \Rightarrow X \to Z$
3. Kompozice: $(X \to Y \wedge X \to Z) \Rightarrow X \to YZ$
4. Dekompozice: $X \to YZ \Rightarrow (X \to Y \wedge X \to Z)$

Funkční uzávěr

- *Uzávěr množiny funkčních závislosti* F je množina všech funkčních závislostí, které lze odvodit z funkčních závislostí v F pomocí Armstrongových axiomů
- Značí se F^+

Pokrytí

- *Pokrytí množiny funkčních závislostí* F je nějaká množina funkčních závislostí G taková, že platí $F^+ = G^+$
- *Kanonické pokrytí* je takové pokrytí, které obsahuje pouze elementární funkční závislosti

Redundance

- Funkční závislost f je *redundantní* v množině F, pokud platí $(F - \{f\}^+ = F^+)$
- To znamená, že f je odvoditelné ze zbytku množiny F
- *Neredundantní pokrytí* množiny F je pak takové pokrytí, které vznikne po odstranění všech redundantních funkčních závislostí z F

Uzávěr atributů

- *Uzávěr atributů* X^+ je podmnožina všech atributů z A, které lze odvodit z množiny atributů X pomocí funkčních závislostí
- Pokud $X^+ = A$, pak říkáme, že X je *super-klíč* (nadklíč)
- Ve funkční závislosti $X \to Y$ je atribut a *redundantní* když $Y \subseteq (X - a)^+$
- *Redukovaná fukční závislost* neobsahuje žádné redundantní atributy
- Pro $R(A)$ je *klíč* je takový super-klíč, kde je funkční závislost $K \to A$ redukovaná
- *Klíčový atribut* je atribut, který se nachází v jakémkoliv klíči (klíčů může být víc)

Minimální pokrytí

- *Minimální pokrytí* je neredundantní kanonické pokrytí, které obsahuje pouze redukované funkční závislosti

Nalezení jednoho klíče

- Z triviální funkčí závislosti $A \to A$ postupně odstraňujeme redundantní atributy na levé straně, dokud tam žádné nezůstanou
- Na levé straně funkčí závislosti pak dostaneme jeden z klíčů A

Lucchesi-Osbornův algoritmus

- Algoritmus k nalezení všech klíčů

1. Najdeme jakýkoliv klíč K
2. Vybereme funkční závislost $X \to y$ z F takovou, že $y \in K$, nebo ukončíme algoritmus pokud neexistuje
3. Protože, platí $X\{K - y\} \to A$, $X\{K - y\}$ je super-klíč
4. Redukujeme $X\{K - y\} \to A$ a na levé straně dostaneme nový klíč K', který je jiný než K
5. Pokud K' ještě není v množině nalezených klíčů, uložíme jej, položíme $K := K'$ a pokračujeme na krok 2. Jinak algoritmus končí

Normální formy

První normální forma (1NF)

- Relace je v 1NF právě tehdy, když platí současně:
  - atributy jsou atomické (dále nedělitelné)
  - k řádkům relace lze přistupovat podle obsahu (klíčových) atributů
  - řádky tabulky jsou jedinečné
- Relace nesplňující 1NF:

jmeno	**prijmeni**	**adresa**
Josef	Novák	Technická 2, Praha 16627

Petr	Pan	Karlovo náměstí 13, Praha 12135

- Relace v 1NF:

jmeno	prijmeni	ulice	cislo	mesto	psc
Josef	Novák	Technivká	2	Praha	16627
Petr	Pan	Karlovo náměstí	13	Praha	12135

Druhá normální forma (2NF)

- Relace je v 2NF právě tehdy, když platí zároveň:
  - relace je v 1NF
  - každý neklíčový atribut je plně závislý na každém kandidátním klíči
- Příklad:
  - Mějme relaci *{**IdStudenta, IdPredmetu**, JmenoStudenta, SemestrLZ},* kde *IdStudenta* a *IdPredmetu* tvoří primární klíč.
  - Tato relace není v 2NF, protože *JmenoStudenta* je závislé pouze na *IdStudenta* a *SemestrLZ* je závislé pouze na *IdPredmetu.*
- Řešení:
  - Rozdělení relace do tří tabulek
    - *{**IdStudenta**, IdPredmetu__}*
    - *{__IdStudenta, JmenoStudenta}*
    - *{IdPredmetu, Semestr}*

Třetí normální forma (3NF)

- Relace je v 3NF právě tehdy, když platí:
  - relace je v **2NF**
  - všechny neklíčové atributy musí být vzájemně nezávislé
- Příklad:
  - Mějme relaci *{**IdStudenta**, JmenoStudenta, Fakulta, Dekan} ta* není ve **3NF**. Sice je ve **2NF**, ale atribut Dekan je funkčně závislý na *Fakulta* a *Fakulta* je funkčně závislá na *IdStudenta* (předpokládáme, že student nemůže být současně studentem více fakult téže university)
  - *IdStudenta* není funkčně závislá na *Fakulta*. Atribut _Deka_n je tedy transitivně závislý na klíči.
- Řešení:
  - Rozdělíme relaci do tabulek:
    - *{__IdStudenta, JmenoStudenta, Fakulta}*
    - *{Fakulta, Dekan}*

Transakce

Transakce (základní pojmy, ACID vlastnosti, BEGIN a COMMIT, rozvrh, historie, uspořádatelnost, serializovatelnost), konfliktová, uspořádatelnost (konflikty WR, RW, WW, precedenční graf), zamykací protokoly (dvoufázové a striktní dvoufázové zamykání), uváznutí (deadlock, graf čekání, Coffmanovy podmínky, strategie wait-die a wound-wait), fantom.

Transakční zpracování

- transakce je posloupnost operací (část programu), která přistupuje a aktualizuje (mění) data.
- Transakce pracuje s konzistentní databází.
- Během spouštění transakce může být databáze v nekonzistentním stavu.
- Ve chvíli, kdy je transakce úspěšně ukončena, databáze musí být konzistentní.
- Dva hlavní problémy:
  - Různé výpadky, např. chyba hardware nebo pád systému
  - Souběžné spouštění více transakci

ACID vlastnosti

- K zachování konzistence a integrity databáze, transakční mechanismus musí zajistit:
  - **A**tomicity : transakce atomická - buď se podaří a provede se celá nebo nic. Nelze vykonat jen část transakce.
  - **C**onsistency : transakce - konkrétní transformace stavu, zachování invariant - integritní omezení.
  - **I**solation : (isolace = serializovatelnost). I když jsou transakce vykonány zároveň, tak výsledek je stejný. jako by byly vykonány jedna po druhé.

- **D**urability : po úspěšném vykonání transakce (commit) jsou změny stavu databáze trvalé a to i v případě poruchy systému - zotavení chyb.

Otázka 35

Dvě paralelně zpracovávané transakce nemohou poškodit konzistenci dat právě tehdy, když:
1. Jsou vykonávány sériově.
2. Jsou vykonávány na stupni izolovanosti SERIALIZABLE.
3. Nedojde k jejich vzájemnému uváznutí.

Která tvrzení platí:

A Pouze 1

B Pouze 2

C Pouze 3

D Pouze 1 a 2

E Pouze 1 a 3

Akce

- **akce na objektech :** READ, WRITE, XLOCK, SLOCK, UNLOCK
- **akce globální :** BEGIN, COMMIT, ROLLBACK

Transakční historie

- Posloupnost akcí několika transakcí, jež zachovává pořadí akcí, v němž byly prováděny
- Historie (rozvrh) se nazývá *sériová*, pokud jsou všechny kroky jedné transakce provedeny před všemi kroky druhé transakce.

Serializovatelnost

- Teorie: Nechť se transakce T i skládá z následujících elementárních akcí:
  - $\mathrm{READ}_i(A)$ - čtení objektu A v rámci transakce T_i
  - $\mathrm{WRITE}_i(A)$ - zápis (přepis) objektu A v rámci transakce T_i
  - $\mathrm{ROLLBACK}_i(A)$ - přerušení transakce T_i
  - $\mathrm{COMMIT}_i(A)$ - potvrzení transakce T_i
- Jsou možné 4 případy:

READI(A) - READJ(A)	Není konflikt	Na pořadí nezávisí
READI(A) - WRITEJ(A)	**Konflikt**	Pořadí má význam
WRITEI(A) - READJ(A)	**Konflikt**	Pořadí má význam
WRITEI(A) - WRITEJ(A)	**Konflikt**	Pořadí má význam

Konflikty

- *WR konflikt* - jedna transakce data zapíše, druhá je čte předtím než byla commited, tzv. *dirty read*
- *RW konflikt* - transakce čte data a druhá je zapíše předtím než byla ta první ukončena, tzv. *unrepeatable read*
- *WW konflikt* - přepis dat, která ještě nebyla commited, tzv. *blind write*
- Dvě historie jsou *konfliktově ekvivalentní*, pokud sdílí stejné konfliktní páry
- Historie je *konfliktově uspořádatelná*, pokud je konfliktově ekvivalentní s nějakou sériovou historií
- *Precedenční graf* je směrový graf, kde vrcholy jsou transakce a hrany jsou konfliktní páry mezi transakcemi
  - Pokud je acyklický, je historie sériově uspořádatelná

Zamykání

- Způsob tzv. *pesimistické kontroly* v plánovači transakcí
- *Zamykací protokol* je protokol, který zamyká entity tak, aby byla zabezpečena konfliktní serializovatelnost
- Dva druhy zámků

    - *Exkluzivní zámek* $X(A)$ je zámek, který může držet pouze jedna transakce, pouze transakce, která drží zámek $X(A)$, pak může přistupovat k entitě A
    - *Sdílený zámek* $S(A)$ je zámek, který může být držen více transakcemi, ale umožňuje pouze zápis, nikoliv čtení
- Navíc je definována operace odemykání $U(A)$

Dvoufázové zamykání (2PL)

- Pokud transakce chce číst (zapisovat) entitu A, musí nejprve získat sdílený (exkluzivní) zámek na tuto entitu
- Transakce nesmí zažádat o zámek na entitu A pokud jej předtím už uvolnila
- Zajišťuje acykličnost precedenčního grafu
- Nezajišťuje zotavitelnost (může se stále stát, že transakce uvolní všechny zámky, jiná po ní pokračuje a je ukončena a potom je první transakce přerušena)

Striktní dvoufázové zamykání (Strict 2PL)

- Zajišťuje zotavitelnost a kaskádovité přerušování
- Všechny zámky může transakce pustit až poté co je ukončena

Uváznutí (deadlock)

- Situace, která nastane, když všechny transakce čekají až jiná transakce uvolní zámek
- Lze identifikovat pomocí tzv. *grafu čekání*
    - Vrcholy jsou jednotlivé transakce
    - Hrana z T_1 do T_2 říká, že transakce T_1 čeká až T_2 uvolní zámek
    - Pokud je v grafu cyklus, došlo k uváznutí

Metody prevence/řešení uváznutí

- Přerušení transakce a restart
- Detekce uváznutí pomocí grafu čekání a přerušení transakce na základě nějakého kritéria (například transakce, která vykonala nejméně práce)
- Dvě strategie na předcházení uváznutí, využívá priority transakcí:
    - *wait-die* - pokud má čekající transakce vyšší prioritu tak čeká dále, pokud ne, tak je ukončena
    - *wound-wait* - pokud má čekající transakce vyšší prioritu tak ukončí transakci, která drží zámek, který potřebuje, jinak čeká dále

Coffmanovy podmínky

- K uváznutí může nastat pokud platí všechny z následujících podmínek

1. *Vzájemné vyloučení* - zdroje nejsou plně sdíleny a jsou drženy exkluzivně
2. *Držení zdrojů* - další zdroje mohou být vyžadovány i když už jsou jiné drženy
3. *Žádná preempce* - zdroje mohou být vypuštěny pouze dobrovolně
4. *Kruhové čekání* - transakce čekají a vyžadují zdroje v kruhu

- Nenaplnění jakékoliv z těchto podmínek stačí na prevenci uváznutí

Fantom

- Může nastat v databázi, kde jsou vkládány a mazány nové položky
- Jedna transakce pracuje s nějakou množinou entit, zatímco druhá transakce tuto množinu změní
- Může vést k nekonzistentní databázi
- Může nastat i při striktním dvoufázovém zamykání