

n13. Konceptuální a logický datový model, dotazovací jazyk SQL, transakční zpracování, objektově-relační mapování, noSQL databáze.

DBS, materialy, zkousky, testy:

https://drive.google.com/drive/folders/1FeJNQvQx2UMgfU8omwNU_FOtSn-hZa_O?usp=sharing

Konceptuální a logický datový model

1. konceptuální model - mapování skutečnosti na databázovou reprezentaci, používá se UML a ER diagramů
2. logický model - specifikuje jak bude konceptualni model vypadat v databazi
3. fyzický model - jak je databaze implementovana a jak se budou data ukladat

Relační model

Základní pojmy

- **relace** (relation): Jsou dvourozměrné struktury tvořené záhlavím a tělem (databázové tabulky).
- **entitní typ** : je nějaká "věc" nebo "objekt" jednoznačně odlišitelná od ostatních (entitní typ je dán množinou svých atributů). Obvykle je vyjádřen podstatným jménem.
- **entita** : instance entitního typu (konkrétní řádek v tabulce, obsahuje nějaké hodnoty)
- **atribut** : vlastnost entitního typu. *např.*: Entitní typ student může obsahovat atributy: *jmeno, prijmeni*,
- **doména atributu** : přípustné hodnoty pro atribut
- **vztah** (relationship): zachycuje, jakým způsobem jsou dvě nebo více entit vztahované mezi sebou. Nezaměňovat s relací. Existují 3 typy vztahu mezi relacemi:

1:1, 1:N (cizí klíč na straně N), **M:N** (využívá vazební tabulku). Obvykle vyjádřen slovesem.

- **klíč (primární klíč)** : je atribut (nejčastěji id) nebo množina atributů (např. autor, název v tabulce knížek). Klíč jednoznačně určuje entitu.
- **super klíč** : Super klíč množiny entit je množina jednoho nebo více atributů, jejichž hodnoty jednoznačně určují entitu (tedy klíč je podmnožina atributů – např. všechny atributy).
- **kandidátní klíč** : Kandidátní klíč množiny entit je minimální super klíč. Rodné číslo je kandidátní klíč entity zákazník, číslo účtu je kandidátní klíč entity účet.
- **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).
- **slabá množina entit** : při modelování reality se někdy vytváří entitní typy, které nemají samy o sobě význam, Existence slabé množiny entit závisí na množině definujících entit.

Parcialita vs Kardinalita

Kardinalita říká zdali se může entita (tabulka) účastnit vztahu vícekrát, parcialita zdali se daná entita (tabulka) musí (nebo nemusí účastnit vztahu. V modelech se povinnost značí plnou čarou / 1 a nepovinnost přerušovanou čarou / 0.

Kardinalita vs. Konektivita

Kardinalita určuje počet prvků asociované množiny entit (entitního typu) prostřednictvím množiny vztahů.

Pro binární vztahy existují 4 typy kardinality:

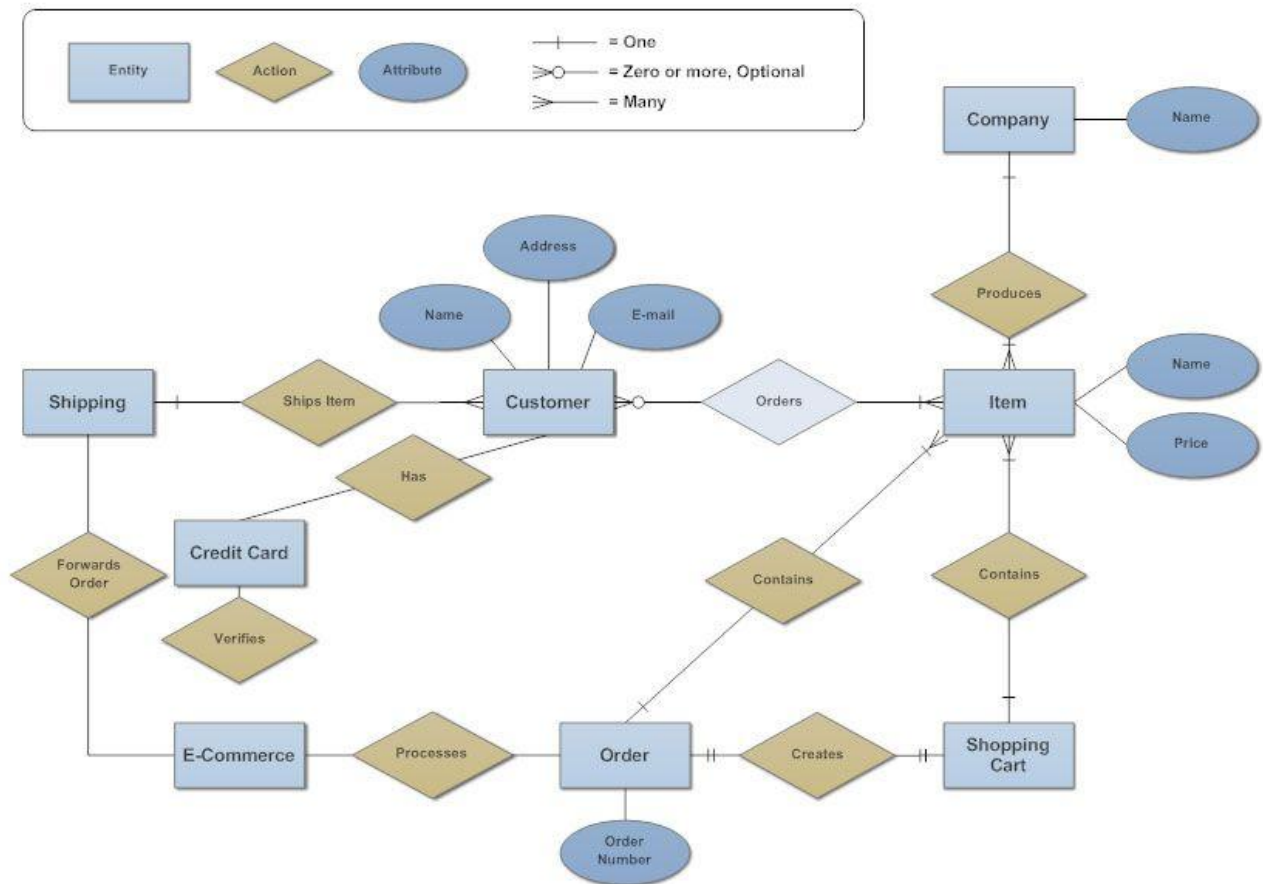
- **1:1** : přiřazuje jednomu záznamu jeden jediný záznam v jiné tabulce. Tento vztah se užívá jen ojediněle, protože většinou není důvod proč takové záznamy neumístit do jedné tabulky.
- **1:N** : přiřadí jednomu záznamu více záznamů v tabulce jiné. Nejpoužívanější typ relace, odpovídá mnoha situacím v reálném životě.
- **N:1** : obdobně jako **1:N**.
- **M:N** : méně častý. Umožňuje několika záznamům v jedné tabulce přiřadit několik záznamů v tabulce jiné. V databázové praxi bývá tento vztah z praktických důvodů nejčastěji realizován kombinací dvou vztahů 1:N a 1:M.

E-R diagramy

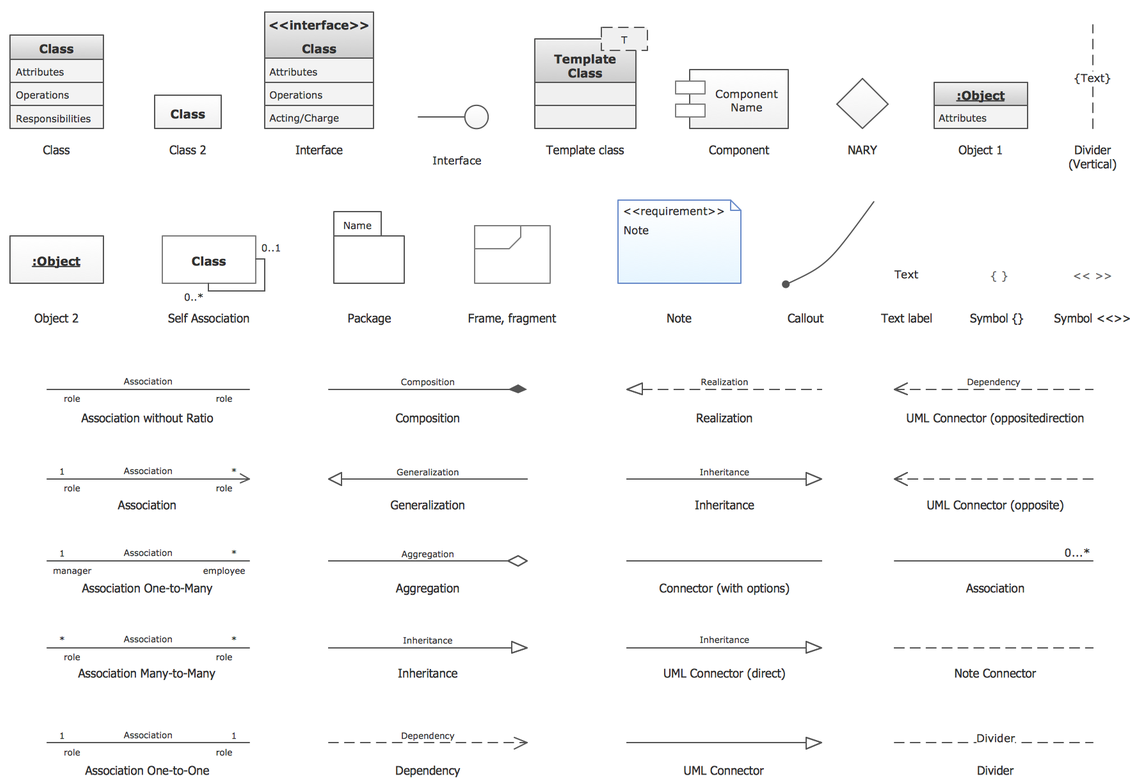
E-R diagram je grafická reprezentace E-R (entity-relationship) modelu.

- **obdelníky** - množiny entit (entitní typy)
- **kosočtverce** - množiny vztahů
- **ovály** - atributy
 - zdvojené ovály se používají pro více hodnotové atributy
 - čárkované ovály značí odvozené (počítané) atributy
- **podtržené atributy** - značí primární klíče

Entity Relationship Diagram - Internet Sales Model



UML –Unified Modeling Language



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

Příklad:

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	Technická	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 atribut, který není primárním klíčem je na primárním klíči úplně závislý

Příklad:

Mějme relaci $\{ \underline{IdStudenta}, \underline{IdPredmetu}, JmenoStudenta, Semestr \}$, kde $\underline{IdStudenta}$ a $\underline{IdPredmetu}$ tvoří primární klíč. Tato relace není v 2NF, protože $JmenoStudenta$ je závislé pouze na $\underline{IdStudenta}$ a $Semestr$ je závislé pouze na $\underline{IdPredmetu}$.

Řešení:

Rozdělení relace do tří tabulek

- $\{ \underline{IdStudenta}, \underline{IdPredmetu} \}$
- $\{ \underline{IdStudenta}, JmenoStudenta \}$
- $\{ \underline{IdPredmetu}, Semestr \}$

0.3.3 Třetí normální forma (3NF)

Relace je v **3NF** právě tehdy, když platí:

- relace je v **2NF**
- žádný atribut, který není primárním klíčem, není tranzitivně závislý na žádném klíči

Příklad:

Mějme relaci $\{ \underline{IdStudenta}, JmenoStudenta, Fakulta, Dekan \}$ ta není ve **3NF**. Sice je ve **2NF**, ale atribut $Dekan$ je funkčně závislý na $Fakulta$ a $Fakulta$ je funkčně závislá na $\underline{IdStudenta}$ (předpokládáme, že student nemůže být současně studentem více fakult téže university)

. $\underline{IdStudenta}$ není funkčně závislá na $Fakulta$. Atribut $Dekan$ je tedy transitivně závislý na klíči.

Řešení:

Rozdělíme relaci do tabulek:

- $\{ \underline{IdStudenta}, JmenoStudenta, Fakulta \}$
- $\{ \underline{Fakulta}, Dekan \}$

Integritní omezení

- **Entitní** – povinné integritní omezení, které zajišťuje úplnost primárního klíče tabulky; zamezí uložení dat, která neobsahují všechna pole sdružená do primárního klíče, nebo data, jež by v těchto polích byla stejná jako v nějakém jiném, již zapsaném, řádku tabulky. To znamená, že sloupce zvolené jako primární klíč by měly být unikátní a nenulové.

- **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

Dotazovací 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).

Příkazy pro manipulaci s daty

Jsou to příkazy pro získání dat z databáze a pro jejich úpravy. Označují se zkráceně DML – Data Manipulation Language („jazyk pro manipulaci s daty“).

- **SELECT** – vybírá data z databáze, umožňuje výběr podmnožiny a řazení dat.
- **INSERT** – vkládá do databáze nová data.
- **UPDATE** – mění data v databázi (editace).
- **MERGE** – kombinace INSERT a UPDATE – data buď vloží (pokud neexistuje odpovídající klíč), pokud existuje, pak je upraví ve stylu UPDATE.
- **DELETE** – 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

Příkazy pro definici dat

Těmito příkazy se vytvářejí struktury databáze – tabulky, indexy, pohledy a další objekty. Vytvořené struktury lze také upravovat, doplňovat a mazat. Tato skupina příkazů se nazývá zkráceně DDL – Data Definition Language („jazyk pro definici dat“).

- **CREATE** – vytváření nových objektů.
- **ALTER** – změny existujících objektů.
- **DROP** – odstraňování objektů.

Příkazy pro řízení dat

Do této skupiny patří příkazy pro nastavování přístupových práv a řízení transakcí. Označují se jako DCL – Data Control Language („jazyk pro ovládání dat“), někdy také TCC – Transaction Control Commands („jazyk pro ovládání transakcí“).

- **GRANT** – příkaz pro přidělení oprávnění uživateli k určitým objektům.

- REVOKE – příkaz pro odnětí práv uživateli.
- START TRANSACTION – zahájení transakce.
- COMMIT – potvrzení transakce.
- ROLLBACK – zrušení transakce, návrat do původního stavu.

Referenční integrita

Referenční integrita je nástroj databázového stroje, který pomáhá udržovat vztahy v relačně propojených databázových tabulkách. Referenční integrita se definuje cizím klíčem, a to pro dvojici tabulek, nebo nad jednou tabulkou, která obsahuje na sobě závislá data (například stromové struktury). Tabulka, v níž je pravidlo uvedeno, se nazývá podřízená tabulka (používá se také anglický termín *slave*). Tabulka, jejíž jméno je v omezení uvedeno, je nadřízená tabulka (*master*). Pravidlo referenční integrity vyžaduje, aby pro každý záznam v podřízené tabulce, pokud tento obsahuje data vztahující se k nadřízené tabulce, odpovídající záznam v nadřízené tabulce existoval. To znamená, že každý záznam v podřízené tabulce musí v cizím klíči obsahovat hodnoty odpovídající primárnímu klíči nějakého záznamu v nadřízené tabulce, nebo NULL.

Otázka 16

Máte zadanou relaci tvořenou spojením dvou tabulek T1 a T2. Primární klíč tabulky T1 je reprezentován atributem PK, odpovídající cizí klíč tabulky T2 je reprezentován atributem FK. Definice referenční integrity, jež zajistí, aby při změně hodnoty primárního klíče tabulky T1 došlo i k příslušné změně hodnoty cizího klíče tabulky T2,

- A je vlastností primárního klíče a tudíž je součástí definice tabulky T1.
- B je vlastností cizího klíče a tudíž je součástí definice tabulky T2.
- C vzhledem k tomu, že je vlastností páru tabulek T1 a T2, nemůže být součástí definice jednotlivých tabulek T1 či T2.

Agregační funkce

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. Agregace se v SQL používají s konstrukcí GROUP BY. Agregace pracují s kolekcí hodnot a vrací jedinou výslednou hodnotu.

- **avg** : průměrná hodnota
- **min** : minimum
- **max** : maximum
- **sum** : součet hodnot

- **count** : počet hodnot

Vnořené dotazy (pod-dotaz)

Poddotaz je takový dotaz na databázi, který je umístěn uvnitř jiného „vnějšího“ dotazu a výsledky z něj se používají v nějaké podmínce v tom vnějším dotazu.

Poddotaz je nejčastěji příkaz SELECT a poskytuje hodnoty do porovnávací podmínky (klauzuli WHERE) pro nadřazený dotaz (jiné části dotazu jen velmi zřídka). Používá se tam, kde není vhodné nebo možné použít agregační funkce nebo (pro dodržení kompatibility) uložené procedury.

Příklad:

```
SELECT * FROM tabulka1 WHERE sloupec1=(SELECT sloupec2 FROM tabulka2
WHERE podmínka);
```

Příklad:

```
DELETE FROM tabulka1 WHERE sloupec1 IN (SELECT sloupec2 FROM tabulka2
WHERE podmínka);
```

Příklad:

```
UPDATE tabulka1 SET sloupec1=hodnota1 WHERE EXISTS(SELECT sloupec2 FROM
tabulka2 WHERE podmínka);
```

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 transakcí

ACID vlastnosti

K zachování konzistence a integrity databáze, transakční mechanismus musí zajistit:

- **Atomicity** : transakce atomická - buď se podaří a provede se celá nebo nic. Nelze vykonat jen část transakce.
- **Consistency** : transakce - konkrétní transformace stavu, zachování invariant - integritní omezení.

- Isolation : (isolation = serializovatelnost). I když jsou transakce vykonány zároveň, tak výsledek je stejný, jako by byly vykonány jedna po druhé.
- Durability : 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

2 Akce

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

Stavy transakce

- Aktivní – počáteční stav; transakce zůstává v tomto stavu, dokud běží
- Částečně potvrzená (Partially Committed) – jakmile byla provedena poslední operace transakce
- Chybující (Failed) – po zjištění, že normální běh transakce nemůže pokračovat
- Zrušená (Aborted) – poté, co byla transakce vrácena (rolled back) a databáze byla vrácena do stavu před spuštěním transakce. Dvě možnosti po zrušení transakce:
 - Znovu spustit transakci – pouze pokud nedošlo k logické chybě
 - Zamítnout transakci
- Potvrzená (Committed) – po úspěšném dokončení

Transakční historie (rozvrh transakcí)

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: Necht' se transakce T_i skládá z následujících elementárních akcí:

- **READ i (A)** - čtení objektu A v rámci transakce T_i
- **WRITE i (A)** - zápis (přepis) objektu A v rámci transakce T_i
- **ROLLBACK i** - přerušení transakce T_i
- **COMMIT i** - 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

Zajímavé jsou navzájem konfliktní operace: Dvě historie **H1** a **H2** (na téže množině transakcí) jsou **ekvivalentní**, pokud jsou **všechny konfliktní operace** (nepřerušných) transakcí **provedeny v témže pořadí**.

To znamená, že pro dvě ekvivalentní historie a uspořádání $\langle H1 \text{ indukované historií } H1 \text{ a } \langle H2 \text{ indukované historií } H2$ platí: pokud p_i a q_j jsou konfliktní operace takové, že $p_i < H1 \ q_j$, musí platit také $p_i < H2 \ q_j$. Pořadí nekonfliktních operací není zajímavé.

- Základní předpoklady – každá transakce zachovává konzistenci databáze
- Tedy sériový plán zachovává konzistenci databáze
- Plán je serializovatelný, když je ekvivalentní sériovému plánu. Různé formy ekvivalence plánů vedou k následujícím pojmům:
 - Konfliktní serializovatelnost
 - Pohledová serializovatelnost
- Ignorujeme všechny instrukce kromě **čtení** a **zápisu** a předpokládáme, že transakce mohou provádět libovolné výpočty na datech v lokálních vyrovnávacích pamětech mezi čteními a zápisy. Naše zjednodušené plány se skládají pouze z operací **čtení** a **zápisu**.

Historie H je serializovatelná právě tehdy, když její závislostní graf nemá cykly.

Transakce je serializovatelná (s výjimkou fantom problémů), právě když:

- je dobře formulovaná (všechny akce prokryty zámky)
- zamykat výhradně všechny data, jejichž obsah modifikuje (legální)

- je dvoufázová - neměla by uvolňovat zámky dříve než budou všechny zámky aplikovány
- výhradní zámky drží až do COMMIT/ROLLBACK

Serializovatelnost se řeší pomocí:

- zamykání (locking) na různé úrovni granularity:
- zamykání celého systému (=>sériovost)
- jednotlivých tabulek
- Jednotlivých záznamů (vět)
- časové značky
- MVCC (multiversion concurrency control)
- predikátové zámky

Zamykání

Jedním ze způsobů, jak zajistit požadavek sériovosti, je zpřístupnit data vždy jen jediné transakci. Když jedna transakce získá k údajům výlučný (exklusivní) přístup, pak tento údaj nemůže modifikovat jiná transakce dříve, než první transakce skončí a uvolní přístup k údajům - a to i v případě, že byla při paralelním zpracování několikrát přerušena. Říkáme, že údaje jsou zamčeny. Jediný klíč ke každému zámku (při modifikaci) přiděluje systém pro řízení paralelního zpracování těm transakcím, které o něj požádají.

Existuje několik úrovní zamykání údajů (CO se zamyká):

1. Na úrovni operačního systému definujeme soubor typu read-only a tak zakážeme zápis a modifikaci všem.
2. Na úrovni SŘBD (Systém řízení báze dat, DBMS) v aplikačním programu definujeme svůj pracovní soubor jako soubor s výlučným přístupem (exclusive). Tak zamezíme přístup všem ostatním procesům, dokud náš program neskončí a neuvolní soubor. Použijeme příkaz k uzamčení a uvolnění souboru, říkáme, že soubor zamykáme explicitně. V SŘBD existují příkazy pro práci se souborem, které vyžadují výlučný přístup k souboru a tak si uzamykají soubor automaticky.
3. V aplikačním programu stačí často zamknout jen jeden nebo několik záznamů, ne celý soubor, aby tak byly ostatní záznamy přístupné ostatním uživatelům. Opět zamykání záznamů může být explicitní nebo automatické.
4. Některé SŘBD umožňují zamknout dokonce jen jednotlivé položky.

Rozlišujeme **zámky dvou** základních **druhů** (JAK se zamyká):

1. zámky pro sdílený přístup (shared) umožňují údaje jen číst více transakcím současně, ne však do nich zapisovat (**SLOCK**)

2. zámky výlučné (exclusive) umožní čtení i zápis vždy pouze jediné transakci (**XLOCK**).

Pokud má jedna transakce údaj (soubor, záznam) uzamčený a další transakce jej chce uzamknout také, může dojít ke kolizi. Proto v SŘBD existují funkce testující, zda je údaj volný. Pokud není, je nutno situaci programově řešit (počkat na uvolnění, zrušit transakci ap.).

Způsob zamykání (KDO zamyká):

1. Aplikační program (programátor) explicitním příkazem
2. SŘBD automaticky (implicitně) současně s některým příkazem pro manipulaci s daty

Použití zámků však není jednoduché, nesprávné použití může vést k nesprávným výsledkům. Důvodem může být například uvolnění zámku příliš brzy (může dojít k nekonzistenci)

Dobře definovaná transakce:

- Před každou operací READ se na daném DB objektu uplatní zámek SLOCK,
- před každou operací WRITE se na daném DB objektu uplatní zámek XLOCK
- operace UNLOCK se na daném DB objektu může provést pouze tehdy, když je na daném DB objektu uplatněn zámek SLOCK/XLOCK
- každá operace SLOCK/XLOCK je v někdy v následujícím běhu transakce následována příslušnou akcí UNLOCK.

Jednoduchá transakce

1. Obsahuje akce READ, WRITE, XLOCK, SLOCK a UNLOCK
2. COMMIT se nahradí sekvencí příkazů UNLOCK A, pro každý objekt A, na který bylo v průběhu transakce aplikováno SLOCK A nebo XLOCK
3. ROLLBACK se nahradí sekvencí:
 - WRITE A pro každý objekt A, na nějž T aplikovala akci WRITE A
 - UNLOCK A pro každý objekt A, na nějž T aplikovala akci SLOCK A nebo XLOCK A

Dvoufázová transakce

Dvoufázové transakce Všechny akce LOCK jsou provedeny před všemi akcemi UNLOCK. Fáze vzrůstu (growing phase) - během ní se provedou všechny akce LOCK
Fáze poklesu (shrinking phase) - během ní se provedou všechny akce UNLOCK. U dvoufázové transakce se fáze vzrůstu a fáze poklesu nepřekrývají.

Stupně izolace

Uváznutí transakcí a jeho prevence

Takovéto situaci, kdy obě transakce čekají, nelze žádný požadavek uspokojit a celý proces uvázne v mrtvém bodě nazýváme uváznutím (deadlock). Problém tedy je v tom, že pokud používáme zámků málo, hrozí nekonzistence, používáme-li zámků mnoho, hrozí uváznutí.

Máme nyní dva problémy: splnění požadavku sériovosti a řešení uváznutí v mrtvém bodě.

1 Požadavek sériovosti

K řešení prvního problému, požadavku sériovosti, se používá tzv. protokolu o zámcích. Je to řada pravidel udávajících, kdy může transakce zamknout a uvolnit objekty.

Pro prevenci uváznutí existuje více technik.

Nejjednodušší metodou prevence uváznutí je uzamčení všech položek, které transakce používá, hned na začátku transakce ještě před operacemi a jejich uvolnění až na konci transakce. Tak se transakce nezahájí dříve, dokud nemá k dispozici všechny potřebné údaje a nemůže dojít k uváznutí uprostřed transakce. Tato metoda však má dvě velké nevýhody:

1. využití přístupu k položkám je nízké, protože jsou dlouhou dobu zbytečně zamčené
2. transakce musí čekat až budou volné současně všechny údaje, které chce na začátku zamknout, a to může trvat velmi dlouho.

Jiná metoda prevence uváznutí využívá faktu, že k uváznutí nedojde, jestliže transakce zamykají objekty v pořadí respektujícím nějaké lineární uspořádání, definované nad těmito objekty (např. abecední ap.). Z hlediska uživatelského však takový požadavek je příliš omezující.

Plánovače

Některé systémy řeší problém uváznutí synchronizací paralelních transakcí pomocí plánovače. V SŘBD jsou zabudovány tyto programové moduly

- Modul řízení transakcí (RT); je to fronta, na kterou se transakce obracejí se žádostí o vykonání operací READ(X) a WRITE(X). Každá transakce je doplněna příkazy BEGIN TRANSACTION a END TRANSACTION.
- Modul řízení dat (RD) realizuje čtení a zápis objektů dle požadavků plánovače a dává plánovači zprávu o výsledku a ukončení.
- Plánovač zabezpečuje synchronizaci požadavků z fronty dle realizované strategie a řadí požadavky do schémat.

- Schéma pro množinu transakcí je pořadí, ve kterém se operace těchto transakcí realizují.

Nejjednodušší schéma je sériové (vždy proběhne celá transakce, pak další), ovšem je málo průchodné. Cílem celé strategie je větší průchodnost systému.

Plánovač při dvoufázovém zamykání vykonává tyto operace

- řídí zamykání objektů
- operace čtení a modifikace objektů povoluje jen těm transakcím, které mají příslušné objekty zamknuté
- sleduje, jestli transakce dodržují protokol dvoufázového zamykání; pokud zjistí jeho porušení, transakci zruší
- předchází uvážnutí nebo ho detekují a řeší zrušením transakce.

Jestliže systém nepoužívá prevenci uvážnutí, musí mít prostředky pro detekci (rozpoznání) uvážnutí a obnovu činnosti umrtvených transakcí. Detekce se provádí obvykle použitím grafu relace "kdo na koho čeká". Je to graf, jehož uzly jsou transakce a orientované hrany představují uvedenou závislost. Záznamem a analýzou grafu čekání se rozpoznává uvážnutí. Je-li v grafu cyklus, systém uvázl v mrtvém bodě.

Jestliže taková situace nastane, systém musí jednu nebo více transakcí vrátit zpět (pomocí souboru log), čímž se zablokováný přístup k datům (pro tuto transakci) odblokuje a umožní provést ostatní transakce. Připomíná to situaci, kdy se dva automobily potkají na úzké cestě a jeden musí vycouvat.

Obnovení činnosti se provádí pomocí souboru log, popsaného v předchozí kapitole. V případě potřeby je možno kteroukoliv transakci vrátit. Jde jen o to, kdy a které transakce se mají provést znovu. Systém vybírá takové transakce, aby s celým postupem byly spojeny co nejmenší náklady, k tomu bere v úvahu:

- jaká část transakce již byla provedena,
- kolik dat transakce použila a kolik jich ještě potřebuje pro dokončení,
- kolik transakcí bude třeba celkem vrátit.

Podle těchto kritérií by se mohlo dále stát, že bude vracena stále tatáž transakce a její dokončení by bylo stále odkládáno. Je vhodné, aby systém měl evidenci o vracených transakcích a při výběru bral v úvahu i tuto skutečnost.

Indexy

A **database index** is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is

accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

- B+ trees
- Bitmaps
- Sstables

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.

Persistence 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.

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 škálovatnost 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) nízkourovňových dotazovacích jazyků, nedostatečná standardizace rozhraní a vysoké realizované investice podniků do SQL v minulosti.

Databáze

- Column stores → Cassandra, Scylla
 - podobné jako relační databáze, máme jednotlivé sloupce se jmény, hodnoty a zpravidla i timestamp s dobou platnosti
 - nezaručují ACID, jsou jednoduché na horizontální škálování, využívají pro high-read, high-write účely
 - vysoká využití SSTables - sorted string tables, soubor se seřazenými páry klíč : hodnota. Ty se zásadně nemění ale slouží jako append only. Výsledná hodnota daného sloupce je pak sestavena až při samotném dotazu.
- Document stores → MongoDB, CouchDB, RethinkDB
 - pro semi-strukturovaná data, položky mohou mít více či méně dat ale nevadí to
 - trochu vyšší úroveň než pouze key value store, umí z dokumentu dostávat metadata, optimalizovat, lepe vyhledávat atd.
- Key-Value store → Redis, Memcache
 - v podstatě velké hashtably, in-memory, nekonzistentní, hlavní je rychlost
 - používají se často k cacheování
- Graph store → Neo4j, FlockDB
 - grafové databáze, například pro geografická data
-