

Obecné info

08 February 2024 22:18

Shift + ě = '

Zarovnání = ctrl + fn + f7

Hodnocení předmětu:

Úkol	Body
Test SQL SELECT 20 bodů (10 příkladů po 2 bodech)	20
Závěrečný test 0 až 40 bodů	38
Otázky z obsahu přednášek: 10 otázeek á 1 bod = 10 bodů.	
Datové modelování: 10 otázeek á 2 body = 20 bodů.	
SQL DDL 4x otázka á 2 body = 8 bodů.	
SQL DCL 1x otázka á 2 body.	
Semestrální práce 0 až 40 bodů	40
Suma	98

V tomto souboru jsou téměř slovo od slova přepsány přednášky Chlapka, ale i tak velmi doporučuj na ně chodit.

V části s otázkama na ZT jsem sesbírala snad všechny otázky co jsem kde našla, většina by jich měla být správně, já sama jsem jich v ZT měla asi 23 z tohoto dokumentu.

V další příloze vkládám i svou seminární práci - u Vedrala za plný počet, ale ten nám sám říkal že to máme udělat opravdu jednoduché ať tam nenaděláme chyby, někdo jiný by to možná vyžadoval složitější.

Základní pojmy

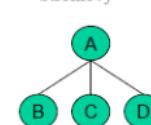
08 February 2024 22:42

Základní pojmy

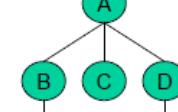
- Data
 - formalizované a fyzicky zaznamenané znalosti, poznatky, zkušenosti, výsledky pozorování procesů, projevů, činností a prvků světa
- Informace
 - smysluplná interpretace dat
 - -> něco, co těm datům dává smysl
 - Spousta dat se zaznamenává, ale málo kdy se čtou -> ne ze všech dat se musí stát informace
- Databáze
 - = datová základna
 - integrovaná počítacově zpracovávaná množina persistentních dat
 - Integrovaná = provázaná
 - Persistentní = trvale uložená
- Systém řízení bází dat (SŘBD / DBMS)
 - množina programových prostředků, který umožňuje:
 - vytvoření databáze,
 - použití databáze (manipulaci s daty v databází - S,I,U,D),
 - Select, Insert into, update, delete
 - údržbu a správu databáze
 - DBMS = database management systém
 - Programy, které umožňují manipulaci s daty
- Databázový systém (DBS) = SŘBD + Databáze
- Přínosy databázového přístupu:
 - sdílení dat (všechna data jsou na jednom místě)
 - snížení redundancy dat (redundance = nadbytečnost)
 - snazší zabránění vzniku nekonzistencek (když si žena změní příjmení, tak ho musíme změnit ve všech výskytech dat o tom člověku - kdybychom to neudělali, tak databáze je nekonzistentní, nemůžeme jí věřit, data nejsou kvalitní)
 - podpora transakčního zpracování (= způsob jak zajistit vnitřní integritu databáze, např. když odebereme peníze z jednoho účtu, musí se připsat na jiný úč. -> systém nemůže skončit v nějakém nedefinovaném stavu)
 - Transakce = množina operací, kterou nějak ohraničím, chci aby se promítla celá do databáze
 - údržba integrity databáze (integritní omezení) (= databáze vyhovuje zadaným pravidlům)
 - zajištění ochrany databáze před havárií a neautorizovaným přístupem
- Model dat:
 - Každý typ má množinu operací, kterou s těmi daty můžu dělat
 - Způsob uspořádání a způsob manipulace s daty:
 - Lineární
 - Z A do D se dostaneme jedině přes B a C
 - Hierarchická struktura
 - V praxi velmi užívaná
 - Př. Vysoké školy -> školy -> fakulty -> obor
 - Stromový
 - ◆ Potřebuji operace od kořene k listům nebo od listů ke kořenům
 - Relační
 - Objektově relační
 - Databázové systémy



Stromový



Síťový



Relační

Tabulka A	*	*	*	*	*	*
Tabulka B	*	*	*	*	*	*

Objektově relační



```
SELECT klausule ..... CO?  
FROM klausule ..... Z ČEHO ?  
[ WHERE klausule ] ..... PLATÍ-LI  
[ GROUP BY klausule]  
[ HAVING klausule]  
[ ORDER BY klausule]
```

Operace relační algebry

20 February 2024 19:28

Operace relační algebry

- Základní množinové operace: sjednocení, průnik, rozdíl, kartézský součin
 - Speciální operace relační algebry: projekce, restrikce, join
 - Pozn.
 - Operace sjednocení, průnik, rozdíl je možné provádět **pouze nad kompatibilními relačními tabulkami**
 - Výsledkem použité operace RA je nová (odvozená) relační tabulka
 - Kompatibilní atributy
 - Atributy definované na shodnou doménou (např. PK a FK)
 - Kompatibilní relační tabulky
 - Tabulky, které jsou stejného stupně (mají stejný počet sloupců) a které obsahují pouze kompatibilní atributy
- Programátoři Analytici
- | Jméno | Osob._číslo | Datum_narození |
|-------|-------------|----------------|
| Novák | 385 | 13.1985 |
| Rosák | 450 | 5.3.1959 |
- | Jméno | Osob._číslo | Datum_narození |
|-------|-------------|----------------|
| Novák | 385 | 13.1985 |
| Novák | 435 | 8.9.1965 |
| Paták | 411 | 4.6.1974 |
- Tyto tabulky jsou kompatibilní, protože obsahují dvojice atributů (osob._číslo programátorů a osob._číslo analytiků je nadefinováno nad stejnou doménovou množinou osobních čísel zaměstnanců v té dané firmě), stejně tak jméno a datum_narození

Základní množinové

Sjednocení

- Odvození relační tabulky, která bude obsahovat údaje o všech zaměstnancích (analytici i programátoři)
- Zápis: Programátoři_Analytici = Programátoři U Analytici

Programátoři_Analytici

Jméno	Osob._číslo	Datum_narození
Novák	385	13.1985
Novák	435	8.9.1965
Paták	411	4.6.1974
Rosák	450	5.3.1959

- Výsledkem jsou jenom 4 řádky, protože zaměstnanec č. 385 je programátor i analytik -> protože to sjednocuje, tak není možné aby tam byl víckrát (DISTINCT)

Průnik

- Odvození relační tabulky, která bude obsahovat údaje o zaměstnancích, kteří jsou analytiky a současně i programátory
- Zápis: Všeestranný = Programátoři ∩ Analytici

Všeestranný

Jméno	Osob._číslo	Datum_narození
Novák	385	13.1985

Rozdíl

- Odvození relační tabulky, která bude obsahovat údaje o všech specialistech - programátorech
- Zápis: Pouze_programující = Programátoři - Analytici

Pouze_programující

Jméno	Osob._číslo	Datum_narození
Rosák	450	5.3.1959

Symetrický rozdíl

- Odvození relační tabulky, která bude obsahovat údaje o všech specialistech pouze na jednu činnost
- Zápis: Specialisté = Programátoři Δ - Analytici

Specialisté

Jméno	Osob._číslo	Datum_narození
Novák	435	8.9.1965
Paták	411	4.6.1974
Rosák	450	5.3.1959

- Pozn: Operace symetrický rozdíl není základní operací RA. Je odvoditelná s použitím jiných operací RA. Například kombinací tří operací:
 - Všichni = Programátoři U Analytici

- Všeobecní = Programátoři ∩ Analytici
- Specialisté = Všichni - Všeobecní

Speciální operace

Projekce

- Odvození relační tabulky, která bude obsahovat údaje o jménech všech zaměstnanců

Zaměstnanci

Jméno	Oсоб_číslo	Datum_narození
Novák	385	1.3.1985
Novák	435	8.9.1965
Paták	411	4.6.1974
Rosák	450	5.3.1959

- Zápis: Jména_zaměstnanců = π Zaměstnanci (Jméno)

Jména_zaměstnanců

Jméno
Novák
Paták
Rosák

- Prostě to co zapíšu za to klíčové slovo SELECT (sloupečky oddělené čárkou)

Restrikce (selekce)

- Aplikuje se podmínka na jednotlivé řádky = vybírám takové řádky, kde je splněna nějaká podmínka
- Odvození relační tabulky, která bude obsahovat údaje o odděleních sídlících ve druhém patře

Zaměstnanci

Jméno	Oсоб_číslo	Datum_narození	Cis_odd
Novák	385	1.3.1985	01
Novák	435	8.9.1965	02
Paták	411	4.6.1974	02
Rosák	450	5.3.1959	01

Oddělení

Cis_odd	Název	Patro
01	Finančních systémů	1
02	Integrace	2
03	Služeb IS/ICT	2

- Zápis: Oddělení_Patro2 = ρ Oddělení (Patro = 2)

Oddělení

Cis_odd	Název	Patro
02	Integrace	2
03	Služeb IS/ICT	2

JOIN (equi)

- Operace, kdy potřebuju nějaké dvě tabulky propojit (potřebuju informace, které jsou v různých tabulkách, dostat do jedné výsledné tabulky)
- Př. Databáze tvořená zaměstnanci a odděleními
 - Zaměstnanci: jméno, osob_číslo (PK), datum_narození, cis_odd (FK, který ukazuje na primární klíč do tabulky oddělení)
 - Oddělení cis_odd (PK), název, patro

Zaměstnanci

Jméno	Oсоб_číslo	Datum_narození	Cis_odd
Novák	385	1.3.1985	01
Novák	435	8.9.1965	02
Paták	411	4.6.1974	02
Rosák	450	5.3.1959	01

Oddělení

Cis_odd	Název	Patro
01	Finančních systémů	1
02	Integrace	2
03	Služeb IS/ICT	2

- Odvození relační tabulky, která bude obsahovat údaje o zaměstnancích a názvech jejich oddělení a umístění v patře

- Zápis: Zam_s_nazvy = join (Zaměstnanci.Cis_odd = Oddělení.Cis_odd)

Zam_s_nazvy

Jméno	Oсоб_číslo	Datum_narození	Cis_odd	Cis_odd	Název	Patro
Novák	385	1.3.1985	01	01	Finančních systémů	1
Novák	435	8.9.1965	02	02	Integrace	2
Paták	411	4.6.1974	02	02	Integrace	2
Rosák	450	5.3.1959	01	01	Finančních systémů	1

- Pozn: Operace join není základní operací RA. Je odvoditelná s použitím operace kartézský součin a restrikce:

- V prvním kroku si mezi sebou vynásobím tyto dvě tabulky: Součin = Zaměstnanci x Oddělení
 - 4 + 3 sloupce -> výsledná tabulka bude mít 7 sloupců
 - 4 a 3 řádky -> výsledná tabulka bude mít 12 řádků (4x3)
 - Vynásobím každý řádek s každým řádkem

Součin

Jméno	Oсоб_číslo	Datum_narození	Cis_odd	Cis_odd	Název	Patro
Novák	385	1.3.1985	01	01	Finančních systémů	1
Novák	385	1.3.1985	01	02	Integrace	2

Součin	Jméno	Oсоб_číslo	Datum_narození	Cis_odd	Cis_odd	Název	Patro
Novák	385	1.3.1985	01	01	Finančních systémů	1	←
Novák	385	1.3.1985	01	02	Integrace	2	←
Novák	385	1.3.1985	01	03	Služeb IS/ICT	2	←
Novák	435	8.9.1965	02	01	Finančních systémů	1	←
Novák	435	8.9.1965	02	02	Integrace	2	←
Novák	435	8.9.1965	02	03	Služeb IS/ICT	2	←
Paták	411	4.6.1974	02	01	Finančních systémů	1	←
Paták	411	4.6.1974	02	02	Integrace	2	←
Paták	411	4.6.1974	02	03	Služeb IS/ICT	2	←
Rosák	450	5.3.1959	01	01	Finančních systémů	1	←
Rosák	450	5.3.1959	01	02	Integrace	2	←
Rosák	450	5.3.1959	01	03	Služeb IS/ICT	2	←

3.: 4HT218 Databáze

- V druhém kroku budu procházet jeden řádek za druhým, a vybírat řádky, ve kterých se hodnoty ve sloupcích Cis_odd shodují.
 - Zam_s_nazvy = ρ Součin (Zaměstnanci.Cis_odd = Oddělení.Cis_odd)
- Equi join = porovnáváme operátorem equi (=)

OUTER JOIN

Zaměstnanci

Jméno	Oсоб_číslo	Datum_narození	Cis_odd
Novák	385	1.3.1985	01
Novák	435	8.9.1965	02
Paták	411	4.6.1974	NULL
Rosák	450	5.3.1959	01

Oddělení

Cis_odd	Název	Patro
01	Finančních systémů	1
02	Integrace	2
03	Služeb IS/ICT	2

- Skoro stejná databáze, až na jednu věc -> připustil jsem, že ve sloupci Cis_odd v Tabulce zaměstnanců můžu mít neuvedenou hodnotu (NULL = neurčená hodnota)
 - Operace join v případě výskytu neurčené hodnoty (NULL VALUE) nemusí správně fungovat při požadavku vytvořit tabulku s údaji o všech zaměstnancích a jejich odděleních.
 - Klasický (INNER) join nevypíše seznam všech zaměstnanců, ale pouze ty řádky, které vyhovují restrikční podmínce („=“).
 - Zápis: Zam_s_nazvy_odd = join (Zaměstnanci.Cis_odd = Oddělení.Cis_odd)
- Zam_s_nazvy_odd
- | Jméno | Oсоб_číslo | Datum_narození | Cis_odd | Cis_odd | Název | Patro |
|-------|------------|----------------|---------|---------|--------------------|-------|
| Novák | 385 | 1.3.1985 | 01 | 01 | Finančních systémů | 1 |
| Novák | 435 | 8.9.1965 | 02 | 02 | Integrace | 2 |
| Rosák | 450 | 5.3.1959 | 01 | 01 | Finančních systémů | 1 |
- Nevypíše se Paták, protože má ve sloupci Cis_odd NULL -> ale já ho tam chci mít -> musím použít OUTER JOIN
- OUTER JOIN umožní, abychom řekli, ze které tabulky chceme vzít i ty řádky, které nevyhověli restrikční podmínce v Inner joinu
 - Záleží na pořadí, jak to napíšeme
 - 3 typy:
 - LEFT
 - RIGHT
 - FULL OUTER JOIN
 - Pozn: Operace join v případě výskytu neurčené hodnoty (NULL VALUE)
 - Left join -> protože zaměstnanci jsou napsáni vlevo od toho operátoru
 - Udělej klasický join, a potom z tabulky zaměstnanci přidej řádky, které nevyhověli restrikční podmínce
 - Right join -> funguje obráceně
 - Když bych to chtěl spojit tak, abych tam dal všechna oddělení (i oddělení 03, které se nevyskytuje v tabulce zaměstnanců)

Zam_s_nazvy_LEFT == LEFT join (Zaměstnanci.Cis_odd = Oddělení.Cis_odd)

Jméno	Oсоб_číslo	Datum_narození	Cis_odd	Cis_odd	Název	Patro
Novák	385	1.3.1985	01	01	Finančních systémů	1
Novák	435	8.9.1965	02	02	Integrace	2
Rosák	450	5.3.1959	01	01	Finančních systémů	1
Paták	411	4.6.1974	NULL			

Zam_s_nazvy_RIGHT == RIGHT join (Zaměstnanci.Cis_odd = Oddělení.Cis_odd)

Jméno	Oсоб_číslo	Datum_narození	Cis_odd	Cis_odd	Název	Patro
Novák	385	1.3.1985	01	01	Finančních systémů	1
Novák	435	8.9.1965	02	02	Integrace	2
Rosák	450	5.3.1959	01	01	Finančních systémů	1
Paták	411	4.6.1974	NULL		Služeb IS/ICT	2

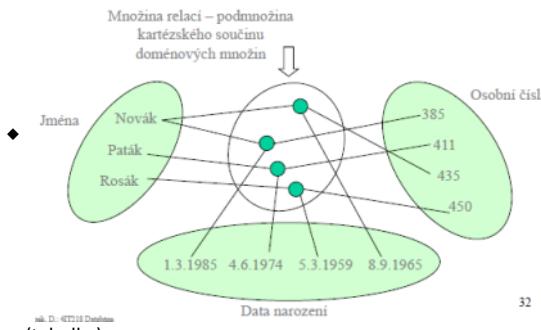
- Kdybych chtěl udělat obě dvě varianty (kombinace left i right join) -> použiji full outer join

Relační model dat

20 February 2024 19:26

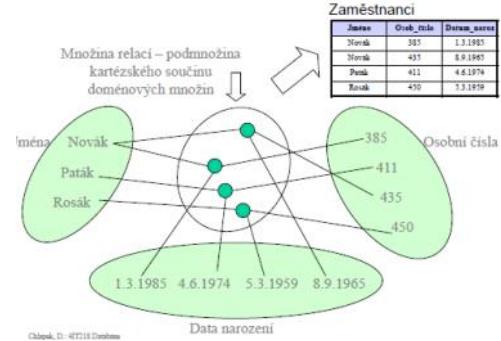
Relační model dat

- Autorem E.F.Codd (publikace 1969 a 1970)
- První komerční implementace 1977 IBM System R
- Model dat založený na predikátové logice prvního řádu
- K manipulaci s daty možno použít relační kalkul nebo operace relační algebry
- Základní pojmy:
 - Doména - množina hodnot stejného významového typu
 - Doména = definiční obor hodnot určitého atributu, vlastnosti nějakého objektu který v tom světě existuje
 - 3 doménové množiny: množina jmen, osobních čísel, dat narození.



32

- vizualizace ve formě relační tabulky



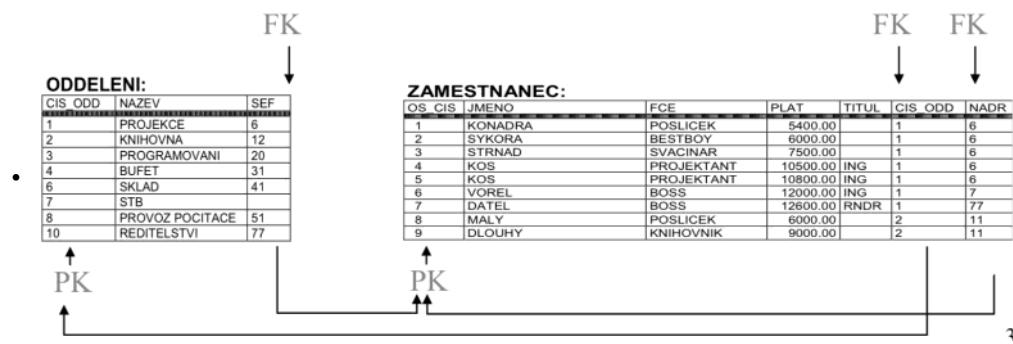
Chlápek, D. - 47718 Databáze

- Relační množina (tabulka)
 - množina vztahů mezi prvky několika domén
 - Podmnožina kartézskeho součinu nad několika doménami D1, D2, ... Dn
 - Nemusí platit $D_i \neq D_j$
 - Relační množina = množina relací = podmnožina kartézskeho součinu doménových množin
- Atribut relace - jméno pro použití hodnoty z domény v relaci
- Relační schéma
 - záhlaví relační tabulky (v čase stálé), obsahuje jméno relační množiny a jména atributů
- Tělo relace (v čase proměnné)
 - obsahuje množinu kombinací hodnot atributů v jedné relaci (n-tic kartézskeho součinu, řádků relační tabulky)
- Zobrazení relační množiny ve formě relační tabulky je možné pouze za předpokladů:
 - Všechny řádky jsou shodné struktury
 - Řádky mají stejný počet hodnot
 - V rámci sloupce jsou všechny hodnoty jednoho typu (z jedné domény)
 - Žádné dva řádky nejsou shodné
 - Nezáleží na pořadí řádků a sloupců
 - Každá relační tabulka má jednoznačný název
 - Každý sloupec v rámci tabulky má jednoznačný název
 - Každá relační tabulka má určený „primární klíč“
 - Všechny data v databázi musí být logicky přístupná kombinací názvu tabulky
- Klíč = atribut (nebo skupina atributů) jehož hodnota (kombinace hodnot) identifikuje prvky relační množiny (řádky v relační tabulce), platí funkci vyhledávací a pořadací
- Primární klíč (PK)
 - Má za úkol odlišit jeden řádek od ostatních řádků
 - = klíč, který splňuje vlastnosti:
 - Jednoznačnost – jednoznačně identifikuje prvek relační množiny (jeden řádek v relační tabulce)
 - Minimálnost – PK se skládá z minimálního počtu atributů, žádný z atributů tvořících PK nelze vyněchat, aniž by byla porušena jednoznačnost PK
 - V praxi se nejčastěji používají umělé klíče (někdo, něco, co se přiřadí k nějakému objektu)
 - Největší problém je identifikovat živé objekty
 - Když pracují s tabulkami, vždycky se nejdříve musí podívat jaké atributy tvoří primární klíč
 - Nemusí být jednoznačný, může být i složený
- Kandidát primárního klíče:
 - může existovat více atributů, které splňují požadavky PK, zvolen může být pouze jeden
 - Např. xname, číslo isicu
- Sekundární klíč (SK):
 - klíč, který identifikuje (potenciálně) množinu řádků (není jednoznačný)
- Cizí klíč (FK):
 - PK použitý v dalším výskytu k vyjádření vazeb mezi objekty zachycenými v relační databázi

Zaměstnanci		
Jméno	Osb._číslo	Datum_narození
Novák	385	1.3.1985
Novák	435	8.9.1965
Paták	411	4.6.1974
Rosák	450	5.3.1959

Relační schéma, možno zapsat také:
Zaměstnanci (Jméno, Osob._číslo, Datum_narození)

Tělo relace



- V relačním prostředí se pro vyjádření vztahů používají cizí klíče = použití klíče k vyjádření vazby
 - Sloupec "šéf" obsahuje čísla zaměstnanců, kteří vedou to oddělení
- Použitím primárního klíče vyjadřujeme cizí klíče -> modelujeme v relačním prostředí vazby

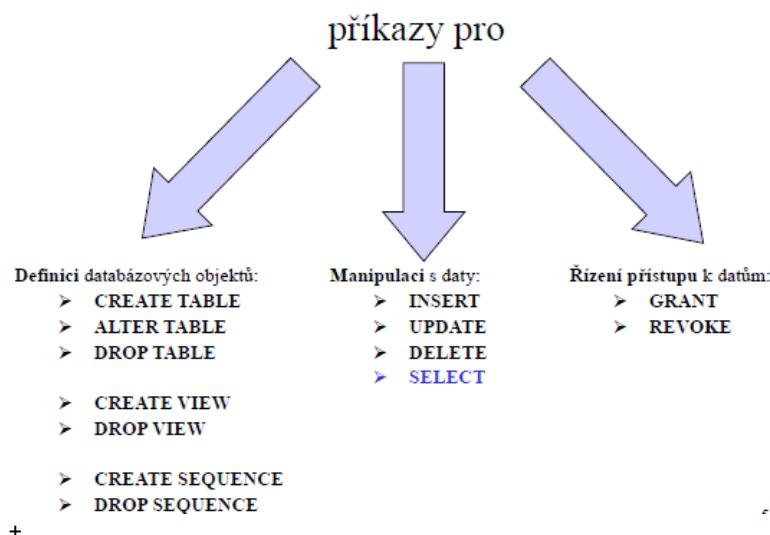
Jazyk SQL

22 February 2024 10:04

Databázové jazyky

- Prostředek komunikace člověka s databázovým systémem.
- Nejčastěji rozlišovány části databázového jazyka:
 - DDL (Data Definition Language)
 - definice databázových objektů, příkazy **Create, Drop, Alter**
 - DML (Data Manipulation Language)
 - manipulace s daty, příkazy **Select, Insert, Update, Delete,**
 - DCL (Data Control Language)
 - řízení přístupu k datům, příkazy **Grant, Revoke**
- Databázových jazyků celá řada (Quel, Progress 4GL, ...).
- V současné době jednoznačně nejrozšířenější databázový jazyk SQL
- Jazyk SQL byl postupně silně standardizován organizacemi (ISO, IEC, ANSI)

Databázový jazyk SQL



DDL (Data Definition Language)

- Příkazy pro definici databázových objektů
- Základním databázovým objektem jsou tabulky:

Druhy relačních tabulek (Tables)

- Tabulky = základní databázový objekt
 - TRVALÉ
 - KMENOVÉ (ZÁKLADNÍ, „base table“)
 - ◆ Do těchto tabulek se vkládají data
 - ODVOZENÉ:
 - ◆ STATICKE („snapshot“)
 - ◇ Časový snímek stavu tabulky v databázi
 - ◆ DYNAMICKE (VIEW, „průhledy“, „pohledy“)
 - ◇ Vidíme podmnožiny tabulek
 - ◇ Nemá vlastní data, přesto můžu přes view vkládat data do tabulek, ze kterých view bylo odvozeno (pokud jsou dodržena pravidla)
 - ◇ V semestrální práci používáme pro kontrolu konzistence (nad rámec standartních integritních omezení)
 - DOČASNÉ
- Základní příkazy **CREATE, ALTER, DROP**
 - Objekty databázového schématu

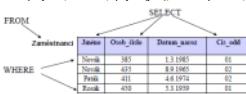
- DB schéma je kolekce logických struktur dat nebo databázových objektů.
- Schéma je vlastněno uživatelem databáze a má název shodný se jménem uživatele. Každý uživatel vlastní jedno schéma.
- DB Objekty jsou vytvářeny a upravovány pomocí jazyka SQL nebo s pomocí dalších nástrojů (např. v dbs ORACLE to může být Enterprise Manager, SQL Developper, ..).
- Seznam typů **databázových objektů** (modře vyznačeny ty, které budou předmětem práce na cvičeních):
 - **Tables**
 - **Views**
 - **Synonyms**
 - **Constraints**
 - **Sequences**
 - **Database triggers**
 - Clusters
 - Database links
 - Dimensions
 - External procedure libraries
 - Index-organized tables
 - Indexes
 - Indextypes
 - Java classes, Java resources, Java sources
 - Materialized views
 - Materialized view logs
 - Object tables
 - Object types
 - Object view
 - Operators
 - Packages
 - Stored functions, stored procedures
- Pozn.: V databázových systémech existuje i řada typů objektů, které nejsou uloženy do databázového schématu, např. Users, Roles, Tablespaces, Rollback segments.
-

DML - SELECT

22 February 2024 10:04

Klausule

- slouží pro výběr a zobrazení požadovaných dat z databáze,
- výsledek je vždy jediný řádek nebo soubor řádků,
- v příkazu SELECT může být pouze CO, nelze výstupem nikoliv JAK toho má být dosaženo,
- příkaz SELECT umožňuje realizovat operace R4:
 - projekce, restrikce, spojení (join), kartézský součin, průnik, rozdíl, sjednocení, agregační funkce (max, min, avg, count, ...).



Klausule SELECT:

- SELECT Co? Jaké sloupec?
- FROM Z čeho? Z jakých tabulek?
- (WHERE) Platí-li? Za jakých podmínek?
- (GROUP BY)
- (HAVING)
- (ORDER BY)

Pouze klausule SELECT a FROM jsou povinné!

Základní syntaxe:

- SELECT (ALL | DISTINCT)
 - (*) | (název sloupu | výraz) | AS název | , název sloupcu AS název, ...)
 - FROM klausule
 - [WHERE klausule]
 - [GROUP BY klausule]
 - [HAVING klausule]
 - [ORDER BY klausule]
- FROM: název tabulky resp. view (tabulek) pro výběr
- klausule = (název-relace (alias-název)) [...]
- WHERE: jedna či více výběrových podmínek spojených logickými operátory AND resp. OR :
- klausule
 - = výraz1 relační operator výraz2
 - test (ne)výnosnosti hodnoty výrazu1 a výrazu2
 - = výraz1 NOT BETWEEN výraz2 AND výraz3
 - test na interval vymezený hodnotami výrazů 2 a 3
 - = název-sloupcu IS (NOT) NULL
 - test na (ne)výsledku hodnotu
 - = název-atributu (NOT) LIKE "fetézec" [ESCAPE "znak"]
 - test (ne)existence fetézce
 - = výraz [NOT] IN (význam-hodnot | vložený výraz) SELECT
 - test (ne)výnosnosti výrazu s hodnotami významu vloženým vloženým příkazem
- SELECT -> hodnoty ve výčtu oddělit čárkou, znakové hodnoty větš v apostrofach
- GROUP BY: určení skupiny a případné výběrové podmínky pro skupinu
- klausule = výčet-sloupců [HAVING výběrová podmínka aplikovaná na skupinu řádků]
- ORDER BY: určení kritéria na řazení řádků výstupu tabulky
- klausule = (výčet-sloupců | potřeb.) [ASC | DESC] [...]

ZAM	ODD
01 Matěj	00
02 Tomáš	00
03 David	NULL
04 Pavla	00
05 Hana	00

CIS	JM	CIS_VED	PLAT	CIS_0000
01	Matěj	00	16000	12
02	Tomáš	00	25000	NULL
03	David	NULL	25500	13
04	Pavla	00	18000	12
05	Hana	00	36000	13

Klausule SELECT

Vypsaní všech sloupců:

```

SELECT *
FROM zam;
  
```

▪ Vybere všechny sloupce z tabulky zaměstnanci

CIS	JM	CIS_VED	PLAT	CIS_0000
01	Matěj	00	16000	12
02	Tomáš	00	25000	NULL
03	David	NULL	25500	13
04	Pavla	00	18000	12
05	Hana	00	36000	13

Odstranění duplicit:

```

SELECT DISTINCT CIS_ODD
FROM zam;
  
```

- Vypíše všechny hodnoty ze sloupcu cis_odd tak, že se hodnoty neopakují
- | CIS_ODD |
|---------|
| 12 |
| 13 |
| 14 |

- DISTINCT = odstranění duplicit
- Když nenapsal distinct, tak mi to vypíše sloupec těchto hodnot: 12, null, 13, 12, 13 (celkem 5 řádků)

Přejmenování sloupců:

```

SELECT cis, jm Jméno, cis_ved AS "Číslo vedoucího", plat * 12 "Roční plát"
FROM zam;
  
```

▪ Vypíše hodnoty ze sloupu jm jako jméno, cis_ved jako Číslo vedoucího atp.

CIS	JM	CIS_VED	PLAT	CIS_0000
01	Matěj	00	16000	12
02	Tomáš	00	25000	NULL
03	David	NULL	25500	13
04	Pavla	00	18000	12
05	Hana	00	36000	13

SELECT a AGREGAČNÍ FUNCE

- Do selectu můžeme přidávat agregační funkce

- Lze použít na všechny řádky tabulky, nebo na skupiny řádků - viz dalej klausule GROUP BY.
 - AVG ((DISTINCT x) | x)
 - průměr hodnot výrazu x
 - MAX ((DISTINCT x) | x)
 - maximální hodnota výrazu x
 - MIN ((DISTINCT x) | x)
 - minimální hodnota výrazu x
 - SUM ((DISTINCT x) | x)
 - součet hodnot výrazu x
 - COUNT ((DISTINCT x) | x)
 - počet určených (různých) hodnot atributu x
 - COUNT(*)
 - počet řádků
 - klausule DISTINCT eliminuje před vloženou agregační hodnotou výrazu x
- Nejdůležitější je COUNT = spočítá počet řádků, hodnot
- SELECT COUNT(*), COUNT(cis_odd), COUNT(DISTINCT cis_odd), MAX(plat)
- ```

SELECT COUNT(*), COUNT(cis_odd), COUNT(DISTINCT cis_odd), MAX(plat)
FROM zam;

```
- Vypočítá počet všech řádků v tabulce Zam
- COUNT(cis, odd): počet vložených (určených hodnot = nejsou NULL) ve sloupci cis\_odd
- COUNT(DISTINCT cis, odd): kolik různých hodnot je ve sloupci cis\_odd
- MAX (plat): vypočítá nejvyšší hodnotu ze sloupu "PLAT"
- | COUNT(*) | COUNT(cis_odd) | COUNT(DISTINCT cis_odd) | MAX(plat) |
|----------|----------------|-------------------------|-----------|
| 5        | 4              | 3                       | 36000     |

#### Díky COUNT si můžu zjistit jestli mám použit inner nebo outer join

- Vidím že mám 5 zaměstnanců, ale mám jenom 4 vyplňené hodnoty ve sloupci cis\_odd -> kdykoliv budu potřebovat JOIN na sloupeček cis\_odd z tabulky zaměstnanci (např. s tabulkou oddělení), musím použít OUTER join (jinak při použití inner join bych nikdy nevypsal všechny zaměstnance, protože by tam chyběl ten co má neurčenou hodnotu ve sloupci cis\_odd)

### Klausule FROM

- FROM: název tabulky resp. view (tabulek) pro výběr klausule = (název-relace (alias-název)) [...]

```

SELECT *
FROM zam, zam zam;

```

◦ Přejmenování tabulky zam -> Zaměstnanci (tady nelze použít AS, lze použít pouze mezera)

| ZAM | JM    | CIS_VED | PLAT  | CIS_0000 |
|-----|-------|---------|-------|----------|
| 01  | Matěj | 00      | 16000 | 12       |
| 02  | Tomáš | 00      | 25000 | NULL     |
| 03  | David | NULL    | 25500 | 13       |
| 04  | Pavla | 00      | 18000 | 12       |
| 05  | Hana  | 00      | 36000 | 13       |

#### Jednu tabulku můžu použít kolikrát budu chtít, ale musím si zajistit jednoznačnou identifikaci sloupců

```

SELECT DISTINCT CIS_ODD
FROM zam, zam zam;

```

◦ Máme 2 tabulky -> tabuku zam, a tabuku zam2, která je identická s tabulkou zam

◦ Přejmenování tabulky zam a kartézský součin tabulky zam a zam2, 25 řádků, 10 sloupců

### Klausule WHERE

- WHERE: jedna či více výběrových podmínek spojených logickými operátory AND resp. OR :

- Klausule:
  - = výraz1 relační operátor výraz2
  - = test na (ne)prázdnost výrazu a výrazu
  - = výraz1 [NOT] BETWEEN výraz AND výraz3
    - test na interval vymezený hodnotami výrazů 2 a 3
  - = název-sloupu IS [NOT] NULL
    - test na (ne)určenou hodnotu

• Posloupnost vyhodnocování:  
 (\*, /, -, NOT AND OR)

```

SELECT *
FROM zam
WHERE (plat * 12) < 350000 OR plat BETWEEN 25000 AND 35000 OR cis_ved IN (02, 03);

SELECT *
FROM zam
WHERE cis_ved IS NOT NULL;
 • Vybírá všechny sloupcy z tabulky zaměstnanců, a vypisuje pouze takové řádky, které neobsahují neurčenou hodnotu (= obsahují pouze určené hodnoty)
 => Zaměstnanci, kteří mají určeného vedoucího

SELECT *
FROM zam
WHERE cis_ved IS NULL;
 • => Zaměstnanci, kteří NEmají určeného vedoucího

```

#### Klausule WHERE a LIKE

- WHERE: jedna či více vyběrových podmínek spojených logickými operátory AND resp. OR:
  - o klausule == název-atributu [NOT] LIKE "řetězec" [ESCAPE "znak"]
  - o zástupné znaky:
    - % - libovolný počet libovolných znaků
    - \_ (podtržko) - jeden libovolný znak
  - Závisí na velikosti písmen v jednoduchých uvozovkách!!

```

SELECT *
FROM zam
WHERE jm LIKE '%y%';
 •
 • SELECT *
 FROM zam
 WHERE jm LIKE 'Now_';
 •

```

#### Klausule GROUP BY

- Shuknu řádky podle určitého sloupu do nějakých skupin hodnot
- GROUP BY: určení skupiny a případné vyběrové podmínky pro skupinu
- Skupina řádků = množina řádků relační tabulky se stejnou hodnotou určeného atributu (atributu)
- 1 skupině řádků odpovídá na výstupu právě jeden řádek
- Nelze konstruovat dotazy přes skupiny řádků a současně se odkazovat "dovnitř" skupiny.
- Pro řádky obsahující neurčenou hodnotu se vytváří samostatná skupina řádků.
- SELECT cis\_odd, COUNT(\*), COUNT(cis\_ved)
 FROM zam
 GROUP BY cis\_odd;
 •
- Chybou:
  - o SELECT cis\_odd, COUNT(\*), COUNT(cis\_ved), plat
 FROM zam
 GROUP BY cis\_odd;
- Pozn:
  - o Lze vytvářet i víceúrovnové skupiny řádků - v klausuli GROUP BY je uvedeno více atributů (názvů sloupců) oddělených čárkou.
  - o Při použití GROUP BY jsou řádky na výstupu tříšeny VZESTUPNĚ. Pokud je třeba jiné třídění, nutno použít klausuli ORDER BY.
- Když použiju Group by, tak můžu použít jen klausuli select, jen to co je použito v klausuli group by a agregační funkce

#### Klausule GROUP BY a HAVING

- Having = když chceme aplikovat podmínu na skupiny/řádky
- Having má smysl až po group by
- Agregované hodnoty (výsledky agregačních funkcí - count, sum, avg, ...) použijte ve vyběrových podmínkách.
- SELECT cis\_odd, COUNT(cis)
 FROM zam
 GROUP BY cis\_odd
 HAVING COUNT(cis) > 1;
 •
- Pořadí správné - viz pořadové čísla
  - o SELECT cis\_odd, COUNT(cis)
 FROM zam
 1. WHERE PLAT > 25000
 2. GROUP BY cis\_odd
 3. HAVING COUNT(cis) > 1;
 •

#### Klausule ORDER BY

- Třídění řádků
- Základní třídění:
  - o ASC - vzestupné, defaultní, není nutno explicitně uvádět;
  - o DESC - sestupné třídění;
  - o je možné třídit podle více sloupců a s různými způsoby třídění, sloupec musí být oddělený čárkou;
  - o místo názvu sloupce je možno uvést číslo sloupce, které odpovídají pořadí sloupců
- Sestupné třídění čísla oddělení:
  - o SELECT \*
 FROM zam
 ORDER BY cis\_odd DESC;
 •
- Třídění podle tří sloupců, podle cis\_odd sestupně, podle cis\_ved a jm vzestupně
  - o SELECT \*
 FROM zam
 ORDER BY cis\_odd DESC, cis\_ved, 2;
 •

JON  
 Spojení dvou a více tabulek (základní table, či odvozených view). Počet tabulek není omezen.

- Při spojení více než 2 tabulek se provede spojení nejprve dvou tabulek a následně se výsledek spojení spojí s další tabulkou atd. Pořadí spojovaných tabulek určuje u některých dbms optimalizátor dbms (viz samostatná přednáška).
- Operace join je složena z operací kartezského součinu a restrikce (viz výklad operaci RA)
- Jedinomocnost jmen sloupců se zajíždí předřazením názvu spojované tabulky, " " a názvem sloupce, např. zam.cis.

- Rozlišujeme:
  - o EQUI JOIN
    - případ, kdy spojovací podminka je ="
  - o SELF JOIN
    - join, když je tabulka spojována sama se sebou
  - o INNER JOIN
    - také označován jako jednoduchý join. Výsledkem jsou pouze řádky, které odpovídají spojovací podmínce.
  - o OUTER JOIN
    - vrácí řádky, které využívají spojovací podmínku a některé řádky (v závislosti na typu outer join LEFT, RIGHT, FULL), které spojovací podmínce nevyhovují.

#### JOIN INNER a EQUI

| ZAM |      | ODD     |       |         |       |       |
|-----|------|---------|-------|---------|-------|-------|
| CIS | JM   | CIS_VED | PLAT  | CIS_ODD | NAZEV | PATRO |
| 01  | Marc |         | 12345 | 11      |       |       |
| 02  | Mark | 00      | 21500 | 102     |       |       |
| 03  | Dave |         | 21500 | 11      |       |       |
| 04  | Paul | 05      | 18900 | 12      |       |       |
| 05  | John | 07      | 18900 | 12      |       |       |

• Spojení dvou a více tabulek (základní table, či odvozených view).

```

SELECT *
FROM zam, odd
WHERE zam.cis_odd = odd.cis_odd;
 •

```

- Starší forma zápisu = kartézský součin a doplnění **restriktivní podmínky**: NEPOUŽÍVAT!!
  - o SELECT \*
 FROM zam, odd
 WHERE zam.cis\_odd = odd.cis\_odd;
 •

- Vhodnější forma zápisu operace join:
  - o Použíti v případě, že se sloupcy jmennu různě -> musim použít kontextové jméno nazev.tabulky.nazev\_sloupcu abychom zajistili jednoznačnou identifikaci sloupcu

- SELECT \*  
FROM zam JOIN odd ON zam.cis\_odd = odd.cis\_odd;
- Použití v případě, že se sloupec přes které spojují jmenuji shodně. Funguje jako přirozený join, tj. ve výstupní tabulce je uveden jeden sloupec:

  - SELECT \*  
FROM zam JOIN odd USING (cis\_odd);

| CIS | JM     | CIS_VED | PLAT  | CIS_ODD |
|-----|--------|---------|-------|---------|
| 01  | Hornák | 02      | 15600 | 12      |
| 02  | Dražek | 03      | 21000 | 12      |
| 03  | Dutý   | 04      | 18900 | 12      |
| 04  | Petrů  | 05      | 18900 | 12      |

• Když napíšu do příkazu jenom slovo JOIN, tak se automaticky bere jako INNER JOIN-> do výstupu se dostanou pouze takové řádky, kde je splněna ta joinovací podmínka

- Např. číslo oddělení z tabulky zam se rovná hodnotám v tom sloupci odd.cis\_odd (= spojujeme to přes dva červené označené sloupečky)
- Když chom ctihlavou vypsat všechny zaměstnance a k nim oddělení, tak při použití JOIN (= inner join) výstup nebude obsahovat všechny zaměstnance -> bude obsahovat jen ty zaměstnance, kteří mají ve sloupci cis\_odd určenou hodnotu -> MUSÍM použít OUTER JOIN!!!
- Použití OUTER JOIN - dotazy typu: pro každého zaměstnance, pro každého zaměstnance, vypíše patřa kde sídlí oddělení apod.

▪ Vypíše po každé oddělení počet zaměstnanců - musím použít OUTER JOIN (při použití jednoduchého joinu by to nevpysalo oddělení 14, protože v tomto odd nepracuje žádný zaměstnanec)

#### JOIN v kombinaci s GROUP BY.

- SELECT cis\_odd, nazev, SUM (plat)  
FROM zam JOIN odd USING (cis\_cis)  
GROUP BY cis\_odd, nazev;

| CIS_ODD | NAZEV | SUM (PLAT) |
|---------|-------|------------|
| 12      | Glosa | 14400      |
| 13      | Obrub | 18900      |

#### SELF JOIN

- SELECT zam.cis, zam.jm, zam.plat, sef.plat  
FROM zam JOIN zam sef ON zam.cis\_yed = sef.cis  
WHERE zam.plat > sef.plat;

| zam.cis | zam.jm | zam.plat | sef.plat |
|---------|--------|----------|----------|
| 01      | Hornák | 15600    | 14100    |
| 02      | Dražek | 21000    | 14100    |

#### OUTER JOIN

- Umírá připojit i řádky, které nevyhovují výběrové podmínce operace INNER JOIN.
- K dispozici jsou RIGHT, LEFT, FULL OUTER JOIN.

#### OUTER JOIN

- ke KAŽDÉMU zaměstnanci potřebujeme vypsat údaje o odděleních, ve kterých zaměstnanec pracuje.

◦ SELECT \*  
FROM zam LEFT JOIN odd USING (cis\_odd);

- → 5 řádků, 7 sloupců (5 + 3 - 1 díly USING)

| CIS | JM     | CIS_VED | PLAT  | CIS_ODD | PLAT1 | PLAT2 |
|-----|--------|---------|-------|---------|-------|-------|
| 01  | Hornák | 02      | 15600 | 12      | 15600 | 12    |
| 02  | Dražek | 03      | 21000 | 12      | 21000 | 12    |
| 03  | Dutý   | 04      | 18900 | 12      | 18900 | 12    |
| 04  | Petrů  | 05      | 18900 | 12      | 21000 | 12    |

◦ Left a right join znamená pořadí od počtu od klíčového slova join

◦ Left join = vlevo od klíčového slova join -> Dávám příkaz: spoj normálně ty tabulky zam a odd, a potom přidej k té tabulky vlevo ty řádky, které nevyhovely joinovací podmínce

#### x INNER JOIN

- Inner join neuumožní vypsat všechny (KAŽDÉHO) zaměstnance, ale pouze ty řádky, které vyhovují podmínce ZAM.CIS\_ODD=ODDEL.CIS\_ODD, tj. ty zaměstnance, kteří mají určené číslo oddělení.

◦ SELECT \*  
FROM zam JOIN odd USING (cis\_odd);

- → 4 řádky, 7 sloupců (5 + 3 - 1 díly USING)

| CIS | JM     | CIS_VED | PLAT  | CIS_ODD | PLAT1 | PLAT2 |
|-----|--------|---------|-------|---------|-------|-------|
| 01  | Hornák | 02      | 15600 | 12      | 15600 | 12    |
| 02  | Dražek | 03      | 21000 | 12      | 21000 | 12    |
| 03  | Dutý   | 04      | 18900 | 12      | 18900 | 12    |

#### • FULL OUTER JOIN

- Dobírá řádky které nevyhovují joinovací podmínce z obou dvou směrů (= z obou dvou tabulek)

◦ Do jedné tabulky spojíme zaměstnance i údaje o odděleních. Vypíše se data, která vyhovují join podmínce a z každé tabulky se doplní také řádky, které join podmínce nevyhověly.

#### ◦ SELECT \* FROM zam FULL JOIN odd USING (cis\_odd);

- → 6 řádků, 7 sloupců (5 + 3 - 1 díly USING)

| CIS | JM     | CIS_VED | PLAT  | CIS_ODD | PLAT1 | PLAT2 |
|-----|--------|---------|-------|---------|-------|-------|
| 01  | Hornák | 02      | 15600 | 12      | 15600 | 12    |
| 02  | Dražek | 03      | 21000 | 12      | 21000 | 12    |
| 03  | Dutý   | 04      | 18900 | 12      | 18900 | 12    |
| 04  | Petrů  | 05      | 18900 | 12      | 21000 | 12    |
|     |        |         |       |         | 15600 | 12    |
|     |        |         |       |         | 21000 | 12    |

#### ◦ OUTER JOIN A AGREGACE

##### ◦ INNER JOIN

- Pozn.: Tento typem operace join není možno realizovat dataz, který požaduje vypsat agregaci ke všem oddělením.

◦ SELECT cis, cis\_yed, nazev, SUM (plat), COUNT (cis)  
FROM zam JOIN odd USING (cis\_odd);

◦ GROUP BY cis\_odd, nazev;

| CIS_ODD | NAZEV | SUM (PLAT) | COUNT (CIS) |
|---------|-------|------------|-------------|
| 12      | Glosa | 14400      | 1           |

##### ◦ OUTER JOIN

◦ SELECT cis\_odd, nazev, SUM (plat), COUNT (cis)  
FROM zam RIGHT JOIN odd USING (cis\_odd);

◦ GROUP BY cis\_odd, nazev;

| CIS_ODD | NAZEV | SUM (PLAT) | COUNT (CIS) |
|---------|-------|------------|-------------|
| 12      | Glosa | 14400      | 1           |

◦ Pozn: cis je primární klíč v tabulce zaměstnanců -> chcí se zistit zaměstnance -> scítání přes primární klíč!!

#### STRUKTUROVANÉ DOTAZY

- Jednoduše - „zvěnit ven“, vnitřní dotaz vyhodnocen a hodnota (hodnoty) dosazený do vnějšího

◦ Související - vnitřní příkaz select je vyhodnocován na základě hodnot v vnějším příkazu SELECT

- Umírá vložit další příkaz SELECT do klauzuli jiného (vnějšího) příkazu SELECT

◦ SELECT ...  
FROM ...  
WHERE ... (SELECT ...  
FROM ...  
WHERE ...);

#### ◦ Bez strukturovaných dotazů:

- Který zaměstnanec má plat nižší než zaměstnanec Dutý z oddělení číslo 13?

◦ 1. Dotaz, který vypíše jaký plat má zaměstnanec Dutý pracující v oddělení číslo 13:

◦ SELECT plat  
FROM zam  
WHERE jm LIKE 'Dutý' AND cis\_odd = 13;

| PLAT  |
|-------|
| 21000 |

◦ 2. Doplňme číslo, které nám vypsal první dotaz:

◦ SELECT \*  
FROM zam  
WHERE plat < 23 500;

| CIS | JM     | CIS_VED | PLAT  | CIS_ODD |
|-----|--------|---------|-------|---------|
| 01  | Hornák | 02      | 15600 | 12      |
| 04  | Petrů  | 05      | 18900 | 12      |

◦ S použitím vloženého dotazu: Který zaměstnanec má plat nižší než zaměstnanec Dutý z oddělení číslo 13?

◦ SELECT \*  
FROM zam  
WHERE plat < (SELECT plat  
FROM zam  
WHERE jm LIKE 'Dutý' AND cis\_odd = 13);

| CIS | JM     | CIS_VED | PLAT  | CIS_ODD |
|-----|--------|---------|-------|---------|
| 01  | Hornák | 02      | 15600 | 12      |
| 04  | Petrů  | 05      | 18900 | 12      |

◦ Lze i bez vloženého dotazu:

◦ SELECT \*  
FROM zam  
WHERE zamduuty.jm like 'Dutý' and  
zamduuty.cis\_odd=13  
and zam.plat < zamduuty.plat

◦ Počet ūrovnění není normou omezen

◦ Rozlišujeme strukturované dotazy:

- JEDNOUCHÉ - „zvěnit ven“, vnitřní dotaz vyhodnocen a hodnota (hodnoty) dosazený do vnějšího
- SOUVTÁZNÉ - vnitřní příkaz SELECT je vyhodnocován na základě hodnot v vnějším příkazu SELECT

#### ◦ Typy SUBDOTAZŮ:

- vložený dotaz vrátí MNÖZNU HODNOT - musí být uvozeny klíčovými slovy IN, ALL, ANY, SOME
  - ALL - výsledek musí být pravidliv pro všechny hodnoty vloženého dotazem
  - ANY - pravidliv alespoň pro jednu vloženou hodnotu
  - množnost dotaz vrátí MAXIMÁLNÉ JEDNU HODNOTU,
  - funguje jako TEST EXISTENCE - klíčové slovo EXISTS,
  - Podmínka EXISTS je vyhodnocena jako pravidliv, je-li výsledkem subdotazu alespoň jeden řádek
  - fungující jako tabulka v klauzuli FROM

#### ◦ IN (množina hodnot)

- Klauzule IN = v množině hodnot

◦ Můžeme se vyhnout dotazu JOIN

◦ SELECT cis, jm

| ZAM |        |         |       |         |
|-----|--------|---------|-------|---------|
| CIS | JM     | CIS_VED | PLAT  | CIS_ODD |
| 01  | Hornák | 02      | 15600 | 12      |
| 02  | Števít | 03      | 21000 | NULL    |
| 03  | Dutý   | 04      | 18900 | 12      |
| 04  | Petrů  | 05      | 18900 | 12      |

| ODD |       |         |       |         |
|-----|-------|---------|-------|---------|
| CIS | JM    | CIS_VED | PLAT  | CIS_ODD |
| 12  | Glosa | 02      | 14400 | 12      |
| 13  | Dutý  | 05      | 21000 | 12      |

- fungující jako tabulka v klauzuli FROM

#### IN (množina hodnot)

- Klauzule IN = v množinu hodnot
  - Můžeme se vyhnout dotazu JOIN
- ```
SELECT cis, jm
FROM zam
WHERE cis_odd IN (SELECT cis_odd
                   FROM odd
                   WHERE patro = 1);
```

ZAM	odd
01 Novák	02
02 Nováček	03
03 Nováček	04
04 Novák	05
05 Novák	06

odd	cis	jm
02	Novák	X
03	Nováček	+
04	Nováček	+
05	Novák	+
06	Novák	+

- Lze i bez vnořeného dotazu:

```
SELECT cis, jm
FROM zam
JOIN odd USING (cis_odd)
WHERE patro = 1;
```

- Zápis s vnořeným selectem je výhodný v tom, že:
 - odpovídá vše běžnému psátku ("vypíš mi jména z čísla zaměstnanců, kteří pracují v oddělení, která se nachází v oddělení v prvním patře" -> nejdříve si sjistíme která oddělení sídlí v prvním patře, pak čísla oddělení dosadíme do vnějšího selectu)
 - z příkladu můžeme snadno udelet množinový rozdíl (doplňek té množiny) - to s jinou nejdé

- Z IN uděláme jednoduše NOT IN, ale s jinoum to nejdé:

```
SELECT cis, jm
FROM zam
WHERE cis_odd NOT IN (SELECT cis_odd
                       FROM odd
                       WHERE patro = 1);
```



```
SELECT cis, jm
FROM zam
WHERE cis_odd = (SELECT cis_odd
                  FROM odd
                  WHERE patro = 1);
```



Čísla a jména zaměstnanců kteří sedí v odděleních která sídlí v jiných patrech než je patro číslo 1

Relační operátory

- Vnořený dotaz musí vracet pouze jednu hodnotu!

- Místo IN můžeme napsat "=" -> v ten okamžik databázový systém předpokládá, že ten vnořený select vraci jenom jednu hodnotu
 - Když v databázi v tabulce oddělení bylo více oddělení, která se liší jenom číslem, ale název je stejný (např. obchod) -> vnořený dotaz by vrátil více hodnot než jednu -> dotaz skončí chybou
 - Je lepší místo operátoru "=" používat NOT IN!!

SELECT cis, jm

```
FROM zam
WHERE cis_odd = (SELECT cis_odd
                  FROM odd
                  WHERE nazev like 'Obchod');
```

```
SELECT cis, jm
FROM zam
WHERE plat > (SELECT AVG(plat)
                 FROM zam);
```

- Tady mám jistotu, že mi vnořený select vráti jen jedno číslo (protože počítám průměr za všechny zaměstnance)

```
o Nelze:
  SELECT cis, jm
  FROM zam
  WHERE cis_odd = (SELECT cis_odd
                     FROM odd);
```

- Když si nejsme jisti že nám vnořený dotaz vraci jen jednu hodnotu, tak máme možnost uvést ten operátor nějakým klíčovým slovem:
 - All
 - Any
 - Some

ALL, ANY (SOME)

- ALL
 - výsledek porovnání musí být pravdivý pro všechny hodnoty vrácené subdotazem
- ANY (= SOME)
 - pravdivost alespoň pro jednu vrácenou hodnotu

- Čísla a jména zaměstnanců z oddělení číslo 12, kteří mají plat vyšší než kterýkoliv zaměstnanec (každý ze zaměstnanců z oddělení 12)
 - Plat je větší než každý z hodnot kterou vraci ten vnořený dotaz -> než každý z plátů zaměstnanců kteří pracují v oddělení 12

```
SELECT cis, jm
FROM zam
WHERE cis_odd = 12
AND plat > ALL (SELECT plat
                  FROM zam
                  WHERE cis_odd = 12);
```

Stejný zápis bez "> ALL" -> select max(plat)
 SELECT cis, jm
 FROM zam
 WHERE cis_odd = 12
 AND plat > (SELECT max(plat)
 FROM zam
 WHERE cis_odd = 12);

= ANY ⇔ IN

- Čísla a jména zaměstnanců z oddělení číslo 12, kteří mají plat větší než střední z oddělení 13.


```
SELECT cis, jm
FROM zam
WHERE cis_odd = 12
AND plat > ANY (SELECT plat
                  FROM zam
                  WHERE cis_odd = 13);
```

- Použití "ANY" je stejně jako IN
 - Používat IN!!
 - Vybírám do výstupu takové řádky, ve kterých se vyskytuje hodnota, která je v množině hodnot, kterou mi vraci něco v té závorce (výčet hodnot nebo vnořený select)

Závislé subdotazy

- Počet zpracování vnořeného dotazu fidi vnořený dotaz
 - Vnořený select se počítá tolíkřat, kolik hodnot mu dodá vnitřní select
- vnitřní dotaz dodává hodnotu i kdežto zpracovávané řádky relate do vnitřního dotazu, který je pro každou z těchto hodnot opakovaně vyhodnocován
- Jak poznat, že se jedná o závisly subdotaz?
 - Poznáme to tak, že v dotazu, který je vnořený, se odkažujeme na tabulku, která není uvedena v klauzuli from vnořeného selectu

```
SELECT cis, jm
FROM zam
WHERE cis_ved = 03
AND z IN (SELECT patro
           FROM odd
           WHERE zam_cis_odd = odd.cis_odd);
```

• Vypisuji

○ Pořadí zpracování příkazu:
 1. první řádek z tabulky zam,
 2. dosazen hodnoty do vnitřního příkazu SELECT,
 3. výsledek subdotazu zpětně dosazen do vnitřního příkazu SELECT
 4. druhý řádek z tabulky zam...

- Čísla jména a platy zaměstnanců, kteří mají plat vyšší než je průměrný plat v jejich oddělení.


```
SELECT cis, jm, plat
FROM zam vnejsi
WHERE plat > (SELECT AVG(plat)
                  FROM zam
                  WHERE zam.cis_odd = vnejsi.cis_odd);
```

Exists

- Pracuje s výsledkem subdotazu jako celkem
- Podmínka s EXISTS je vyhodnocena jako pravdivá, jeli výsledkem subdotazu alespoň jeden řádek

- Čísla a jména všech vedoucích.


```
SELECT cis, jm
FROM zam vnejsi
WHERE EXISTS (SELECT *
               FROM zam
               WHERE zam.cis_ved = vnejsi.cis);
```

Kombinace strukturovaných dotazů s klauzulemi GROUP BY a HAVING

- Čísla a průměrný plat v oddělení, jehož průměrný plat je vyšší než průměrný plat za celou organizaci (všechny zaměstnance).

```
SELECT cis_odd, AVG(plat)
GROUP BY cis_odd
HAVING AVG(plat) > (SELECT AVG(plat)
                           FROM zam);
```

Poddotazy v klauzuli FROM

- Císla zaměstnanců, jejichž plat je nižší, než průměrný plat v jejich oddělení.

```
SELECT cis
FROM zam JOIN (SELECT AVG(plat) AS prumplat, cis_odd
                FROM zam
                GROUP BY cis_odd)
                ON cis = cis_odd
WHERE cis < prumplat;
```

Neboli:

```
SELECT cis
FROM zam JOIN (SELECT AVG(plat) AS plat, cis_odd
                FROM zam
                GROUP BY cis_odd) prumery
                USING (cis_odd)
WHERE zam.plat < prumery.plat;
```

SQL a základní množinové operace

kartézský součin	<pre>SELECT * FROM zna, odd;</pre>
sjednocení	<pre>SELECT * FROM beh UNION SELECT * FROM skok;</pre>
průnik	<pre>SELECT * FROM beh INTERSECT SELECT * FROM skok;</pre>
rozdíl (v normě EXCEPT, v ORACLE MINUS)	<pre>SELECT * FROM beh MINUS SELECT * FROM skok; nebo také v použití subdotazů s NOT EXISTS nebo s NOT IN</pre>

U rozdílu primárně používat NOT IN!

DML - INSERT

25 February 2024 21:05

- INSERT = vložení nového řádku do definované tabulky:
 - INSERT INTO název-tabulky [(výčet-sloupců)]
 - {{VALUES (výčet-hodnot)} | SELECTpríkaz}
- varianta VALUES **vkládá jeden řádek**, resp. hodnoty do jednoho řádku
- Uvést seznam sloupců do kterých vkládáme
- **INSERT INTO zam(cis, jm, plat) VALUES (06, 'Rosák', 55500);**
- Pokud by sloupec cis_ved byl povinný (nesmí obsahovat neurčenou hodnotu) -> skončilo by to chybou a příkaz by se nepovedl

ZAM

CIS	JM	CIS_VED	PLAT	CIS_ODD
01	Horák	02	15600	12
02	Nový	03	25500	NULL
03	Dutý	NULL	23500	13
04	Petrů	05	18900	12
05	Nová	03	35600	13
06	Rosák	NULL	55500	NULL

DML - UPDATE

25 February 2024 21:08

- UPDATE název-tabulky SET název-sloupce = výraz [...]
 - [WHERE klausule]1)
 - 1)viz klausule WHERE v popisu syntaxe SELECT příkazu
- Aktualizuje množinu řádků určenou klauzulí where

```
UPDATE zam
SET plat = plat * 1,04
WHERE cis_odd = 13;
```

```
UPDATE zam
SET plat = plat * 1,04
WHERE plat <(SELECT AVG(plat)
    FROM zam);
```

DML - DELETE

25 February 2024 21:05

- DELETE FROM název-tabulky [WHERE klausule]
- Zruší množinu řádků určenou klauzulí WHERE

```
DELETE FROM zam
WHERE cis_odd = 13;
```

```
DELETE FROM zam
WHERE plat < (SELECT AVG(plat)
    FROM zam);
```

```
DELETE FROM zam
WHERE cis_odd IN (SELECT cis_odd
    FROM odd
    WHERE patro = 2);
```

DDL - CREATE TABLE

25 February 2024 21:18

CREATE TABLE - INTEGRITNÍ OMEZENÍ

- Důležité pojmy
 - Konzistence databáze
 - data v databázi zachycují stav, které jsou v popisovaném světě MOŽNÉ
 - Databáze nabývá jen takových stavů, které jsou v tom světě, který se v té databázi zrcadlí, možné
 - *Cvičím si nesmí podávat založit nekonzistentní data do naší databáze!*
 - Integrity databáze
 - databáze je v konzistentním stavu

- Podpora databázových systémů - nástroje pro údržbu databáze v konzistentním stavu, mimo jiné i INTEGRITNÍ OMEZENÍ

- Integrity omezení = pravidla pro zajištění správnosti a konzistence uložených dat. Rozlišujeme 3 typy integritních omezení:
 - Doménová integrita
 - Entitní integrita
 - Referenční integrita

1. Doménová integrita

- zajištění, aby každá hodnota atributu byla v souladu s množinou připustných hodnot
 - = definiční obor doménové množiny
 - Např. hodnota musí být mezi 60-100, musí to být číslo nebo znak, nemůžeme zaměstnance zaměstnat dříve než se narodil,...
- podpora v SQL:
 - od SQL89 klauzule CHECK (Příklad: PLATNUMBER(10,2) check (PLAT between 100 and 60000)
 - od SQL92 CREATE DOMAIN (Příklad: create domain PLATDOM NUMBER(10,2) check PLATDOM between 100 and 60000; v příkazu create table zam pak u sloupce PLAT uveden odkaz na doménu: PLAT PLATDOM)

2. Entitní integrita

- Entita = cokoliv co pro nás má smysl, o čem potřebujeme něco rozlišovat
- Základní princip: zajištění jednoznačné identifikace každého řádku relační tabulky
 - Ve sloupcích tabulky, které označím jako primary key (nemusí být jeden, může být i složený) nesmí být možnost vložit duplicitní hodnotu (databázový systém odmítne vykonat příkaz insert nebo update)
- podpora v SQL:
 - od SQL86 pouze s použitím kombinace klíčových slov UNIQUE a NOT NULL (Příklad: create table ZAM (CIS INTEGER unique not null)
 - od SQL89 PRIMARY KEY (Příklad: create table ZAM (CIS INTEGER primary key, ...)

3. Referenční integrita

- cízí klíč (tj. použití primárního klíče - atributu nebo skupiny atributů jednoznačně identifikujících jednotlivé řádky relační tabulky - pro zachycení vazby mezi daty) nemůže nabývat jiných hodnot, než těch které jsou obsaženy ve sloupcích - atributech tvorících primární klíč
- Vezmu primární klíč z tabulky cis_odd a dám ho do tabulky zaměstnanci, tak říkám že zaměstnanec pracuje v oddělení jehož číslo jsem tam vložil a ostatní údaje o tom oddělení si uživateli dohledej v tabulce oddělení (proto joinujeme)

- Nejdříve musíme nadefinovat vazbu vazbu cízí klíč vs ten klíč na který se odkazuje = reference

- Př. Do tabulky zam nemůžu napsat číslo oddělení 15 (můžu tam napsat pouze čísla oddělení, o kterých je údaj v tabulce odd nebo null - nemá určené oddělení)
 - Když je reference dobré naprogramovaná, tak databázový systém sám hľídá že do cis_odd můžu vložit pouze hodnoty které jsou uvedeny v odkazované tabulce ve sloupci který tvoří PK

- Musíme vyřešit co se bude dít při aktualizaci databáze
 - Př. Chci zrušit oddělení 12 - účtárna

ZAM				
CIS	JM	CIS_VED	PLAT	CIS_ODD
01	Horsk	02	15600	12
02	Nový	03	25500	NULL
03	Durý	NULL	23500	13
04	Petrú	05	18900	12
05	Nová	03	35600	13

ODD		
CIS_ODD	NAZEV	PATRO
12	Účtárna	1
13	Obychod	2
14	Controlling	1

- Druhy ošetření referenční integrity - pro operace UPDATE a DELETE:
 - RESTRICT
 - SET NULL
 - CASCADE

- Update nikdy neměnit z varianty restrict!
- Pro delete si můžeme vybrat ze všech 3

- RESTRICT on delete
 - → když bych chtěl udělat update v tabulce zam kde cis_odd místo 12 bych chtěl napsat 15 -> databázový systém to odmítne vykonat
 - Restrict se tam dává automaticky, nemusí se psát
 - Když bych chtěl oddělení 12 zrušit (delete from odd where cis_odd = 12), protože jsou oddělení v zam už 2x použita (2x je referencováno oddělení číslo 12) -> když napišu restrict tak systém odmítne vykonat příkaz delete
 - Když bych napsala delete from odd where cis_odd = 14 tak systém zruší řádek 14, protože není použit v jiné tabulky
- U varianty CASCADE v ten okamžik databázový systém předpokládá, že jsme navrhli databázi správně
 - Když bych udělal delete na řádek 12 v tabulce odd, tak se vymažou další 2 řádky z tabulky zam
- SET NULL funguje tak, že systém řekne že to je možné použít jen tehdy, když ten sloupec může nabývat i neurčené hodnoty
 - Definoval bych to u tabulky zam
 - Když bych udělal delete na řádek 12 v tabulce odd, tak ten databázový systém sám do sloupce cis_odd v tabulce zam místo 12 napše neurčenou hodnotu null

- podpora v SQL na deklarativní úrovni:
 - od SQL89 pouze restriktivní varianta s použitím klíčového slova REFERENCES (Příklad: create table ZAM (..., CIS_ODD INTEGER references ODD(CIS_ODD) ...;)
 - od SQL92 klíčová slova ON DELETE CASCADE, ON DELETE SET NULL, ON UPDATE CASCADE, ON UPDATE SET NULL, CONSTRAINT (Příklady: viz následující snímky)

CREATE TABLE

```
1 create table ODD (
2   CIS_ODD SMALLINT not null,
3   NAZEV VARCHAR2(40) not null,
4   PATRO CHAR(3) not null,
5   constraint PK_ODD primary key (CIS_ODD)
6 );
```

Restriktivní varianta referenční integrity

```
1 create table ZAM (
2   CIS INTEGER not null,
3   CIS_ODD SMALLINT,
4   CIS_VED INTEGER,
5   JM VARCHAR2(40) not null,
6   PLAT NUMBER(10,2) not null constraint CKC_PLAT_ZAM check (PLAT between 100 and 60000),
7
8   constraint PK_ZAM primary key (CIS),
9
10  constraint FK_ZAM_ZARAZENI_ODD foreign key (CIS_ODD) references ODD (CIS_ODD),
11
12  constraint FK_ZAM_NADRIZ_ZAM foreign key (CIS_VED)
13  references ZAM (CIS)
14 );
```

Není tam napsané "on delete ..." -> je defaultně restriktivní
Když tam není na update nic tak update znamená, že umožníme změnit primární klíč, což bychom neměli nikdy dovolit!

Kaskádová varianta referenční integrity

```
1 create table ZAM (
2 CIS INTEGER not null,
3 CIS_ODD SMALLINT,
4 CIS_VED INTEGER,
5 JM VARCHAR2(40) not null,
6 PLAT NUMBER(10,2) not null constraint CKC_PLAT_ZAM check (PLAT between 100 and 60000),
7
8 constraint PK_ZAM primary key (CIS),
9
10 constraint FK_ZAM_ZARAZENI_ODD foreign key (CIS_ODD) references ODD (CIS_ODD) on delete cascade,
11
12 constraint FK_ZAM_NADRIZ_ZAM foreign key (CIS_VED) on delete cascade
13 references ZAM (CIS)
14 );
```

SET NULL varianta referenční integrity

```
1 create table ZAM (
2 CIS INTEGER not null,
3 CIS_ODD SMALLINT,
4 CIS_VED INTEGER,
5 JM VARCHAR2(40) not null,
6 PLAT NUMBER(10,2) not null constraint CKC_PLAT_ZAM check (PLAT between 100 and 60000),
7
8 constraint PK_ZAM primary key (CIS),
9
10 constraint FK_ZAM_ZARAZENI_ODD foreign key (CIS_ODD) references ODD (CIS_ODD) on delete set null,
11
12 constraint FK_ZAM_NADRIZ_ZAM foreign key (CIS_VED) on delete set null
13 references ZAM (CIS)
14 );
```

DDL - ALTER TABLE

25 February 2024 21:30

- změna definice tabulky, struktury a vlastností sloupců, integritních omezení a dalších vlastností tabulky
- používána klíčová slova ADD, DROP, MODIFY
- příklad zrušení integritních omezení a nadefinování nových

```
alter table ZAM
drop constraint FK_ZAM_NADRIZ_ZAM;

alter table ZAM
drop constraint FK_ZAM_ZARAZENI_ODD;

alter table ZAM
add constraint FK_ZAM_NADRIZ_ZAM foreign key (CIS_VED)
references ZAM (CIS);

alter table ZAM
add constraint FK_ZAM_ZARAZENI_ODD foreign key (CIS_ODD)
references ODD (CIS_ODD);
```

DDL - DROP TABLE

25 February 2024 21:32

- Zrušení tabulky

```
drop table ODD cascade constraints;
```

```
drop table ZAM cascade constraints;
```

DDL - TRIGGER

25 February 2024 21:34

- Trigger = databázový objekt -> leží na serveru a spouští se na serveru
- Sekvence příkazů
- Spouští se při vykonání nějakého příkazu
- Dají se navést na nějakou událost
 - Událostí je použití nějaké operace nad nějakou tabulkou
- Může se s ním hlídat stav zásob na skladě, když to klesá pod určitou úroveň tak se automaticky vystaví např. objednávka
- Používá se pro kontrolu konzistence - můžeme si tím napsat nějaká vlastní omezení
 - Např. ruší oddelení a ty zaměstnance chci přesunout do jiného oddelení
 - Zruší oddelení, a nechci připustit aby zaměstnanec neměl žádné oddelení (každý zaměstnanec musí být v nějakém oddelení - neplatí to null)
 - Např. je přesunu do oddelení HR, kde zůstanou než si je rozeberou ostatní oddelení

INTEGRITNÍ OMEZENÍ S použitím typu objektu TRIGGER

- TRIGGER je pojmenovaná množina příkazů, která bude spuštěna v souvislosti (AFTER, BEFORE) s určitou událostí (použitím určité DML operace, např. INSERT, UPDATE, DELETE) nad určitou částí databáze (nejčastěji relační tabulkou)
- TRIGGER se používá pro zajištění složitějších integritních omezení,
- TRIGGER - databázový systém provádí spouštění automaticky dle definovaných podmínek
- TRIGGER je databázový objekt a proto:
 - je vytvářen příkazem CREATE TRIGGER,
 - je rušen příkazem DROP TRIGGER,
 - je možno provést aktivaci nebo deaktivaci (ENABLE, DISABLE) příkazem ALTER TRIGGER nebo ALTER TABLE

```
1 create trigger ODD_DEL after delete
2 on ODD for each row
3 update ZAM
4     set ZAM.CIS_ODD = NULL
5     where ZAM.CIS_ODD = :old.CIS_ODD;
```

Tady je vlastně napsána referenční integrita Set null

DDL - CREATE SEQUENCE

25 February 2024 21:37

- **Sequence = databázový objekt**
- Automaticky zajistí zvyšování hodnot u nějakých umělých klíčů
- objekty typu SEQUENCE se používá pro nastavení pravidel pro automatické pořadové číslování
- **nejčastěji se používá pro generování hodnot do atributů tvořících umělý primární klíč**
- objekt typu SEQUENCE se používá nezávisle na relační tabulce - z jednoho objektu typu sekvence možno generovat hodnoty PK do více tabulek
- hodnoty v objekty typu SEQUENCE jsou přístupné přes dva pseudosloupce
 - CURRVAL - aktuální hodnota
 - NEXTVAL - nová hodnota
 - Dej mi to číslo, které ještě nebylo použito, dej mi tu další hodnotu v té sekvenci
- ruší se příkazem DROP SEQUENCE, lze měnit příkazem ALTER SEQUENCE
- příklad:

```
create sequence CISLA_ZAM start with 6 increment by 1;
```

```
INSERT INTO zam (cis, jm, plat) VALUES (CISLA_ZAM.NEXTVAL, 'Rosa', 53500);
```

DDL - CREATE VIEW

25 February 2024 21:38

- **dynamické odvozené relační tabulky - průhledy do databáze**
 - Vytvoří se jedna tabulka o zaměstnancích, a nad ní se vytvoří 7 view, pro každého děkana
 - Děkaní mají přístup jenom k zaměstnancům kteří jsou pro jejich fakultu
 - Můžou insertovat, updatovat
 - Personální má přístup ke všem zaměstnancům
 - Přes view řídíme práva k některým řádkům a sloupcům té základní tabulky
- VIEW neobsahuje vlastní data
- VIEW může být odvozeno z objektů typu TABLE nebo z jiných objektů typu VIEW,
- mohou se vrátit do více úrovní,
- VIEW za jistých předpokladů mohou být aktualizovatelná, tj. data ze základních tabulek jsou aktualizovatelná přes VIEW (dohledejte podmínky aktualizovatelnosti view v příručce SQL),
- mohou být použita pro zpřístupnění určité podmnožiny dat z databáze, používají se jako nástroj pro snazší řízení přístupových práv,
- **klaузule with check option umožňuje kontrolovat vkládaná data**

```
create [or replace]view název_view  
as subdotaz  
[with check option];
```

```
create view OBCHODNICI  
as select CIS, JM, PLAT, CIS_VED  
      from ZAM join ODD using (CIS_ODD)  
     where NAZEV like 'Obchod';
```

```
Create view OBCHODNICI2  
as select*  
      From ZAM  
     Where CIS_ODD = 13  
with check option;
```

- Přes view možno vkládat pouze údaje o zaměstnancích z oddelení 13
- Když jsem vytvořil to view, tak já přes to view kontroluji vkládaná data
 - Nedovolím, aby ten kdo dělá insert do view obchodníci2, aby vložil do sloupce cis_odd jinou hodnotu, než 13
 - Kdyby tam to "with check option" nebylo, tak v tu chvíli můžu vložit i oddelení 14, ale když si pak dám select tak to oddelení neuvidím

ZAM

CIS	JM	CIS_VED	PLAT	CIS_ODD
01	Horák	02	15600	12
02	Nový	03	25500	NULL
03	Dutý	NULL	23500	13
04	Petrů	05	18900	12
05	Nová	03	35600	13

ODD

CIS_ODD	NAZEV	PATRO
12	Účtárna	1
13	Obchod	2
14	Controlling	1

OBCHODNICI

CIS	JM	PLAT	CIS_VED
03	Dutý	23500	NULL
05	Nová	35600	03

OBCHODNICI2

CIS	JM	CIS_VED	PLAT	CIS_ODD
03	Dutý	NULL	23500	13
05	Nová	03	35600	13

DDL - ALTER, DROP VIEW

25 February 2024 21:44

- ALTER VIEW umožní změnit omezení (constraints) view
- pro redefinici obsahu view nutno použít příkaz create or replace
- DROP VIEW název_view - ruší view

```
create or replace view OBCHODNICI
as select CIS, JM, PLAT, CIS_VED
from ZAM join ODD using (CIS_ODD)
where NAZEV like 'Obchod';
```

```
drop view OBCHODNICI2;
```

Většinou se alter nepoužívá, většinou se view dropne a vytvoří se nové

DDL - CREATE SYNONYM

25 February 2024 21:45

- Synonyma slouží k vytvoření dvou názvů téhož (dvě slova, která označují tutéž skutečnost)
- objekty typu synonym použity pro vytvoření uživatelsky přívětivějších názvů,
- synonyma se ruší příkazem DROP SYNONYM
- příklad z testovací databáze
 - uživatel IT218 vytvořil tabulku ZAM;
 - uživatel IT218 přidělil práva na operaci SELECT pro tabulku ZAM uživateli STUDENT;
 - uživatel student by bez vytvoření synonyma musel používat plné jméno tabulky = jméno_tvůrce.název_tabulky, tj.:
`select *
from IT218.ZAM;`
 - uživatel student si vytvoří synonymum:
`create synonym ZAM for IT218.ZAM;`
 - uživatel student může používat synonymum ZAM pro přístup k tabulce IT218.ZAM.

DCL

27 April 2024 11:18

- přidělování a odebírání přístupových práv v prostředí jazyka SQL pro
 - systémová oprávnění pro role a uživatele,
 - přiřazení rolí k uživatelům,
 - objektová práva, tj. práva k určitým objektům (např. typu TABLE, VIEW, SEQUENCE, procedura, funkce, balík, UDT) pro určité uživatele, role nebo všechny uživatele (PUBLIC)
- Hodně často se pro přidělování práv používá view (dynamická odvozená relační tabulka)
 - Podmnožina nějaké tabulky
- používají se příkazy
 - GRANT - přidělení práv
 - REVOKE - odebrání práv
- pro potřeby předmětu se budeme zabývat pouze přidělováním práv k vybraným typům objektů -TABLE, VIEW, a to pouze pro uživatele
- Ten kdo vytvoří ten objekt = ten kdo udělá create, ten má práva, která jsou neodebratelná
 - Vlastníkovi tabulky práva nejdou odebrat
 - Jde zrušit ten uživatela, se všemi databázovými objekty, ale nemůžu odebrat (revoke) ta práva tomu uživateli, který udělal ten create
 - Tento uživatel má právo přidělovat práva ostatním uživatelům - grant

GRANT

- vlastník (tvůrce) objektu má práva k objektu všechna a neodebratelná
- vlastník práv k objektu, nebo jím určený (klauzule WITH GRANT OPTION) uživatel může přidělovat práva ostatním uživatelům
- grant {all [privileges]} { select | insert | delete | update [(výčet_atributů)] }
on název_objektu
to {výčet_uživatelů |public}
[with grant option];
- **grant select on ZAM to public;**
- **grant select, update (NAZEV, PATRO) on ODD to student1;**
- **grant all privileges on ZAM to student with grant option;**

REVOKE

- odebrání přidělených práv
- ALL odejmoutí všech práv,
- PUBLIC - odebírá práva přidělená pro všechny uživatele,
- uživatel, který má přidělena práva s volbou WITH GRANT OPTION, může odebírat pouze práva, která sám přidělil,
- odejmoutí práv uživateli, který díky volbě WITH GRANT OPTION předal práva na další uživatele, znamená odejmoutí těchto práv i všem, kteří je od něj získali,
- vlastníkovi (tvůrci) objektu nelze odebrat žádná práva
- revoke {all [privileges]} { select | insert | delete | update [(výčet_atributů)] }
on název_objektu
from {výčet_uživatelů |public};
- **revoke select on ZAM from public;**
- **revoke select, update (NAZEV, PATRO) on ODD from student1;**
- **revoke all privileges on ZAM from student;**

Normy jazyka SQL

27 April 2024 11:24

- Jazyk SQL –postupně standardizován organizacemi:
 - ISO
 - **Mezinárodní organizace pro normalizaci** (International Organization for Standardization), označovaná jako ISO, je světovou federací národních normalizačních organizací se sídlem v Ženevě. Byla založena v roce 1947. Zabývá se tvorbou mezinárodních norem ISO a jiných druhů dokumentů ve všech oblastech normalizace kromě elektrotechniky. Jsou to např. TS –technické specifikace, TR –technické zprávy, PAS –veřejně dostupné specifikace, TTA –dohody o technických trendech.
 - IEC
 - **Mezinárodní elektrotechnická komise** (International Electrotechnical Commission) se sídlem v Ženevě, založena v roce 1906, je světovou organizací, která vypracovává a publikuje mezinárodní normy IEC a jiné druhy dokumentů (technické specifikace -TS, technické zprávy -TR, technické dohody průmyslu -ITA, a veřejně dostupné specifikace -PAS) v oblasti elektrotechnika elektroniky.
 - ANSI
 - **americká standardizační organizace**, sídlící ve Washingtonu (American National Standards Institute). Je to nezisková organizace, která vytváří průmyslové standardy ve Spojených státech. Je členem organizace ISO a IEC. Organizace ANSI byla založena v roce 1918 jako American Engineering Standards Committee a reorganizován na American Standards Association v roce 1928. Roku 1966 byl reorganizován jako United States of America Standards Institute. Roku 1969 bylo změněno jméno na American National Standards Institute.
- Postupný vývoj standardů:

Rok	Označení	Charakteristika
1986	SQL-86	První verze (ANSI / ISO 1987).
1989	SQL-89	Oprava a mírné rozšíření normy jazyka SQL-86.
1992	SQL-92	Označení SQL2, komplexní pohled na jazyk, tři úrovně
1999	SQL:1999	Nepřesně označováno jako SQL3. Výrazné rozšíření jazyka, regulární výrazy, rekursivní dotazy, triggery, objektově orientované vlastnosti
2003	SQL:2003	XML, GUI funkce, multimédia
2006	SQL:2006	Import, export a způsob použití XML dat, vazba na XML Query Language (W3C)
2008	SQL:2008	Kompletní přepracování všech hlavních částí jazyka SQL
2011	SQL:2011	Přepracování verze z 2008 a rozšíření směrem k temporálním databázim
2016	SQL:2016	Revize verzi z 2011

Standardizace databázového jazyka SQL

- ISO (the International Organization for Standardization) (www.iso.org)
- IEC (the International Electrotechnical Commission)
- In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC
- International Standard ISO/IEC9075 was prepared by Joint Technical Committee ISO/IEC JTC1. Information technology, Subcommittee SC32, Data management and interchange.
- ISO/IEC9075 consists of the following parts, under the general title Information technology— Database languages—SQL :
 - Part1: Framework (SQL /Framework)
 - Part2: Foundation (SQL /Foundation)
 - Part3: Call-Level Interface (SQL /CLI)
 - Part4: Persistent Stored Modules (SQL /PSM)
 - Part9: Management of External Data (SQL /MED)
 - Part10: Object Language Bindings (SQL /OLB)
 - Part11: Information and Definition Schema (SQL /Schemata)

- Part13: SQL Routines and Types Using the Java Programming Language (SQL /JRT)
- Part14: XML-Related Specifications (SQL /XML)
- Part15: Multi dimensional arrays (Underdevelopment)

- Platné standardy

- Minimální požadavky

ISO/IEC 9075-1:2016	Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)
ISO/IEC 9075-2:2016	Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation)
ISO/IEC 9075-11:2016	Information technology -- Database languages -- SQL -- Part 11: Information and Definition Schemas (SQL/Schemata)

ISO/IEC 9075-3:2016	Information technology -- Database languages -- SQL -- Part 3: Call-Level Interface (SQL/CLI)
ISO/IEC 9075-4:2016	Information technology -- Database languages -- SQL -- Part 4: Persistent Stored Modules (SQL/PSM)
ISO/IEC 9075-9:2016	Information technology -- Database languages -- SQL -- Part 9: Management of External Data (SQL/MED)
ISO/IEC 9075-10:2008	Information technology -- Database languages -- SQL -- Part 10: Object Language Binding (SQL/OLB)
ISO/IEC 9075-13:2016	Information technology -- Database languages -- SQL -- Part 13: SQL Routines and Types Using the Java TM Programming Language (SQL/JRT)
ISO/IEC 9075-14:2016	Information technology -- Database languages -- SQL -- Part 14: XML-Related Specifications (SQL/XML)
ISO/IEC 13249-1:2016	Information technology -- Database languages -- SQL multimedia and application packages -- Part 1: Framework
ISO/IEC 13249-2:2003	Information technology -- Database languages -- SQL multimedia and application packages -- Part 2: Full-Text
ISO/IEC 13249-3:2016	Information technology -- Database languages -- SQL multimedia and application packages -- Part 3: Spatial
ISO/IEC 13249-5:2003	Information technology -- Database languages -- SQL multimedia and application packages -- Part 5: Still image
ISO/IEC 13249-6:2006	Information technology -- Database languages -- SQL multimedia and application packages -- Part 6: Data mining
ISO/IEC 13249-7:2013	Information technology -- Database languages -- SQL multimedia and application packages -- Part 7: History
ISO/IEC 9579:2000	Information technology -- Remote database access for SQL with security enhancement
ISO 19125-2:2004	Geographic information -- Simple feature access -- Part 2: SQL option

- Stále však přežívá (z důvodů zpětné kompatibility) řada odchylek (dialektů) - implementací jazyka SQL výrobci jednotlivých databázových systémů.

Vlastnosti relačních DBS

27 April 2024 11:29

Relační DBS - min. požadavky

- E. F. Codd –Minimální podmínky relačnosti DBS:
 - všechna data v databázi jsou uložena ve formě relačních tabulek,
 - neexistují uživateli viditelné přístupové cesty (mechanismy, např. nutnost mít vytvořeny indexy),
 - existuje databázový jazyk, který umožní realizovat - v libovolné syntaxi (bez použití příkazů pro iteraci a rekurzi) – operace:
 - Restrikce - vybírám nějaké řádky (where)
 - Projekce - vybírám jen nějaké sloupce (select)
 - Equi Join - data musí být rozfragmentována - musí být co nejbližší tomu světu, objektům, typům, třídám, entitním množinám, které se snažím v databázi zrcadlit
- C. J. Date –specifikoval podrobnější vlastnosti relačních DBS –viz další snímky

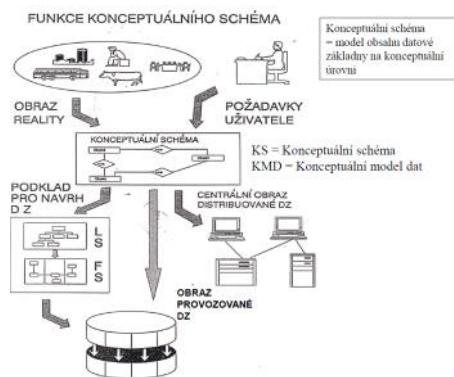
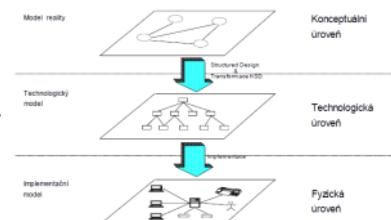
Vlastnosti relačních DBS

1. Všechna data v relační db musí být reprezentována na logické úrovni pomocí relačních tabulek.
2. Všechna data uložená v relační db musí být přístupná pomocí
 - názvu tabulky,
 - názvu sloupce
 - a hodnoty primárního klíče.
 - Primární klíč umožňuje nalézt příslušný řádek
3. DBS musí umožnit použití v databázi i v relačních operacích neurčené hodnoty "null value" nezávisle na typu dat.
4. Popis databáze musí být reprezentován na logické úrovni dynamicky podobně vlastním datům tak, že oprávnění uživatelé mohou použít stejný databázový jazyk k dotazům jak na vlastní data, tak i na jejich popisy, tzv. "metadata".
5. Není podstatné kolik databázových jazyků je pro daný dbs k dispozici, ale nejméně jeden jazyk musí být uživatelský přívětivý (klíčová slova - znakové řetězce, s dobré definovanou syntaxí), musí podporovat jak interaktivní, tak i programový režim a musí umožňovat:
 - definici dat
 - realizaci integritních omezení (entitní integrita, doménová, referenční integrita, další IO)
 - manipulaci s daty (vkládání, aktualizaci, rušení a vyhledávání dat)
 - vytváření dynamických odvozených relací (view)
 - definici transakcí
 - definici přístupových práv
6. DBS musí poskytovat způsob, jak při definici view určit, zda view bude použit pro vkládání, rušení řádků nebo aktualizaci sloupců základních tabulek nad kterými je view vytvářen.
7. DBS musí umožňovat provádět množinové operace s daty v relačních tabulkách nejen při vyhledávání dat, ale i při operacích vkladání, aktualizaci a rušení řádků v tabulkách.
8. Změna způsobu uložení dat v db (fyzická struktura) či metody přístupu k datům nesmí způsobit nutnost změny programu či interaktivně zadávaného příkazu (**fyzická datová nezávislost**).
9. Změna struktury dat v db či metody přístupu k datům nesmí způsobit nutnost změny programu či interaktivně zadávaného příkazu (**logická datová nezávislost**).
 - a. Např. přidání nového sloupce do tabulky by nemělo vyvolat nutnost změny či překladu programu, který s měněnou tabulkou pracuje aniž by používal nový sloupec.
10. Změna integritních omezení definovaných pomocí databázového jazyka a uložených v katalogu dat nesmí způsobit nutnost změny programu či interaktivně zadávaného příkazu.
11. Změna umístění (distribuce) dat nesmí způsobit nutnost změny programu či interaktivně zadávaného příkazu.
12. Pokud má DBS má nízkourovňový (procedurální) jazyk, tomuto jazyku nesmí být dovoleno obejít integritní omezení nebo přístupová práva vyjádřená prostředky vysokoúrovňového relačního databázového jazyka.

Datové modelování

27 April 2024 11:33

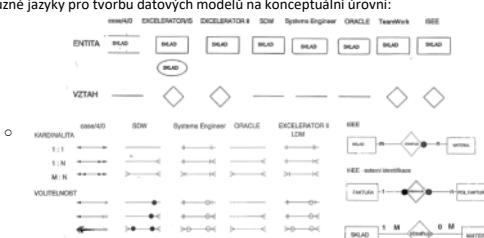
Princip tří architektur



- Konceptuální úroveň** je tvořena dvěma typy modelů
 - Model obsahu databáze, zabýváme se jedinou otázkou - co do té databáze potřebujeme uložit za data
 - Nesmíme se zabývat tím jak to bude vypadat!** Řešíme jen otázku co
- Konceptuální schéma(reality)**
 - = model obsahující základní entitní množiny, vztahy a jejich atributy.
 - Způsob jednoznačného popisu toho světa
 - Jedná se o model vytvořený za účelem poznání zkoumaného světa.
 - Nemusí být úplný, potřebujeme hlavně poznat jaké základní množiny objektů se v tom světě vyskytují a jaké jsou tam základní typy vazeb
 - Tužka papír, pero tablet
- Konceptuální model dat**
 - = popis obsahu datové základny na úrovni, která je nezávislá na vlastním implementačním a technologickém prostředí.
 - Je vytvořen za účelem přesného zobrazení obsahu datové základny.
 - Tady musí být všechny entitní množiny, všechny atributy, všechny vazby a atributy vazeb
 - To už je zadání, které zadáme někomu (stroji, sami sobě) kdo už navrhne strukturu např. relačních tabulek
 - Potom se z toho vygeneruje fyzický způsob
 - Dokumentace provozované databáze, datové základny
- Konceptuální úroveň je nejdůležitější - tvůrčí čin, všechno ostatní dál se dá do jisté míry algoritmizovat
- Technologický (logický) model**
 - = popis způsobu realizace systému v termínech jisté třídy technologického prostředí (lineární, relační, hierarchické nebo síťové logické datové struktury). Například pro relační databázový model jsou na této úrovni do relačních tabulek doplněvány cizí klíče reálnizující vazby mezi entitami z konceptuálního modelu.
- Implementační (fyzický) model**
 - = popis vlastní realizace systému v konkrétním implementačním prostředí, např. doplnění údajů o typech indexů, velikostech a rozmištění pracovních prostorů v konkrétním databázovém systému.

Funkce modelů na konceptuální úrovni

- Modely na konceptuální úrovni plní následující funkce
 - prostředek poznávání zkoumané výseče reality,
 - prostředek komunikace mezi členy řešitelského týmu,
 - platforma pro diskusi s uživateli,
 - podklad pro návrh datové základny na technologické implementační úrovni,
 - prostředek dokumentace existující datové základny.
- Různé jazyky pro tvorbu datových modelů na konceptuální úrovni:
 - ER diagram (Entity Relationship diagram) - standard pro modelování datových struktur
 - IDEF0 (Information System Functional Decomposition) - standard pro modelování funkcionálních procesů
 - IDEF1X (Information System Functional Decomposition Extended) - rozšířený standard pro modelování funkcionálních procesů
 - UML (Unified Modeling Language) - všeobecný jazyk pro modelování softwarových systémů
 - SQL (Structured Query Language) - jazyk pro manipulaci s daty v databázi
 - XML (Extensible Markup Language) - jazyk pro reprezentaci dat v textovém formátu



- V předmětu 4IT218 (na cvičeních) budeme používat speciální notace pro jednotlivé úrovně návrhu struktury obsahu databáze:
 - Konceptuální schéma reality => ručně používaná a kreslená notace - viz tato a další přednášky a skripta CHLAPEK, Dušan, KUČERA, Jan, PALOVSKÁ, Helena. Datové modelování a návrh relační databáze -Sbírka řešených úloh [online]. 1. vyd. Praha : Oeconomica, 2019. 168 s. ISBN 978-80-245-2331-6. Dostupné z: <https://oeconomica.vse.cz/publikace/datove-modelovani-a-navrh-relacni-databaze-sbırka-resených-úloh/>
 - Konceptuální schéma dat
 - v nástroji Sybase Power Designer (Conceptual Data Model – CDM) – Notace E/R + Merise
 - nebo v nástroji Oracle Data Modeler (model Logical) – notace Information Engineering
 - Logická úroveň návrhu
 - v nástroji Sybase Power Designer (Physical Data Model – PDM)
 - nebo v nástroji Oracle Data Modeler (model Relational)

Činnosti řešených při datovém modelování na konceptuální úrovni

- Rozlišení množin objektů (entitních množin).
- Pojmenování entitních množin a identifikace entit.
- Rozlišení entitních podmnožin.
- Určení vztahů mezi entitními množinami, určení kardinality a parciality vztahů.
- Určení atributů entitních množin a vztahů.
- Vyřešení problémů synonym a homonym.
- Rozlišení množin objektů (entitních množin).**
- Pojmenování entitních množin a identifikace entit.**
 - Zaměstnanci
 - Oddělení
- Entita** je rozlišitelný a identifikovatelný objekt světa, který popisujeme
 - Entitou je např. Karel Novák, oddělení mzdové účtárny.
 - Entity se na základě podobnosti sloučují do entitních množin (typu entit).
- Každá entitní množina musí mít uveden **identifikátor**, tj. minimální množinu atributů, které zajišťují jednoznačnou identifikaci entit této množiny.
 - Identifikátor nemusí být tvoren pouze vlastními atributy entitní množiny, ale entitní množina může být identifikována na jiné entitné množině. Pak se jedná o "externí" identifikaci.
 - Je to vlastně PK, ten ale může být tvoren i z cizích klíčů -> v konceptuálních modelech žádné cizí klíče být nesmí (cizí klíč je jen

- způsob, jak v relačním prostředí vyjadřujeme vztahy (v konceptuálním modelu na to budeme mít speciální slovo)
- Pozn.: V ER diagramech je často posouván význam pojmu - pod entitou je chápána entitní množina a pod pojmem výskyt entity - entita modelovaného světa.

Jazyk pro tvorbu konceptuálního schématu reality

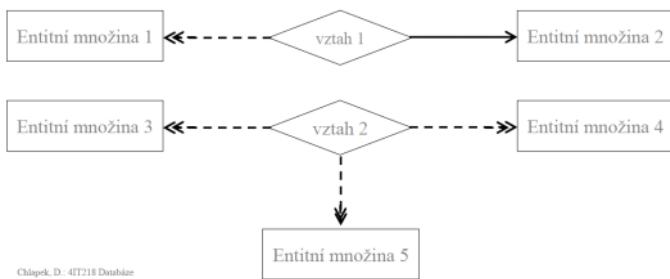
- Entitní množina a její atributy. Určujeme povinnost (plná čára) a volitelnost (čárkovaná čára) atributů.
- Povinný = not null (nemůže obsahovat neurčenou hodnotu)
 - Volitelný = Může obsahovat i neurčenou hodnotu (nemusí pro každý prvek, pro každou entitu být uveden atribut)
- Entitní množina a její identifikace
 - Vlastní identifikace - identifikátor tvořen atributy entitní množiny
 - Šípka tam a zpátky = atribut je součástí identifikátoru
 - Externí identifikace - identifikátor tvořen částečně či plně atributy jiné entitní množiny
 - V konceptuálním modelu se nesmí vyjadřovat cizí klíče, ale ani se **nesmí jeden atribut vyskytovat vícekrát** -> my musíme říct že nějaká entitní množina je externě identifikována jinou entitní množinou
 - Entitní množina 1 externě identifikuje entitní množinu 2
 - Atribut 4 nestáčí tu entitní množinu 2 identifikovat, a je potřeba ji identifikovat ještě něčím jiným
- Př.
 - Vlastní identifikace
 - Šípka tam a zpátky = atribut je součástí identifikátoru
 - Externí identifikace
 - Sám název ZP není unikátní, je unikátní až v kombinaci s externí identifikací -> ZP je entitní množina identifikačně slabá, její identifikace je závislá na jiné entitní množině, to vyjadřujeme šípkou v červeném kolečku

Rozlišení entitních podmnožin.

- ISA hierarchie
 - Množina zaměstnanců, které podle typu smlouvy (atribut) můžeme rozdělit na podmnožiny (podtypy) - podmnožina učitelů, vědců a administrativních pracovníků
 - Atributy os.číslo, jméno a datum narození jsou společné pro všechny zaměstnance bez ohledu na typ smlouvy který mají
 - Atribut pedag. titul je společný jen pro tu podmnožinu (jenom učitelé můžou mít vyplňený tento atribut, a ještě ně všichni (přeširovaná čára - volitelný atribut))
 - Administrativní zaměstnanec zůstává jen na 3 společných atributech
 - Musíme rozhodnout, jestli rozdělíme zaměstnance do této množin tak, že průnik této množin bude prázdná množina (nemůže existovat zaměstnanec, který by byl současně vědcem a učitelem)
 - identifikátor je jeden společný pro všechny
 - Musíme určit, jestli tomu tak je nebo ne -> podle toho se pak rozhoduje, jak to budu realizovat v relačních tabulkách

Určení vztahů mezi entitními množinami

- Vztahy se modelují kosočvercem
- Vztahy mohou mít atributy, bez ohledu na svou kardinalitu
- Vztah musí mít název
 - Do case nástroje můžu zaznamenat nejen název vztahu, ale i název role té entitní množiny kterou hraje v tom daném vztahu
 - Vztah, který vyjadřuje zařazení zaměstnance do nějakého oddělení
 - Ale současně vyjadřuje, že oddělení zaměstnává nějaké zaměstnance a zaměstnanec je zařazen v nějakém oddělení
 - Vztahy se čtou v směru šípek (pouze u binárních vztahů)
 - Vztahy v KSR rozlišujeme na vztahy:
 - binární, tj. vztahy mezi 2 entitními množinami
 - polyární, tj. vztahy mezi více než dvěma entitními množinami (také označované jako vztahy n-ární, kde n >2).



Chlápek, D.: 4IT218 Database

- Určení kardinality a parciality binárních vztahů mezi entitními množinami

- Kardinality může být 1:1, 1:N, M:N (=horní)



- Ve světě, který chci převést do databáze, existují množina mužů a množina žen, a mezi nimi existuje vztah manželství, který se týká a propojuje jednu entitu z množiny mužů (jednoho muže) s právě jednou entitou z množiny žen
 - Každý vztah má dva směry, musím to číst i z druhé strany: Má méně entitu žen která vstupuje do vztahu manželství s právě jedním mužem (s právě jednou entitou z té entitní množiny mužů)
- Kdyby mohl jeden muž vstoupit do manželství s více ženami, byly by ty dvě šípky (N) na straně u ženy -> vztah 1:N (jeden muž ve stejném manželství s více ženami) -> jeden muž vstupuje do vztahu manželství s více ženami
- Kdybychom tam chtěli zohlednit i rozvodovost -> chtěli bychom sledovat nejen platný manželství (1 muž a 1 žena), ale i minulá manželství -> tento model by nám nestačil, ke vztahu bychom museli přidat dva atributy - datum sňatku (povinný) a datum rozvodu (nepovinný) -> pokud by datum rozvodu nebylo vyplňeno, znamenalo by to že manželství ještě platí -> vztah by byl poté M:N -> jeden muž může vstupovat do více manželství s různými ženami, ale pouze v jednom manželství může setrvavat
- Jedna žena vstupuje do vztahu mateřství s více muži (je matkou více synů), jeden muž má ve vztahu mateřství právě jednu matku
- Jeden muž se může přátelit s více prvky z množiny žen, jedna žena se může přátelit s více prvky z množiny mužů

- Potřebujeme zachytit i dolní kardinalitu = povinnost nebo volitelnost vazeb -> parcialita

- Parcialita vyjadřuje volitelnost vztahu (=dolní)

- Vyjadřujeme ji čárkovánou nebo plnou čarou. Čteme ve směru šípek.



- Čísla 1, N a M jsou jen duplicitní vyjádření šípek -> standartně stačí jen to jestli je čára plná nebo volitelná, jestli je tam jedna nebo dvě šípky

- Rozlišují množinu zaměstnanců a množinu oddělení, a platí tam:
 - Může existovat oddělení, ve kterém není zařazen ani jeden zaměstnanec (čárková čára) ale může v něm být zařazen jeden, nebo i více zaměstnanců
 - Každý zaměstnanec musí být zařazen právě do jednoho oddělení (nemůže nastat ani jeden případ, kdy by zaměstnanec nebyl zařazen do oddělení)

- Nastavujeme pravidla, jak se potom budou chovat ty business procesy, které budou nad to databází navrženy

- Vazba vedení (1:1)

- Zaměstnanec nemusí vést žádné oddělení, ale pokud vede nějaké oddělení, tak vede maximálně jedno oddělení
 - Mám množinu zaměstnanců, ne každý zaměstnanec musí být vedoucím nějakého oddělení, ale pokud je vedoucím, tak vede maximálně jedno oddělení

- Máme množinu oddělení, která jsou vedena nějakými zaměstnanci, a však každé oddělení je vedeno právě jedním zaměstnancem (nemůže být žádný, nemůže jich být více) (plná čára, jedna šípka)

- V semestrální práci musíme vždy popsat oba směry vztahů, vždy horní i dolní kardinalitu!**

- Zaměstnanec hodnotí nějaké oddělení, nemusí hodnotit žádné, může jich hodnotit více, oddělení je hodnoceno více zaměstnanci (0 až N) -> oboustranně volitelná M:N vazba, tak je to nejjednodušší (na její vyjádření v relačním prostředí potřebujeme 3 tabulky)

- Když je něco povinné, tak musíme vynaložit úsilí k tomu, abychom tu povinnost zajistili! (not null)

- Např. u vedení bych ke každému oddělení v tabulce oddělení přiřadil číslo zaměstnance, který je vedoucím daného oddělení, a tento sloupec by byl povinný (tím vyjádřím tu povinnost - plná čára), současně bych na tom sloupci ale musel nadefinovat podmíinku unique (nesmí se opakovat jedna hodnota ve více řádcích) -> tím říkám že jeden zaměstnanec může vést jen maximálně jedno oddělení

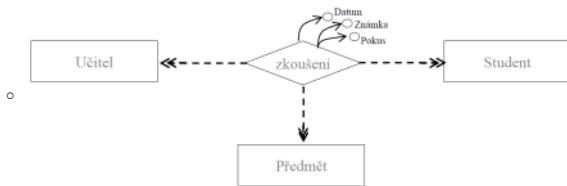
- Kdybych ale měla že každý zaměstnanec musí pracovat alespoň v jednom oddělení (plná čára a 1-N), tak to už databázový systém neumí jednoduše zajistit, a my to musíme vynutit nějak (proaktivně nebo reaktivně)

- Každou povinnost a volitelnost je třeba velmi pečlivě zvážit! -> povinnost musíme něčím vynutit**

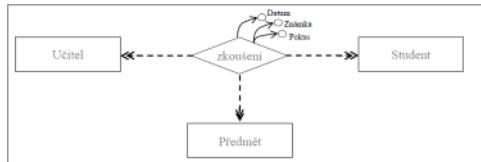
- Kardinality a parcialita se v některých notacích vyjadřují také čísla a písmeny, které označují "dolní" a "horní" kardinalitu role entitní množiny ve vztahu.



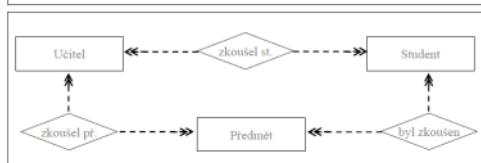
- Převod polárních vztahů na binární vztahy může probíhat i na konceptuální úrovni.



- Ternární vazba = vazby složitější než binární
- Složitější vazby než binární jsou velmi efektivní pro ty první konceptuální modely, pro které vytváříme za účelem poznání
- Pokud bychom šli níž a chtěli to převádět do relačních tabulek, tak je dobré si ještě na konceptuální úrovni vyřešit co bude ten identifikátor (u vazeb se identifikátory neuvedá)
- Snaha o převod jedné ternární vazby do binárních vazeb:



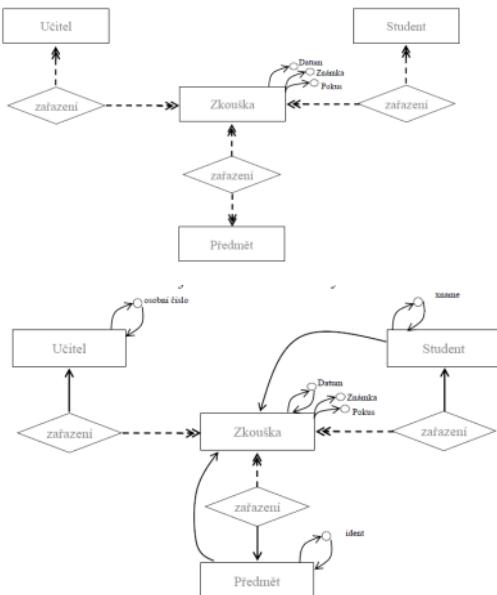
3 entitní množiny a 1 ternární vazba



3 entitní množiny a 3 binární vazby

- Dokážu zachytit svět, který je popsaný horním modelem? Vypovídá to o stejných světech?
 - ◆ Není to dobré, nejde to vyjádřit.
 - ◆ Dva učitelé - chlapek a kučera. Oba dva učí magisterský předmět řízení projektů (4it414) a databáze (4it218). Představme si že máme dva studenty (student 1 a student 2) -> každá entitní množina má dvě entity
 - ◆ Představme si, že každý ze studentů byl zkoušen u každého z těch předmětů -> počet instancí vazby jsou 4 (student 1 - databáze, student 1 - řízení, student 2 - databáze, student 2 - řízení)
 - ◆ Chlapek i kučera zkouší -> 4 instance vazby - chlapek zkouší databáze i řízení, kučera zkouší databáze i řízení
 - ◆ Dva studenti mají zkoušku od každého z učitelů -> student 1 byl zkoušen od chlapka i od kučery, ale nepoznáme z jakého předmětu -> oba dva zkoušeli oba ty předměty ze kterých byl student zkoušen, ale nejsme schopni v databázích dohledat kdo ho zkoušel -> ten dolní model není přesný!

- Tento je lepší:



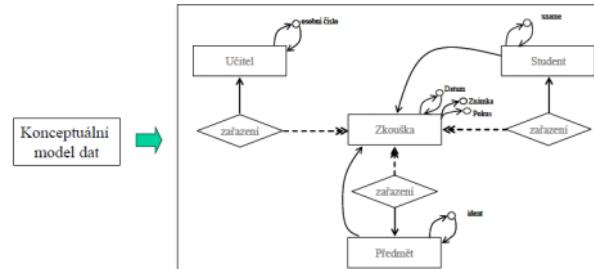
Pozn.: Cílem je využití vlastní a nákladového atributu entitních množin.

- Vazby "zkoušení" jsme udělali entitní množinu
- Ta entitní množina je spojená s učitelem který zkouší, se studentem který byl zkoušen a s předmětem kterého se zkouška týká
- Datum, xname studenta a ident předmětu zajistí jasnou identifikaci té zkoušky

• Úrovně konceptuálních modelů



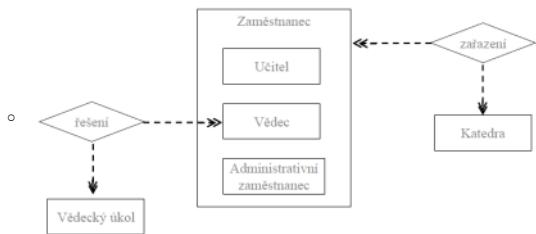
Koncepcionalni schéma reality



- Ternární vazby se do binárních rozkládají vždy tak, že z té vazby n-ární (kde n > 2) uděláme entitní množinu, a doplníme ji binárními vazbami mezi těmito původními množinami a nouzou

- Konceptuální schéma stačí k popsání světa (máme složitější vztah kde jsou tyto 3 entitní množiny v nějakém vztahu, ten má tyto atributy) a tím můžeme skončit v tom co kreslíme ručně -> ve světě se vyznáváme, potřebujeme ho pochopit
- Když budeme potřebovat navrhnut databázi, tak musíme jít do detailů, a n-ární vazby ke $n > 2$ bychom měli transformovat do binárních vazeb a jasně identifikovat i identifikátory té vazební entitní množiny

- Rozlišení entitních podmnožin je také důležité pro přesnější zachycení vztahů mezi entitními množinami



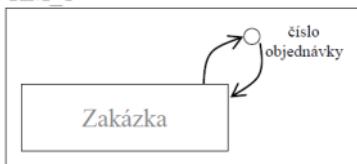
- Katedra zařazení (vazba M:N) říká, že na jedné katedře může být zařazeno více zaměstnanců (vazba je vedena k tomu nadtypu, k té entitní množině zaměstnanců - zaměstnanec může být sekretářka, vědec nebo učitel) ale říkám, že tahle vazba platí pro všechny bez ohledu na podtypy, a současně říkám že zaměstnanec může být zařazen na více katedrách. Můžou existovat zaměstnanci, kteří nejsou zařazeni ke katedrám, a může být katedra bez zaměstnanců = obostranně volitelná m:n vazba)
- Může být ale entitní množina která má vztah k jednomu podtypu (jenom vědci mají zadání řešitelské úkoly, mají speciální typ vědeckého úvazku -> má množinu vědců, ty mohou mít přiřazené vědecké úkoly, na jednom vědeckém úkolu může pracovat více vědců, můžou mít úkol, ke kterému jsem zatím nepřiřadil vědce kteří ho budou řešit, a současně můžou mít vědce, který zatím není přiřazen k řešení nějakého vědeckého úkolu)

- Ukázky různých řešení:

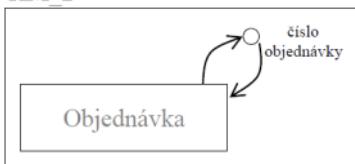
- Dva modelující či dva týmy vytvoří dva modely, které je nutno integrovat do jednoho.

- Při integraci je třeba vyřešit několik problémů.

KM_1



KM_2



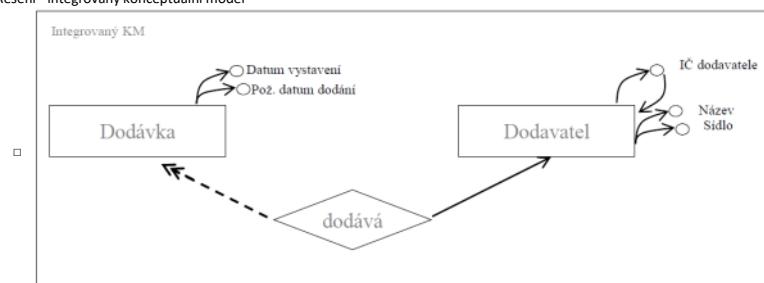
- Jeden konceptuální model obsahuje zakázku která má identifikátor číslo objednávky, druhý tým vytvořil druhý konceptuální model který má entitní objednávku, a identifikátor číslo objednávky -> casé nástroje nás neupustí dál (není možné aby se identifikátor použil u dvou entitních množin)
 - Každý atribut se v datovém modelu na konceptuální úrovni objevuje pouze jednou.
 - Oba konceptuální modely nutno integrovat. Existují však 2 entitní množiny se stejným identifikátorem, což není přípustné.

- Tři varianty řešení konfliktu dvou integrovaných konceptuálních modelů:

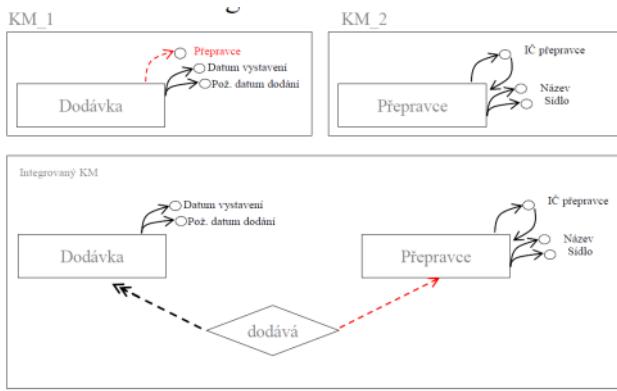
1. Týmy se neshodli na názvu a jsou to Synonyma -> je nutno změnit název jedné z entitních množin.
2. Jedna z entitních množin je podmožinou druhé entitní množiny.
3. Existuje třetí entitní množina, obě stávající jsou podmožinami nově vzniklé entitní množiny (podtypy)



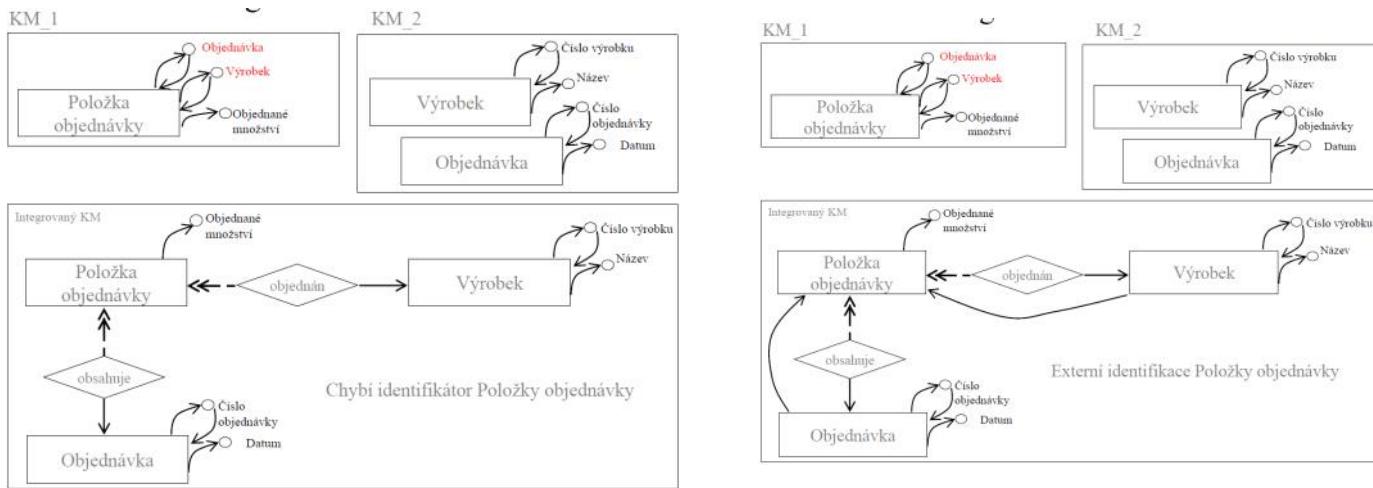
- Jeden tým vytvoří entitní množinu dodávka, kde má atribut dodavatel, a jiný tým zjistí že je zajímají i další údaje o tom dodavateli -> vytvoří si entitní množinu dodavatel
- Řešení - integrovaný konceptuální model



- Každý atribut může být potenciální entitní množinou
- Atribut v jednom z integrovaných modelů byl nahrazen vztahem k v druhém KM existující entitní množině.
- U dodávky už neexistuje atribut dodavatel, protože jsme ho nahradili samostatnou entitní množinou dodavatel
- Každá dodávka musí mít svého právě jednoho dodavatele



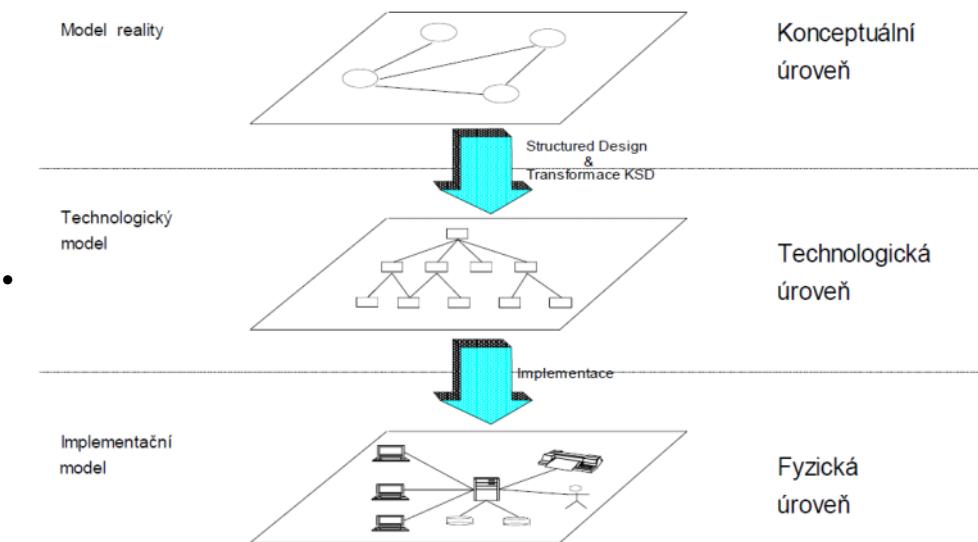
- Atribut v jednom z integrovaných modelů byl nahrazen vztahem k v druhém KM existující entitní množině. Volitelnost atributu se promítá do volitelnosti vztahu na straně 1
- Dodávka nemusí mít uvedeného přepravce



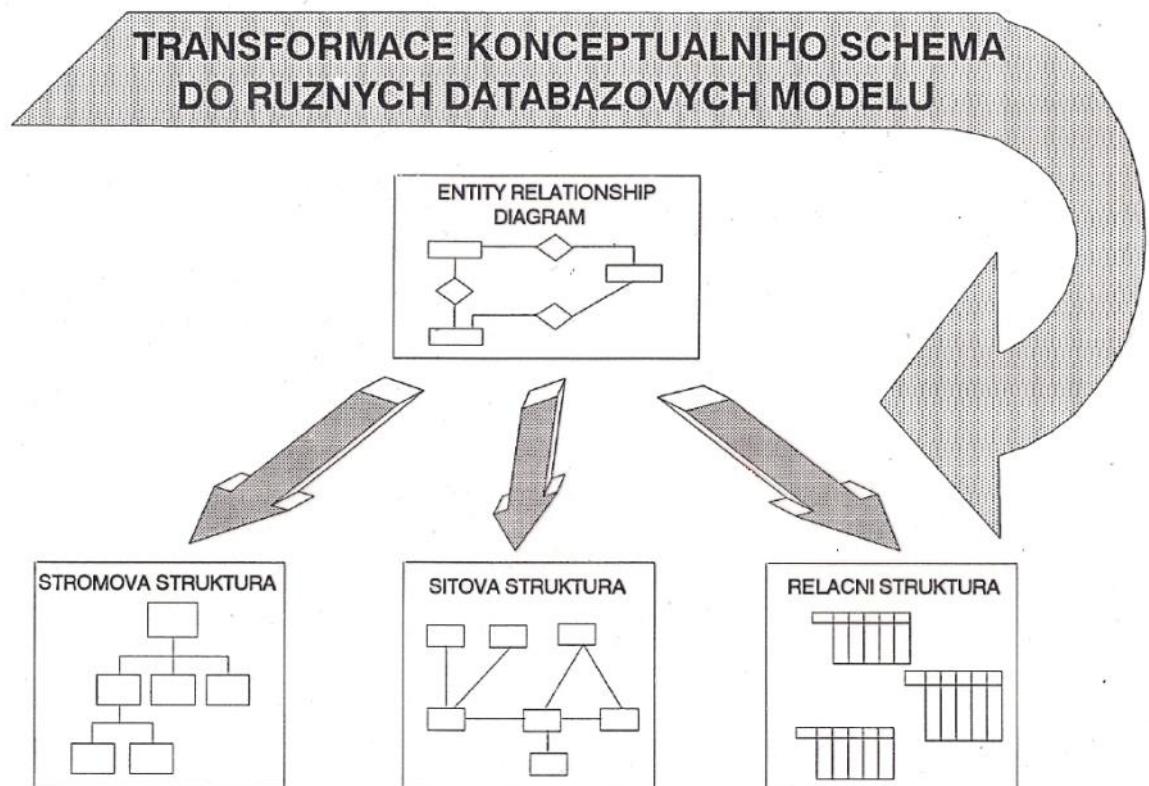
Princip tří architektur

15 May 2024 12:27

Princip tří architektur



- Transformace z konceptuálního schématu do logických struktur (technologické úrovni)

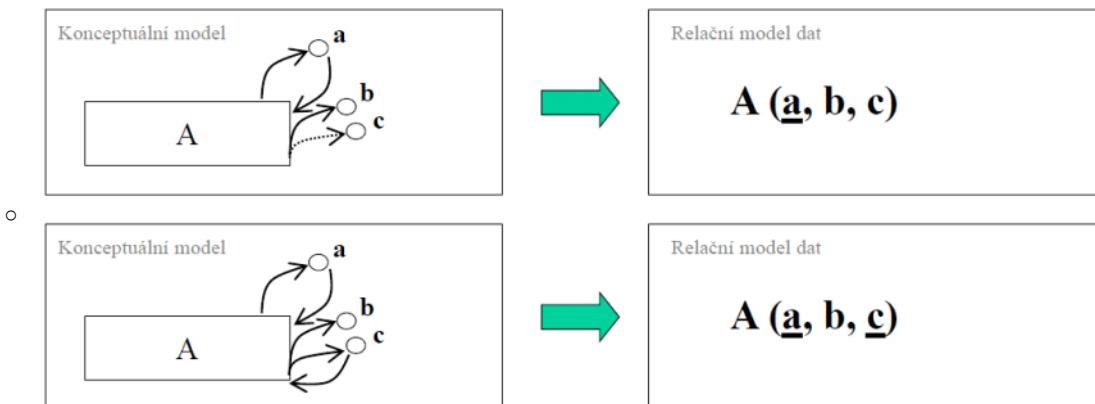


Pravidla transformace konceptuálního modelu do relačního modelu

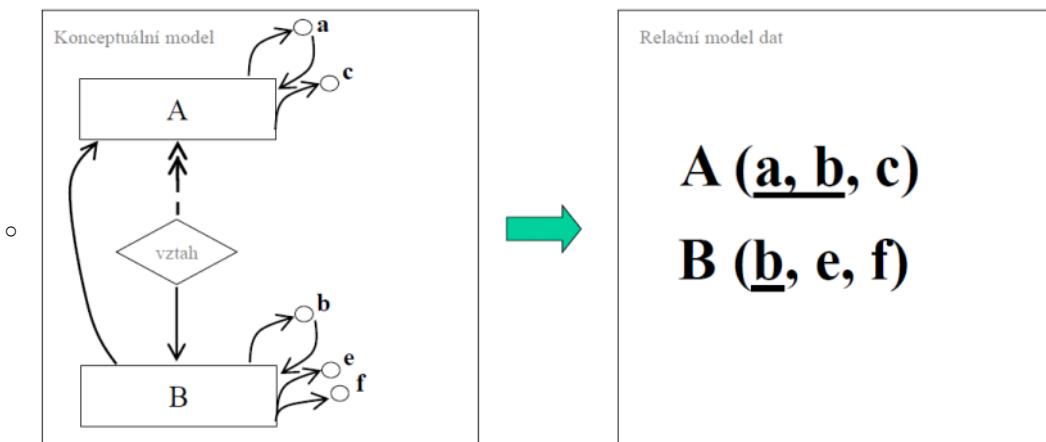
15 May 2024 12:28

Pravidla transformace konceptuálního modelu do relačního modelu

- Pravidlo č. 1: Každá entitní množina je transformována do jedné relační tabulky.
Identifikátory entitní množiny se stanou atributy tvořícími primární klíč relační tabulky.



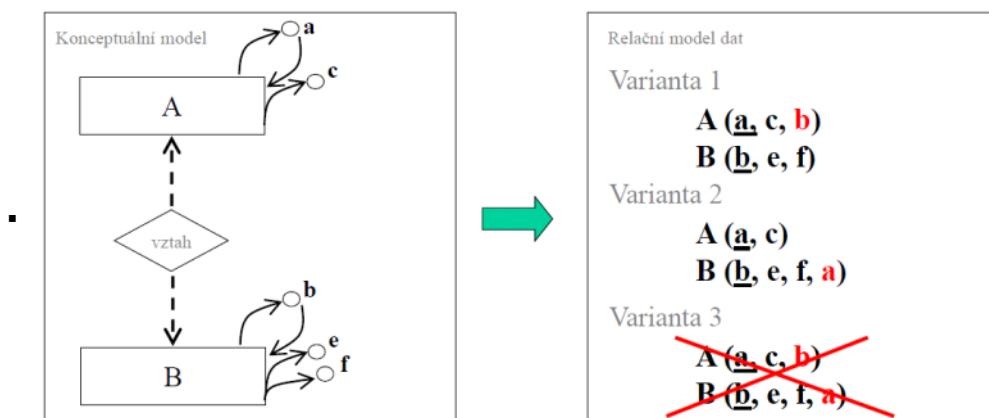
- Název relační tabulky (názvy sloupců) -> Atributy tvořící primární klíč podtrháváme
 - a je primární klíč, b je not null atribut, c je null atribut (může nabývat neurčených hodnot)



- Externí identifikace -> identifikačně slabá entitní množina A (která je závislá na množině B)
 - Pro vyjádření vztahu musím do množiny A zadat cizí klíč, který současně funguje i jako část primárního klíče
 - Entitní množina A, která je závislá identifikačně na B -> v entitní množině A do primárního klíče zahrnu oba dva atributy, přičemž atribut b se navíc odkazuje do entitní množiny B

- Pravidlo č. 2: Vztah z konceptuálního modelu je v relačním modelu vyjádřen cizím klíčem.

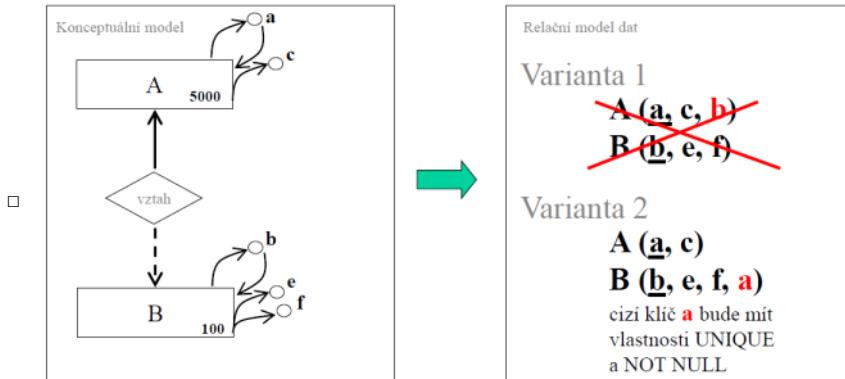
- **Vztah 1 : 1.**



- Nejjednodušší vazba je nejsložitější
 - Case nástroje očekávají, že uživatel toho nástroje ví co má říct -> musíme říct dominantní roli

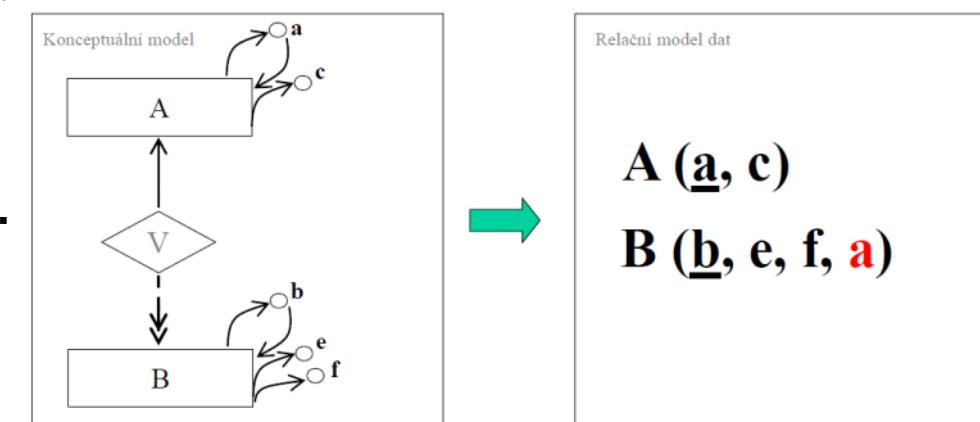
- 1 B má vztah k maximálně jednomu A, jedno A má vztah k maximálně jednomu B
- Když neřekneme dominantní roli, tak nástroj udělá tu nejhorší variantu 3 -> pro triviální věc uděláme dva cizí klíče
- Jedna změna ve světě by měla znamenat jeden update databáze
- Cizí klíč dáme pouze do jedné nebo druhé relační tabulky

■ Jaká varianta je vhodnější pro uvedený příklad vztahu 1:1?



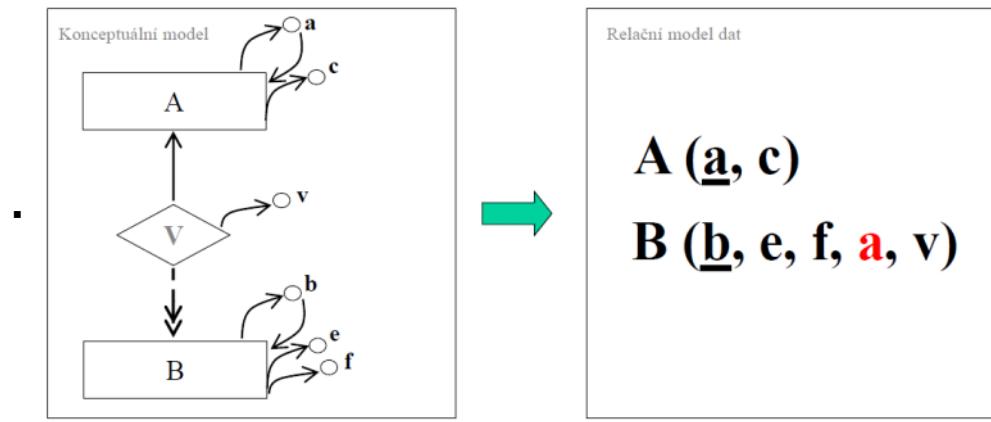
- 5000 entit v entitní množině A, ta má dva povinné atributy a c
- 100 entit v entitní množině B
- Vazba nemůže být povinná z obou stran (nemáme stejné množství entit)
- Bude existovat 4900 výskytů entitní množiny A, která nebude mít vztah k nějakému B
- Ve varianci 1 říkám, že dávám cizí klíč k tomu A -> relační tabulka má 3 sloupce a c, bude mít 5000 řádků a ve 4900 řádcích musí být neurčena hodnota ve sloupci b
 - ♦ V relační tabulce B s těmito 3 sloupcemi je 100 řádků
- Ve varianci 2 máme relační množinu A, která má 5000 řádků, obsahuje sloupce a a c
 - ♦ Relační tabulka B má 100 řádků, v každém řádku musí být uvedena hodnota a
- Varianta 2 je správně -> tím že řeknu že cizí klíč bude unique zajistím, že ta vazba je opravdu na jedno A, že se nemůže v různých řádcích opakovat stejný cizí klíč

○ **Vztah 1:N**



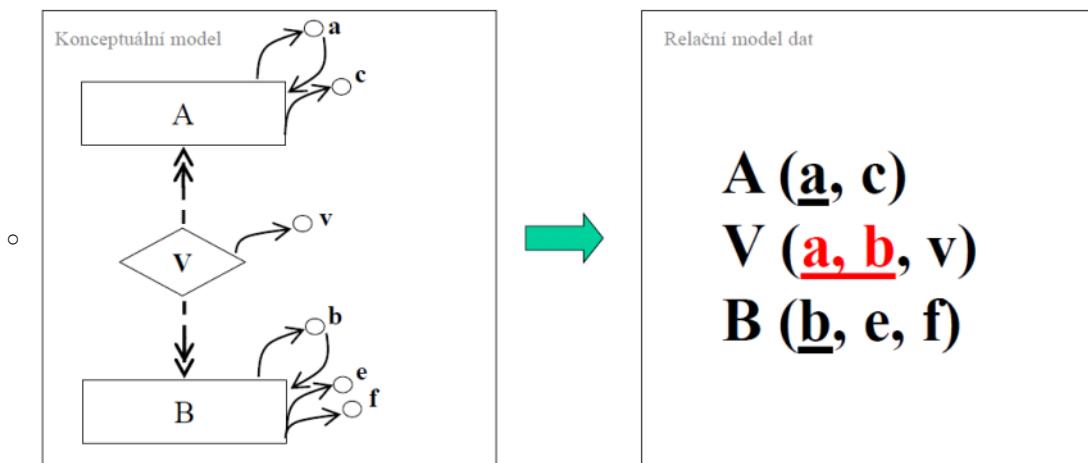
- Nejjednodušší vyjádření
- Cizí klíč vždy dáváme v té relační tabulce která vyjadřuje entitní množinu na straně N

○ **Vztah 1:N a stributem**



- Pokud má vztah atributy, přidáváme ty atributy z toho vztahu i k té relační tabulce na straně N

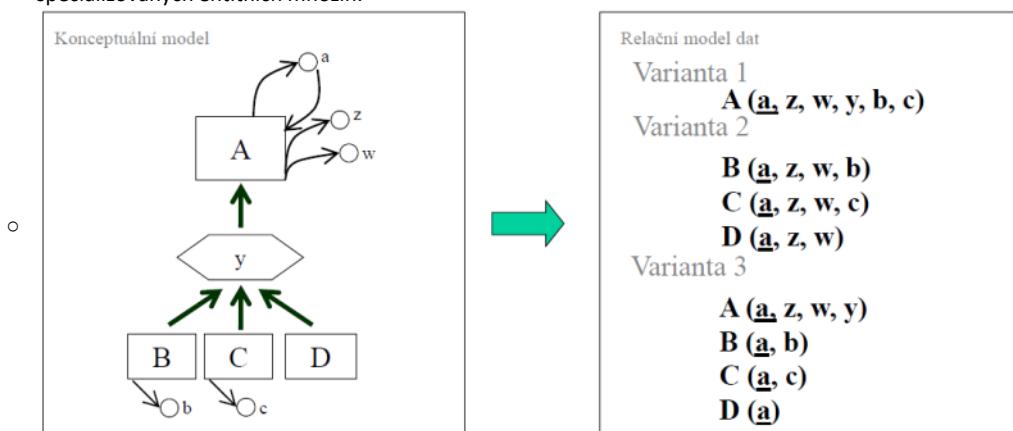
- Pravidlo č. 3: **Vztah M:N z konceptuálního modelu je v relačním modelu vyjádřen vazební relační tabulkou a dvěma cizími klíči.**

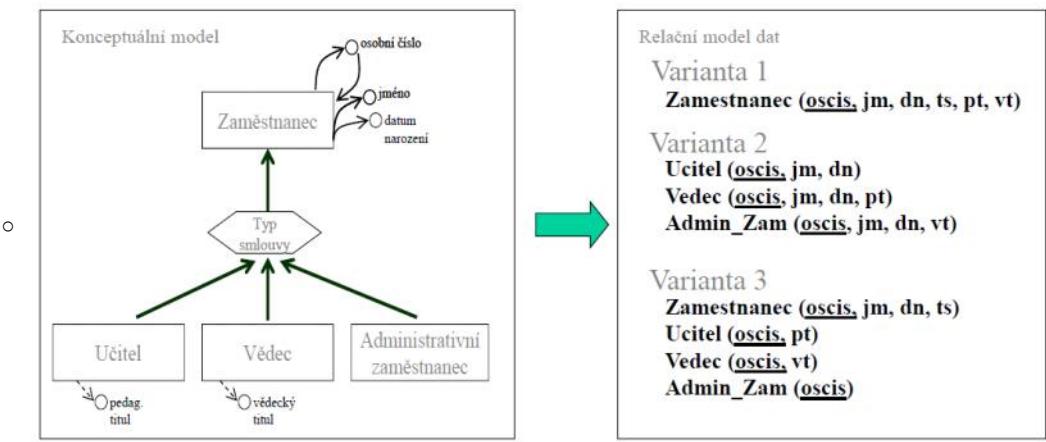


- Vazbu M:N musíme vyjádřit 3 relačními tabulkami!
- Třetí relační tabulka je pro tu vazbu, přičemž do té vazby použiji cizí klíče a atribut

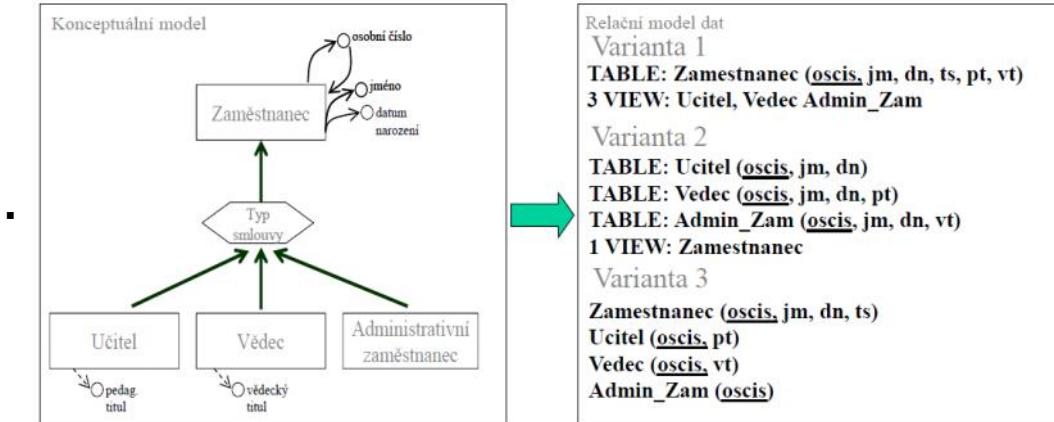
- Pravidlo č. 4: **Generalizace / specializace je transformována do relačního modelu dat několika variantami:**

- a) jednou relační tabulkou na úrovni celé hierarchie,
- b) relačními tabulkami na úrovni specializovaných entitních množin,
- c) relačními tabulkami na úrovni generalizující entitní množiny i na úrovni specializovaných entitních množin.



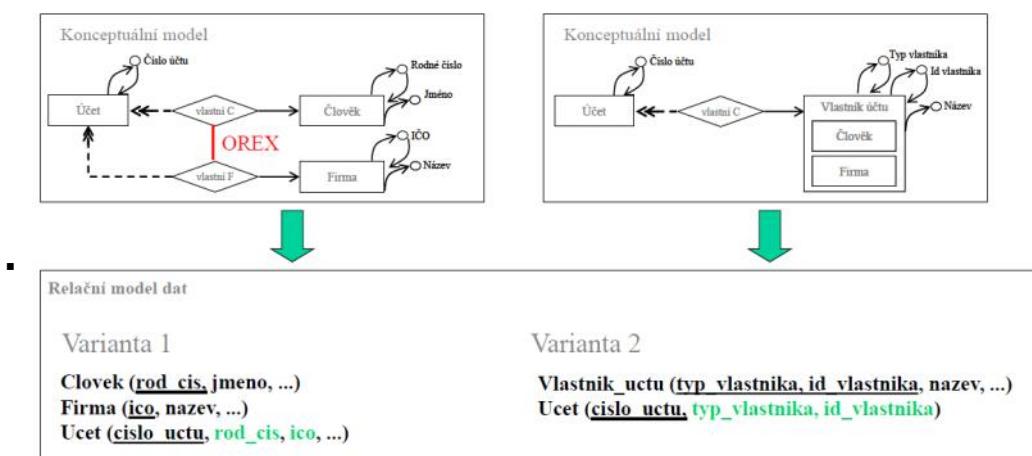


- Varianta 1: všechno vyjádříme jako 1 relační tabulku (pt a vt budou obsahovat neurčené hodnoty)
- Varianta 2: uděláme relační tabulky jenom na úrovni podtypů
- Varianta 3: Budu mít tabulku dohromady za všechny zaměstnance bez ohledu na to, jaký mají typ smlouvy. Pro ty, kteří mají specifický typ smlouvy, uděláme nějaké další relační tabulky
- **Podmnožiny spojené do jedné entitní množiny se počítají v semináře jako jedna entitní množina!**
- Pozn.: Všechny varianty je možno kombinovat s dynamickými odvozenými rel. tabulkami -objektem typu VIEW (dynamická odvozená relační tabulka)



- V tomto řešení je nejužívanější varianta 1 - vytvořím 1 základní tabulku, kam dám všechny řádky a všechny sloupce, a pak si vytvořím 3 view
- Ve druhé variantě děláme view jako sjednocení tabulek
- 3 varianta se moc nepoužívá

- Ukázka výlučnosti vztahů realizace formou generalizace / specializace a převod do relačního modelu dat.



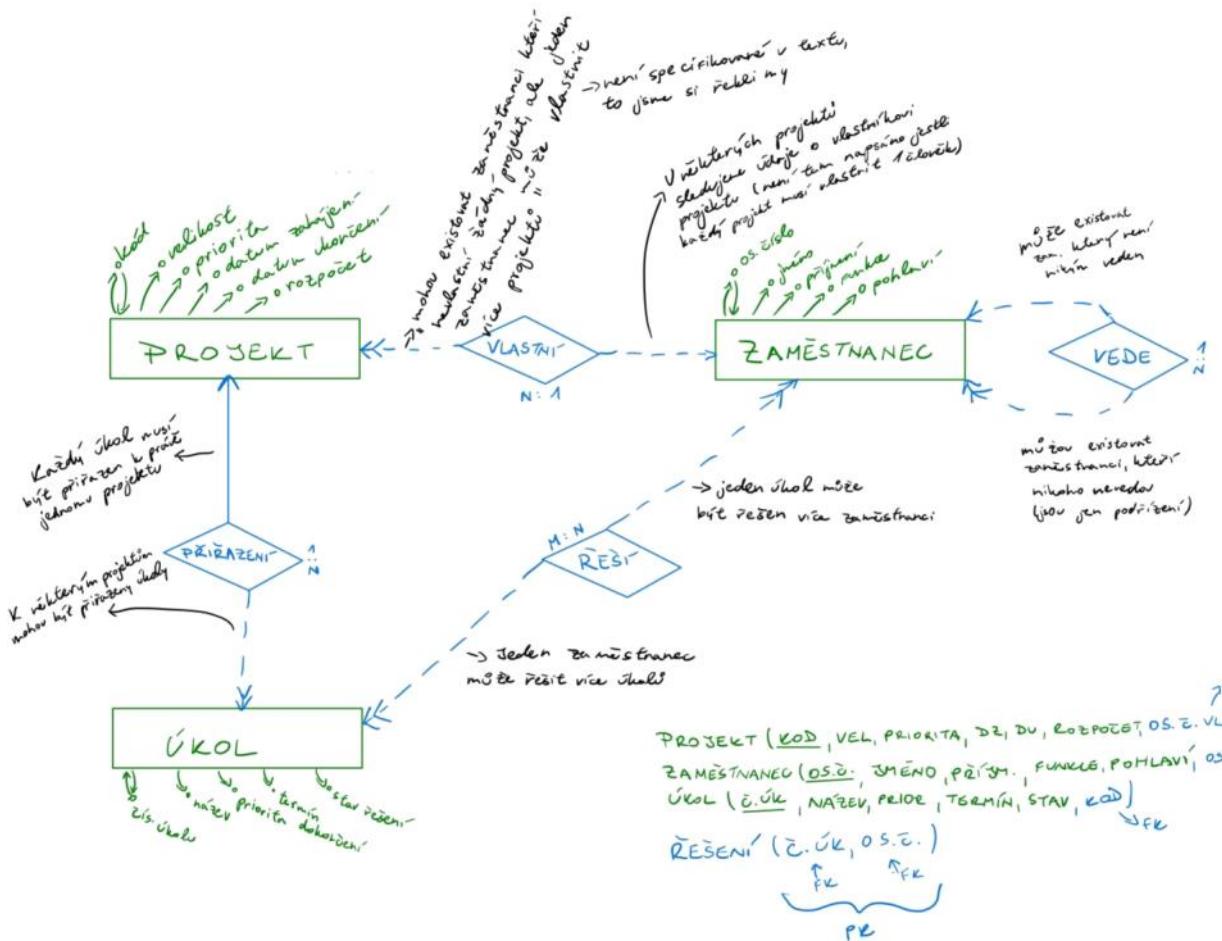
Kterou z variant se dá lépe vyjádřit výlučnost vztahů?

- 3 Entitní množiny - účet, člověk a firma. Každý účet má právě jednoho vlastníka, kterým je člověk, a právě jednoho vlastníka, kterým je firma. (V reálné situaci toto nemůže nastat) -> model přesně neodpovídá světu
- Pokud chceme, aby se ty dva vztahy vyučovali -> v některých case nástrojích existují notace které umožňují říct, že nějaké vztahy jsou vzájemně výlučné =

- OREX = exkluzivní nebo -> každý účet má právě jednoho vlastníka, a tím je buď člověk, nebo firma
- Můžu to udělat jinak: účet má právě jednoho vlastníka (vlastník účtu nemusí vlastnit žádný účet, může jich vlastnit více) -> vztah je jeden -> z dvou entitních množin člověk a firma jsem udělal podtypy -> vytvořil jsem nad nimi generalizační entitní množinu -> potřebuji odlišit člověka a firmu -> může nastat situace, že id člověka a firmy by mohlo být stejné -> řeší se to tak, že identifikátor uděláme složený -> v prvním identifikátoru bude typ (=zkratka, která označuje podtypy) a ve druhém je číslo toho podtypu (c když se jedná o člověka a za tím nějaký identifikátor který je jednoznačný v rámci množiny lidí, a potom je tam f jako firma a za tím např. ičo) -> nemusíme řešit, jestli id vlastníka se překrývají nebo ne, protože identifikátor je vždy složený z těch dvou atributů a tím prvním (c nebo f) jasné identifikujeme ten podtyp -> takhle to můžu udělat v situaci, kdy se to nepřekrývá (např. když je člověk živnostník, tak i tehdy má nějaké ičo)
 - Druhá varianta by byla, že by se udělali podtypy na straně účtů (ta tady není zakreslená) -> účet soukromé osoby, účet firemní
 - Můžu to tedy realizovat buďto 3 tabulkami, nebo 2 tabulkami -> 1. varianta není úplně ideální, protože u účtu jsou dva cizí klíče, tu výlučnost nedokážeme na úrovni databáze zachytit (museli bychom si na to napsat nějaký trigger), protože ani jeden z těch dvou atributů nesmí obsahovat neurčenou hodnotu, ale ani se nesmí vyskytovat ani jeden účet kde by byly vyplňené oba dva údaje (identifikátor člověka a identifikátor firmy) -> pak by to znamenalo, že jeden účet má dva vlastníky, což ten model zakazuje
 - Druhá varianta -> jedna relační tabulka vypovídá o vlastnících -> u účtu jsou také dva atributy, ale oba dva dohromady tvoří jeden cizí klíč (typ vlastníka a id vlastníka musí být oba dva not null)
 - 2. varianta je lepší!

Souhrnný příklad

- Stručný popis světa, pro který má být navržena databáze:
- Je třeba evidovat údaje o **projektech**, které jsou identifikovány jednoznačným kódem a u kterých sledujeme velikost, prioritu, datum zahájení, datum ukončení a rozpočet. U některých projektů sledujeme údaje o vlastníkovi projektu, kterým může být některý ze zaměstnanců. K některým projektům mohou být přiřazeny úkoly.
- Dále potřebujeme evidovat údaje o
 - **zaměstnancích**, kterí jsou identifikováni osobním číslem a obsahují další atributy: jméno, příjmení, funkci, pohlaví. Potřebujeme také zachytit, kteří zaměstnanci
 - vlastní který projekt,
 - řeší úkoly, přičemž jeden zaměstnanec může řešit více úkolů a jeden úkol může být řešen více zaměstnanci,
 - vedou jiné zaměstnance;
 - **úkolech**, které jsou identifikovány číslem úkolu a mají další atributy: název úkolu, prioritu, termín dokončení úkolu a stav řešení úkolu. Každý úkol musí být přiřazen k právě jednomu projektu.



Normalizace dat

15 May 2024 12:24

- Technika datové analýzy. Zabývá se vztahy na úrovni datových položek (atributů) typů záznamů.
- Když je databáze normalizovaná tak odpovídá objektům v tom světě který v té databázi zachycujeme
- Cílem je:
 - vytvořit co nejvěrnější obraz v modelovaném světě existujících entitních množin (vytvořit věrný obraz světa který v té databázi chceme mít)
 - zajistit interní konzistence datového modelu, resp. databáze,
 - minimalizovat redundancy,
 - maximalizovat stabilitu datových struktur.
- Technika vytvořena pro ověření správnosti struktury relačních tabulek v relační databázi.
- Technika je ale použitelná i pro ověření správnosti navrženého konceptuálního modelu.
- Co se dá normalizovat?
 - struktura tabulek v existující relační databázi,
 - struktura tabulek v navrhované relační databázi,
 - entitní množiny z konceptuálního datového modelu,
 - jakákoli množina datových položek, např. obsah nějakého formuláře.
- S využitím automatizovaných nástrojů (CASE) lze techniku využít i pro návrh části databáze.
- Pozn.: Data získaná normalizací nemusí být optimalizovaná z hlediska výkonnosti provozovaného databázového systému.
- Sada normálních norem -podmínek, které musí splňovat normalizovaný záznam:
 - První normální forma,
 - Druhá normální forma,
 - Třetí normální forma,
 - Boyce / Coddova normální forma,
 - Čtvrtá normální forma,
 - Pátá normální forma.
- Pozn.:
 - Číslovky v názvech vyjadřují že normální formy aplikujeme postupně za sebou
 - Nejčastěji uváděny pouze první tři normální formy
 - První čtyři normální formy zkoumají vztah neklíčových položek na primárním klíči.
 - Poslední dvě normy zkoumají vztahy uvnitř složených primárních klíčů.
- **První normální forma**
 - Záznam je v první normální formě, pokud neobsahuje ani jednu opakující se položku nebo skupinu položek

➤ Záznam PRACE:

- ◻ OS_CISLO (PRIMÁRNÍ KLÍČ)
- ◻ JMENO
- ◻ PLAT
- ◻ TYM
- ◻ NAZEV PROGRAMU
- ◻ DAT_ZAH_PRACE
- ◻ PLAN_DOKONC_PRACE
- ◻ VELIKOST PROGRAMU
- ◻ PROGRAMOVACI JAZYK

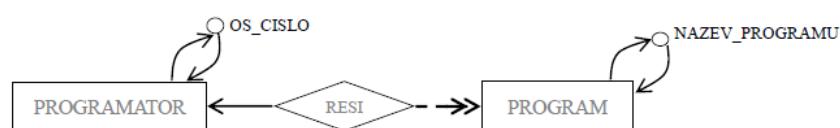
Záznam je v první normální formě, pokud neobsahuje opakující se položku nebo skupinu položek.

} 1n_x

- Záznam PROGRAMATOR:
 - ◻ OS_CISLO (PRIMÁRNÍ KLÍČ)
 - ◻ JMENO
 - ◻ PLAT
 - ◻ TYM

➤ Záznam PROGRAM:

- ◻ NAZEV PROGRAMU (PRIMÁRNÍ KLÍČ)
- ◻ DAT_ZAH_PRACE
- ◻ PLAN_DOKONC_PRACE
- ◻ VELIKOST PROGRAMU
- ◻ PROGRAMOVACI JAZYK
- ◻ OS_CISLO



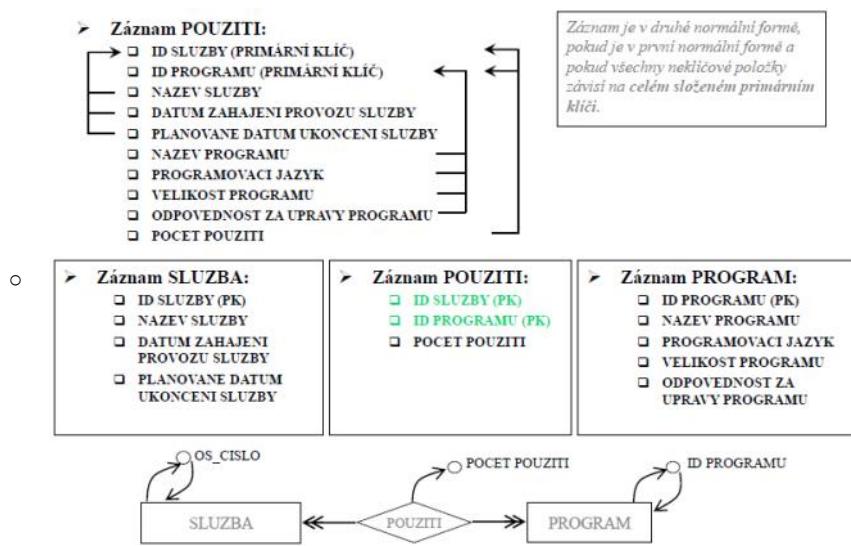
- Máme záznam, kterému říkáme Práce
- Je tam osobní číslo, tedy záznam odpovídá objektu zaměstnanec (vypovídá o nějakém člověku), jméno, plat a tým ve kterém člověk pracuje, název programu, ve kterém

pracuje, datum kdy na tom programu začal pracovat, plánované dokončení práce na programu, jak velký program je (počet příkazů) a v jakém programovacím jazyce to píše

- V excelu - co člověk, to jeden list sešitu v excelu, záhlaví jsou položky co se neopakují (os.cis, jmeno, plat, tym), a na listu v excelu je tabulka která má nadepsáno 5 záhlaví (nazev programu,...) a jeden člověk pracuje na jednom programu, jiný na 5 programech -> na každém listu sešitu se může objevit jiný počet řádků -> toto nepůjde dát do relační struktury (šlo by to jedině tak, že by tam bylo hrozně moc sloupců kde by se těch 5 záhlaví pořád opakovalo)
- Máme tady n opakujících se položek (5) -> tento záznam nevyhovuje první normální formě a musí se opravit
- Úprava je vždy taková, že se záznam roztrhne -> roztrhnu to přesně podle toho, co se opakuje a co se neopakuje -> z původního záznamu zbydou pouze 4 položky které se neopakovali, a objeví se nový záznam který si pojmenujeme program, a v něm se objeví 5 položek které se opakovali
 - Protože musíme být schopni původní datový informační význam, tak tam musíme přidat i primární klíč původního záznamu (os.cislo se tady objeví v roli cizího klíče)

• Druhá normální forma

- Záznam je v druhé normální formě, pokud je v první normální formě a pokud všechny neklíčové položky závisí na celém složeném primárním klíci.

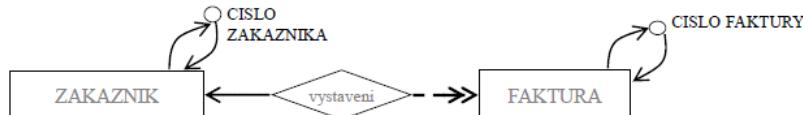
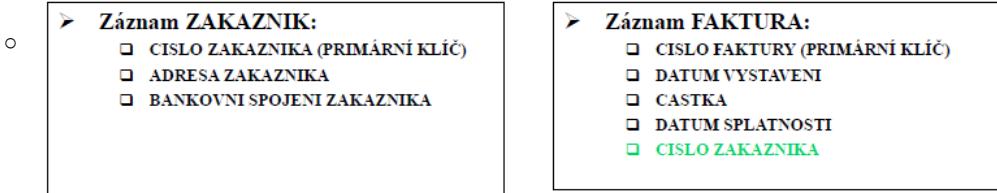
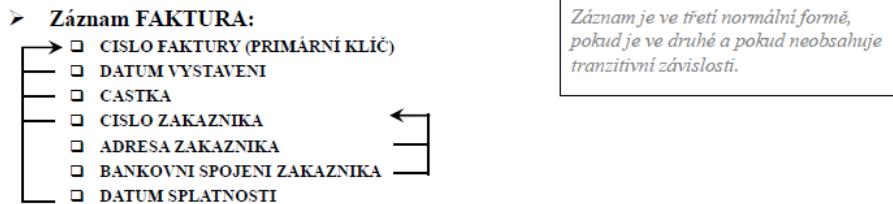


- Zkoumá se pouze u záznamů, které mají složený primární klíč (= primární klíč je tvořen 2 a více atributy) -> když záznam nemá složený primární klíč tak je automaticky v druhé normální formě a ihned postupuje na zkoumání třetí normální formy
- Jsou zde softwarové služby, které používají nějaké programové komponenty (id programu) a já v tomto záznamu mám identifikaci služby, identifikaci programu který je pro realizaci té služby zapotřebí, název služby, datum zahájení a ukončení provozu služby, název programu, programovací jazyk, velikost programu, kdo za ten program zodpovídá a počet použití (kolikrát se volá v té službě)
- Je to relační tabulka - neplatí že by jeden řádek měl jiný počet sloupců než druhý řádek, všude je tady pevný počet sloupců ale přesto je tady obrovský problém - násobí se řádky, opakují se atributy v celé té relační tabulce
- Druhá normální forma má za úkol zkoumat, zda všechny neklíčové položky (= položky které netvoří primární klíč) závisí na celém složeném primárním klíci
- Náš složený klíč je id programu a id služby
 - Vezmeme neklíčovou položku název služby -> je závislý název služby na celém složeném primárním klíči -> ne, je závislý jen na části primárního klíče
 - Takto jdeme postupně položku po položce a zkoumáme zda ty neklíčové položky závisí na celém primárním klíči
 - Jenom položka "počet použití" splňuje podmínu druhé normální formy a závisí na celém složeném klíči (id služby, id programu)
- -> záznam není ve druhé normální formě, musíme ho upravit -> jak máme graficky naznačeno co na čem závisí, tak z toho to natrháme na samostatné záznamy -> zbytek původního záznamu je záznam Pouziti (jen položka počet použití splňovala požadavek druhé normální formy) -> místo jedné tabulky budeme mít v databázi 3 tabulky -> přinese mi to větší šanci že bude databáze konzistentní
- S druhou normální formou se setkáme často (opravdu všechno patří do jedné tabulky)? Nedá se to rozdělit do více tabulek aby se mi nenásobili řádky?)

• Třetí normální forma

- Záznam je ve třetí normální formě, pokud je ve druhé a pokud neobsahuje tranzitivní

závislosti.



- Tranzitivní = zprostředkovaná, přenesená závislost
 - Faktura má číslo faktury, datum vystavení, částka, číslo zákazníka, adresa zákazníka, bankovní spojení na zákazníka, datum splatnosti
 - Třetí normální forma se ptá, zda každá neklíčová položka závisí přímo na primárním klíči (zda nezávisí až zprostředkování, přes jinou položku)
 - Datum vystavení je jasné vlastnost faktury,
 - částka je vlastnost faktury
 - Číslo zákazníka (vystavují to pro nějakého zákazníka, je tam jako identifikátor z informačního systému, např. ičo)
 - Adresa zákazníka - to není adresa, na kterou posílám tu fakturu, je to adresa sídla toho zákazníka -> mezi adresou a číslem zákazníka je těsnější vazba než mezi adresou zákazníka a číslem faktury
 - Datum splatnosti je atribut vlastní té faktuře
 - Nic se neopakuje, ale problém je v tom, že dva atributy je mezi nimi těsnější závislost než mezi každou z těch položek a primárním klíčem -> položky adresa a číslo zákazníka závisí na čísle faktury přes číslo zákazníka = tranzitivní závislost
 - Řešení je, že to rozdělíme na dva záznamy
- První 3 normální formy zkoumají, jestli ten neklíčový atribut závisí na primárním klíči (je jedno, jestli je složený nebo jednoduchý)
 - **Boyce / Codova normální forma (4. v pořadí)**
 - Říká nám, že může být více variant identifikátorů (např. id na isicu, login name -> jako primární klíč musíme zvolit pouze 1 z nich)
 - Eliminuje potencionální redundancy v záznamu, kde existuje více variant primárních klíčů (kandidátů primárního klíče).
 - Závislosti na vybraném primárním klíči nemusí postihovat všechny možné problémy.
 - Pokud existuje jiná varianta primárního klíče, je zvolena a opětovně je provedeno prověření prvních tří normálních forem.
 - Protože první 3 normální formy zkoumaly vztah neklíčových a klíčových položek, co když když zvolíme jinou variantu primárního klíče najdeme nějaká další zjištění -> pokud existuje, změřte jinou variantu PK a projděte znovu všechny 3 normální formy zda nezjistíme nějaký nedostatek, problém, který nám předtím unikl
 - Čtvrtá a pátá normální forma nezkoumají vztah klíčová vs neklíčová položka, ale zkoumají vztah uvnitř složeného primárního klíče (složeného minimálně ze 3 atributů) -> když najdeme složený primární klíč, většinou se jedná o tabulkou nebo o záznam který vznikl z nějakého vztahu (binární, ternární,...)
 - Zaměřují se na situace, které jsou relativně hodně komplikované (když půjdeme přes konceptuální model, neměli bychom se k nim dostat)
 - Takové záznamy vznikají nejčastěji z nebinárních vazeb = vazeb mezi 3 a více entitními množinami -> často zde dochází k chybám
 - **Čtvrtá normální forma**
 - Umožňuje rozlišení a oddělení nezávislých vícehodnotových atributů vytvářejících složený primární klíč.

➤ **Záznam KVALIFIKACE A CÍLE ZAMESTNANCIU:**

- ID ZAMESTNANCE (PK)
- KVALIFIKACE (PK)
- CIL (PK)

Umozňuje rozlišení a oddělení nezávislých vícehodnotových atributů vytvářejících složený primární klíč.

○

ID ZAMESTNANCE	KVALIFIKACE	CIL
Rosák	Účetnictví	více peněz
Rosák	Těsnopis	pochvala od vedoucího
Rosák	Účetnictví	pochvala od vedoucího
Rosák	Těsnopis	více peněz

➤ **Záznam KVALIFIKACE ZAMESTNANCE:**

- ID ZAMESTNANCE (PK)
- KVALIFIKACE (PK)

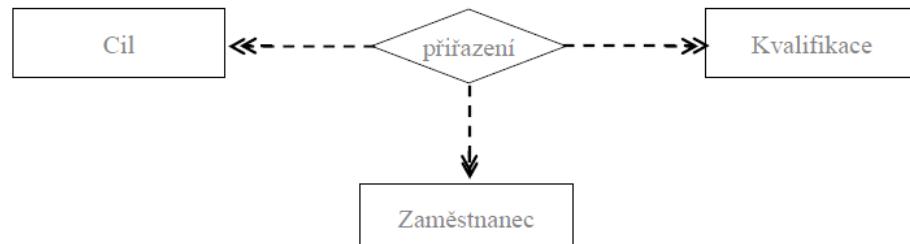
➤ **Záznam CÍLE ZAMESTNANCE:**

- ID ZAMESTNANCE (PK)
- CIL (PK)

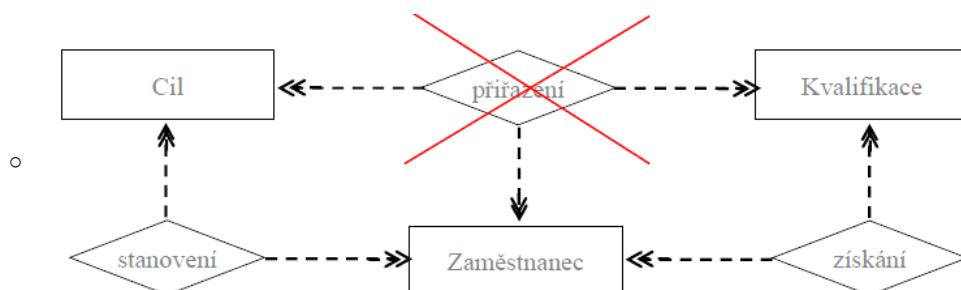
ID ZAMESTNANCE	KVALIFIKACE
Rosák	Účetnictví
Rosák	Těsnopis

ID ZAMESTNANCE	CIL
Rosák	více peněz
Rosák	pochvala od vedoucího

- Všechny 3 sloupce dohromady tvoří složený primární klíč (id zam, kvalifikace, cíl) -> nesmí se opakovat kombinace těchto hodnot v té tabulce
- Když je atribut součástí primárního klíče, tak nesmí být null, musí být vždy určena hodnota
- Kdybych chtěl přidat, že se rosák naučil čínsky musel bych to vyjádřit dvěma řádky: rosák-čínština-peníze a rosák-čínština-pochvala
- Kdyby měl 3 kvalifikace (účto, těsnopis, čínština) a přidal si další cíl být vedoucím -> musel bych přidat 3 nové řádky:
 - Rosák-účto-být vedoucím
 - Rosák-těsnopis-být vedoucím
 - Rosák-čínština-být vedoucím
- U první tabulky máme ternární vazbu:



- Budeme zkoumat, jestli opravdu všechny ty 3 hodnoty jsou spolu závislé -> říkáme, že spolu musí všechny ty 3 entitní množiny souviseť
- ale co když je to jinak, co když člověk jenom získává kvalifikace a jenom si stanovuje cíle a nemusí to spolu nutně souviseť
- Musíme to zanalyzovat, podívat se jak to v tom světě je a pokud zjistíme že jsou ty atributy nezávislé ty atributy kvalifikace a cíle a jsou závislé pouze kvalifikace na zaměstnanci a cíle na zaměstnanci, tak původní záznam zahodíme a místo něj uděláme dva záznamy, respektive vyjádříme 2 binární vazby
 - Když se rosák naučí čínsky, tak bez ohledu na to kolik měl cílů vložím jeden řádek rosák-čínština
 - Když chce rosák nový cíl, tak prostě založím v tabulce cíl jeden nový řádek rosák-být vedoucím bez ohledu na to, kolik už má kvalifikací
- **Jedna změna znamená jeden nový záznam v databázi** (předtím jsme museli zadat dva záznamy)



- Pátá normální forma

- umožňuje rozlišení a oddělení párových cyklických závislostí, které se objevují v primárním klíči složeném ze tří a více atributů.

- Záznam o VZTAHU MEZI NÁKUPČÍMI, PRODEJCI A ZBOŽÍ:
- NÁKUPČÍ (PK)
 - PRODEJCE (PK)
 - ZBOŽÍ (PK)
 - MNOZSTVÍ

Umožňuje rozlišení a oddělení párových cyklických závislostí, které se objevují v primárním klíči složeném ze tří a více atributů.

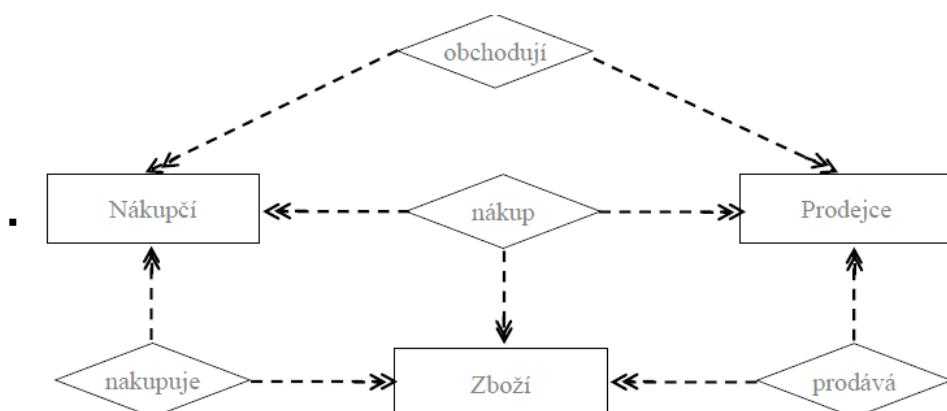
-

NAKUPCI	PRODEJCE	ZBOZI	MNOZSTVI
Rosák	Potraviny Praha	Becher	100
Novák	Potraviny Praha	Becher	200
Rosák	Potraviny Praha	Chleba	1000
Novák	Tesla Karlín	Televize	50
Rosák	Tesla Karlín	Televize	20

Předpokládejme:

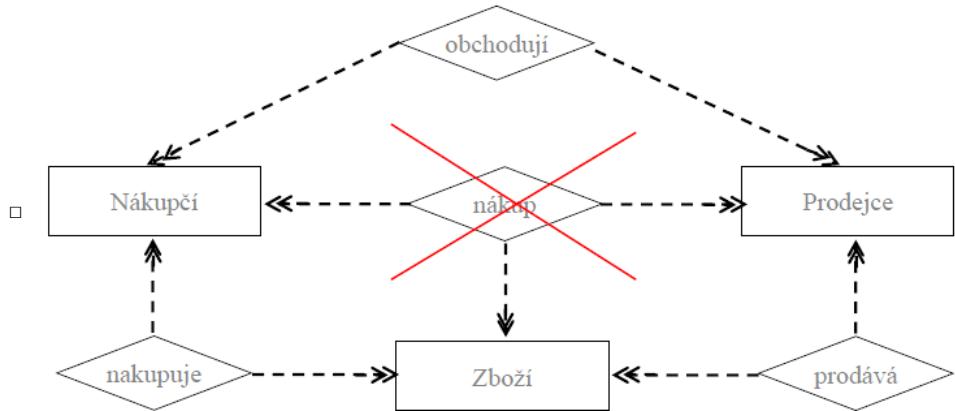
jestliže nákupčí A nakupuje od prodejce B
a současně prodejce B prodává zboží C
a současně nákupčí A potřebuje nakoupit zboží C,
pak zboží C nakoupí od prodejce B.

- Záznam obsahuje 4 atributy, ale 3 atributy (sloupce) tvoří primární klíč -> kombinace nákupčího, prodejce a zboží jednoznačně odlišuje jeden řádek od jiného řádku => nesmí se vyskytnout duplicita v těchto 3 sloupcích v nějakém řádku vůči jinému už existujícímu řádku
- U 4 normální formy jsme zkoumali to, že někdo to navrhl jako ternární vazbu, ale ve skutečnosti tam byly binární vazby -> mezi klíči nebyla závislost mezi některými atributy toho primárního klíče
- U 5 normální formy hledáme, že je tady nějaká cyklická závislost nějakých dvojic údajů, které jsou poskládány v tom klíči
- Ve světě, který popisujeme, platí následující pravidla
 - Když nějaký nákupčí nakupuje od nějakého prodejce nějaké zboží, a ten prodejce začne prodávat jiné zboží, které ale ten nákupčí má za úkol nakupovat, tak protože už mají mezi sebou ten nákupčí a prodejce vytvořené vztahy (už od sebe něco koupili), tak nákupčí si od prodejce může kupit (ale nemusí) to zboží, které začne prodejce prodávat
 - Tesla karlín začne prodávat bechera -> rosák i novák bechera už nakupují (ale od někoho jiného) -> ale protože rosák i novák od tesli nakupují už ty televize, tak protože už mají vytvořené obchodní kontakty a protože tesla karlín začala nabízet produkt becher, tak abych tuhle jednu novou skutečnost zachytily, tak bych musel přidat 2 řádky:
 - Novák - tesla karlín - becher - null
 - Rosák - tesla karlín - becher - null
 - Null protože ještě nebyl uskutečněn nákup, pouze to vyjadřuje že může nakoupit
- Zkoumáme, zda v těch složených primárních klíčích nenastávají takové nějaké závislosti a hraje velkou roli, zda v tom neklíčovém sloupci (atributech, které nejsou součástí primárního klíče) se můžou vyskytovat neurčené hodnoty

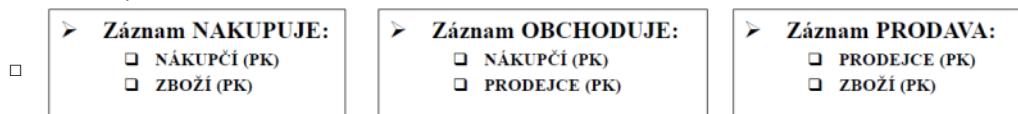


- Když začneme modelovat tak se dostaneme k tomu, že tak jak jsme naznačili to pravidlo, tak bychom mohli říct "tady to je ten původní stav, protože to odpovídá"

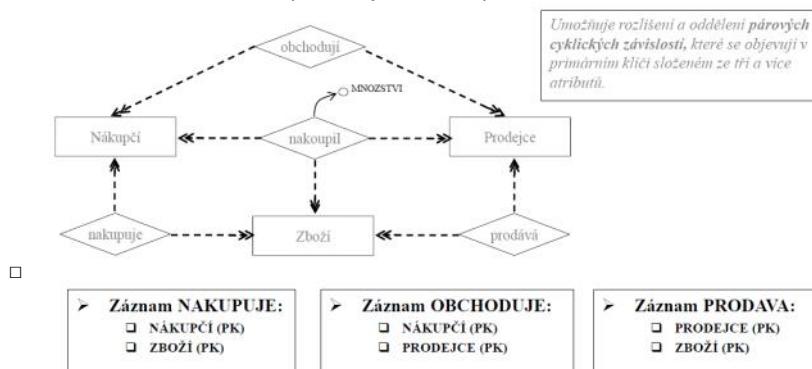
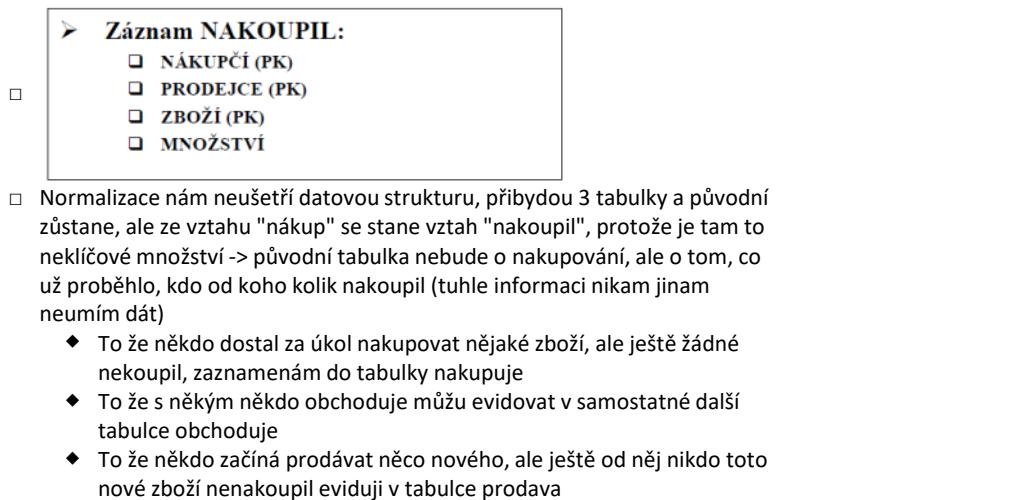
- Vztah nákup (nakupování) odpovídá tabulce výše bez sloupečku množství, a zkoumáme:
 - Zajímá nás kdo co prodává? -> ano, to jsme popsali v pravidlech
 - Stačilo by nám to vědět který prodejce prodává které zboží? -> ano
 - Zajímá nás kdo od koho co nakoupil? (kdo spolu obchoduje, kdo má uzavřené obchodní vztahy) -> ano
 - Zjistíme, jaký nákupčí má nakupovat jaké zboží? -> ano
- Když se podíváme na tento model, tak vlastně zjistíme, že k tomu abychom zachytili ten svět pořád zapomínáme (naschvál abstrahuji od toho neklíčového sloupce) tak zjistíme že ternární vztah vůbec nepotřebujeme, a že nám stačí když ho nahradíme třemi dvojicemi vazeb:

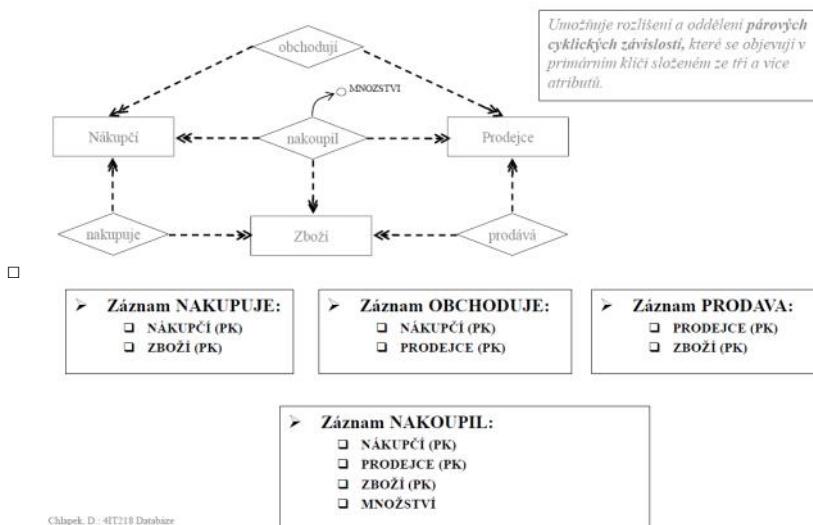


- Když vím, s kým má kdo vytvořené vztahy, když vím, kdo co prodává a když vím co kdo nakupuje, jsem schopen obsloužit to pravidlo, které bylo výše napsané
- Když bych nechtěl tam mít to množství (kolik kdo skutečně nakoupil), tak v ten okamžik můžu horní tabulku zrušit, a místo ní vytvořit 3 tabulky které by obsahovali vždy dva cizí klíče



- Ale když tam to neklíčové množství mám, tak si nemůžu dovolit tu tabulku výše zrušit



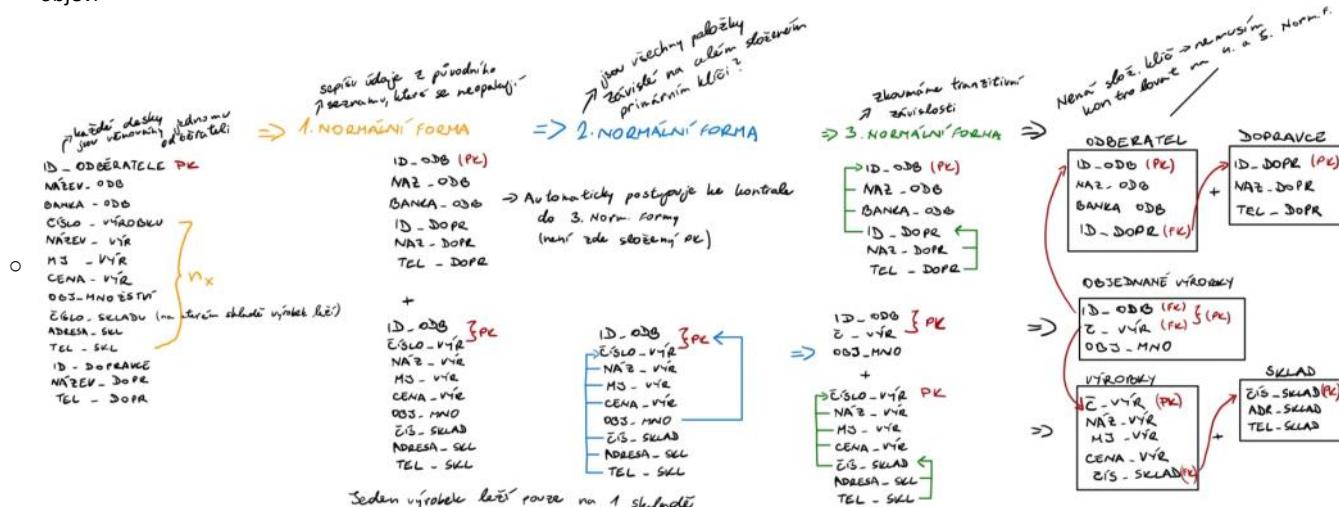


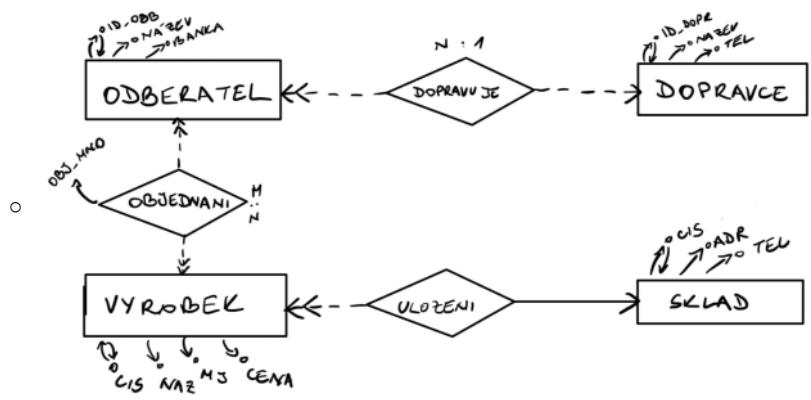
- Když tesla karlín začne prodávat bechera, tak se nedotknou tabulky nakoupil, ale pouze udělám insert na jeden řádek do tabulky prodeje (prodejce prodává jedno nové zboží) -> žádné jiné struktury nemusím aktualizovat
- V okamžiku, kdyby někdo zrealizoval ten nákup, tak to vkládám do toho rozsahu

- Rozdíl je v tom že promýšíme tu strukturu, a někdy se stane to, že v tabulce (nákup) popisuj co může být, a i to co proběhlo (zachycuji v tom vazby a současně v tom zachycuji skutečný průběh) -> z jedné tabulky jsme udělali 4 -> ušetřili jsme násobení řádků a něco, co do toho nepatří (co vypovídá o tom co se skutečně stalo, co jsme skutečně nakoupil)

Příklad:

- Nějaká firma prodává nějaké výrobky, jejich odběratelé (zákazníci) si volají objednávky na nějaké prodejní oddělení, a v prodejním oddělení sedí pracovníci, kteří si vedou jednotlivé sešity v excelu pro jednotlivé odběratele (každý odběratel má nějaký sešit), a oni podle toho kolik si ten objednatel objedná výrobku, tak mu tam zakládají tolik listů v tom sešitu -> na konci týdne se z toho udělá souhrnná objednávka
- Úkol: navrhnout databázi (ukážeme si, jak je možné normalizaci využít pro návrh databáze)
 - Podle toho jak jsme ten svět popsali, tak už z něj můžeme identifikovat některé entitní množiny, atributy, a pak už jen budeme aplikovat pravidla z přednášky před 14 dny
- Hypoteticky si vezme sešity a sepíšeme si všechny atributy, které se v těch sešitech objeví





Transakční zpracování

15 May 2024 12:23

Databázová zpracování

- Rozlišujeme 2 základní databázová zpracování:
 - **OLTP - Online TRANSACTION Processing**
 - Většina systémů (insis, bankovnictví,...)
 - zpracování velkého počtu relativně malých transakcí;
 - pojem transakce odvozen z bankovnictví
 - Bankovní transakce spočívá v tom že si někdo vybere peníze z jednoho účtu a převede je na jiný účet -> když odepřeše peníze z jednoho účtu, tak by měl informační systém garantovat, že je připřeše na druhý účet -> že situace nezůstane vmezistavu
 - V databázovém prostředí musíme připravit takové předpoklady, aby buď proběhli obě dvě operace (odebrání a připsání) a nebo žádná z nich
 - Vytváříme si množiny operací, které označíme jako jednu transakci
 - **výkonnost se měří v počtu transakcí za minutu.**
 - Pumpují se z nich data do analytických systémů které slouží pro rozhodování
 - Pracují nad nimi uživatelé, chtějí tam něco vkládat, vytvořit, vybrat a uložit peníze,...
 - **OLAP - Online ANALYTICAL Processing**
 - zpracování vícedimensionálních dotazů (OLAP kostky);
 - používá se zejména pro aplikace na podporu rozhodování, aplikace BI (Business Intelligence);
 - **výkonnost se měří v počtu dotazů za hodinu**

Transakce v databázovém zpracování

- logická posloupnost operací, která je promítána do databáze jako celek,
- slouží k zajištění konzistence databáze a k řízení víceuživatelského přístupu k datům.
- **Vlastnosti transakce (ACID):**
 - **Atomicity**
 - transakce se tváří jako jeden celek, musí proběhnout celá nebo vůbec ne
 - Když začneme něco vkládat, aktualizovat -> vkládáme 20 prací, u 21. uděláme chybu a chceme potvrdit transakci -> v ten okamžik systém vyhodnotí že uvnitř transakce vznikla chyba, a vrátí stav databáze do stavu před začátkem kdy jsme to začali vkládat (protože se vyskytla chyba - nebyla dokončena správně transakce)
 - Když vkládáme insertem, a někde v průběhu vkládání těch 20 prací napíšeme select, a podíváme se na data, tak zjistíme, že my jakožto uživatel který má právo na vkládání vidíme závěrečných prací o 20 víc, než když se k tomu přihlásí někdo jiný -> protože transakce nebyla potvrzena, tak v ten okamžik ti jiní uživatelé ze stejně tabulky vidí o těch 20 závěrečných prací méně (protože jsme ještě nepotvrdili tu transakci, a teprve při potvrzení transakce se stanou ty změny viditelné i ostatním procesům)
 - **Consistency**
 - transakce transformuje databázi z jednoho konzistentního stavu do jiného konzistentního stavu
 - Musím dokončit ty transakce do konce (musíme opravit 21. a pak dát commit) -> pak se změny stanou viditelné a tudíž jsou uloženy do databáze
 - **Independence**
 - transakce jsou nezávislé, změny prováděné jednou transakcí nejsou viditelné ostatním transakcím
 - **Durability**
 - efekty potvrzené transakce jsou uloženy do databáze
- Při úspěšném dokončení transakce se stávají všechny změny v databázi platnými.
- **Při neúspěšném ukončení transakce dojde k odstranění změn dosud vykonaných, je provedena obnova stavu databáze před zahájením zpracování neúspěšně ukončené transakce.**
- Používané příkazy:
 - BEGIN TRANSACTION
 - END TRANSACTION, **COMMIT**
 - ROLLBACK TRANSACTION.
- Slouží k:
 - **k zajištění konzistence databáze (nevznikají nám tam chyby)**
 - k řízení víceuživatelského přístupu k datům.

Nástroje pro obnovu databáze

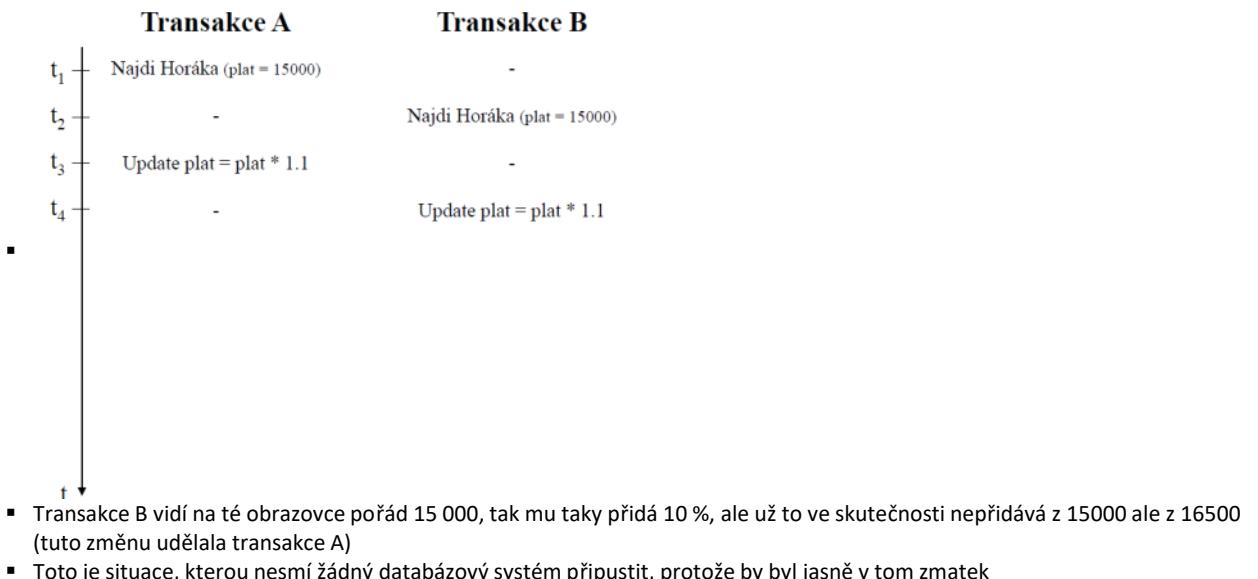
- U současných databázových systémů jsou kladený vysoké požadavky na dostupnost databáze
 - Pozn.: dostupnost 99,9 % znamená, že databáze může být nedostupná max. necelých 9 hodin za kalendářní rok.
- Nedostupnost databáze můžezpůsobit vysoké ztráty. Na druhé straně zajištění vysoké dostupnosti může znamenat velmi vysoké náklady.
- Typy ohrožení databáze, resp. její dostupnosti:
 - živelní katastrofy,
 - chyba systému, např. výpadek proudu, chyba operátora, ztráta komunikace,
 - zničení databázových souborů, např. z důvodu chyby paměťového zařízení,

- chybná data,
 - nedokončené transakce.
- zabudované do databázových systémů, stále více automatizované,
- **vytváření záloh databázových souborů**
 - **Datových souborů**
 - obsahují vlastní data, databázové objekty,
 - **Řídicích souborů**
 - obsahují seznam všech datových a žurnálových souborů náležících k databázi,
 - **Žurnálových souborů**
 - viz dále
- Databázové systémy nepracují se soubory napřímo, ale pracují přes tzv. instanci databáze ->
- vedení žurnálových (LOG) souborů, které obsahují záznamy o provedených transakcích a změnách prováděných v databázi
 - (zjednodušená struktura LOG souboru: čas, identifikace transakce, id uživatele, tabulka, řádek, hodnota před změnou - Before Image, hodnota po změně - After Image),
- kontrolní body (Checkpoint) - synchronizace "bufferů", žurnálových a datových souborů, dokončují se všechny rozpracované transakce; je zapisován záznam do řídicího souboru. Při obnově databáze je využíván poslední kontrolní bod.
- zrcadlení disků
- zrcadlení serverů
- Databázový systém musí při obnově zajistit:
 - Eliminaci nedokončených transakcí(UNDO, ROLLBACK)
 - odstranění z kopie databáze změn nedokončených transakcí,
 - využívá Before Image záznamy z LOG souboru,
 - Zotavení dokončených transakcí (REDO, ROLLFORWARD)
 - dle záznamů v žurnálovém (LOG) souboru jsou do kopie databáze doplněny změny dokončených transakcí,
 - využívá After Image záznamy z LOG souboru.

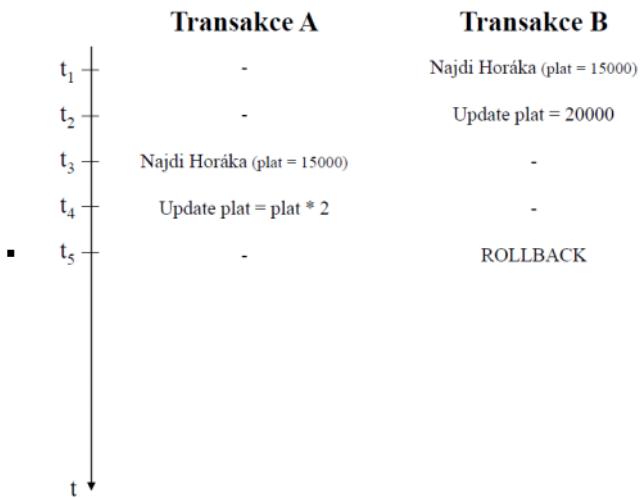
Transakce v databázovém zpracování

Řízení víceuživatelského přístupu k datům

- Nutnost chránit data zpracovávaná jednou transakcí před zásahy jiných transakcí.
- Základní typy problémů při víceuživatelském zpracování:
 - **Současná aktualizace** - aktualizace stejného záznamu více transakcemi,

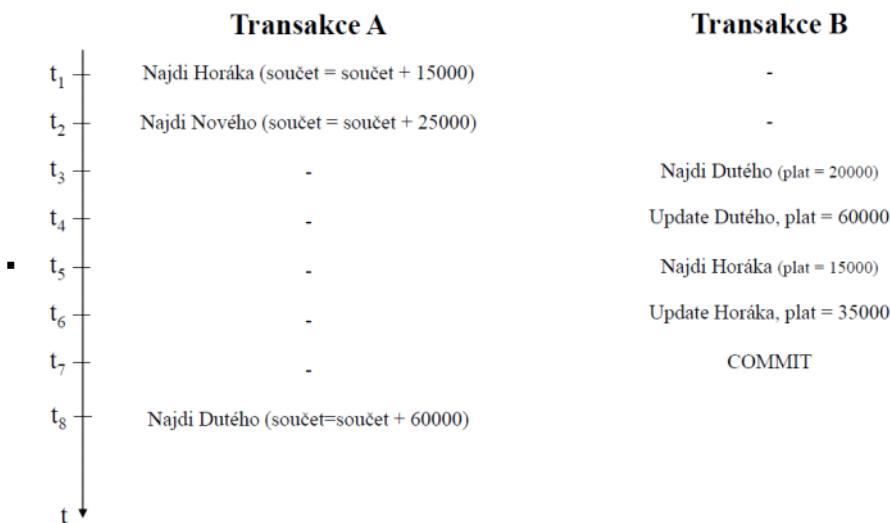


- **Závislost nepotvrzených transakcí** - transakce A vyhledá (aktualizuje) data, která transakce B změnila, ale ještě nedošlo k potvrzení transakce B, tj. existuje možnost provedení ROLLBACK,



- Transakce B nepotvrdí -> transakce A vidí ta data původní (v čase před začátkem transakce B) -> transakce B v čase t5 udělá rollback (vrať mi stav databáze před začátek) -> situace ke které nesmí dojít (horák by neměl ani 30k, ani 20k, ale měl by zase původních 15k)

- **Nekonzistentní analýza** - transakce A vyhledává údaje, které současně jiná transakce aktualizuje.



- Na konci dostaneme výsledek, který není pravda

- Toto vše jsou běžné situace, které mohou nastávat, a ty databázové systémy musí toto opečovat -> jednoduché řešení: data, nebo nějaké jejich části (řádky) se zamykají

- **Řešení problémů při víceuživatelském zpracování**

- 1. USPOŘÁDÁNÍ TRANSAKcí,
- 2. ZAMYKÁNÍM
 - tj. zajištěním, že po dobu práce s daty určitou transakcí nedojde k jejich změně v důsledku práce jiné transakce.

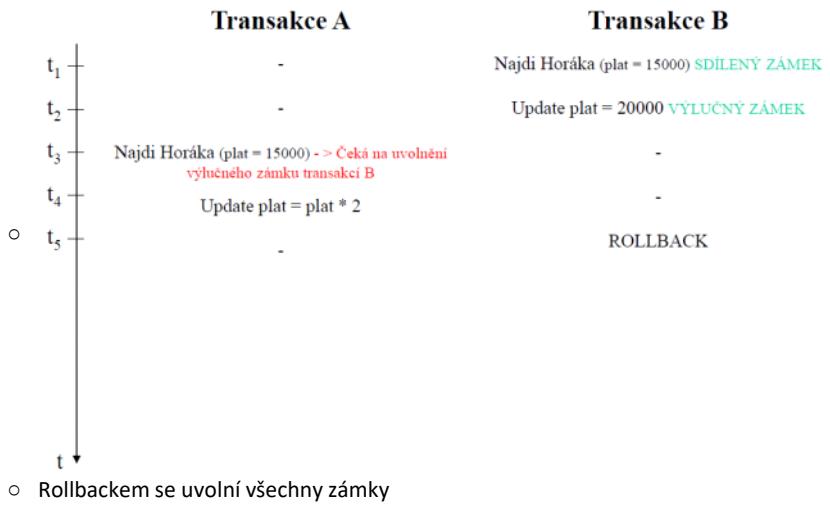
- **Typy zámků:**

- **VÝLUČNÉ (EXCLUSIVE)**
 - pro operace INSERT, UPDATE, DELETE,
 - pokud transakce A zamkne záznam Z, ostatní transakce čekají až do uvolnění zamknutého záznamu transakcí A
 - Když zamknu data žádní jiná transakce k nim nemůže
- **SDÍLENÉ (SHARED)**
 - pro operaci SELECT,
 - pokud transakce A zamkne záznam Z sdíleným zámkkem, ostatní transakce, které potřebují záznam zamknout výlučným zámkkem, čekají až do uvolnění zamknutého záznamu transakcí A,
 - pokud transakce A zamkne záznam Z sdíleným zámkkem, ostatní transakce mohou záznam Z zamknout sdíleným zámkkem

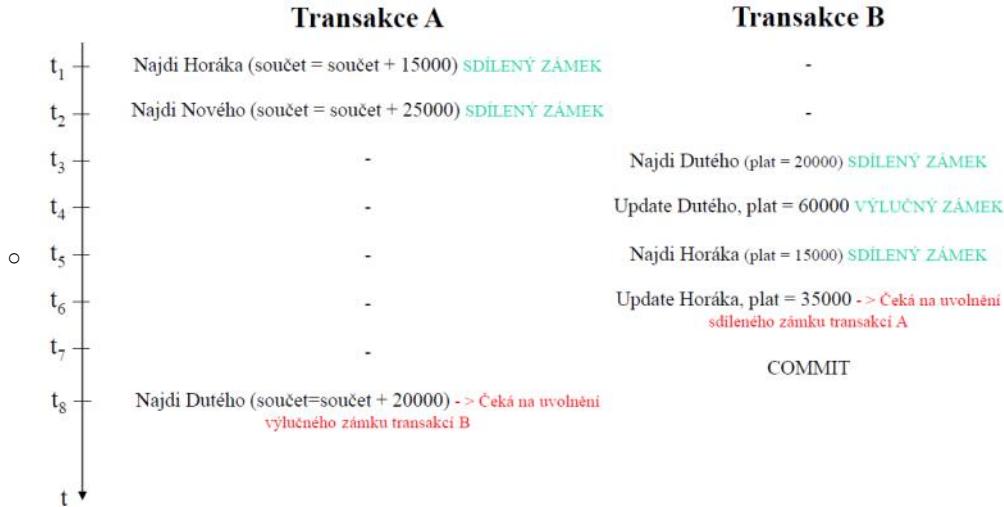
- **Úrovně zamykání:**

- databáze,
- celá relační tabulka,
- databázová stránka (datový blok),
- záznam, řádek relační tabulky,
- položka záznamu (atribut -sloupec řádky relační tabulky).

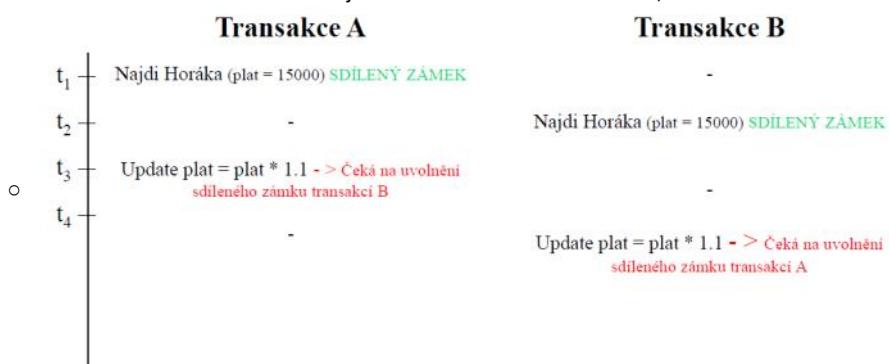
- Nejčastěji se zamykají řádky
- Na čím nižší úrovni se zamyká, tím je řešení otevřenější
- **Závislost nepotvrzených transakcí** - transakce A vyhledá (aktualizuje) data, která transakce B změnila, ale ještě nedošlo k potvrzení transakce B, tj. existuje možnost provedení ROLLBACK,



- **Nekonzistentní analýza** - transakce A vyhledává údaje, které současně jiná transakce aktualizuje.



- **Současná aktualizace** - aktualizace stejného záznamu více transakcemi,



- **DEADLOCK**

- = stav, ve kterém dvě nebo více transakcí čekají na podmínky, které nikdy nenastanou.
- Deadlock se řeší:
 - prevencí
 - při zahájení transakce jsou zamčeny všechny záznamy (omezené použití, obtížná predikce záznamů, které bude potřebovat transakce zamknout)
 - detekcí (např. čerpáním časového limitu)
 - Po detekování zablokovaných transakcí musí DBS určit ty transakce, které budou "násilně" ukončeny a provést jejich ROLLBACK (uvolnění zámků) a opětovné nastartování ukončených transakcí.
- Transakce se používají pro měření výkonnosti databázových systémů

Měření výkonu databázových systémů

- TPC (Transaction Processing Performance Council)
 - organizace zabývající se rozvojem testů, rozhoduje o korektnosti výsledků
 - publikuje výsledky testů na <http://www.tpc.org>
 - členové zástupci hw a sw firem, standardizační organizace, odborníci
- Hlavní členové TPC :

Full Members				
ACTIAN	Alibaba.com	AMD	柏睿数据	CISCO
DELL Technologies	FUJITSU	Hewlett Packard Enterprise	HITACHI	HUAWEI
IBM	inspur	intel	Lenovo	Microsoft
Netrix	NUTANIX	NVIDIA	ORACLE	Red Hat
TTA	vmware			

Associate Members				
中电飞腾	innec	清华大学	中国电子科技大学	

Používané testy TPC

- TPC-C**
 - Používán od roku 1992 **pro OLTP databázové systémy**.
 - Simuluje informační systém velké dodavatelské firmy s množstvím skladů s kapacitou 100 tis. kusů, ze kterých je zásobováno deset oblastí, každá s 3 tisíci zákazníky.
 - Data jsou uložena v devíti tabulkách s pevně definovanými atributy a určenými primárními a cizími klíči.
 - Nad daty se provádí 5 různě složitých transakcí. Jsou definovány i další parametry, např. počet kontrolních bodů, minimální testovaný čas, poměry prováděných transakcí.
 - Výsledky udávány vpočtu transakcí za minutu (tpm), ceně systému vůči počtu transakcí za minutu (cena/tpm).**
- TPC-E**
 - Test **pro OLTP databázové systémy**.
 - Simulace makléřské firmy, se zákazníky kteří vytváří obchodní transakce, účetní a marketingové průzkumy.
 - Makléřská firma komunikuje s finančními trhy pro realizaci objednávek zákazníků a provádí aktualizaci jednotlivých účtů.
 - Test je škálovatelný pro simulaci různě velkých zatížení databázových systémů.
 - Test definuje poměr různých transakcí.
 - Výsledky udávány vpočtu transakcí za sekundu (tps) a v poměru ceny vůči výkonu (price/tps)**
- TPC-H, TPC-DS, TPC-DI**
 - Test **pro OLAP databázové zpracování**, systémy pro podporu rozhodování.
 - Test simuluje ad-hoc komplexní dotazy a současnou aktualizaci velkých objemů dat.
 - Výsledky udávány vpočtu dotazů za hodinu (QphH) a v poměru ceny vůči výkonu (price/ QphH)**
- další testy pro:
 - Big Data (TPCx-HS, TPCx-BB),
 - Virtualizaci (TPCx-V, TPCx-HCI),
 - Internet věcí (TPCx-IoT),
 - Umělou inteligenci (TPCx-AI),
 - Energetickou náročnost, ceny (TPC-Energy, TPC-Pricing)

Fyzické struktury – implementační úroveň návrhu datové základny

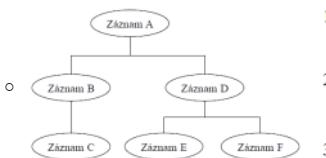
15 May 2024 12:51

• Datová struktura

- účelné uspořádání datových prvků v paměti počítače,
- soubor pravidel a omezení, která určují vztahy mezi jednotlivými prvky datové struktury.
- = Způsob jak uspořádáme nějaké prvky (např. do hierarchické struktury nebo do relační -> relační tabulka je podmnožina kartézského součinu)
- Každá ta struktura s sebou nese i množinu operací, kterou s nimi můžeme dělat
 - V relačním prostředí si představujeme data jako relační tabulky (ve skutečnosti to relační tabulky nejsou, je to jenom nejdokonalejší, ale stále nepřesná, vizualizace množin, které vznikly z kartézských součinů)
- Data můžeme interpretovat různě

• Logická datová struktura

- abstraktní představa vztahů mezi daty.



- Hierarchická struktura -> chceme nějaký svět, který je uspořádán hierarchicky (strom -> kořen -> uzly -> listy) -> jak to fyzicky v té paměti udělat? Jak data navkládat? -> několik možností:

• Fyzická datová struktura

- způsob realizace logické datové struktury v paměti počítače.

Varianta 1 (k záznamům přidán údaj o úrovni, na které se záznam ve stromu nachází)



Varianta 2 (k záznamům přidány dva směrniky odkazující na podřízené záznamy)

- | | | | | | | | | | | | | | | | | |
|---|----|----|---|----|--|---|--|---|----|----|---|--|--|---|--|--|
| A | SB | SD | B | SC | | C | | D | SE | SF | E | | | F | | |
|---|----|----|---|----|--|---|--|---|----|----|---|--|--|---|--|--|

Varianta 3 (k záznamům přidán směrnik odkazující na nadřízený záznam)



- Varianta 1 -> sice poznám na které úrovni ten kurzor je, ale nepoznám, kdo je jeho bezprostředně nadřízený
- Varianta 3 -> a nemá nadřízený záznam -> je to kořen stromu

- Musíme vědět, jak ten databázový systém který používáme si ta data ve skutečnosti ukládá -> podle toho nastavíme způsob uložení vhodný tak, aby vyhověl tomu způsobu jak to chceme

• Organizace dat

- konkrétní způsob (varianta) uspořádání záznamů na paměťových médiích počítače,
- Pozn.: V některých databázových systémech je možné vybírat či ovlivnit způsob organizace dat.

• Způsob fyzické realizace datové struktury je dán

- způsobem uložení dat
 - sekvenční
 - přímé
- způsobem přístupu k datům
 - sekvenční
 - přímý
 - s využitím dodatečných přístupových mechanismů (řetězení, indexy)
- způsobem vyhledávání dat

Sekvenční způsob uložení dat

- záznamy jsou ukládány v pořadí odpovídajícím hodnotám primárního klíče (setřídění dle PK).
 - Je možné využít zrychlených způsobů vyhledávání, např. postupné, intervalové, binární)
- | | | | | | | | | | | | |
|----|------|----|-------|----|-------|----|-------|----|------|----|-------|
| 27 | Nový | 29 | Paták | 31 | Rosák | 42 | Bosák | 57 | Nový | 61 | Novák |
|----|------|----|-------|----|-------|----|-------|----|------|----|-------|
- záznamy jsou ukládány v pořadí daném pořadím v jakém jsou vkládány
 - Bez dodatečných přístupových mechanismů je možné použít pouze úplné vyhledávání.
- | | | | | | | | | | | | |
|----|------|----|-------|----|------|----|-------|----|-------|----|-------|
| 27 | Nový | 61 | Novák | 57 | Nový | 31 | Rosák | 42 | Bosák | 29 | Paták |
|----|------|----|-------|----|------|----|-------|----|-------|----|-------|
- Pozn.: Sekvenční způsob uložení dat není využíván pro uložení dat v databázových systémech. Bývá používán pro zálohování databázových souborů.
- když máme setříděna data, a když hledám zaměstnance 43, tak procházím postupně záznamy a když narazím na záznam který má PK osobní číslo větší než 43 (a já jsem ještě záznam s PK 43 nenašel) -> vím že už nemusím číst dál.
- Problém -> Ale když mám data nesetříděná, tak je musím přečíst všechny (to se často děje v databázových systémech)

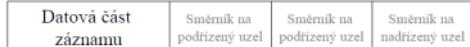
Přímý způsob uložení dat

- hodnotě primárního klíče jednotlivých záznamů je přiřazena (vypočtena) určitým algoritmem (hashing algorithm) adresa uložení záznamu na paměťovém médiu,
 - umožní nejrychlejší způsob přístupu k datům, avšak pouze podle primárního klíče,
- hodnota PK záznamu → Převodní algoritmus → vypočtená adresa, na kterou bude záznam uložen
- existují různé druhy algoritmů,
 - obecně rozlišujeme
 - přímé adresování
 - 1 hodnotě PK je vypočtena 1 adresa
 - (PK <-> ADRESA)
 - nepřímé adresování
 - více hodnotám PK je vypočtena stejná adresa
 - (PK <<-> ADRESA)

Dodatečné přístupové mechanismy - ŘETĚZENÍ DAT

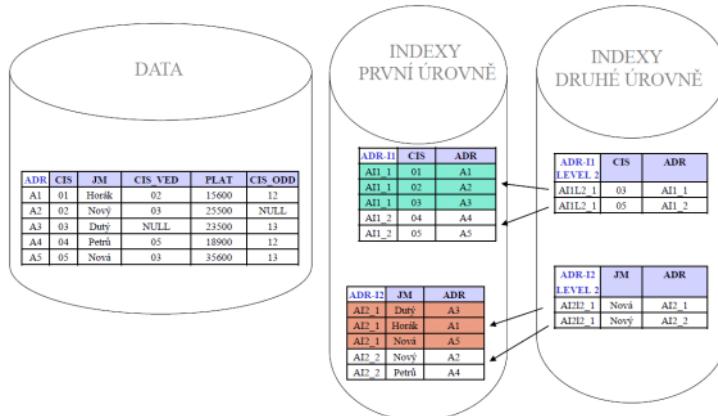
- umožňuje vyjádřit mezi záznamy logicke vztahy,

- záznamy jsou rozšířeny o směrnikovou část, tj. jeden nebo více směrníků
- směrníkem může být:
 - absolutní adresa,
 - relativní adresa,
 - symbolický směrnik
 - hodnota primárního klíče,
 - hodnota umělého klíče.
- pro každý typ logické vazby mezi daty je použit jeden směrnik



Dodatečné přístupové mechanismy - INDEXNÍ STRUKTURY

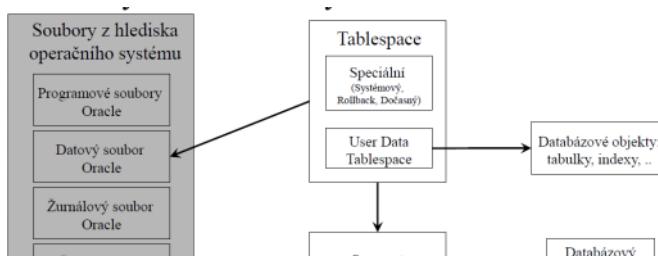
- INDEX = dvojice údajů:
 - HODNOTA KLÍČE ADRESA ZÁZNAMU
- Vytvořím někde další strukturu, která obsahuje údaje pouze o hodnotě klíče, přes který indexuju, a adresu, na které se nachází záznam s tím klíčem
- index je možné přirovnat k indexům v knihách,
- indexovány nemusí být pouze atributy tvorící primární klíč, ale i sekundární klíče,
- indexy jsou nejčastěji ukládány v samostatných strukturách (indexní stromy, indexní tabulky) mimo vlastní data,
- pro každý index je vytvářena samostatná indexní struktura,
- indexy slouží pro zrychlení vyhledávání dat, současně mohou způsobit zpomalení aktualizačních operací v databázi, neboť je třeba mimo vlastní data provádět aktualizace i všech s aktualizovanými daty souvisejících indexních struktur,
- rozlišujeme indexní struktury:
 - statické (počet úrovní indexů je dopředu pevně stanoven, počet úrovní se nemění v závislosti na počtu indexovaných záznamů),
 - dynamické (počet úrovní indexů závisí na množství indexovaných záznamů) - například B - stromy.



- dynamické (počet úrovní indexů závisí na množství indexovaných záznamů)
 - B - stromy (1972 -Bayer, McCreight)
 - každý uzel má maximálně m dětí,
 - každý uzel kromě kořene a listů má nejméně m/2 dětí,
 - kořen má nejméně 2 děti, pokud není listem,
 - všechny listy jsou na stejně úrovni,
 - každý uzel, který není listem a má n dětí obsahuje n-1 hodnot klíče,
 - jestliže uzel U je podřízený uzel (dítětem) uzlu V s hodnotami klíčů v1, v2, .. vn-1, potom platí pro všechny hodnoty klíčů ui v uzlu U:
 - ui < v1, jestliže U je první dítě
 - vj-1 ≤ ui < vj, jestliže U je j-té dítě, 1 < j < n
 - ui ≥ vn-1, jestliže U je n-té dítě
 - uzel, který není listem obsahuje n-1 hodnot klíče a n směrníků
 - Pozn.:
 - strom se vytváří ze zdroje nahoru,
 - uzly mohou obsahovat jak hodnoty klíčů a adresy (indexy), tak i vlastní data

Fyzické struktury - ORACLE

- Databáze obsahuje následující datové struktury:
 - (fyzické) datové soubory,
 - žurnálové soubory (Redo Log Files),
 - řídící soubory,
 - objekty schématu,
 - datový blok,
 - oblast (extent),
 - segment,
 - tabulkový prostor (tablespace).



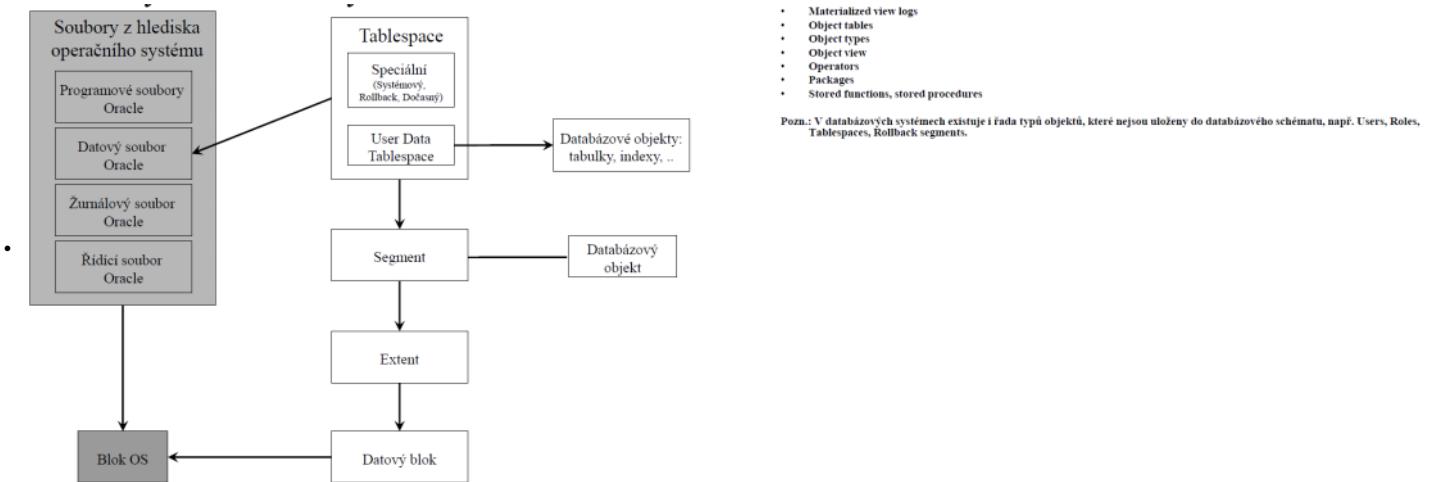
Objekty databázového schématu

DB schéma je kolekce logických struktur dat nebo databázových objektů. Schéma je vlastněno uživatelem databáze a má název shodný se jménem uživatele. Každý uživatel vlastní jedno schéma. DB Objekty jsou vytvářeny a upravovány pomocí jazyka SQL nebo s pomocí dalších nástrojů (např. v DB Oracle, SQL Developer, ...).

Seznam typů databázových objektů

- Tables
- Views
- Synonyms
- Constraints
- Sequences
- Database triggers
- Clusters
- Database links
- Dimensions
- External procedure libraries
- Index-organized tables
- Indexes
- Indextypes
- Java classes, Java resources, Java sources
- Materialized views
- Materialized view logs
- Object tables
- Object types
- Object view
- Operators
- Packages
- Stored functions, stored procedures

Pozn.: V databázových systémech existuje i řada typů objektů, které nejsou uloženy do databázového schématu, např. Users, Roles, Tablespaces, Rollback segments.



(fyzický) datový soubor

- Ukládají se vlastní data, indexy, ...
- každá databáze má alespoň jeden datový soubor
- může být přiřazen pouze jedné databázi
- může být nastaven pro automatické rozšiřování velikosti

žurnálové soubory (Redo Log Files)

- určeny pro záznam všech změn dat provedených v databázi
- slouží pro obnovu databáze a zajištění konzistence
- každá databáze má dva nebo více žurnálových souborů
- může existovat více kopií žurnálových souborů (z důvodu zajištění vyšší bezpečnosti zpracování a obnovy na různých discích)

řídící soubory

- každá databáze má řídící soubor
- obsahuje údaje o fyzické struktuře databáze, např.: jméno databáze, jména a umístění fyzických datových souborů a žurnálových souborů, datum vytvoření databáze
- mohou existovat kopie uložené na různých paměťových médiích
- dojde-li ke změně fyzických souborů (datových či žurnálových souborů), je automaticky provedena změna řídícího souboru

Schéma

- je množina databázových objektů (např. tabulky, view, sekvence, indexy, uložené procedure atd. Schéma je vlastně uživatelem databáze a má shodné jméno se jménem uživatele db).

datový blok

- je základní fyzickou jednotkou uložení a přenosu mezi vnější a vnitřní pamětí počítače. Velikost datového bloku je určena při vytváření databáze.
- Každý datový blok má následující strukturu:
 - záhlaví
 - adresář záznamů, umístěných do bloku
 - volný prostor
 - prostor obsazený záznamy

extent

- je určitý počet spojitéch datových bloků

segment

- je množina oblastí alokovaných pro určitou logickou strukturu - databázový objekt.
- Segment nemusí být spojity.
- Např. každá tabulka má přiřazen datový segment, každý index má svůj segment atd.
- Databázový systém dynamicky alokuje prostor. Je-li segment zaplněn, je přidán další extent.

- Každá databáze je logicky rozdělena do jednoho nebo více logických tabulkových prostorů, které jsou uloženy na vnějších paměťových médiích prostřednictvím fyzických datových souborů
 - fyzický datový soubor << ----- > logický tabulkový prostor

Tabulkový prostor (table space)

- databázová struktura, která sdružuje související logické struktury,
- obsahuje charakteristiky, které určují, jak objekty prostor využívají (např. procentuální vyjádření volného místa rezervovaného pro změny),
- tabulkový prostor může být nadefinován také tak, že se automaticky zvětší, čímž je možné předejít jeho přetečení,
- prostor používaný tabulkovým prostorem však nemůže být větší, než je fyzický prostor datových souborů přiřazených tabulkovému prostoru,
- každý objekt vytvořený v databázi je součástí nějakého tabulkového prostoru. Při vytváření objektu je možné určit, kolik prostoru si má objekt alokovat. Tuto hodnotu je možno později změnit,
- celkový prostor používaný objektem nemůže být větší, než je celkový prostor v daném tabulkovém prostoru,
- tabulkový prostor lze deaktivovat.

Způsob uložení databázových tabulek v dbs Oracle

- do jednoho tabulkového prostoru je přiřazena jedna či více tabulek.
- řádky tabulek jsou uloženy v datových blocích. Pokud se nejdá o tabulku s clustery či dělenou tabulkou, řádky nemusí být uloženy v přesném pořadí.
- nové řádky jsou přidávány na konec bloku, zaplň-li se datový blok na hranici danou požadavkem na volné místo pro změny, jsou data uložena do následujícího datového bloku
- není-li v datovém bloku dostatek místa pro uložení řádky tabulky, jsou data uložena do více datových bloků, které se zřetězí. Zřetězení snižuje výkonnost dbs.
- není možné, aby dvě tabulky sdílely jeden datový blok. Tato omezení neplatí pro
 - dělené tabulky
 - rozsáhlé tabulky, je možné rozdělit do více částí (partitions), každá část může být přiřazena do samostatného tabulkového prostoru a každému tabulkovému prostoru odpovídají jiné fyzické datové soubory.
 - tabulky s clustery
 - několik tabulek, které spolu těsně souvisí a nad kterými jsou často prováděny společné dotazy, je možné sdružit do clustera (shluku). Při

sdružení tabulek dochází ke sloučení řádků dvou tabulek, takže tabulky sdílejí stejné datové bloky.

Optimalizace v databázových systémech

15 May 2024 13:11

Podle čeho poznáme, že není potřeba optimalizovat? -> podle doby odevzdy (max 3s)

- důvod:
 - nedostatečná výkonnost systému vzhledem k stanoveným výkonnébním požadavkům IS/ICT
- cíle:
 - optimalizovat výkonnost dbs
 - optimalizovat náklady (strojový čas, kapacita paměti, čas přenosu, práce programátorů, čas uživatelů, ..)
 - nalezení optimálního poměru mezi kapacitou paměti, časem potřebným pro výběr dat, časem potřebným pro aktualizaci dat.
- Výkonnost je charakterizována
 - dobu odevzdy (čas mezi odesláním požadavku a získáním výsledku)
 - počtem transakcí zpracovaných za minutu
 - dostupnosti (po jakou dobu je systém dostupný a plní své funkce)
 - škálovatelnosti (možnost přizpůsobit výkon vzhledem ke změně nároků požadavků)
- Kvalitu datové základny ovlivňují
 - tvůrci základního programového vybavení (operační systémy, databázové systémy, ..),
 - analytické a návrháří IS/ICT,
 - tvůrci aplikačního programového vybavení,
 - správce databáze.
- Výkonnost dbs je ovlivňována činnostmi ve všech etapách životního cyklu IS/ICT dle MMDIS (od informační strategie až po provoz a údržbu IS/ICT).
- Používané kategorie technik :
 - analytické a výpočtové (např. výpočty různých variant zpracování dotazu)
 - monitorovací a měřící (např. snímkování, simulace, testování)

Způsoby zvýšení výkonnosti dbs

- konfigurace a úprava struktury dbs s cílem snížení spotřeby zdrojů (např. snížení počtu I/O operací, zátěže pevných disků, přenosů dat po síti)
- navýšení kapacit (většinou hw komponent)
- úprava požadavků na dbs (např. jejich rozložení s cílem dosáhnout nižší zatížení systému, či využití paralelního zpracování)
- Normalizace vs Denormalizace
 - Normalizace - redukuje množství dat v db odstraňováním redundancí
 - Denormalizace - proces porušování pravidel daných normalizací dat (vzdalování se od modelu světa v db) s cílem zvýšení výkonnosti systému – nevýhody musí být převáženy přínosy takového řešení
 - výhodou je např. snižování potřeby spojování tabulek, uchování aktuálních hodnot odvozených atributů
 - nevýhodou je např. snížení rychlosti aktualizačních operací (Insert,Update,Delete), často komplikovanější aplikační řešení, obtížnější zajištění integrity databáze
- Umístění dat do vnitřní paměti
- Vhodný výběr datových typů
 - číselné typy oproti CHAR, zvážení nezbytnosti použití UNICODE a BLOB, použití neurčené hodnoty (NULL), v některých dbs je vymezen prostor pro typ CHAR, i když žádána hodnota není vložena.
- Komprese tabulek
 - komprese zvyšuje nároky na CPU, ale současně snižuje nároky
 - na prostor na vnějších médiích,
 - na I/O operace,
 - na prostor a rychlosť zálohování.
 - Např. v ORACLE existují následující typy komprese tabulek:
 - základní - pouze pro dávkové nahrávání dat, komprese na úrovni bloků, kompresní poměr min. 50%,
 - OLTP - data jsou komprimována při všech DML operacích,
 - Hybrid Columnar Compression (HCC) - kombinace sloupcového a řádkového uložení dat, Oracle Exadata
- Vyvážená konfigurace a parallelizace
 - Je třeba pečlivě navrhovat:
 - CPU a RAM
 - velikost úložiště dat
 - propustnost I/O operací.
 - a zkoumat, jak jsou jednotlivé části dimenzovány a ve skutečnosti využívány
- Parallelizace
 - nastavení, aby CPU bylo využito alespoň na 50%
 - viz samostatná přednáška o Oracle
- Rozmístění dat na databázových stránkách, tabulkových prostorech
- Využití indexů
 - množství, druh a způsob organizace indexů – zrychlení výběru, zpomalení aktualizace.
 - Čas na uložení řádku tabulky se skládá z času na uložení dat (relativní konstantní doba) a z času na záznam indexu (doba závislá na nutnosti reorganizovat – využít indexní struktury)
- Využití shlukování dat (cluster)
 - více tabulek (často propojovaných) sdílí stejné databázové stránky
- Optimalizace dotazů (cílem vybrat plán provedení operací nad relačními tabulkami tak, aby výhodnocení bylo co nejefektivnější),

Typy optimalizátorů

- optimalizace řízená syntaxí
- optimalizace založená na indexech a velikostech relací
- optimalizace založená na relativní selektivitě relačních operátorů (pro daný operátor se určuje v procentech velikost selekcí z původní relace) – např. předpoklad že operátor = má menší selektivitu než operátor >.
- statisticky řízená optimalizace

Kroky optimalizace

1. Kontrola syntaxe a sémantiky příkazu
 2. Převedení dotazu do interní reprezentace (např. operací relační algebry).
 3. Optimalizace použitých operací na základě obecných pravidel pro možné úpravy posloupnosti operací, např.: $(A \text{ join } B) \text{ where } (\text{restrikce } B) \Rightarrow (A \text{ join } (B \text{ where restrikce } B))$
 4. Výběr vhodných způsobů realizace příkazů (výběr manipulačních procedur s fyzickými datovými strukturami). Pozn.: Tepřve v tomto kroku jsou zohledněny přístupové mechanismy, aktuální hodnoty dat a způsoby uložení.
 5. Vytvoření variant, ohodnocení a výběr nejvhodnější varianty realizace příkazu - hledání "nejlevnějšího" plánu provedení příkazu.
 6. Generování kódu.
- Příklad:

MATERIAL (CISLO, NAZEV, MJ, CENAMJ) 5.000ř
 ULOZENI (MATERIAL, SKLAD, MNOZSTVI) 20.000ř
 Dotaz: Select nazev from material, ulozeni where cislo=material and sklad = 3;

Var. A: (cca 300.000.000 I/O operací)
 1. R1 = MATERIAL X ULOZENI R,W: 100.000.000 ř
 2. R2 = p R1 (CISLO = MATERIAL AND SKLAD = 3) R: 100.000.000 ř, red. na 50ř
 3. R3 = π R2 (NAZEV)

Var. B: (cca 500.000 I/O operací)
 1. R1 = ULOZENI (SKLAD = 3) R: 20.000ř, W 50 ř
 2. R2 = MATERIAL X R1 R: 5.000ř x 50 ř = W: 250.000ř
 3. R3 = p R2 (CISLO = MATERIAL) R: 250.000ř, red. na 50ř
 4. R4 = π R3 (NAZEV)

Nejčastější chyby při použití dbs

1. Nevyvážená architektura - silný výpočetní výkon, ale poddimenzované I/O systémy
2. Špatná správa spojení – časté připojování a odpojování aplikace od databáze.
3. Chybné využití kursorů a předem přeložených dotazů – nepoužívání kursorů má za následek opakováný překlad dotazů. Pozor na aplikace generující dynamické SQL.
4. Nesprávný kód SQL – špatný SQL kód je takový, který využívá více systémových zdrojů než je přiměřené pro prostředí dané aplikace..
5. Použití nestandardních inicializačních parametrů
- Většina systémů pracuje s přijatelným výkonom, jestliže jsou nastaveny pouze základní parametry.
6. Špatná nastavení Redo logů – málo a malé redo logy mohou zapříčinit nadměrnou vytíženosť buffer cache (cache pro zápis do datových souborů) a následně také I/O systému. Zápis do logů pak „zdržuje“ celou databází.
7. Serializace datových bloků v buffer cache kvůli nedostatku volných listů, skupin listů, transakčních slotů nebo rollback segmentů – to je běžné u aplikací, které vkládají hodně záznamů a byla nastavena velikost databázového bloku více než 8KB nebo u aplikací s velkým množstvím aktivních uživatelů a málo rollback segmenty.
8. Zbytečné full table scany – může být způsobeno špatným designem transakcí, chybějícími indexy nebo špatnou optimalizací SQL příkazů. Způsobují zejména přetížení I/O systému.
9. Vysoké objemy rekurzivního SQL – pokud se hodně rekurzivních SQL příkazů vykonává pod uživatelem SYS..
10. Chyby při nasazení a migraci – při migraci z vývojového prostředí nebo starší implementace se může stát, že schéma vlastníci tabulků nebylo úspěšně zmigrrováno. Příkladem jsou chybějící indexy nebo chyběné statistiky. Dochází pak k přetěžování databáze.
11. Nevyužití komprese tabulek - t.j. zejména zbytečné místo na disku, zatěžování I/O systémů, nároky na zálohování.

Distribuované databázové systémy

- DB obsahuje data rozmištěná do dvou či více uzlů sítě,
- uzly sítě jsou spojeny komunikační sítí,
- v libovolném uzlu mohou uživatelé i programy zpracovávat data jako by byla umístěna v jedné globální DB umístěné v tomto uzlu,
- všechna data nacházející se v uzlu X a používaná v globální DB mohou být zpracována uživateli v uzlu X stejným způsobem jako by šlo o lokální DB izolovanou od zbytku sítě.

Distribuční nezávislost

Nezávislost umístění

Techniky při distribuci

- **Fragmentace** (části relační tabulky jsou rozděleny do lokálních databází):
 - horizontální fragmentace (restriky)
 - vertikální fragmentace (projekce)
- **Replikace** (opakováný výskyt identické kopie tabulky v různých uzlech sítě)
- **Alokace** (kombinace ad a) a b))
- **2PC (Two Phase Commit) - zajistění integrity distribuované databáze, distribuovaná transakce se provede ve všech uzlech sítě, nebo v žádném.**
- **Údržba replikovaných dat** - strategie řešení:
 - současná aktualizace všech replik (kopii) - není-li jeden uzel sítě dostupný nemůže být transakce dokončena,
 - provádění aktualizace se zpožděním - určení primární repliky, po aktualizaci primární repliky, je tento uzel DDBS zodpovědný za provedení aktualizace i na všech ostatních replikách.

Základní vlastnosti DDBS (C.J.Date)

- k uživateli se DDBS chová jako nedistribuovaná databáze,
- musí být zajištěna lokální autonomie,
- nezávislost na centrálním uzlu (rovnopravnost uzlů),
- nepřetržitý provoz,
- nezávislost na umístění dat,
- nezávislost na fragmentaci dat,
- nezávislost na replikaci dat,
- zajištění optimalizace datou nad DDB,
- řízení distribuovaných transakcí (2PC),
- nezávislost na HW,
- nezávislost na OS,
- nezávislost na síti,
- nezávislost na DBS (integrace heterogenních DBS)

Distribuované databázové systémy v prostředí webu

CAP Brewerův teorém:

- ACID vlastnosti transakcí nemusí být vždy vyžadovány
- platí tři (CAP) požadavky, které však nemohou být v distribuovaném systému splněny současně:
 - C (Consistency) - všechny uzly mají shodná data, která věrně odražejí zobrazován svět
 - A (Availability) - každá operace skončí v požadované době odezvy, resp. bude poskytnuta informace o úspěšnosti, či neúspěšnosti provedení operace
 - P (Partitioning tolerance) - jestliže se část sítě přeruší nebo dojde ke ztrátě přenosu dat bude systém stále schopný realizovat požadované operace.
- Řeší se replikačními mechanismy a odloženou aktualizací
- dokázán Brewerův teorém tvrdí, že pro jakýkoliv distribuovaný systém sdílení dat je nemožné garantovat všechny tři CAP vlastnosti současně.
- zejména u webových aplikací je nutné se rozhodnout mezi C a A
- klasické DBS preferují C před A a P

Principy OLAP technologie

1. Multidimensionální koncept data a manipulace s daty
2. Přístup k primárním datům - OLAP funguje jako mediator
3. Dynamická manipulace s "řídkými" maticemi
4. Zpracování nonormalizovaných dat
5. Intuitivní operace s daty a multidimensionálními strukturami
6. Uložení výsledků OLAP, jejich uchování mimo zdrojovou data
7. Analytické modely OLAP
8. Volný počet dimenzií a agregačních úrovní
9. Standardní databázové operace (výkonnost, dávkové zpracování, zpracování neurčených hodnot, multiúživatelský provoz, bezpečnost, ..)

Pozn.:

a) použití pro aplikace typu DSS (Decision Support System), BI (Business Intelligence, BW (Business Warehouse, DW (Data Warehouse).

- b) ROLAP - rozšíření relačních DBS o funkce OLAP, např. OLAP Services v MS SQL Serveru.
- c) využívány i postrelační databázové systémy - viz další přednáška
- d) velký důraz na katalogizaci, popis dat (ontologie možno znázornit konceptuálním modelem - objektovým či datovým)

Modely dat a databázové systémy

- DBS budovány nad určitým modelem dat
- **Hierarchické DBS** (1967) – IMS, System M
 - Nemá standard
- **Sítové DBS** (1969, 1971) – IDMS
 - Standardizace CODASYL, DBTG (Database task Group)
- **Relační** (1970, 2. pol. 70. let, 80. léta) a **objektově relační** (90. léta) DBS – ORACLE, Informix, Sybase, MS SQL Server, Progress, DB/2, MySQL, ...
 - Robustní teorie, prof. Codd
 - První implementace 1978
 - Standardizace SQL
- **Objektové DBS** (90. léta) – Orion, GemStone, Ontos, Object Store
 - de facto standard ODMG-93
 - komerčně nepříliš rozšířen
- **NoSQL DBS** (2005+) – BigTable (Google), HBase (Apache), SimpleDB (Amazon), Dynamo, Voldemort (LinkedIn)
 - ➔ Systémy vyvíjené zejména poskytovateli internetových služeb

Objektově orientované DBS

- Dva směry vývoje:
 - revoluční (vznik zcela nových systémů, většinou na základě OO programovacích nástrojů) - ObjectStore (C++), Ontos (C++), Orion (LISP), GemStone (Smalltalk),
 - evoluční (doplňování OO principů do relačních DBS) - např. Oracle (od verze 8i)
- Standardizace: ODMG, ISO SQL 99
- **Charakteristika OODBS:**
 - data v OODBS jsou chápána jako objekty odpovídající entitám (objektům) zachycovaného světa,
 - objekty zachycují data i chování (tj. funkčnost, která je v jiných typech DBS zajišťována aplikačními programy).
- Vlastnosti
 - komplexní objekty (krom relačních tabulek i další typy objektů, např. seznamy, pole). Komplexní objekty možno navzájem skládat, mohou být hodnotou atributu nebo mohou existovat jako samostatné objekty v DB)
 - identita objektů (DBS zajírá unikátní identifikaci objektu v rámci celé DB - OID),
 - logická datová nezávislost (zapouzdření, skrývání dat + zveřejnění rozhraní, tj. hodnoty atributů nejsou přístupné přímo, ale přes definované rozhraní),
 - třídy a typy (možnost definice typu, resp. třídy, jako popisu společné struktury množiny objektů se stejnými vlastnostmi),
 - dědičnost (odvozování nových tříd z existujících, nové třídy dědi všechny atributy a chování existující třídy, vícenásobná dědičnost),
 - polymorfismus (schopnost operací fungovat na objektech více než jednoho typu),
 - rozšiřitelnost (možnost definovat nové základní typy).

Relační DBS	Objektové DBS
+ silné teoretické zázemí	+ schopnost zachycovat komplexní objekty
+ vysoká míra standardizace	+ Zapouzdření dat s asociovanými věcnými pravidly a metodami
+ pokročilé metody výběru dat, včetně využití služeb vestavěných optimalizátorů	+ dědičnost a polymorfismus
+ definice pravidel pro zajištění konzistence a integrity dat	+ navigace mezi objekty
+ optimalizace pro práci na víceprocesových systémech	+ možnost vytváření nových datových typů
+ paralelní zpracování	- nejednotné teoretické zázemí
+ možnost vytváření transparentních distribuovaných systémů, replikativní mechanismy	- nepropracované technologické vlastnosti DBS (transakční zpracování, paralelní zpracování, distribuované vlastnosti, zajištění konzistence a integrity dat)
+ pokročilé transakční zpracování (včetně 2PC)	
- omezená množina podporovaných datových typů	
- obtížné zachycení komplexních objektů pomocí tabulek	
- obtížná navigace mezi objekty v SQL	
- omezené možnosti analytických operací s použitím SQL	

Důvody hledání nových db řešení

- nedostatečná flexibilita relačních databázových schémat,
- obtížná škálovatelnost stávajících db,
- náklady na robustní řešení
- rostoucí objem dat a převažující podíl (cca 80%) nestrukturovaných dat
- potřeba ukládat a zpracovat v reálném čase obrovské objemy dat (řády desítek exabyte)
 - NoSQL hnútí

Trendy DBS

- Web a internet, umožňující uložit a zpřístupňovat informace v jednom rozsáhlém „globálním informačním systému“
 - zpřístupnění dat novým způsobem (portály, dynamické webové stránky)
 - e-commerce (sdílení obchodních informací, udržování obchodních vztahů a řízení obchodních transakcí pomocí telekomunikační sítě)
 - B2B
 - B2C
 - Personalizace dat
- Stále nová aplikativní prostředí vyžadující integrovat data a programy
 - Rozvoj OR DBS
 - Zahrnující databáze v aplikativních programech (Progress)
 - Rozšíření databázových serverů o aplikativní služby
- Pokroky v HW
 - Velikost vnitřní paměti umožňuje umístit tabulky a objekty do vnitřní paměti a radikálně mění vyhodnocování dotazů a jejich optimalizaci
 - Možnost současné práce až desetitisíců uživatelů – nároky na škálovatelnost DBS a dostupnost dat
 - Zvyšování počtu transakcí – milióny transakcí za minutu
- Velikost databází
 - Dosavadní velikost databází v GB a TB
 - S rozvojem nových typů aplikací (multimedialní data, archivy, lidský genom, ..) zvětšení do řádu petabajtů, exabajtů a zetabajtů (10^{21})
- Heterogenost databází
 - Přístup k více informačním zdrojům, které se vyvíjely samostatně a jsou nyní dostupné prostřednictvím internetu pro uživatele

Kritéria výběru DBS

1. Základní funkční a technologická kritéria
2. Výkonnost
3. Zabezpečení a utajení dat
4. Rozšířenost
5. Podpora ze strany dodavatele
6. Cena
7. Další rozvoj

Rozšířenost robustních DBS: Oracle, Microsoft SQL Server, IBM DB2, Microsoft Access, Teradata

Rozšířenost Open Source DBS: MySQL, PostgreSQL, MongoDB, Redis, SQLite

Cloud computing

- je model poskytování ICT služeb, který umožnuje všudypřítomný, pohodlný přístup na poždání prostřednictvím sítě ke sdílené skupině konfigurovatelných výpočetních zdrojů (jako síť, serverů, úložišť a aplikací), které mohou být rychle poskytnuty a spouštěny s minimálním úsilím či interakcí poskytovatele služeb. musí být zajištěna lokální autonomie,
- Služba cloud computingu má pět základních charakteristik:
 - **samoobslužnost** – zákazník si může jednostranně zajistit možnost poskytnutí výpočetních zdrojů v případě potřeby automaticky bez nutnosti lidského zásahu ze strany poskytovatele,
 - **přístup v síti** – funkce jsou dostupné v síti a přístupné přes standardní mechanismy, které podporují různé heterogenní tenké či tlusté klientské platformy (internet, mobilní telefony, PDA atd.),
 - **sdružování zdrojů** – zdroje poskytovatele se sloučují, aby obsloužily více zákazníků, velikost, umístění a struktura jsou uživatelům utajeny,
 - **rychlosť pružnosti služby** – zdroje mohou být rychle a pružně poskytnuty, automaticky přidány v případě potřeby, a nepotřebné zdroje zase mohou být rychle uvolněny. Zákazníkovi se zdroje dostupné pro poskytnutí jeví obvykle jako neomezené a mohou být nakoupeny kdykoli v jakémkoli množství,
 - **měřená služba** – provoz a využití zdrojů jsou automaticky monitorovány, měřeny a optimalizovány.
- V rámci cloud computingu se obvykle rozlišují tři základní typy ICT služeb:
 - **IaaS** (Infrastructure as a Service) – poskytovatel dodává zákazníkům jako službu výkon technologické infrastruktury. Zákazník si sám ukládá své data do pronajatých datových uložišť, resp. si na pronajatém infrastruktuře sám provozuje svoje aplikace,
 - **PaaS** (Platform as a Service) – poskytovatel dodává jako službu kromě technologické infrastruktury i vývojové a integrační nástroje. Tzn., že zákazník si na pronajaté platformě sám vyvine a provozuje svoje aplikace,
 - **SaaS** (Software as a Service) – poskytovatel provozuje na své infrastruktuře aplikace, jejichž funkcionalitu nabízí širokému okruhu zákazníků.

NoSQL a NewSQL databázové systémy

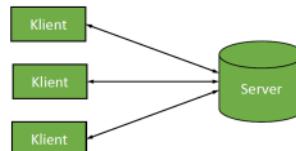
16 May 2024 10:09

- NoSQL databázové systémy
 - Vymezení NoSQL databázových systémů
 - Databázové systémy typu klíč-hodnota
 - Dokumentově orientované NoSQL databázové systémy
 - Sloupcově orientované NoSQL databázové systémy
 - Grafové databázové systémy
- NewSQL databázové systémy

Vymezení NoSQL databázových systémů

Klasická architektura klient-server

- Předpokládá se dostatečně výkonný server, aby byl schopen vyřizovat požadavky velkého množství uživatelů (klientů)
- Tradiční SŘDB:
 - ACID transakce
 - (viz 6. přednáška)
 - Podpora architektury klient-server
- Dobrá vertikální škálovatelnost
- Není jednoduché dosáhnout horizontální škálovatelnosti

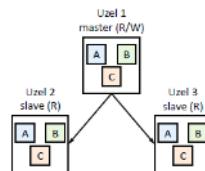


Motivace pro využívání jiných než relačních SŘBD

- Data v relační databázi jsou ukládána v jiné struktuře, než jsou struktury využívané programátory v rámci aplikací.
 - Normalizované relační schéma vs. objekty.
 - Objektové databázové systémy se ale příliš neujaly.
- Rozvoj webu vede k potřebě **rychleji reagovat** na požadavky zákazníků či uživatelů, **zpracovávat velké objemy dat, kombinovat data z různých zdrojů a zároveň zajistit rychlou odezvu a vysokou dostupnost služeb**.
- **Big Data – 3V** (Gartner, ©2023):
 - High volume – velký objem dat.
 - High velocity – rychle se měnící data.
 - High variety – velmi různorodá data.
 - Pozn.: Další autoři později přidali k definici Big Data několik dalších V.

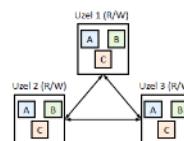
Vertikální a horizontální škálovatelnost

- Vertikální škálovatelnost – schopnost systému:
 - vypořádat se s měnící se zátěží přidáváním nebo ubíráním systémových prostředků jako jsou CPU nebo RAM k jedné instanci systému
 - jsme schopni měnit výkonnost serveru (přibírání/odebírání hardware prostředků – paměť...)
- Horizontální škálovatelnost – schopnost systému:
 - vypořádat se s měnící se zátěží přidáváním nebo ubíráním instancí systému.



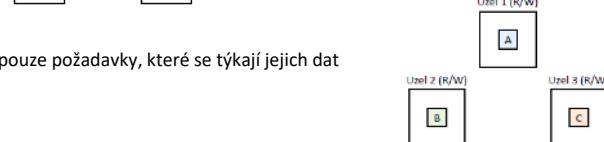
Master-slave replikace dat

- Replika všech dat je udržována na všech uzlech clusteru
 - Cluster = skupina spolupracujících počítačů, "dohromady tvoří jeden pevný disk"
 - Libovolný uzel musí být schopen vyřídit požadavku
- Požadavky na zápis dat zpracovává master, který tyto požadavky propaguje na uzly typu slave.
- Uzly typu slave mohou zpracovávat pouze požadavky na čtení dat.



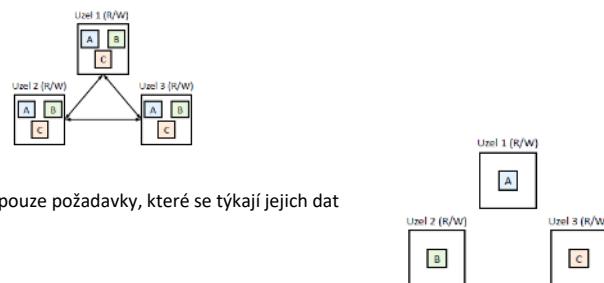
Peer-to-peer replikace dat

- Replika všech dat je udržována na všech uzlech clusteru.
- Každý uzel může zpracovávat požadavky na zápis i čtení dat.
- Každý uzel propaguje jím zpracované požadavky na ostatní uzly.



Rozdelení dat (sharding)

- Data jsou rozdělena mezi uzly clusteru.
- Každý uzel obsahuje jen určitou část dat – clustery jsou schopné vyřídit pouze požadavky, které se týkají jejich dat
- Každý uzel zpracovává požadavky na čtení i zápis dat



Kombinace replikace a rozdelení dat

- Master slave replikace + rozdelení
 - Pro každou část dat existuje jeden uzel typu master
 - Uzel může být uzlem typu master pro určitou část dat a pro ostatní části dat může být uzlem typu slave
- Peer to peer replikace + rozdelení
 - Každá část dat je replikována na určený počet uzlů.
 - Každý uzel nemusí obsahovat všechny části dat.

NoSQL databázové systémy

- NoSQL lze přeložit jako „ne pouze SQL“ nebo „zádné SQL vůbec“
- Nová generace databázových systémů, které se vyznačují některými z následujících vlastností: jsou ne relační, distribuované, open source a horizontálně škálovatelné.
- Další vlastnosti NoSQL databázových systémů: bez datového schématu, snadná replikace, jednoduché API, občas konzistentní (BASE namísto ACID), navržené pro zpracování velkých objemů dat

ACID transakce

- Všechny příkazy musí proběhnout (pokud selže příkaz, musí selhat celá transakce)
- Vzájemné transakce jsou nezávislé
- Po provedení transakce musí proběhnout commit
- **Atomicity:** transakce se tváří jako jeden celek, musí proběhnout celá nebo vůbec ne.

- **Consistency:** transakce transformuje databázi z jednoho konzistentního stavu do jiného konzistentního stavu.
- **Independence:** transakce jsou nezávislé, změny prováděné jednou transakcí nejsou viditelné ostatním transakcím.
- **Durability:** efekty potvrzené transakce jsou uloženy do databáze.

CAP teorém

- **Distribuovaný systém, v rámci kterého jsou sdílena data, se může vyznačovat pouze dvěma z následujících vlastností**
 - **Consistency** (konzistence): ekvivalent situace, kdy existuje jedna aktuální kopie dat.
 - **Availability** (dostupnost): všechny požadavky na čtení a zápis dat jsou systémem zpracovány.
 - **Partition tolerance** (odolnost vůči výpadku sítě): i když dojde k rozdělení systému na části, systém bude stále schopen zpracovávat požadavky.
- Pozor, konzistence v CAP je chápána jinak než v ACID.

BASE systém

- **Basically Available** (převážná dostupnost): systém je převážně dostupný po celou dobu; výpadek systému může být pouze částečný
- **Soft state** (volný stav): v systému neustále dochází ke změnám.
 - Jednotlivé uzly nemusí být vzájemně konzistentní v každém okamžiku.
 - Je definován způsob, jak se uzel dostane do konzistentního stavu.
- **Eventual consistency** (občasné konzistence): systém se občas dostane do konzistentního stavu, ale není zaručeno, že se v něm bude nacházet

Relační vs. NoSQL databáze

Relační databáze	NoSQL databáze
Integrita dat je zásadní.	Stačí, když je většina dat většinu času v pořadku.
Datový formát je konzistentní a dobře definován.	Datový formát nemusí být známý nebo konzistentní.
Předpokládáme dlouhodobé uložení dat.	Vzhledem k velkému množství dat často ukládáme pouze určité „časové okno“ (např. poslední měsíc, poslední rok).
Aktualizace dat jsou časté.	„Write-once/read-many“, tedy vložená data už typicky nejsou dále modifikována (nebo alespoň ne příliš často). Ovyklo data neustále přibývají, aniž by byla modifikována. Již nepotřebné záznamy jsou pak smazány.
Předvídatelný (lineární) nárůst velikosti dat.	Nepředvídatelný (exponenciální) nárůst velikosti dat.
Nástroje pro dotazování dat umožňují přístup i ne-programátorům.	Typicky pouze programátoři piší (implementují) zpracování dat.
Prohibuje pravidelné zálohy dat.	Pro řešení výpadků je využívána replikace dat.
Přístup k datům zajišťuje jediný server.	Data jsou umístěna na více serverech, přístupujeme tedy ke clusteru uzlů.

Typy NoSQL databázových systémů

- Databázové systémy typu **klíč-hodnota**
- **Dokumentově orientované** NoSQL databázové systémy
- **Sloupcově orientované** NoSQL databázové systémy
- **Grafové** databázové systémy

Databázové systémy typu klíč-hodnota

- K unikátnímu klíči je uložena hodnota
- Hodnotou mohou být obecně data libovolného typu
- Některé databázové systémy umožňují seskupit související dvojice klíč-hodnota do tzv. bucketů
- **Pro získání uložených dat je třeba znát příslušný klíč**
- Nemusí být možné vyhledávat záznamy prohledáváním obsahu uložených hodnot
 - Některé systémy typu klíč hodnota ale umožňují definovat sekundární indexy, které pak umožní podle indexovaných hodnot vyhledávat.
- Využití:
 - **Výhody**
 - Rychlost zpracování dotazů
 - Data lze dobré distribuovat
 - **Příklady využití:**
 - Ukládání dat relací (sessions) ve webových
 - Ukládání dat nákupního košíku v e shopu
 - Ukládání profilů a preferencí uživatelů.
 - Ukládání dat chatů /zpráv
- Příklady: Redis, Amazon DynamoDB, Microsoft Azure Cosmos DB, Hazelcast

Ukázka dvojic klíč-hodnota



Využití Riak pro League of Legends

- Riot Games využívá Riak – databázový systém typu klíč-hodnota – pro ukládání chatu.
 - Seznam přátel a blokovaných uživatelů.
 - Historie chatu.
- Zprvu byly využívány instance MySQL s master-slave replikací dat.
- Rostoucí počet hráčů způsoboval zahlcení master serveru zprávami.
- Přechod na Riak
 - umožnil zvládat požadavky dané rostoucím počtem hráčů;
 - pomohl zkrátit čas přihlášení hráče; pomohl odstranit problém neúspěšného přidávání přátel, které bylo způsobeno
 - vypršením časového limitu požadavku;
 - umožnil vývoj nových vlastností jako import přátel z Facebooku.

Ukázka uložení JSON dokumentu

```
Klíč = os_cis
{
  "os_cis": "1",
  "jmeno": "Pavel",
  "prijmeni": "Novák",
  "telefon": "+420 555 555 555",
  "adresa": {
    "ulice": "Stará cesta",
    "c_popisne": "15",
    "obec": "Nová Ves",
    "psc": "11111",
    "stat": "Česká republika"
  }
}
```

Dokumentově orientované NoSQL databázové systémy

- Uložená a zpracovávaná data mají podobu dokumentu
 - Dokument je hodnotou uloženou k určitému unikátnímu klíči.
- **Dokument je samopopisná datová struktura**
 - Typický data ve formátu JSON, BSON nebo XML
- **Lze provádět dotazy nad obsahem dokumentů**
 - Lze definovat sekundární indexy.
 - Neexistuje standardizovaný dotazovací jazyk.
 - Dokumentově orientované SŘBD zpravidla definují vlastní jazyk pro manipulaci s daty.

- Využití:
 - Výhody
 - Dokumenty mohou přímo odpovídat třídám objektů v aplikaci - v porovnání s relačními databázemi není třeba provádět objektově relační mapování.
 - Datový model může být velmi flexibilní - lze ukládat různé dokumenty s různou strukturou.
 - Příklady využití:
 - Content management systémy, blogovací platformy.
 - Analytické úlohy.
 - E shopy a obdobná e commerce řešení.
- Příklady: MongoDB, Amazon DynamoDB, Microsoft Azure Cosmos DB, Couchbase, CouchDB

Využití MongoDB městem Chicago

- Chicago využívá MongoDB pro systém WindyGrid, který slouží k vizualizaci a analýze dat z 15 zdrojových systémů (např. policie, doprava, hasiči).
- Integrace systémů umožněna díky flexibilnímu datovému modelu.
- Systém umožňuje:
 - v reálném čase analyzovat, co se ve městě děje;
 - provádět analýzu závislosti událostí (např.: do sedmi dnů od stížnosti na nedovezený odpad se zpravidla objeví stížnost na výskyt hlodavců);
 - ve vývoji je podpora predikce problémů ve městě s cílem problémům předcházet namísto toho, aby byly řešeny reaktivně.

Sloupcově orientované NoSQL databázové systémy

- Uložená a zpracovávaná data mají podobu řádků
 - Každý řádek má unikátní klíč řádku.
 - Řádek tvořen sloupcí, kterých může být i velké množství.
 - Každý sloupec má název sloupce a hodnotu
 - Jednotlivé řádky nemusí mít stejný počet sloupců
 - Sloupce mohou být sdruženy do rodin sloupců/supersloupců
- Lze provádět dotazy nad řádky a sloupcí
- Využití:
 - Hlavní výhody:
 - Datový model může být velmi flexibilní.
 - Data lze dobře distribuovat
 - Příklady využití:
 - Systémy vyžadující vysokou dostupnost a zároveň škálovatelnost.
 - Content management systémy, blogovací platformy.
 - Systémy s velkými objemy zapisovaných dat, např. zapisování logů.
- Příklady: Cassandra, HBase, Microsoft Azure Cosmos DB, DataStax Enterprise, ScyllaDB
- Anglicky column family stores nebo wide column stores
- Existují i sloupcově orientované relační databázové systémy (column oriented RDBMS)
 - Data ukládána po sloupcích.
 - Lepší výkonnost analytických dotazů než v situaci, kdy jsou data uložena po řádcích.
 - Lepší možnosti komprese dat.
 - Nejedná se o NoSQL
 - Příklady: Infobright DB, Vertica
 - Podporu sloupcového ukládání dat lze nalézt i u systémů, které nejsou primárně sloupcově orientované

Ukázka dat ve sloupcově orientovaném NoSQL databázovém systému

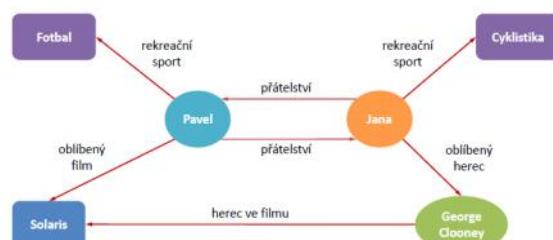
Klíč řádku 1234	Jméno 'Jana'	Příjmení 'Novotná'	
Klíč řádku 4444	Jméno 'Pavel'	Příjmení 'Novák'	
Klíč řádku 5678	E-mail pavel@email.test	Tel. 111 222 333	
Kontakty			
Název 'Kamna a krby'	E-mail kamna@krby.test	Tel. 1 555 222 333	Tel. 2 555 222 444

Využití DataStax Enterprise pro podporu rozhodování v témař reálném čase

- Banka Capital One využila DataStax Enterprise pro svoji platformu umožňující analýzu dat z různých zdrojů v témař reálném čase.
- Před vytvořením nové platformy pro analýzu dat banka využívala vícero datových skladů a databází.
- Ze 450 milionů řádků v původní databázi bylo při budování platformy pro analýzu vytvořeno pouze 11,5 milionů řádků.
- Platforma pro analýzu dat zpracovává 21 000 transakcí za sekundu.
- Dostupnost platformy je 99,99 %.

Grafové databázové systémy

- Uložená a zpracovávaná data mají podobu grafu, tj. uzlů vzájemně propojených hranami
- Typy grafů:
 - Orientované (jednostranné vztahy) a neorientované (oboustranné vztahy)
 - Jednovztahouvé (všechny hrany mají stejný typ) a vícevztahouvé (hrany mohou mít různé typy)
 - Hypergraf - graf, ve kterém může hrana spojovat více než dva vrcholy.
- Grafová data je obtížně distribuovat
- Využití:
 - Výhody: lze vyjádřit vztahy
 - Příklady použití:
 - Sociální síť
 - Doporučování např. souvisejícího zboží v e-shopu
 - Úlohy související s umístěním v prostoru např. doručování nebo plánování cest
 - Detekce podvodů
- Příklady: Neo4j, Microsoft Azure Cosmos DB, ArangoDB, Aerospike, Virtuoso



Využití Neo4j pro detekci podvodů

- Společnost TODO1 Services Inc. poskytuje řešení pro detekci podvodů bankám v Latinské Americe.
- TODO1 využívá ve svém řešení Neo4j.
- Nahrazením dřívějšího systému systémem využívajícím Neo4j byl počet odhalených podvodů zdvojnásoben při stejném procentu transakcí nesprávně označených za podvod.
- Systém iuviPROFILER
 - pracuje s grafem, který je tvořen 250 milionům uzel a 2,2 miliardy vztahů;
 - je schopen zpracovat 500 transakcí za vteřinu;
 - ročně ochrání více než 3,2 miliardy transakcí a zabrání škodám převyšujícím částku 40 milionů dolarů.

NewSQL databázové systémy

- NewSQL představuje moderní kategorii relačních SŘBD, které se snaží poskytnout stejnou škálovatelnost a výkonnost jako NoSQL databázové systémy

- pro OLTP úlohy čtení a zápisu dat při současném zajištění ACID vlastností transakcí .."
- Vznik NewSQL databázových systémů reaguje na to, že NoSQL databázové systémy nedisponují vlastnostmi relačních databázových systémů, zejména:
 - absence vlastností, jako jsou například cizí klíče,
 - absence ACID transakcí
 - absence standardního jazyka pro interakci s databází.
- Někdy se lze také setkat s označením Distributed SQL Database.

Proč jsou ACID transakce důležité

- Transakce slouží:
 - k zajištění konzistence databáze
 - k řízení víceuživatelského přístupu datům.
- ACID Rain útoky
 - Zneužití zranitelnosti vyplývající z nesprávné či nevhodné implementace víceuživatelského přístupu k datům skrze aplikační programové rozhraní (API).
- Možné důsledky špatně implementovaného víceuživatelského přístupu k datům v případě eshopu
 - Přečerpání skladových zásob.
 - Vícenásobné uplatnění poukazu, např. na slevu.
 - Neuhrazení zboží v důsledku přidání zboží do košíku současně s platbou.
- Díky chybně implementovanému víceuživatelskému přístupu skončila bitcoinová burza Flexcoin
 - Chyba v implementaci víceuživatelského přístupu umožnila přečerpat účet.
 - Odcizeno 896 bitcoinů v hodnotě 365 000 GBP.

Typy NewSQL databázových systémů

- Systémy představující nové databázové architektury (new database architectures)
- Middleware pro transparentní rozdělení dat (transparent sharding middleware)
- Databáze jako služba (database-as-a-service)

Nové databázové architektury

- NewSQL databázové systémy, které byly od začátku navrženy a vytvořeny jako nový SŘBD
 - Nejsou omezovány architekturou nějakého dřívějšího řešení.
- Od základu koncipovány jako distribuované databázové systémy s podporou řízení transakcí v distribuovaném prostředí
 - Protokol pro komunikaci mezi uzly.
 - Optimalizátor dotazů zohledňuje distribuci dat mezi uzly.
- SŘBD se zpravidla sám stará o distribuci dat mezi uzly. Tj. nespoléhá se na jinou komponentu, která by toto zajišťovala, např. distribuovaný souborový systém.
 - Dotaz putuje za daty spíše než, aby data putovala k dotazu.
- Příklad: CockroachDb, H-Store, HyPer, MemSQL, NuoDB, SAP HANA, VoltDB

Middleware pro transparentní rozdělení dat

- Middleware pro transparentní rozdělení dat je softwarová komponenta umožňující vytváření clusterů tvořených instancemi standardních relačních databází
- Cluster se chová jakoby to byla jedna (logická) databáze
 - Aby aplikace mohla využívat data z clusteru, nemusí ji být nutné modifikovat.
- Middleware typicky zajišťuje směrování dotazů, koordinaci transakcí, replikaci a rozdělení dat
- Příklady: MariaDB MaxScale, MySQL Cluster, ScaleArc

Databáze jako služba

- Databáze jako služba (DBaaS)
 - Poskytovatel služby poskytuje služby SŘBD včetně související infrastruktury, konfigurace a správy zákazníkovi
 - Zákazník zpravidla platí podle využitého objemu služby, nebo na základě předplatného na určité časové období.
- Pavlo a Aslett (2016) za NewSQL DBaaS považují jen takové služby, kde SŘBD představuje novou databázovou architekturu.
 - Existuje řada poskytovatelů, kteří nabízejí tradiční relační databáze formou služby. Např. v rámci Google Cloud SQL lze využívat MySQL a PostgreSQL
- Příklady: Amazon Aurora, ClearDB

Dovětek: Vývoj ohledně konzistence v NoSQL databázových systémech

- NoSQL databázové systémy se snaží doplňovat vlastnosti pro zajištění konzistence dat.
- Někteří výrobci NoSQL databázových systémů tvrdí, že transakce v jejich systémech mají ACID vlastnosti.
 - Např. MongoDB (od verze 4.0) nebo RavenDB.
 - Pozor ale ne různá omezení.
- Distributed Systems Safety Research
 - Iniciativa zaměřená na testování vlastností distribuovaných systémů.
 - <https://jepsen.io/>

Propojená data na webu (a jinde), Formát RDF a jazyk SPARQL

16 May 2024 13:17

- Otevřená data a data na webu
- Propojená data a jejich principy
- Resource Description Framework
- Ontologie a slovníky
- Možnosti přístupu k datům v RDF
- Jazyk SPARQL
- Data ČSSZ v RDF

Otevřená data

- Otevřená data jsou „data, která mohou být svobodně využívána, zpracovávána, upravována a šířena, přičemž může být vyžadováno, aby byl uveden původce dat nebo aby byla dále šířena za stejných podmínek“
- Klíčové vlastnosti otevřených dat:
 - Úplnost
 - snadná dostupnost
 - strojová čitelnost
 - použití standardů s volně dostupnou specifikací (tzv. otevřených standardů)
 - zpřístupnění za jasné definovaných podmínek užití dat (licence) s minimem omezení
 - dostupnost uživatelům při vynaložení minima možných nákladů na jejich získání
- Požadavky na podmínky užití otevřených dat
 - Neomezuji jejich uživatele ve způsobu použití dat.
 - Opravňují uživatele k jejich dalšímu šíření.
 - Musí být uveden autor dat (i při dalším šíření).
 - Při dalším šíření musí i ostatní uživateli mít stejná oprávnění s daty nakládat - během šíření dat nesmí dojít např. k omezení jejich využití pouze pro nekomerční účely.

Stupně otevřenosti dat



Data on the Web Best Practices

- W3C Recommendation od 31. 1. 2017.
- Sada 35 doporučení (Best Practices) pro publikaci dat na webu
- Aplikace doporučení by měla přinést následující přínosy.
 - Lepší pochopení dat lidmi
 - Snazší automatizované zpracování dat
 - Usnadnění automatizovaného vyhledávání dat
 - Usnadnění opětovného použití dat (re use).
 - Zvýšení důvěryhodnosti dat
 - Snazší propojování datasetů a jednotlivých dat (záznamů, položek)
 - Zlepšení přístupnosti dat pro lidi i pro automatizované zpracování
 - Zlepšení interoperability
- V řadě případů je doporučeno využití principů propojených dat
 - Doporučení jsou ale širší a netýkají se pouze propojených dat.
- Kdo využívá propojená data nebo sémantické technologie: EA Games, BBC, Wolters Kluwer, Volkswagen, Britská vláda, Skotská vláda, Česká správa sociálního zabezpečení, generální finanční ředitelství

Propojená data a jejich principy

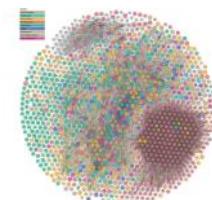
World Wide Web

- World Wide Web změnil způsob, kterým sdílíme dokumenty
 - Hypertextové odkazy propojují jednotlivé dokumenty.
 - Díky odkazům vzniká globální vzájemně propojená síť.
- World Wide Web je postaven na několika málo standardech:
 - Uniform Resource Identifiers (URIs) / International Resource Identifiers (IRIs), Uniform Resource Locators (URLs)
 - Protokol http
 - Jazyk HTML
- Výhody:
 - Využívá relativně jednoduché technologie
 - Je otevřený
 - Použité technologie umožňují škálovatelnost a rozšiřitelnost
 - Využívání standardů zajišťuje interoperabilitu
- Nevýhody:
 - Jazyk HTML byl primárně navržen pro vyjádření obsahu dokumentů na webu. Pro strojové zpracování jsou ale vhodnější strukturovaná data

Web dokumentů



Web dat

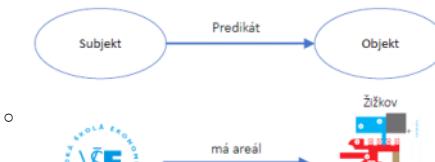


Principy propojených dat (Linked data)

1. Používejte IRI jako identifikátory objektů
2. Používejte HTTP IRI, aby bylo možné objekty vyhledat v prostředí webu a odkazovat na ně.
3. Při přistoupení na IRI objektu poskytněte o daném objektu data pomocí standardů jako RDF či SPARQL
4. Propojte související objekty pomocí IRI

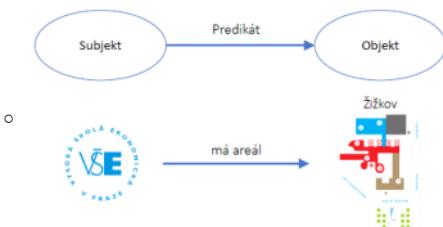
Resource Description Framework (RDF)

- Obecný model pro popis jakýchkoli zdrojů (objektů) a vztahů mezi nimi
- Standard konsorcia W3C
- Data reprezentovaná pomocí RDF mají podobu sítě
 - Uzly sítě představují zdroje (resources)
 - Vazby v síti představují vztahy mezi zdroji
- Data reprezentovaná pomocí RDF jsou tvořena trojicemi



Resource Description Framework (RDF) je framework určený k popisu informací na webu. Používá strukturu zvanou trojice (triple), která se skládá z podmíny "subjekt - predikát - objekt". Subjekt reprezentuje entitu, o které hovoří, predikát reprezentuje vlastnost nebo charakteristiku této entity a objekt je hodnota nebo hodnoty této vlastnosti. RDF se často používá pro popis metadat, což jsou data o datech, která pomáhají kategorizovat, strukturovat a lépe porozumět obsahu na webu. RDF je také součástí základních technologií pro vytváření a popis tzv. "semantického webu" (Semantic Web), což je koncept webu, který umožňuje počítačům a lidem spolupracovat a porozumět informacím lépe.

Z <<https://chatgpt.com/c/1f24e288-3480-4b45-a4a9-689239f287cb>>

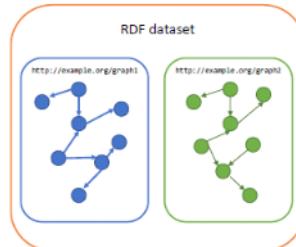


Objekt je rozšířená nebo rozšířený tedy vlastností. RDF se často používá pro popis metadat, což jsou data o datech, která pomáhají kategorizovat, strukturovat a lépe porozumět obsahu na webu. RDF je také součástí základních technologií pro vytváření a popis tzv. "semaforů na webu" (Semantic Web), což je koncept webu, který umožňuje počítačům a lidem spolupracovat a porozumět informacím lépe.

Z <<https://chatgpt.com/c/1f24e288-3480-4b45-a4a9-689239f287cb>>

RDF dataset a RDF grafy

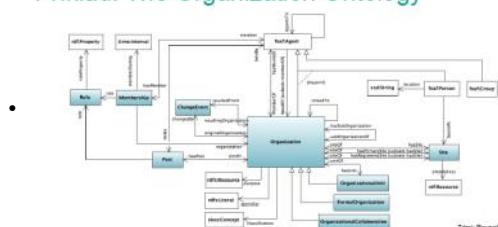
- Tvrzení v RDF (trojice) lze sestavit do RDF grafů
- RDF dataset je kolekce RDF grafů
 - Vždy obsahuje jeden výchozí (default) graf, který nemusí být pojmenován.
 - Může obsahovat 0-N pojmenovaných grafů
 - Jméno grafu musí být v rámci RDF datasetu unikátní.
- Serializace RDF: N-Triples, Turtle, TriG, N-Quads, JSON-LD, RDFa, RDF/XML



Ontologie a slovníky

- RDF představuje generický model nezávislý na jakékoli doméně.
- Pro reprezentaci dat z určité domény jsou využívány ontologie či slovníky.
- Ontologie/slovník definuje třídy a predikáty pro reprezentaci dat z dané domény.
- Ontologie/slovník je opět reprezentována pomocí RDF.

Příklad: The Organization Ontology



Konceptuální schéma a ontologie

- Ontologii lze přirovnat ke konceptuálnímu schématu, které je sdíleno širší komunitou lidí (uživatelů / datových expertů / softwarových inženýrů atd.).
- Je snahou o nalezení konsenzu na konceptuálním způsobu popisu určitého výseku reality
- Technicky je vyjádřena pomocí prostředků, které umožňují sdílení takového popisu a jeho přímé využití pro vyjádření významu dat

Konceptuální schéma	Ontologie / slovník
Entity množina	Třída
Atribut entity množiny	Vlastnost třídy s primitivní hodnotou
Vazba mezi entity množinami	Vlastnost třídy s jinou třídou jako hodnotou

Využití ontologií a slovníků

- Ontologie/slovník definuje třídy a vlastnosti pro reprezentaci dat o objektech či jevech z určitého výseku světa (domény)
- Na ontologii/slovník lze také nahlížet jako na veřejný a sdílený katalog tříd a vlastností pro reprezentaci dat z určité domény
- Pro maximalizaci interoperability dat je doporučeno v maximální míře využívat existujících ontologií/slovníků
 - Pokud pro danou doménu není ontologie/slovník k dispozici, je třeba navrhnut ontologii/slovník vlastní.
 - Lze kombinovat více ontologií/slovníků pro reprezentaci dat.
- Pokud různé ontologie/slovníky zavádějí koncepty pro ekvivalentní třídy a vlastnosti, lze je propojit pomocí vlastnosti owl:equivalentProperty či owl:equivalentClass z ontologie OWL

Sémantický slovník pojmu veřejné správy

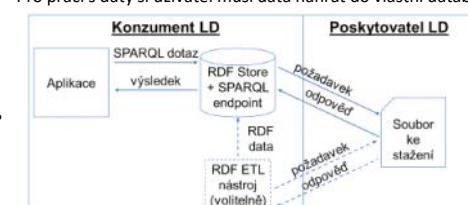
- Prostředek pro harmonizaci významu dat v informačních systémech veřejné správy
- Vznikl v gesci Ministerstva vnitra ČR, dále rozvíjen v gesci Digitální a informační agentury.

Možnosti přístupu k datům v RDF

- Soubor ke stažení (dump)
- Dereferencovatelná HTTP IRI
- SPARQL endpoint
- Triple Pattern Fragments

Soubor ke stažení

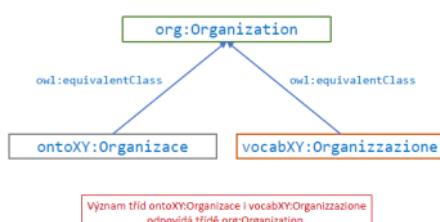
- Zveřejněný soubor ke stažení.
- Umožňuje relativně snadné zveřejnění úplných datasetů.
- Pro práci s daty si uživatel musí data nahrát do vlastní databáze.



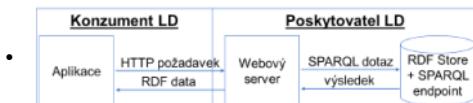
Dereferencovatelná HTTP IRI

- Poskytovatel dat musí zajistit, že po přistoupení na IRI objektu budou o daném objektu vrácena data v RDF.

Příklad: Ekvivalentní třídy

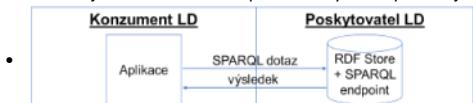


- Umožňuje procházet sítí propojených dat.



SPARQL endpoint

- Umožňuje dotazování a manipulaci s daty v RDF pomocí jazyka SPARQL.



Triple Pattern Fragments

- Linked Data Fragments**
 - Koncept zastřešující všechna možná rozhraní pro přístup k datům v RDF.
- Triple Pattern Fragments**
 - Jeden z typů rozhraní v rámci Linked Data Fragments.
 - Umožňuje rozložit náročnost poskytování a zpracování dat mezi poskytovatele a konzumenta dat.
 - Díky snížení náročnosti na výpočetní zdroje na straně poskytovatele dat v porovnání se SPARQL endpointem usnadňuje dosažení vysoké dostupnosti dat.



RDF úložiště

- Systémy řízení báze dat pro ukládání dat v RDF jsou nazývány RDF store (někdy také quadstore, dříve triplestore).
- Pro výběr úložiště platí obdobná kritéria jako u relačních databázových systémů.
- Navíc je důležité sledovat podporu specifikace SPARQL.
 - Aktuálně by RDF úložiště mělo implementovat SPARQL 1.1.
- Příklady RDF úložišť:
 - Amazon Neptune,
 - Apache Jena,
 - GraphDB,
 - MarkLogic Server,
 - Openlink Virtuouso,
 - Oracle,
 - RDF4J (dříve Sesame),
 - Stardog.
- Benchmarking RDF úložišť viz <http://www.w3.org/wiki/RdfStoreBenchmarking>.

RDFa – strukturovaná data v rámci webových stránek

- RDF lze také využít k poskytování strukturovaných dat v rámci webových stránek pomocí Resource Description Framework in Attributes (RDFa).

Jazyk SPARQL

- SPARQL zahrnuje sadu specifikací, které definují jazyk pro dotazování a manipulaci s daty v RDF a také související protokol.

Dotazovací jazyk SPARQL

- Dotazování pomocí jazyka SPARQL je založeno na formulaci vzorů RDF trojic (triple patterns).
- Při vyhodnocování dotazu se hledají trojice, které odpovídají definovaným vzorům.
- Typ výsledku pak záleží na zvoleném typu dotazu (viz dále).
- Obdobně jako v SQL lze používat i definované funkce.

Typy dotazů ve SPARQL 1.1 Query Language

- Při dotazování lze využít tyto typy dotazů:
 - SELECT** – vrací všechny nebo zvolené atributy odpovídající dotazu.
 - CONSTRUCT** – vrací výsledek jako RDF graf.
 - ASK** – vrací jako výsledek boolean (TRUE/FALSE).
 - DESCRIBE** – vrací RDF graf s popisem zdroje s daným IRI.

SPARQL 1.1 Update

- SPARQL Update je specifikace jazyka pro aktualizaci RDF grafů v RDF úložišti.
- Jsou definovány dva typy operací:
 - Graph Update – přidávání a odebírání trojic v rámci RDF úložiště.
 - Operace v rámci Graph Update:
 - INSERT DATA – vložení trojic do grafu.
 - DELETE DATA – smazání trojic z grafu.
 - DELETE/INSERT – mazání a vkládání trojic na základě definovaných vzorů.
 - LOAD – načtení zdrojového dokumentu obsahujícího grafu jeho nahrání do RDF úložiště.
 - CLEAR – smazání všech trojic z jednoho či více grafů.
 - Operace v rámci Graph Management
 - CREATE – vytvoření nového grafu.
 - DROP – smazání grafu a všech jeho trojic.
 - COPY – aktualizace grafu, aby obsahoval stejný obsah jako jiný graf.
 - MOVE – přesun všech dat z jednoho grafu do druhého.
 - ADD – přidání dat z jednoho grafu do druhého grafu.

Příklad dotazu v jazyce SPARQL

```

PREFIX dcat: <http://www.w3.org/ns/dcat#>
PREFIX qb: <http://purl.org/linked-data/cube#> ← Definice prefixů
PREFIX dcterms: <http://purl.org/dc/terms/>

SELECT DISTINCT ?nazev AS ?datovaSada COUNT(?o) AS ? pocetPozorovani
FROM <http://linked.opendata.cz/resource/dataset/cssz/pensions>
FROM <http://linked.opendata.cz/resource/dataset/cssz/pensions/def>
WHERE {
  ?o a qb:Observation ;
    qb:dataSet ?ds .
  ?ds dcterms:title ?nazev .
  FILTER (LANG(?nazev)="cs")
}
ORDER BY DESC(?pocetPozorovani)
LIMIT 10 ← Omezení počtu výsledků a jejich seřazení
  
```

Data ČSSZ v RDF

Publikace dat důchodové statistiky ČSSZ v průběhu času



- Motivace k řešení výzkumného projektu TD020121
 - Orgány veřejné správy v ČR poskytují řadu statistických dat, ale v různých formátech (PDF, XLS, CSV, XML).
 - Struktura dat a jejich význam nejsou vždy popsány.
 - Existují ale i výjimky, např. data ve Veřejné databázi ČSÚ jsou bohatě popsána metadaty.
 - Heterogenita formátů a chybějící popis struktury a významu dat komplikují jejich zpracování.
- Hlavní cíle projektu TD020121
 - Vytvořit metodiku a typovou architekturu pro publikaci statistických dat v podobě otevřených propojených dat.
 - Ověřit navrženou metodiku a typovou architekturu na publikaci dat důchodové statistiky ČSSZ.
 - Vybudovat nad těmito daty pilotní aplikaci pro zpřístupnění a prezentaci těchto dat.
 - Využít tato data pro v rámci specializované mapy s interpretací regionálních rozdílů.
- Data poskytovaná ČSSZ v RDF
 - Česká správa sociálního zabezpečení (ČSSZ) poskytuje otevřená data od listopadu roku 2015.
 - Otevřená data ČSSZ jsou dostupná data na portálu <https://data.cssz.cz>.
 - Data jsou poskytována ve formátu CSV a v RDF.
 - Data jsou poskytována ke stažení i prostřednictvím SPARQL endpointu.
 - Všechna data jsou opatřena metadaty a jsou registrována v Národním katalogu otevřených dat.

Oázky ZT

16 May 2024 10:13

1. Co řeší následující dotaz jazyka SQL:

```
Select cislo_oddeleni, avg(plat) from zamestnanci group by cislo_oddeleni having avg(plat) > (select avg(plat) from zamestnanci)
```

 - Vypisuje čísla oddělení a průměrné platy oddělení, ve kterých je plat větší než průměrný plat všech zaměstnanců
 - Vypisuje čísla oddělení a průměrné platy oddělení, ve kterých je průměrný plat větší než průměrný plat všech zaměstnanců
 - Vypisuje čísla oddělení a průměrné platy oddělení, ve kterých je plat větší než průměrný plat zaměstnanců jejich oddělení
 - Vypisuje čísla oddělení a průměrné platy oddělení, ve kterých je plat větší než jakýkoliv plat v jiných odděleních
2. SQL příkaz „delete from osoba“ má provést:
 - Zrušení tabulky osoba
 - Zrušení definice tabulky osoba ze systémového katalogu
 - Smazání všech dat (řádků) z tabulky osoba
 - Není dokončen a proto nebude proveden
3. Při měření výkonnosti databází se používají testy TCP, které z následujících metrik **se nepoužívají** k měření výkonnosti OLAP systémů.
 - Transakce za minutu ANO - NE
 - Cena systému / počet dotazů za hodinu ANO - NE
 - Cena systému / počtem transakcí za minutu ANO - NE
 - Počet dotazů za hodinu ANO - NE
4. Při měření výkonnosti databázových systémů se používají testy TCP. Které z následujících metrik **se nepoužívají** pro měření výkonnosti OLTP systémů:
 - Transakce za minutu ANO - NE
 - Cena systému / počet dotazů za hodinu ANO - NE
 - Cena systému / počtem transakcí za minutu ANO - NE
 - Počet dotazů za hodinu ANO - NE
5. Rozdíl je možné v jazyce SQL vyjádřit pomocí klausulí
 - Is not null ANO - NE
 - Not exists ANO - NE
 - Not in a vnořeného dotazu ANO - NE
 - minus ANO - NE
6. Transakce v databázovém zpracování
 - slouží pro vyjádření vazeb mezi daty zpracovanými v databázovém prostředí ANO - NE
 - slouží k zajištění konzistence databáze a řízení uživatelského přístupu k datům ANO - NE
 - vyjadřují vztah mezi primárním a cizím klíčem v relačních tabulkách ANO - NE
 - je logická posloupnost operací, která je promítána do databáze jako celek ANO - NE
7. Co znamená OLTP v databázovém zpracování
 - Není to zkrotka
 - On-line trade processing
 - On-line time processing
 - On-line transaction processing
8. Cizí klíč se v relačních databázových systémech používá pro
 - Vyjádření vztahu mezi řádky tabulky ANO - NE
 - Zachycení vlastnosti dat ANO - NE
 - Zachycení počtu výskytů relační množiny ANO - NE
 - Jednoznačnou identifikaci řádků relační tabulky ANO - NE
9. Primární klíč v databázové tabulce
 - Slouží k tomu, aby se automaticky číšovaly záznamy ANO - NE
 - Má stejné vlastnosti jako sekundární index ANO - NE
 - Může obsahovat duplicitní hodnoty ANO - NE
 - Slouží jako jednoznačný identifikátor řádků této tabulky ANO - NE
10. Jaké jsou základní množinové operace relační algebry:
 - Spojení ANO - NE
 - Rozdíl ANO - NE
 - Průnik ANO - NE
 - Projekce ANO - NE
 - Restriktce ANO - NE
 - Sjednocení ANO - NE
11. Databázový systém ORACLE 10g nebo 11g je databázovým systémem postaveným na
 - Relačně-objektovém modelu
 - Objektovém modelu
 - Hierarchickém datovém modelu
 - Sítovém datovém modelu
12. Jako správnou odpověď či odpovědi označte tvrzení, která představují doporučenou praxi při modelování dat reprezentovaných pomocí Resource description Framework (RDF)
 - Existující slovníky, či ontologie není doporučeno kombinovat ANO - NE
 - IRI sloužící jako identifikátory objektů by mely být tvořeny jako tzv. LD IRI, aby bylo možné využít zvláštního linked data protokolu pro diferencování dat ANO - NE
 - Při modelování je doporučené využít v maximální míře existujících slovníků a ontologií ANO - NE
 - Protože se IRI se strukturou webu budou měnit, není doporučováno využít IRI, podle užitých vzorů ANO - NE
13. Označte pravdivá (správně) a nepravdivá (špatně) tvrzení týkající se ontologií a slovníků v kontextu sémantického webu a dat reprezentovaných pomocí Resource Description Framework (RDF)
 - Ontologie/slovник je reprezentován pomocí RDF ANO-NE
 - Ontologie/slovник definuje trídy pro predikáty pro reprezentaci dat u určité domény pomocí RDF ANO-NE
 - Ontologie/slovник představuje katalog určité databáze ANO-NE
 - Jak ontologie/slovníky, tak konceptuální schéma používají stejnou terminologii, tj. entitní množiny, atributy entitních množin a vazby mezi entitními množinami ANO-NE
 - Na ontologii/slovnik lze také nahlížet jako na veřejný a sdílený katalog tříd a vlastností pro reprezentaci dat u určité domény
14. Jaký je hlavní přístup k datům NoSQL databázovému systému klíč-hodnota?
 - Přístup pomocí specializovaného jazyka umožňujícího vyhledávání v uložených hodnotách
 - Přístup pomocí hodnoty, která vráci hledaný klíč
 - Přístup pomocí klíče záznamu. Dotaz vráci hodnotu k zadanému klíči
 - Přístup pomocí jazyka SQL umožňujícího vyhledávání v uložených hodnotách
15. Mějme tabulku faktura s primárním klíčem číslo_fakt a zakazník s primárním klíčem číslo_zak.
Dále byl vytvořen objekt typu SEQUENCE pomocí následujícího příkazu: CREATE SEQUENCE cisla_seq START WITH 100 INCREMENT BY 1;
Hodnota z objektu cisla_seq dosud nebyla získána žádným příkazem. Při řešení úlohy dále předpokládejme, že tabulky faktura a zakazník mají vedle sloupců představujících primární klíč všechny další sloupce potřebné k tomu, aby dále uvedené příkazy mohly být provedeny
Které z následujících tvrzení je pravdivé?
 - Po provedení příkazu INSERT INTO faktura VALUES (cisla_sq.nextval, 'Pavel Nový', TO_DATE('2018-05-24', 'YYYY-MM-DD')); již nebude možné použít objekt cisla_seq pro generování hodnot při vkládání či aktualizaci dat v tabulce zakazník, protože při prvním použití objektu typu SEQUENCE v rámci manipulace s daty v určité tabulce dojde k asociaci objektu typu SEQUENCE s příslušnou tabulkou
 - Provádění příkazu INSERT INTO faktura VALUES (cisla_sq.nextval, 'Pavel Nový', TO_DATE('2018-05-24', 'YYYY-MM-DD')); skončí chybou. Objekt typu SEQUENCE je třeba asociovat se sloupcem požadované tabulky pomocí příkazu ALTER TABLE.
 - Provádění příkazu INSERT INTO faktura VALUES (cisla_sq.nextval, 'Pavel Nový', TO_DATE('2018-05-24', 'YYYY-MM-DD')); skončí chybou. Objekt typu SEQUENCE je třeba asociovat se sloupcem požadované tabulky pomocí příkazu ALTER TABLE.
 - Žádné z uvedených tvrzení není pravdivé
16. Doplňte následující příkaz tak, aby mezi tabulkami faktura a polozka_fakt byla vytvořena referenční integrita taková, že po smazání záznamu v tabulce faktura budou

zároveň v tabulce položka fakt smazány všechny záznamy představující položky dané faktury.

o ALTER TABLE zam ADD CONSTRAINT fk_zam_cisodd FOREIGN KEY (cis_odd) REFERENCES oddel (cis_odd) ON DELETE CASCADE

17. Který nebo které z následujících příkazů jazyka SQL **neslouží** k přidělování či odebírání oprávnění provádět operace v rámci databáze nebo s databázovými objekty?

- o SELECT ANO-NE
- o ROLLBACK ANO-NE
- o REVOKE ANO-NE
- o COMMIT ANO-NE

18. Indexy v databázi jsou

- o Jediný způsob uložení dat v objektově relačních databázových systémech
- o Tabulky s obráceným pohledem na data
- o Pomocné struktury k urychlení vyhledávání dat
- o Vytvořeny pro každý alfanumericky sloupec každé databázové tabulky

19. Jaké části databázového jazyka jsou nejčastěji rozšiřovány

- o Data Definition Language ANO-NE
- o Data Selection Language ANO-NE
- o Data Control Language ANO-NE
- o Data Manipulation Language ANO-NE
- o Data Description Language ANO-NE

20. Jaké organizace se zabývají standardizací databázového jazyka SQL?

- o ISACA
- o ANSI
- o IEC
- o ISO
- o ICT Unie
- o OGC

21. Která tvrzení o normalizaci dat jsou pravdivá (správně), a která nikoli (špatně)?

- o Normalizovat lze entitní množiny z konceptuálního datového modelu ANO-NE
- o Normalizovat lze strukturu tabulek v navrhované i existující databázi ANO-NE
- o Cílem normalizace dat je optimalizovat data z hlediska výkonnosti ANO-NE
- o Množinu datových položek (např. obsah nějakého formuláře) nelze normalizovat ANO-NE

22. Primární klíč relační databázové tabulky

- o Může být v složen v více poli této tabulky
- o Slouží pouze pro třídění řádků tabulky
- o může v nějakém řádku obsahovat prázdnu hodnotu
- o obsahuje data, která nesouvisí s objektem popisovaným v příslušné řadce tabulky
- o nemusí být v tabulce jen jeden

23. Entita v datovém modelování je

- o rozlišitelný a identifikovatelný objekt světa, který modelujeme s cílem zachytit jej v databázi ANO-NE
- o množina objektů, které budou vloženy do databáze ANO-NE
- o podmnožina kartézského součinu doménových množin ANO-NE
- o třída objektů, které budou vloženy do databáze ANO-NE

24. Binární vztah v datovém modelování umožnuje zachytit vztah mezi

- o Prvky (entitami) a jejich atributy ANO-NE
- o Prvky (entitami) dvou entitních množin ANO-NE
- o Prvky (entitami) tří a více entitních množin ANO-NE
- o Prvky jedné entitní množiny ANO-NE

25. Co řeší následující dotaz jazyka SQL:

Select číslo, jméno from zaměstnanci where plat > (select avg(plat) from zaměstnanci where

zamestnanci.cislo Oddeleni = vnejsi.cislo oddeleni)

- o Vypisuje čísla a jména zaměstnanců, kteří mají plat větší než je průměrný plat v jejich odděleních
- o Vypisuje čísla a jména zaměstnanců, kteří mají plat větší než je plat jejich vedoucích
- o Vypisuje čísla a jména zaměstnanců, kteří mají plat větší než je průměrný plat v ostatních odděleních
- o Vypisuje čísla a jména zaměstnanců, kteří mají plat větší než je průměrný plat všech zaměstnanců

26. Kterí autoři se nejvíce podíleli na vzniku a rozvoji relačního modelu dat?

- o J.A Table
- o C.J.Date
- o A.Kazimir
- o P.Chen
- o E.F.Codd

27. Které či která z následujících možností představují existující serializaci RDF (Resource Description Framework)? Serializací je myšlena syntaxe pro zápis a výměnu dat v RDF.

- o JSON LD
- o Zebra A
- o Turtle
- o TriG

28. Která tvrzení o grafových databázových systémech jsou pravdivá

- o Data mají podobu grafu, tj. užívají propojených hranami ANO-NE
- o Grafové databázové systémy mají široké využití a jsou vhodné např. pro zpracování dat ze sociálních sítí, nebo pro úlohy související s umístěním v prostoru, např. plánování tras, ANO-NE
- o Grafová data lze obtížně distribuovat ANO-NE
- o Grafové databázové systémy mají jen omezené využití, protože v realitě se jen zřídka vyskytuje situace, kde by výskytu určitých entitních množin byly provádzány nějakými vztahy ANO-NE

29. Pokud v normalizované relační databázi potřebujeme uchovávat jména, data narození a telefony osob, s možností více telefonních čísel pro jednu osobu:

- o Lze to realizovat dvěma tabulkami
- o Lze to realizovat jednou tabulkou
- o Potřebujeme alespoň tři tabulky
- o Potřebujeme tolik tabulek, kolik může mít jedna osoba tel. čísel

30. Množinové sjednocení relačních tabulek lze v SQL udělat

- o UNION
- o JOIN,
- o GROUP BY,
- o WHERE

31. Která z tvrzení o dvoufázovém commitu (2PC, two-phase commit) jsou pravdivá

- o Při dvoufázovém commitu jeden uzel distribuované databáze vystupuje v roli koordinátora transakce a zbylé užly vystupují v roli účastníků transakce
- o Dvoufázový commit je technikou pro zajištění integrity distribuované databáze, protože i zde probíhá commit ve dvou fázích: BEGIN TRANSACTION a END TRANSACTION
- o Pomocí dvoufázového commitu je v distribuované databázi zajišťováno, že se distribuovaná transakce provede ve všech uzlech sítě, nebo v žádném

32. Přístupová práva v databázi se řídí principem

- o pouze správce databáze může přidělit práva k datům ostatním uživatelům
- o k datům má přístup jenom správce databáze
- o co není povoleno, je zakázáno
- o co není zakázáno, je povoleno

33. Základní vlastnosti databázové transakce jsou:

- o Atomicity, Collaboration, Traceability
- o Atomicity, Consistency, Independence, Durability

RDF (Resource Description Framework) je standard pro reprezentaci a výměnu strukturovaných informací na webu. Zde jsou klíčové body týkající se RDF a domén dat, která lze pomocí RDF reprezentovat:

Co je RDF?

Reprezentace dat: RDF umožňuje reprezentovat informace ve formě trojic (tripletů), kde každá trojice obsahuje subjekt, predikát a objekt. Například: (subjekt: "Kniha123", predikát: "má autora", objekt: "J.K. Rowling").

Formáty serializace: RDF data mohou být serializována v různých formátech, jako je RDF/XML, Turtle, N-Triples, JSON-LD, a další. Tyto formáty umožňují flexibilní způsoby zápisu a čtení dat.

Jednotná identifikace: RDF používá URI (Uniform Resource Identifiers) k jednoznačné identifikaci zdrojů, což usnadňuje jejich propojení a sdílení napříč různými systémy.

Domény dat reprezentovatelné pomocí RDF

- o co není zakázáno, je povoleno
- 33. Základní vlastnosti databázové transakce jsou:**
- o Atomicity, Collaboration, Traceability
 - o Atomicity, Consistency, Independence, Durability
 - o Plan, Act, Do, Check
 - o Analytical, Treatable, Veritable
- 34. Jaké / jaká z dalej uvedených klíčových slov není / nejsou příkazem jazyka SQL**
- o Display ANO-NE
 - o Update, ANO-NE
 - o Drop, ANO-NE
 - o Select ANO-NE
 - o Insert ANO-NE
- 35. Co platí o RDF a doménách dat, která lze pomocí RDF reprezentovat?**
- o RDF představuje generický model nezávislý na jakékoli doméně
 - o Pomocí RDF lze reprezentovat pouze data o organizacích
 - o Pomocí RDF lze reprezentovat pouze data z domén, pro které v katalogu Linked Open Vocabularies existuje odpovídající slovník nebo ontologie
 - o Nic z předešlého
- 36. Při neúspěšném ukončení databázové transakce dojde**
- o K odstranění změn dosud provedených v rámci transakce ANO-NE
 - o dosud provedené změny jsou promítány do databáze a uživateli spouštějící transakci je požádán o restart transakce ANO-NE
 - o dosud provedené změny jsou promítány do databáze a transakce je restartována ANO-NE
 - o je provedena obnova stavu databáze do stavu před zahájením transakce ANO-NE
- 37. K možným formám výsledku dotazu ve SPARQL 1.1 Query Language přířadte**
- | | |
|---------------------------------------------------------|-----------|
| Vrací RDF graf s popisem zdroje s daným IRI | DESCRIBE |
| Vrací jako výsledek boolean(True/False) | ASK |
| Vrací výsledek jako RDF graf | CONSTRUCT |
| Vrací všechny nebo zvolené atributy odpovídající dotazu | SELECT |
- 38. K možným formám výsledku dotazu ve SPARQL 1.1 Query Language DESCRIBE přířadte**
- o Vrácí jako výsledek boolean(T/F)
 - o Vrácí výsledek jako RDF graf
 - o Vrací RDF graf s popisem zdroje s daným IRI
 - o Vrácí všechny nebo zvolené atributy odpovídající dotazu
- 39. K možným formám výsledku dotazu ve SPARQL 1.1 Query Language ASK přířadte**
- o Vrácí jako výsledek boolean(T/F)
 - o Vrácí výsledek jako RDF graf
 - o Vrací RDF graf s popisem zdroje s daným IRI
 - o Vrácí všechny nebo zvolené atributy odpovídající dotazu
- 40. K možným formám výsledku dotazu ve SPARQL 1.1 Query Language SELECT přířadte**
- o Vrácí jako výsledek boolean(T/F)
 - o Vrácí výsledek jako RDF graf
 - o Vrací RDF graf s popisem zdroje s daným IRI
 - o Vrácí všechny nebo zvolené atributy odpovídající dotazu
- 41. K možným formám výsledku dotazu ve SPARQL 1.1 Query Language CONSTRUCT přířadte**
- o Vrácí jako výsledek boolean(T/F)
 - o Vrací výsledek jako RDF graf
 - o Vrací RDF graf s popisem zdroje s daným IRI
 - o Vrácí všechny nebo zvolené atributy odpovídající dotazu
- 42. Která tvrzení o dokumentově orientovaných NoSQL databázových systémech jsou pravdivá (správně)**
- o Dokument je soubor obsahující text v přirozeném jazyce, který může být členěn do celků, jako jsou např. kapitoly nebo odstavce. Typicky je uložen ve formátu DOCX nebo PDF.
 - o V rámci jedné databáze musí mít všechny dokumenty stejnou strukturu
 - o Lze provádět dotazy nad obsahem dokumentů
 - o Dokument je samopopisná datová struktura typicky ve formátu JSON, BSON nebo XML
- 43. Rozhodněte, zda příkazy pro provádění uvedených operací zavádí specifikace SPARQL 1.1 Query Language nebo SPARQL 1.1 Update**
- | | |
|-------------------------------------------------------------|-------------------|
| Vložení trojic do grafu pomocí příkazu INSERT DATA | SPARQL 1.1 Update |
| Vložení nového grafu pomocí příkazu CREATE | SPARQL 1.1 Update |
| Smažání trojic z grafu pomocí příkazu DELETE DATA | SPARQL 1.1 Update |
| Dotazování pomocí příkazů SELECT, Construct, ASK a DESCRIBE | SPARQL 1.1 QL |
- 44. Co je SPARQL?**
- o SPARQL je pokročilý jazyk pro dotazování v relačních databázích rozšiřující SQL.
 - o SPARQL je dotazovací jazyk obecně použitelný pro dotazování v NoSQL databázích.
 - o SPARQL představuje specifikaci architektury pro vysokou paralelizaci provádění dotazů v relačních databázích, tj. Super-Parallel Architecture for Relational Query Languages
 - o SPARQL představuje sadu specifikací konsorcia W3C, které definují jazyk pro dotazování a manipulaci s daty v RDF a také související protokol.
- 45. Rozhodněte, zda příkazy pro provádění uvedených operací zavádí specifikace SPARQL 1.1 Update**
- o Vložení nového grafu pomocí příkazu CREATE
 - o Dotazování pomocí příkazů SELECT, Construct, ASK a DESCRIBE
 - o Vložení trojic do grafu pomocí příkazu INSERT DATA
 - o Smažání trojic z grafu pomocí příkazu DELETE DATA
- 46. Rozhodněte, zda příkazy pro provádění uvedených operací zavádí specifikace SPARQL 1.1 Query language**
- o Vložení nového grafu pomocí příkazu CREATE
 - o Dotazování pomocí příkazů SELECT, Construct, ASK a DESCRIBE
 - o Vložení trojic do grafu pomocí příkazu INSERT DATA
 - o Smažání trojic z grafu pomocí příkazu DELETE DATA
- 47. Rozhodněte, zda následující tvrzení patří mezi principy propojených dat (LINKED DATA). Tvrzení, která mezi tyto principy patří označte jako správná.**
- o Používejte IRI jako identifikátory objektů ANO-NE
 - o Používejte http://IRI aby bylo možné objekty vyhledat v prostředí webu a odkazovat na ně ANO-NE
 - o Při přístupu na IRI objektu poskytněte o daném objektu data pomocí standardu HTML ANO-NE
 - o Propojte související objekty pomocí IRI ANO-NE
- 48. Jakou podobu mají data reprezentovaná pomocí Resource Description Framework (RDF)?**
- o Data reprezentovaná pomocí RDF představují strom elementů v jazyce HTML
 - o Data reprezentovaná pomocí RDF představují množinu dvoujic klíč-hodnota
 - o Data reprezentovaná pomocí RDF představují síť vzájemně propojených-tabulek
 - o Data reprezentovaná pomocí RDF představují síť (orientovaný graf) tvořenou trojicemi subjekt-predikát-objekt
- 49. Co je podstatou CAP (Brewerova) teoremu**
- o Distribuovaný systém, v rámci kterého jsou sdílena data, se může vyznačovat pouze dvěma z následujících tří vlastností: konzistence, dostupnost a odolnost vůči výpadku site
 - o V distribuovaných databázových systémech je horizontální škálovatelnost typicky zajíždována master-slave replikací, nebo peer-to-peer replikací v kombinaci s rozdělením dat mezi uzly.
 - o Transakce se tváří jako jeden celek, musí proběhnout celá nebo vůbec ne
 - o Nic z předešlého

50. Vztah s kardinalitou n:m v relační databázi

- Nelze realizovat
- Lze realizovat cizím klíčem v tabulce té z entit, pro kterou je tento vztah povinný
- Musí být realizován cizími klíči v tabulkách obou entit, které jsou v tomto vztahu
- **Ize realizovat tabulkou s primárním klíčem složeným z cizích klíčů**

51. Konceptuální model dat se vytváří za účelem:

- Specifikace pravidel pro ochranu a zabezpečení dat v relačním prostředí ANO-NE
- popis obsahu datové základny na úrovni, která je nezávislá na vlastním technologickém a implementačním prostředí ANO-NE
- poznání světa, který má být zobrazen v databázi ANO-NE
- popis obsahu – struktury – databáze v relačním prostředí ANO-NE

52. Implementační model dat se vytváří za účelem

- Specifikace pravidel pro ochranu a zabezpečení dat v relačním prostředí ANO-NE
- Popis obsahu datové základny na úrovni, která je nezávislá na technologickém a implementačním prostředí ANO-NE
- **Popis obsahu – struktury – databáze v konkrétním databázovém prostředí ANO-NE**
- Poznání světa, který má být zobrazen v databázi ANO-NE

53. Jaký model dat je používán v nejrozšířenějších aplikacích typu ERP / ERP II?

- Hierarchický
- Stromový
- **Relační**
- Sítový
- Objektový
- Relačně-objektový

54. OLAP zpracování je typické pro

- ERP systémy ANO-NE
- **Bí systémy ANO-NE**
- CRM systémy ANO-NE
- ERP II systémy ANO-NE

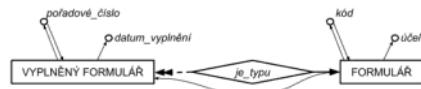
55. Označte pravdivá (správné) a nepravdivá (špatně) tvrzení o situaci označované jako „deadlock“ v kontextu databázového zpracování

- Deadlock označuje situaci, kdy dojde k definitivní ztrátě přihlašovacích údajů k účtu správce databáze ANO-NE
- Deadlock označuje situaci, kdy je použit příkaz DEADLOCK pro trvalé uzamčení záznamu v databázi ANO-NE
- **Deadlock označuje situaci, kdy dvě nebo více transakcí čekají na podmínky, které nikdy nenastanou ANO-NE**
- Deadlock označuje situaci, kdy dojde k uzamčení uživatelského účtu v databázi a uživ se tak nemůže přihlásit ANO-NE

56. Primární klíč relační databázové tabulky

- Může v nějakém řádku obsahoval prázdnou hodnotu
- **Se používá pro jednoznačnou identifikaci řádku**
- Slouží pouze pro třídění řádků tabulky
- Obsahuje data, která nesouvisí s objektem popisovaným v příslušné řádce tabulky
- Nemusí být v tabulce jen jeden

57. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek



- Toto schéma:



- Toto schéma:



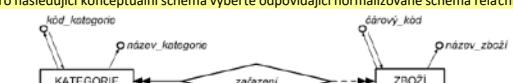
- Obě možnosti jsou správné

58. Schéma



- Jedno a totéž zboží může být zařazeno do více různých kategorií ANO-NE
- V každé kategorii musí být zařazeno nějaké zboží ANO-NE
- Zboží musí být zařazeno v nějaké kategorii ANO-NE
- Do kategorie může být zařazeno více zboží ANO-NE

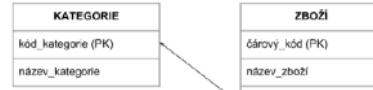
59. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek



- Toto řešení:



- Toto řešení:



- Toto řešení:



60. Schéma

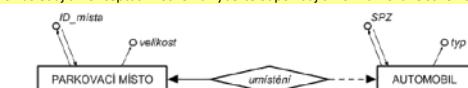


Vyjadřuje:

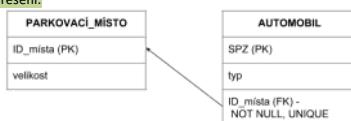
- Každý cestující sedí na nějakém sedadle ANO-NE
- Na každém sedadle sedí nějaký cestující ANO-NE
- **Na žádném sedadle nemohou sedět dva cestující ANO-NE**

- Cestující může sedět na dvou sedadlech ANO-NE

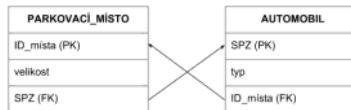
61. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek



Toto řešení:



Toto řešení:



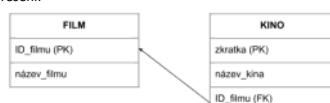
Toto řešení:



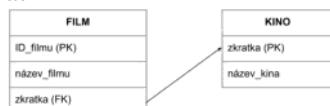
62. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek



Toto řešení:



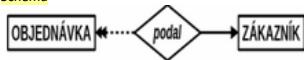
Toto řešení:



Toto řešení:



63. Schéma



- Každou objednávku podal právě jeden zákazník ANO-NE
- Každý zákazník podal jednu objednávku ANO-NE
- Může existovat zákazník, který nepodal žádnou objednávku ANO-NE
- Každý zákazník podal nějakou objednávku ANO-NE

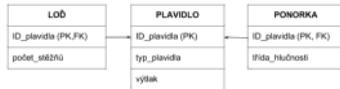
64. Pro následující konceptuální schéma vyberte schéma relačních tabulek, které NENÍ jemu odpovídající a normalizované



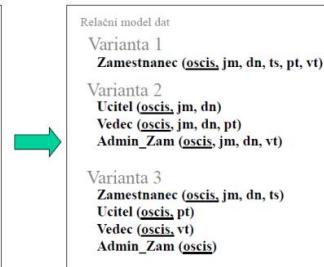
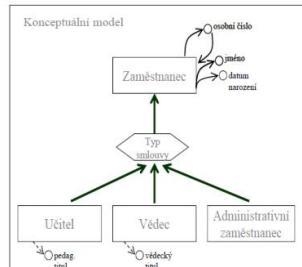
Toto řešení:



Toto řešení:



Toto řešení:

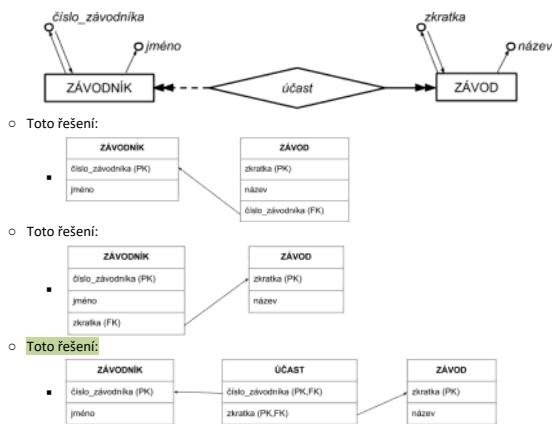


1. Schéma



- Závod se může účastnit více závodníků ANO-NE
- Závodník se musí účastnit nějakého závodu ANO-NE
- Každého závodu se někdo účastní ANO-NE
- Jeden a tenží závodník se může účastnit více různých závodů ANO-NE

2. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek



3. Schéma



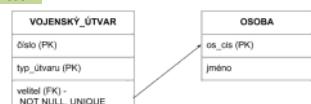
Vyjadřuje:

- Každá osoba něčemu velí ANO-NE
- Každý vojenský útvar má osobu, která mu velí ANO-NE
- Žádná osoba nemůže velet dvěma vojenským útvarym ANO-NE
- Vojenskému útvaru nemohou velet dvě osoby ANO-NE

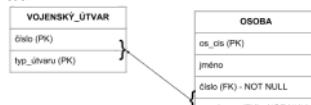
4. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek



○ Toto řešení:

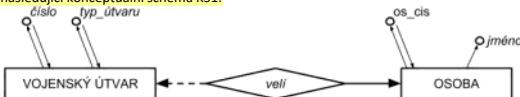


○ Toto řešení:

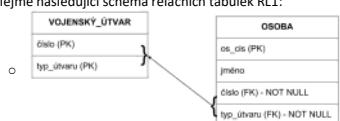


- Obě normalizované schéma relačních tabulek odpovídají zadámu konceptuálnímu schématu. Je možné si zvolit libovolnou z možností.
- Ani jedno z normalizovaných schémát relačních tabulek neodpovídá zadámu konceptuálnímu schématu

5. Mějme následující konceptuální schéma KS1:



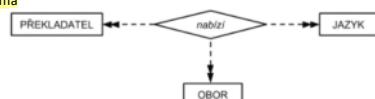
Mějme následující schéma relačních tabulek RL1:



Odpovídá výše uvedené schéma relačních tabulek RL1 správné transformaci konceptuálního schématu KS1 do normalizovaného relačního schématu?

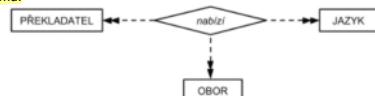
- Relační schéma RL1 by bylo správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu, pokud by kombinace sloupů číslo a typ_útvaru v tabulce OSOBA měla navíc omezení UNIQUE.
- Relační schéma RL1 je správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Relační schéma RL1 není správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Žádná z uvedených možností není správná.

6. Schéma



- Daný jazyk vždy někdo nabízí ANO-NE
- Překladatel v jednom oboru může nabízet různé jazyky ANO-NE
- Daný jazyk může překladatel nabízet v různých oborech ANO-NE
- Daný jazyk může v tomtéž oboru nabízet více překladatelů ANO-NE

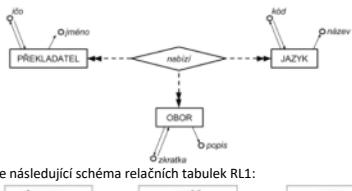
7. Schéma:



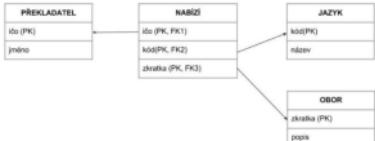
Co výše uvedené konceptuální schéma vyjadřuje?

- Nemůže existovat překladatel, který by nabízel překlady do jednoho jazyku ve více oborech.
- Každý překladatel nabízí překlady alespoň do jednoho jazyka v rámci alespoň jednoho oboru.
- Překladatel v určitém oboru může nabízet překlady jen do jednoho jazyku.
- Může existovat jazyk, pro který není k dispozici překladatel, Jenž by překlady do daného jazyka v určitém oboru nabízel.

8. Mějme následující konceptuální schéma KS1:



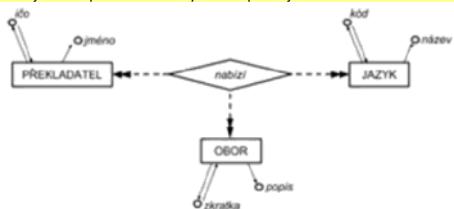
Mějme následující schéma relačních tabulek RL1:



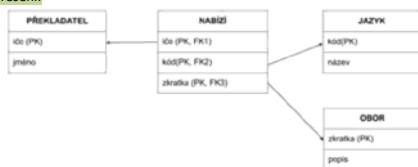
Odpovídá výše uvedené schéma relačních tabulek RL1 správné transformaci konceptuálního schématu KS1 do normalizovaného relačního schématu?

- Relační schéma RL1 **bylo** správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu, pokud by byl primární klíč tabulky NABÍZÍ tvořen pouze kombinací sloupců ičo a kód.
- **Relační schéma RL1 je správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.**
- Relační schéma RL1 **není** správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Žádná z uvedených možností není správné.

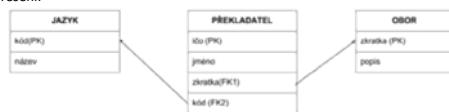
9. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek



○ **Toto řešení:**



○ **Toto řešení:**



- Obě možnosti jsou správné

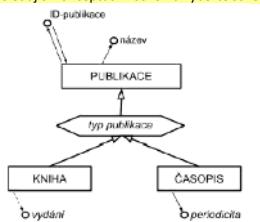
10. Schéma



Vyjadřuje:

- Publikace je buď kniha nebo časopis ANO-NE
- Časopis je publikace ANO-NE
- Kniha je publikace ANO-NE
- Každá publikace se může skládat z nějakých knih a časopisů ANO-NE

11. Pro následující konceptuální schéma vyberte schéma relačních tabulek, které **NENÍ** jemu odpovídající a normalizované



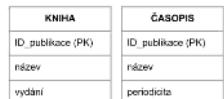
○ **Toto řešení:**



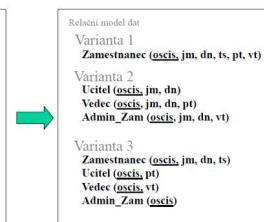
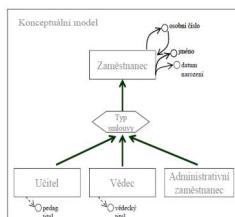
○ **Toto řešení:**



○ **Toto řešení:**



○ **Toto řešení:**



PUBLIKACE
ID_publikace (PK)
typ_publikace
název
vydání
periodicitu

12. Mějme tabulku projekt a omezení chk_projekt_datum vytvořené pomocí následujících příkazů:

```
CREATE TABLE projekt (
    kod INTEGER PRIMARY KEY,
    nazev VARCHAR2(30) NOT NULL,
    datum_zahajeni DATE NOT NULL,
    datum_ukonceni DATE
);
ALTER TABLE projekt ADD CONSTRAINT chk_projekt_datum CHECK (datum_zahajeni <= datum_ukonceni);
```

Který z příkazů nebo příkazy povodou k úspěšnému odstranění omezení chk_projekt_datum?

- ALTER TABLE projekt DROP CONSTRAINT chk_projekt_datum ANO-NE
- DROP TABLE projekt ANO-NE
- DELETE CONSTRAINT chk_projekt_datum ANO-NE
- DROP CONSTRAINT chk_projekt_datum ANO-NE

13. Mějme tabulky zam (sloupec os_cis představuje primární klíč) a oddel (sloupec cis_odd představuje primární klíč) s následující strukturou a obsahem:

Tabulka ZAM		
os_cis	jmeno	cis_odd
1	Jan Nový	11
2	Petra Králová	11
3	Jakub Svec	NULL
4	Anna Pešková	13

Tabulka ODDEL	
cis_odd	název
11	Nákup
13	Prodej

Bylo vytvořeno integritní omezení pomocí následujícího příkazu

```
ALTER TABLE zam ADD CONSTRAINT fk_zam_cisodd
FOREIGN KEY (cis_odd) REFERENCES oddel (cis_odd);
```

Jaké příkazy skončí chybou? Příkazy, jejichž vykonání skončí chybou, označte jako správné

- DELETE FROM oddel WHERE cis_odd = 11; ANO-NE
- INSERT INTO zam VALUES (5, „Pavel Skočidopole“); ANO-NE
- INSERT INTO oddel VALUES (17, „Servis“); ANO-NE
- UPDATE zam SET cis_ODD = NULL where os_cis = 2; ANO-NE

možná i toto? Protože tam není definované delete, předpokládám že je tam restrict -> nemůžu smazat oddel 11, dokud bude mít nějaký zam přidělené oddělení 11

14. V tabulce služba představují sloupcy kod_sluzby a typ_sluzby a typ_sluzby je požadováno, aby tabulce položka byl definován cizí klíč odkazující na primární klíč v tabulce služba. Které z následujících tvrzení je pravdivé?

Cizí klíč bude vytvořen podle následujícího příkazu

```
o ALTER TABLE položka_fakt ADD CONSTRAINT fk_položkafakt_kodsl1_typs1
FOREIGN KEY (kod_sluzby ,typ_sluzby) REFERENCES služba (kod_sluzby ,typ_sluzby);
```

15. Označte pravdivá (správná) a nepravdivá (špatně) tvrzení ohledně pohledu (VIEW)

- Definici obsahu VIEW (subdotaz) lze změnit pomocí příkazu CREATE OR REPLACE VIEW ANO-NE
- Definici obsahu VIEW (subdotaz) lze změnit pomocí příkazu ALTER VIEW ANO-NE
- Odstranit VIEW (zde pomocí příkazu DELETE VIEW ANO-NE)
- Omezení VIEW (CONSTRAINTS) lze změnit pomocí příkazu ALTER VIEW ANO-NE

16. K čemu slouží REVOKE jazyka SQL?

- K přejmenování databázové tabulky ANO-NE
- K odebrání oprávnění přidělených uživatelů ANO-NE
- K odebrání objektových oprávnění přidělených za předpokladu, že SŘDB podporuje role ANO-NE
- K navrácení databáze do stavu před zahájením transakce ANO-NE

17. Který z následujících příkazů jazyka SQL slouží k přidělování či odebrání oprávnění provádět operace v rámci databáze nebo s databázovými objekty?

- DELETE
- REVOKE
- SELECT
- CREATE VIEW

18. Doplňte následující příkaz tak, aby prostřednictvím vytvořeného pohledu (VIEW) „inženýri“ nebylo možné vložit záznam, který by nebyl součástí množiny výsledků použitého SELECT subdotazu. Tj. aby skrze tento pohled nebylo možné vložit např. záznam o osobě s titulem RNDr.

```
o Create VIEW inženýri AS
SELECT os_cis ,jmeno, titul
FROM zam
Where titul like 'ING'
/*chybějící klauzule*/;
```

Jaká klauzule může být doplněna místo textového řetězce /*chybějící klauzule */, aby prostřednictvím vytvořeného pohledu "inženýri" bylo možné vkládat záznamy, ale aby nebylo možné vložit záznam, který by nebyl součástí množiny výsledků použitého SELECT subdotazu? Tj. aby skrze tento pohled nebylo možné vložit např. záznam o osobě s titulem RNDr.

- With read only
- With check option
- With insert option
- With update option

20. Mějme tabulku zam a pohled docenti vytvořené pomocí následujících příkazů

```
CREATE TABLE zam (
    Os_cis INTEGER PRIMARY KEY,
    Jmeno VARCHAR2(50) NOT NULL,
    Titul VARCHAR2(5) NOT NULL,
    Plat INTEGER
);

CREATE VIEW docenti AS
SELECT os_cis ,jmeno, titul
FROM zam
WHERE titul LIKE 'DOC'
WITH CHECK OPTION;
```

Označte jako správné ty z následujících příkazů, jejichž vykonání skončí chybou za předpokladu, že tabulka zam je prázdná

- INSERT INTO docenti VALUES (336, 'Karla Nováková'); ANO-NE
- INSERT INTO docenti VALUES (333, 'Pavel Novák', 'DOC'); ANO-NE
- INSERT INTO docenti VALUES (334, 'Pavla Nováková', 'ING'); ANO-NE
- INSERT INTO docenti VALUES (335, 'Karel Novák', 'DOC', 10000); ANO-NE

21. Tabulka zajezd má sloužit k ukládání dat o zájezdech včetně data zahájení zájezdu (sloupec datum_zacatek) a data ukončení zájezdu (sloupec datum_konec). Je třeba zjistit, aby nebyl uložen záznam na kterém datum ukončení zájezdu je později než datum zahájení zájezdu. Která z násled. Tvrzení o možnostech zajištění tohoto požadavku jsou pravdivá (správné). Uvažujte použití SŘBD Oracle 11g.

- Požadavek lze zjistit pomocí omezení typu CHECK s podmínkou datum_zacatek <= datum_konec, které bude definováno v rámci příkazu CREATE TABLE samostatně mimo definici kteréhožkoliv ze sloupců TJ. Ize použít tzv. „out-of-line“ specifikaci CHECK podmínky. ANO-NE
- Požadavek nelze zjistit na úrovni databáze a je nutné ho zajistit na úrovni aplikace ANO-NE
- Požadavek lze zjistit pomocí omezení typu CHECK s podmínkou datum_zacatek <= datum_konec, které bude definováno v rámci příkazu CREATE TABLE jako součást definice sloupců datum_konec. Ize použít tzv. „inline“ specifikaci CHECK podmínky ANO-NE
- Požadavek lze zjistit pomocí objektu typu TRIGGER ANO-NE

22. Uživatel uživ_a vytvořil tabulku zam a udělil právo provádět operaci SELECT na tabulce zam uživateli uživ_b následujícím příkazem:

GRANT SELECT ON zam TO uživ_b WITH GRANT OPTION;

Následně uživ_b přidělil právo provádět operaci SELECT na tabulce zam uživ_c příkazem:

GRANT SELECT ON zam TO uživ_c;

Co se stane, pokud uživateli uživ_b provede následující příkaz? REVOKE SELECT ON zam FROM uživ_c;

- o Příkaz bude proveden ale uživateli uživ_c bude právo odebráno až poté, co odebrání práva potvrdí uživ_a, protože ten uživateli uživ_b přidělil právo s klauzulí WITH GRANT OPTION
- o Příkaz nebude proveden. Uživatel uživ_a sice přidělil uživateli uživ_b právo s klauzulí WGO, ale již mu nepřidělil toto právo s klauzulí WITH REVOKE OPTION.
- o None from the above options

23. V tabulce zam představuje sloupec cis_odd cízklič odkazující na primární klíč v tabulce oddel. Příslušné integritní omezení bylo vytvořeno pomocí následujícího příkazu:

```
ALTER TABLE zam ADD CONSTRAINT fk_zam_cisodd  
FOREIGN KEY (cis_odd) REFERENCES oddel (cis_odd)  
ON DELETE SET NULL;
```

Uživatel sputí příkaz ke smazání záznamu o oddělení v tabulce oddel, pro které ale v tabulce zam existuje alespoň jeden záznam o zaměstnanci, jenž v něm pracuje. Které z následujících tvrzení je pravdivé?

- o Záznam o takovémto oddělení nelze smazat, proto budou v záznamu o zvoleném oddělení hodnoty všech sloupců, kromě těch tvořících primární klíč, nahrazeny hodnotou NULL.
- o Záznam o takovémto oddělení nelze smazat a provádění příkazu ke smazání záznamu o zvoleném oddělení skončí chybou.
- o Záznam o zvoleném oddělení bude smazán a dosavadní hodnota sloupcu cis_odd v tabulce zam bude v záznamech o zaměstnancích, kteří ve smazaném oddělení pracovali, nahrazena hodnotou NULL.
- o Záznam o zvoleném oddělení bude smazán a dosavadní hodnoty všech sloupců v tabulce zam, kromě těch tvořících prim. Klíč, budou v případě záznamu o zaměstnancích, kteří ve smazaném oddělení pracovali, nahrazeny hodnotou null

24. Schéma



Vyjadruje (doplňte věty):

- o Model musí patřit právě jedné značce. Ke značce může patřit více modelů, ale mohou existovat značky, ke kterým nepatří žádný model.

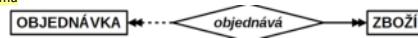
25. Mějme následující konceptuální schéma:



Uvažujte normalizované schéma relačních tabulek odpovídající výše uvedenému konceptuálnímu schématu. Za správné označte ty SQL příkazy, které povedou k vytvoření správných primárních a cizích klíčů.

- o Primární klíč v tabulce VYPLNENY_FORMULAR bude tvořen pouze sloupcem PORADOVE_CISLO. Za předpokladu existence uvedené tabulky a sloupcu bude primární klíč vytvořen pomocí následujícího příkazu:
 - ALTER TABLE vyplneny_formular ADD CONSTRAINT vyplneny_formular_pk PRIMARY KEY (poradove_cislo);
- o Žádný ze sloupců v tabulce FORMULAR nebude představovat cízklič. Primární klíč v tabulce FORMULAR bude tvořen pouze sloupcem KOD. Za předpokladu existence uvedené tabulky a sloupcu bude primární klíč vytvořen pomocí následujícího příkazu:
 - ALTER TABLE formular ADD CONSTRAINT formular_pk PRIMARY KEY (kod);
- o Součástí tabulky VYPLNENY_FORMULAR bude sloupec KOD, který bude cizím klíčem. Za předpokladu existence uvedených tabulek a sloupců bude cízklič vytvořen pomocí následujícího příkazu:
 - ALTER TABLE vyplneny_formular ADD CONSTRAINT vyp_form_kus_fk CIŽÍ KLÍČ (kod)
 - REFERENCES formular (kod);

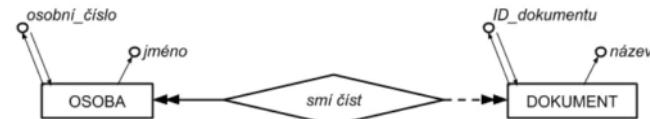
26. Schéma



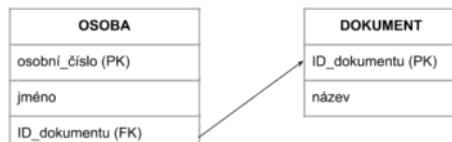
Vyjadruje (doplňte věty):

- o V rámci objednávky musí být objednáno alespoň jedno zboží, ale může být objednáno i více zboží. Zboží může být objednáno v rámci více objednávek, ale může existovat zboží, které nebylo objednáno v rámci žádné objednávky.

79. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek:



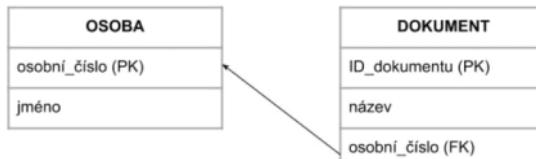
- o Toto řešení:



- o Toto řešení:



- o Toto řešení:



80. Schéma



Vyjadruje:

- o Může existovat osoba, která nesmí číst žádný dokument.
- o Jeden a tentýž dokument může smět číst více různých osob.
- o Osoba může smět číst více dokumentů.
- o Každý dokument musí smět číst nějaká osoba.

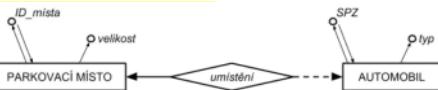
81. Schéma



Vyjadruje (doplňte věty):

- o Osoba může být oprávněna číst více dokumentů, ale mohou existovat osoby, které nejsou oprávněny číst žádný dokument.
- o Dokument musí mít oprávnění číst alespoň jedna osoba, ale oprávnění číst dokument může mít více osob.

82. Mějme následující konceptuální schéma K51:



Mějme následující schéma relačních tabulek RL1:

PARKOVACÍ_MÍSTO	AUTOMOBIL
ID_místa (PK)	
velikost	
SPZ (FK) - NOT NULL, UNIQUE	SPZ (PK)
	typ

Odpovídá výše uvedené schéma relačních tabulek RL1 správné transformaci konceptuálního schématu KS1 do normalizovaného relačního schématu?

- Na základě schématu KS1 a RL1 **nelze posoudit**, zda je relační schéma RL1 správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Relační schéma RL1 je správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- **Relační schéma RL1 není správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.**
- Žádná z uvedených možností není správná.

83. Mějme následující konceptuální schéma:



Co výše uvedené konceptuální schéma vyjadřuje?

- Zákazník je **bud osoba nebo organizace**. ANO-NE
- Některé osoby jsou zákazníky, ale mohou existovat také osoby, které zákazníkem nejsou. ANO-NE
- Osoba pracuje pro organizaci a jak osoba, tak organizace mohou být zákazníkem. ANO-NE
- Každý záznam o zákazníkovi je tvořen údajem o osobě a jeho organizaci. ANO-NE

84. Mějme následující konceptuální schéma KS1:



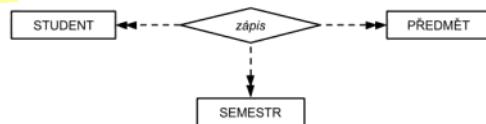
Mějme následující schéma relačních tabulek RL1:

KNIHA	PUBLIKACE	ČASOPIS
ID_publikace (PK, FK)	ID_publikace (PK)	ID_publikace (PK, FK)
vydání	typ_publikace	periodicita

Odpovídá výše uvedené schéma relačních tabulek RL1 správné transformaci konceptuálního schématu KS1 do normalizovaného relačního schématu?

- Relační schéma RL1 **bylo** jednou z možných správných transformací konceptuálního schématu KS1 do normalizovaného schématu, pokud by sloupec "název" byl součástí jak tabulky "kníha", tak též tabulky "časopis".
- Relační schéma RL1 **není** správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Relační schéma RL1 **je** jednou z možných správných transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Žádná z uvedených možností není správná.

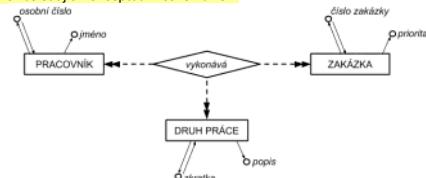
85. Schéma



Vyjadřuje

- Student si může v jednom semestru zapsat více předmětů. ANO-NE
- Student si může dany předmět zapsat ve více semestrech. ANO-NE
- **Dany předmět může mít v téže semestrů zapsáno více studentů.** ANO-NE
- Do daného předmětu se v každém semestru někdo zapsal. ANO-NE

86. Mějme následující konceptuální schéma KS1:



Mějme následující schéma relačních tabulek RL1:

PRACOVNIK	VYKONAVA	ZAKÁZKA	DRUH_PRACE
osobni_cislo (PK)	osobni_cislo (PK, FK)	cislo_zakazky (PK, FK)	zkratka (PK, FK)
jmeno	zkratka (PK)	priorita	
			zkratka (PK)
			popis

Odpovídá výše uvedené schéma relačních tabulek RL1 správné transformaci konceptuálního schématu KS1 do normalizovaného relačního schématu?

- Relační schéma RL1 **není** správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Relační schéma RL1 **bylo** správnou transformací konceptuálního schématu KS1 do normalizovaného schématu, pokud by primární klíč tabulky "vykonává" byl tvořen pouze kombinací sloupců "číslo_zakázky" a "zkratka"
- Relační schéma RL1 **bylo** správnou transformací konceptuálního schématu KS1 do normalizovaného schématu, pokud by primární klíč tabulky "vykonává" byl tvořen pouze kombinací sloupců "číslo_zakázky" a "osobni_cislo"
- Žádná z uvedených možností není správná.

87. Schéma



a.



Vyjadřuje

- Pracovník na jedné zakázce může vykonávat různé druhy práce.
- **Dny druh práce může na téže zakázce vykonávat více pracovníků.**

- Dany druh práce může pracovník vykonávat na různých zakázkách.
- Dany druh práce vždy někdo vykonává.

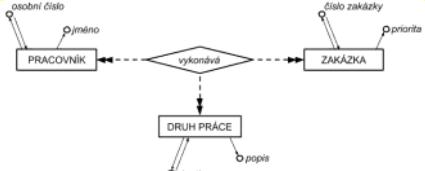
88. Schéma



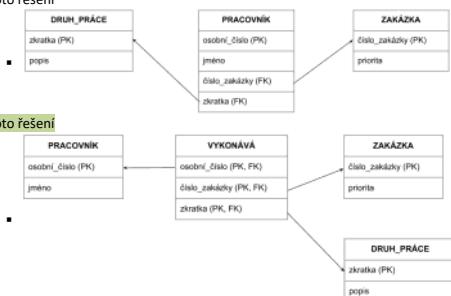
Vyjadřuje

- Na téže zakázce může vykonat práci více pracovníků, ale poukud každý z pracovníků podílejících se na zakázce vykonal jiný druh práce.
- Pracovník vždy na zakázce něco vykonává.
- V rámci zakázky mohou být uvedeny druhy práce, které nevykonal žádný pracovník.
- Pracovník na jedné zakázce může vykonávat různé druhy práce.

89. Mějme následující konceptuální schéma a vyberte odpovídající normalizované schéma relačních tabulek



- Toto řešení



- Obě možnosti jsou správné

90. Mějme tabulku zam se sloupcy os_cis, rodne_cis, jmeno, prijmeni, plat a nadr.

Jako správnou odpověď či odpovědi označte příkaz či příkazy, kterými bude zajištěna entitní integrita.

- ALTER TABLE zam ADD CONSTRAINT chk_plat CHECK (plat >= 0);
- ALTER TABLE zam ADD CONSTRAINT fk_zam_nadr FOREIGN KEY (nadr) REFERENCES zam (os_cis);
- ALTER TABLE zam ADD CONSTRAINT pk_zam_os_cis PRIMARY KEY (os_cis);
- ALTER TABLE zam ADD CONSTRAINT unq_rodne_cis UNIQUE (rodne_cis);

91. Mějme tabulky oddel a zam s následující strukturou a obsahem:

Tabulka ZAM		
os_cis	jmeno	cis_odd
1	Jan Nový	11
2	Petra Králová	11
3	Jakub Švec	15
4	Anna Pešková	13

Tabulka ODDEL	
cis_odd	nazev
11	Nákup
13	Prodej

Je spuštěn následující příkaz:

```
ALTER TABLE zam ADD CONSTRAINT fk_zam_cisodd
FOREIGN KEY (cis_odd) REFERENCES oddel (cis_odd);
```

Které z následujících tvrzení je pravdivé?

- Bude vytvořeno integritní omezení fk_zam_cisodd
- Vykonání příkazu skončí chybou, protože aktuální data v tabulkách zam a oddel nevyhovují vytvořenému integritnímu omezení
- Vykonání příkazu skončí chybou, protože integritní omezení lze vytvářet pouze mezi prázdnými tabulkami
- Zádné z výše uvedených tvrzení není pravdivé.

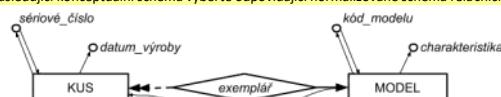
92. Schéma



Vyjadřuje (doplňte věty):

- Vyplněný formulář musí být typem právě jednoho formuláře. K (typu) formuláře může existovat více vyplněných formulářů, ale mohou existovat formuláře, ke kterým neexistuje žádný vyplněný formulář.

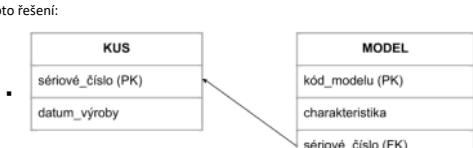
93. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek:



- Toto řešení:

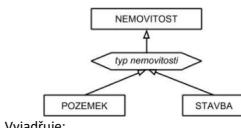


- Toto řešení:



- Obě možnosti jsou správné

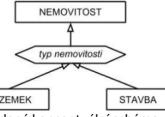
94. Schéma



Vyjadřuje:

- o Pozemek je nemovitost.
- o Ke stavbě musíme vždy přidat pozemek, abychom vytvořili nemovitost.
- o Stavba je nemovitost.
- o Nemovitost je buď pozemek, nebo stavba.

95. Mějme následující konceptuální schéma



Co výše uvedené konceptuální schéma vyjadřuje?

- o Pozemek je typem nemovitosti.
- o Každá stavba stojí na nějakém pozemku.
- o Pozemek, na kterém není stavba, není nemovitost.
- o Na pozemku může stát i jedna nebo více staveb.

96. Mějme tabulky zam a oddel vytvořené pomocí následujících příkazů:

```

CREATE TABLE zam (
    os_cis INTEGER PRIMARY KEY,
    jmeno VARCHAR2(50) NOT NULL,
    cis_odd INTEGER NOT NULL
);

CREATE TABLE oddel (
    cis_odd INTEGER PRIMARY KEY,
    nazev VARCHAR2(30) NOT NULL
);
    
```

Pomočí kterého z následujících příkazů bude vytvořeno integrní omezení takové, že sloupec cis_odd v tabulce zam bude cizím klíčem odkazujícím na primární klíč v tabulce oddel, a zároveň v tabulce oddel nebude možné smazat záznam o oddělení, pokud v tabulce zam bude existovat alespoň jeden záznam o zaměstnanci, který pracuje v oddělení, jenž má být smazáno?

- o ALTER TABLE zam ADD CONSTRAINT fk_zam_cisodd
 FOREIGN KEY (cis_odd) REFERENCES oddel (cis_odd);
- o ALTER TABLE zam ADD CONSTRAINT fk_zam_cisodd
 FOREIGN KEY (cis_odd) REFERENCES oddel (cis_odd)
 ON DELETE CASCADE;
- o ALTER TABLE zam ADD CONSTRAINT fk_zam_cisodd
 FOREIGN KEY (cis_odd) REFERENCES oddel (cis_odd)
 ON DELETE RESTRICT;
- o ALTER TABLE zam ADD CONSTRAINT fk_zam_cisodd
 FOREIGN KEY (cis_odd) REFERENCES oddel (cis_odd);

97. V tabulce zam představuje sloupec cis_odd cizí klíč odkazující na primární klíč v tabulce oddel. Příslušné integrní omezení bylo vytvořeno pomocí následujícího příkazu:

```

ALTER TABLE zam ADD CONSTRAINT fk_zam_cisodd
FOREIGN KEY (cis_odd) REFERENCES oddel (cis_odd)
ON DELETE CASCADE;
    
```

Které z následujících tvrzení je pravdivé?

- o Pokud dojde ke smazání záznamu o oddělení v tabulce oddel, v tabulce zam budou zároveň smazány záznamy o všech zaměstnancích, kteří pracují v oddělení, jenž má být smazáno.
- o V tabulce zam nebude možné smazat záznam o zaměstnanci, dokud bude v tabulce oddel existovat záznam o oddělení, ve kterém daný zaměstnanec pracuje.
- o V tabulce oddel nebude možné smazat záznam o oddělení, dokud bude v tabulce zam existovat alespoň jeden záznam o zaměstnanci, který pracuje v oddělení, jenž má být smazáno.
- o Pokud dojde ke smazání záznamu o zaměstnanci v tabulce zam, v tabulce oddel bude zároveň smazán záznam o oddělení, ve kterém zaměstnanec pracoval.

98. Schéma



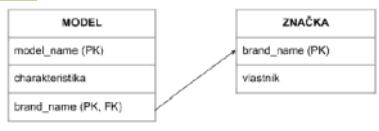
Vyjadřuje (doplňte větu):

- o Produkt **musí** být vyráběn právě jedním výrobcem. Výrobce **může** vyrábět **více** produktů, ale mohou existovat výrobci, kteří nevyrábějí žádný produkt.

99. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek.



- o Toto řešení:

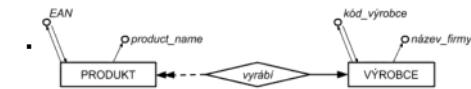


- o Toto řešení:



- o Obě možnosti jsou správné

100. Pro následující konceptuální schéma vyberte odpovídající normalizované schéma relačních tabulek.



- o Toto řešení:



- o Toto řešení:



- o Obě možnosti jsou správné

101. Mějme následující konceptuální schéma KS1:



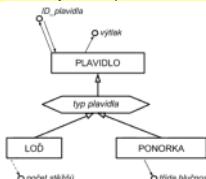
Mějme následující schéma relačních tabulek RL1:

FILM	KINO
ID_filmu (PK)	
název_filmu	
zkratka (FK)	zkratka (PK)

Odpovídá výše uvedené schéma relačních tabulek RL1 správné transformaci konceptuálního schématu KS1 do normalizovaného relačního schématu?

- Relační schéma RL1 je správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Na základě schémat KS1 a RL1 nelze posoudit, zda je relační schéma RL1 správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Relační schéma RL1 bylo správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu, pokud by součástí tabulky KINO byl navíc sloupec ID_filmu, který by zároveň byl cizím klíčem vytvořeným pomocí následujícího příkazu:
 - ALTER TABLE kino ADD CONSTRAINT kino_id_filmu_fk FOREIGN KEY (id_filmu) REFERENCES film (id_filmu)
- Žádná z uvedených možností není správně.

102. Mějme následující konceptuální schéma KS1:



Mějme následující schéma relačních tabulek RL1:

LOĎ	PONORKA
ID_plavidla (PK)	ID_plavidla(PK)
výtlak	výtlak
počet stěžníu	tfida_hlučnost

Odpovídá výše uvedené schéma relačních tabulek RL1 správné transformaci konceptuálního schématu KS1 do normalizovaného relačního schématu?

- Relační schéma RL1 bylo jednou z možných správných transformací konceptuálního schématu KS1 do normalizovaného schématu, pokud by součástí tabulky LOĎ i tabulky PONORKA byl sloupec typ_plavidla, který by v obou tabulkách byl součástí primárního klíče.
- Relační schéma RL1 není správnou transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Relační schéma RL1 je jednou z možných správných transformací konceptuálního schématu KS1 do normalizovaného relačního schématu.
- Žádná z uvedených možností není správně.

103. Mějme tabulku ZAM vytvořenou pomocí následujícího příkazu:

```
CREATE TABLE zam (
    os_cis INTEGER PRIMARY KEY,
    jmeno VARCHAR2 (50) NOT NULL,
    fce VARCHAR2(20)
);
```

Označte, zda jsou následující tvrzení pravdivá (správně) či nepravdivá (špatně). Uvažujte použití SŘBD Oracle 11g.

- Do sloupu jmeno nepůjde zapsat více shodných hodnot, tj. hodnoty ve sloupci jmeno musí být unikátní. ANO-NE
- Příkaz INSERT INTO zam (os_cis, jmeno) VALUES (2, 'Anna Nová'); skončí chybou ANO-NE
- Příkaz INSERT INTO zam VALUES (1, 'Anna Nová', 'Technik'); bude proveden za předpokladu, že v tabulce "zam" nebude obsažen záznam s hodnotou sloupce "os_cis" = 1 ANO-NE
- V tabulce zam představuje sloupec os_cis primární klíč. ANO-NE

104. Označte pravdivá (správně) a nepravdivá (špatně) tvrzení ohledně pohledu (VIEW). Uvažujte použití SŘBD Oracle 11g.

- Příkaz SELECT definující obsah view (subdotaz) musí obsahovat agregační funkci. ANO-NE
- Prostřednictvím view lze vždy vkládat, aktualizovat a mazat záznamy v podkladové tabulce či tabulkách. ANO-NE
- Příkaz SELECT definující obsah VIEW (subdotaz) může obsahovat klauzuli JOIN. ANO-NE
- Změna definice obsahu view (subdotaz) se provádí pomocí příkazu CREATE OR REPLACE VIEW. ANO-NE

105. Který z následujících výčtu neobsahuje žádný příkaz jazyka SQL sloužící k přidělování či odebrání oprávnění provádět operace v rámci databáze nebo s databázovými objekty?

- CREATE USER, ALTER USER, DROP USER
- GRANT, REVOKE, CREATE USER
- SELECT, UPDATE, INSERT, REVOKE
- SET TRANSACTION, COMMIT, REVOKE