Interreg

EVROPSKÁ UNIE

Rakousko-Česká republika

Evropský fond pro regionální rozvoj

INFORMATIKA

Databáze

UNIVERSITY OF APPLIED SCIENCES UPPER AUSTRIA

VŠTE

EVROPSKÁ UNIE

OBSAH

1. Základní pojmy databáze 4
1.1. Systém řízení báze dat 4
1.2. Struktura databáze 6
1.3. Databázová tabulka 6
1.4. Primární klíč 7
1.5. Relace 7
1.6. Primární a sekundární tabulka 7
1.7. Typy relací 8
1.8. Referenční integrita 9
1.9. Obsluha databáze 9
2. Databázové modely 10
2.1. Hierarchický model dat 10
2.2. Síťový model dat 10
2.3. Relační model dat 11
2.4. Objektové databáze 12
2.5. Připojená aplikace 12
2.6. Objektové databáze 12
2.7. Základy objektové orientace. Objekty a třídy 13
2.8. Literály 13
2.9. Operace 14
3. Integrita databáze 16
3.1. Druhy integritních omezení 16
3.2. Dodržování integritních omezení 16
3.3. Vztahy mezi tabulkami 17
3.4. Normální formy 17
3.5. Databázová integrita 18
4. Relační databázový model 20
4.1. 12 pravidel pro relační SŘBD 20
4.2. Relační datový model 22
4.3. Vlastnosti relačního datového modelu 23
4.4. Vazby v relačním modelu 23

5. Zásady SGL – vytváření dotazů ... 25
5.1. Úvod do SQL – popis a vlastnosti ... 25
5.2. Databáze SQL a dotazy ... 27
6. SQL – složitější dotazy ... 34
6.1. Cvičení – dotazy SQL s více tabulkami ... 34
7. Grafové databáze ... 38
7.1. Relační databáze ... 38
7.2. Opravdové databáze ... 38
7.3. Jednosměrné vztahy ... 39
7.4. Obousměrné vztahy ... 40
7.5. Shrnutí ... 41
7.6. Metodika modelování grafové databáze ... 42
7.7. Realizace konceptuálního modelu ... 42
8. NoSQL databáze ... 43
8.1. Co je NoSQL? ... 44
8.2. Kdy použít NoSQL ... 44
9. Transakce ... 47
9.1. Řešení: Transakce ... 47
9.2. Transakce v SQL ... 47
10. Procedury a funkce, triggery a sekvence ... 49
10.1. Formální pohled ... 49
10.2. Metoda ... 51
11. Analytické nástroje OLAP ... 55
11.1. OLAP – informace pod kontrolou ... 55
11.2. Co a jak zjednodušuje práci? ... 55
11.3. Jak tyto informace získat? ... 55
11.4. Analytické nástroje ... 58
11.5. OLAP kostka ... 58
12. Specifika databázových systémů. Technologie přístupu k databázím. Geografické informační systémy. ... 60
12.1. Přehled přístupu k datům v technologii ASP.NET ... 60
13. Databázové objekty ... 61
13.1. Základní pojmy databázové technologie ... 62
13.2. Geografický informační systém ... 63

13.3. Geografická data 64
13.4. Sběr dat 64
13.5. Multidimenzionální kostka 65
13.6. Uložení dat v OLAP databázi 65

1.ZÁKLADNÍ POJMY DATABÁZE

Databáze (neboli datová základna) je určitá uspořádaná množina informací (dat), většinou tabulka se záznamy, uložená na paměťovém médiu. V širším smyslu jsou součástí databáze i softwarové prostředky, které umožňují manipulaci s uloženými daty a přístup k nim.

Databáze je množina záznamů, kterou shromažďujeme za nějakým konkrétním účelem. Databáze používáme zejména pro ukládání obsáhlých informací. Databázové systémy jsou k dispozici jako součást kancelářských balíků (např. MS Access, OpenOffice.org Base). Tyto systémy jsou k dispozici i jako samostatné programy, které se používají pro tvorbu databází velkého rozhlasu. Například se jedná o MySQL, Oracle a další.

Databáze je soubor dat (informací o objektech reálného světa), která spolu nějakým způsobem souvisí. Data je výraz pro údaje, používané pro popis nějakého jevu nebo vlastnosti pozorovaného objektu. Představují formu prezentace reálných objektů (znaky, symboly, obrázky, fakta, události), odrážejí tedy stav reality v určitém časovém okamžiku. Informace je zpráva, že nastal určitý jev. Vzniká přiřazením významu datům a existuje ve vztahu k příjemci. Slouží k informování o změnách ve vnímané realitě. S databázemi se v běžném životě setkáváme velmi často. Uvádíme běžné použití databází velkého rozsahu:

- databáze jízdních řádů,
- databáze státní správy,
- informační systémy bank, škol, úřadů
- nemocniční systémy evidence pacientů,
- databáze knihoven.

1.1. Systém řízení báze dat

Systém řízení báze dat (zkracováno na SŘBD či DBMS podle anglického database management system) je softwarové vybavení, které zajišťuje práci s databází, tzn. tvoří rozhraní mezi aplikačními programy a uloženými daty. Někdy se tento pojem zaměňuje s pojmem databázový systém. Databázový systém však je SŘBD dohromady s bází dat.

Entita

Libovolný objekt (osoba, zvíře, věc či jev) reálného světa, který je zachycen v datovém modelu. Entita musí být rozlišitelná od ostatních entit a existovat nezávisle na nich.

Data

Výraz pro údaje používané pro popis nějakého jevu nebo vlastnosti pozorovaného objektu. Data se získávají měřením nebo pozorováním, a lze je dělit na data spojitá a data atributivní. Data spojitá se přitom vztahují k nějaké spojité stupnici, zatímco data atributivní nikoliv

Informace

Informace jsou data, která nám přinášejí nové poznatky.

Záznamy a atributy

Řádky tabulky s hodnotami atributů pro jeden objekt (entitu), které se musí od sebe lišit. Atributy jsou sloupce tabulky, Atributy mají určen svůj konkrétní datový typ a doménu, což je množina přípustných hodnot daného atributu. Řádek je řezem přes sloupce tabulky a slouží k vlastnímu uložení dat.

Atributy, pole

Vlastnosti, které se u objektů (entit) sledují:

- tvoří sloupce tabulky
- mohou nabývat různých hodnot
- pole jsou určitého datového typu (číslo, text, datum, ...)

Primární klíč

Jednoznačně identifikuje záznam (řádek tabulky).
Je to takový atribut (pole), který má pro každou entitu jedinečnou hodnotu, např. rodné číslo, většinou je to pomocné pole s identifikačním číslem záznamu (ID).

Cizí klíč

Takový atribut, který je v jiné tabulce primárním klíčem.

Index

Jedná se o způsob řazení tabulky.

- pořadí záznamů se v tabulce během „života“ databáze nemění; index pomáhá k rychlému hledání dat v tabulce
- index vytvoří pomocný soubor s řazením tabulky podle určitého pole
- k jedné tabulce může být více indexů s řazením podle různých atributů
- primární klíč je vždy indexem

1.2. Struktura databáze

Nejpoužívanějšími databázemi jsou relační databáze. V nich jsou data ukládána v menších tabulkách, aby se zajistila minimální redundance (nadbytečnost) dat. Tabulky jsou vzájemně propojeny pomocí relací. Relace určují vztahy mezi tabulkami a zjišťují provázanost jednotlivých tabulek. Každá z těchto tabulek by měla obsahovat data týkající se pouze jednoho druhu objektu (např. tabulka objednávek, klientů, cen, zboží atd.). Pro vytvoření dobré databáze je nutné nejdříve navrhnout správnou strukturu jednotlivých tabulek. Tyto tabulky je pak nutné propojit pomocí relací. Tabulky tvoří základ celé struktury databáze. Základní pravidla pro návrh tabulek jsou následující:

- každá informace by měla být v databázi obsažena pouze jednou,
- každá tabulka by měla obsahovat informace o jednom typu objektu,
- při návrhu tabulek by se měl vzít do úvahy rozsah budoucích dat.

1.3. Databázová tabulka

V jedné tabulce by měly být informace o jednom typu objektu. Databázová tabulka je podobná běžné tabulce. Řádky obsahují záznamy (o jednom objektu) a sloupce označujeme položky či pole. Polem je totiž někdy zván i průsečík určitého řádku a sloupce, který obsahuje jedinou hodnotu (datový prvek).

Zaměstnanci

ID_zamestn	Jméno	Příjmení	Datum naro	Pohlaví	Telefonní čí	ID_funkce
1	Tomáš	Novák	28.4.1980	muž	723 123 456	F_03
2	Josef	Koblížek	15.3.1976	muž	728 452 123	F_01
3	Petra	Maková	5.2.1985	žena	724 556 115	F_02
4	Václav	Sýkorka	30.6.1970	muž		F_06
5	Denisa	Rosolová	12.12.1956	žena		F_05
6	Michal	Aspik	3.6.1976	muž		F_04
7	Dominik	Kokeš	14.2.1981	muž		F_04
8	Tereza	Železná	25.10.1978	žena		F_02
9	Vladimíra	Mimořádná	17.11.1989	žena		F_02
10	Jakub	Pekelník	5.12.1973	muž		F_05

V uvedené tabulce zaměstnanci tvoří řádky záznamy s informacemi o jednotlivých zaměstnancích. Sloupce představují pole, ve kterých vidíme vždy jeden typ dat (text, datum číslo). Každý sloupec (položka) má název, zvolený datový typ (např. text, číslo, ano/ne, datum a čas) a velikost. Lze přiřadit i další vlastnosti (formát, výchozí hodnotu, atd.). Více k této problematice ve videonávodech.

1.4. Primární klíč

Ve většině případů potřebujeme každý vložený záznam do tabulky jednoznačně identifikovat. K tomu slouží tzv. primární klíč. Primární klíč je takové pole, které je určeno pro zajištění jednoznačné identifikace jednotlivých záznamů v tabulce. Primární klíč je obvykle tvořen jedním pole (tzv. jednoduchý primární klíč), ale může být tvořen i více poli tabulky (tzv. složený primární klíč).

V každé tabulce může být pouze jeden primární klíč. S pomocí primárního klíče se rychleji vyhledávají informace a také se s ním vytváří relace s ostatními tabulkami. Hodnoty v poli primárního klíče musí být jedinečné pro každý uvedený záznam. Primární klíč je jedním z indexů. Pro rychlejší vyhledávání a třídění záznamů podle určitého pole tabulky je vhodné používat indexování polí (indexy). Pokud potřebujeme vyhledávat podle jiného sloupce tabulky než je primární klíč, tak mu nastavíme index. S pomocí nastaveného indexu se zrychlí řazení, vyhledávání a editace hodnot v tabulkách.

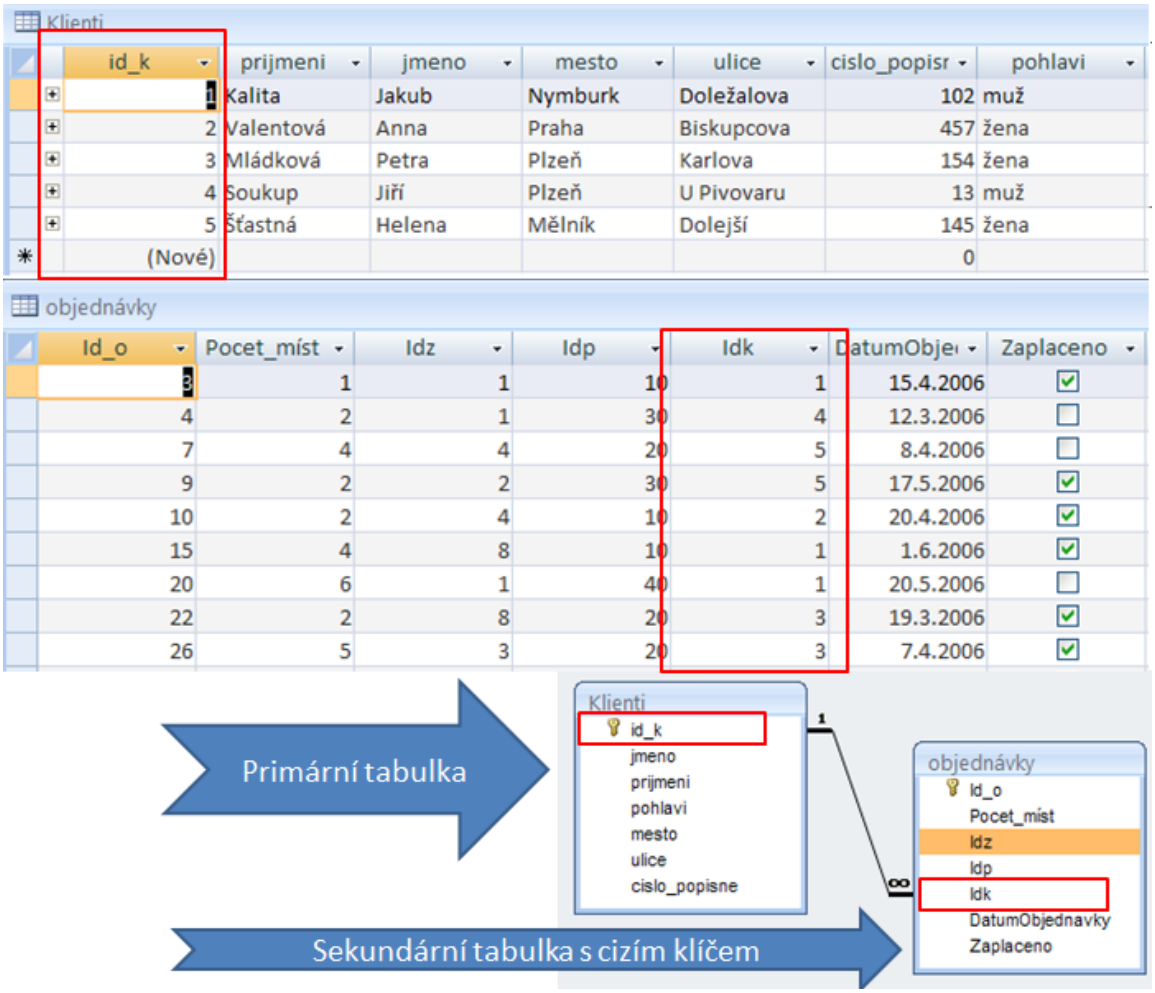
1.5. Relace

V relační databázi jsou jednotlivé tabulky propojeny pomocí relací. Hlavním účelem relací mezi tabulkami je omezení výskytu redundantních (nadbytečných) dat. Data by se neměla v různých tabulkách v rámci jedné databáze opakovat. Relace je postavena na vazbě stejných hodnot mezi unikátním polem (primárním klíčem) jedné tabulky a odpovídajícím polem jiné tabulky. Pro vytvoření relace mezi dvěma tabulkami je nutné mít v každé tabulce speciální pole. V jedné z těchto tabulek je to primární klíč. Takovou tabulku označujeme jako primární tabulku. Ve druhé tabulce vytváříme speciální pole pro účely relace. Toto pole označujeme jako cizí klíč, obsahuje hodnoty primárního klíče z jiné tabulky. Tabulka obsahující cizí klíč je označována jako sekundární tabulka.

1.6. Primární a sekundární tabulka

Pole primárního klíče obsahuje pouze unikátní hodnoty. Pole cizího klíče může obsahovat i stejné hodnoty. Pole primárního i cizího klíče musí obsahovat pro správné vytvoření relace stejné hodnoty. Na obrázku na následující straně uvádíme primární tabulku, která obsahuje záznamy o jednotlivých zákaznících. Primárním klíčem je první pole, které obsahuje číslo zákazníka. Tato tabulka je propojena se sekundární tabulkou, která je

tvořena záznamy o objednávkách zákazníků. Cizím klíčem v sekundární tabulce je pole obsahující číslo zákazníka, který objednávku provedl. Z uvedeného příkladu vidíme, že jednomu záznamu z primární tabulky odpovídá jeden nebo více záznamu ze sekundární tabulky. Jinými slovy jeden zákazník může provést jednu nebo více objednávek. Jedná se o typ relace 1:N.



1.7. Typy relací

Rozlišujeme tři základní typy relací:

- **Relace typu 1:1 (výjimečný)** – jednomu záznamu primární tabulky odpovídá právě jeden záznam v sekundární tabulce. Například jedné osobně je přiděleno právě jedno rodné číslo.
- **Relace typu 1:N (nejčastější)** – jednomu záznamu primární tabulky odpovídá jeden nebo více záznamů v sekundární tabulce. Tento příklad jsme popsali ve výše uvedeném příkladu se zákazníky a objednávkami.

- **Relace typu M:N (velmi často)** – jednomu nebo více záznamům primární tabulky odpovídá jeden nebo více záznamů v sekundární tabulce. Tyto relace řešíme pomocí spojovací tabulky, kterou pomocí dvou relací typu 1:N propojíme s původními tabulkami.

1.8. Referenční integrita

Referenční integrita udržuje neporušenost relací mezi tabulkami. Nedovolí nám vložit do sekundární tabulky záznam, který by neměl odpovídající záznam v primární tabulce. Dále si pohlídá si změnu hodnot cizího klíče při změně primárního klíče. V rámci referenční integrity je možné nastavit pravidla pro odstraňování záznamů.

1.9. Obsluha databáze

Profesionální databáze obsahují velké množství důležitých informací. Osoby, které pracují s takovou databází, můžeme rozdělit do několika skupin:

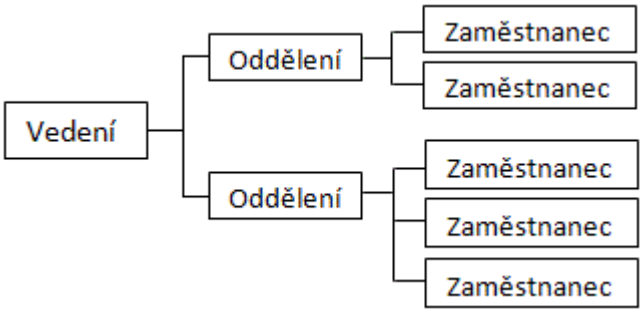
- databázový specialista navrhuje a vytváří profesionální databáze,
- uživatel zadává data, udržuje data a získává informace z databáze,
- správce databáze poskytuje uživatelům oprávnění přístupu k určitým datům, je odpovědný za obnovu databáze po její havárii nebo výskytu závažné chyby.

2.DATABÁZOVÉ MODELY

Z hlediska způsobu ukládání dat a vazeb mezi nimi dělíme databáze do základních typů:

2.1. Hierarchický model dat

Data jsou organizována do stromové struktury. Každý záznam představuje uzel ve stromové struktuře, vzájemný vztah mezi záznamy je typu rodič/potomek. Nalezení dat v hierarchické databázi vyžaduje navigaci přes záznamy směrem na potomka, zpět na rodiče nebo do strany na dalšího potomka. Největšími nevýhodami hierarchického uspořádání je složitá operace vkládání a rušení záznamů a v některých případech i nepřirozená organizace dat.

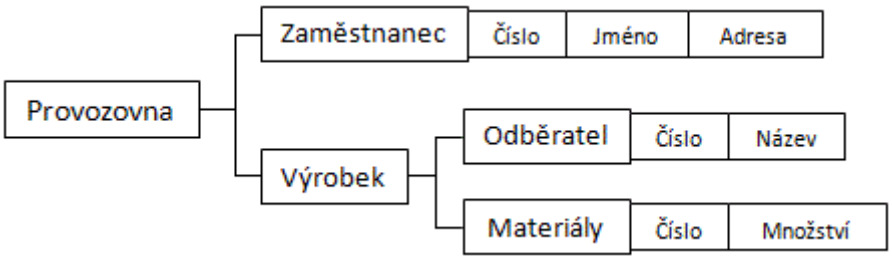


Obr. 1 - Hierarchický model

2.2. Síťový model dat

Síťový model dat je v podstatě zobecněním hierarchického modelu, který doplňuje o mnohonásobné vztahy (sety). Tyto sety propojují záznamy různého či stejného typu, přičemž spojení může být realizováno na jeden nebo více záznamů. Přístup k propojeným záznamům je přímý bez dalšího vyhledávání, k dispozici jsou operace: nalezení záznamu podle klíče, posun na prvního potomka v dílčím setu, posun stranou na dalšího potomka

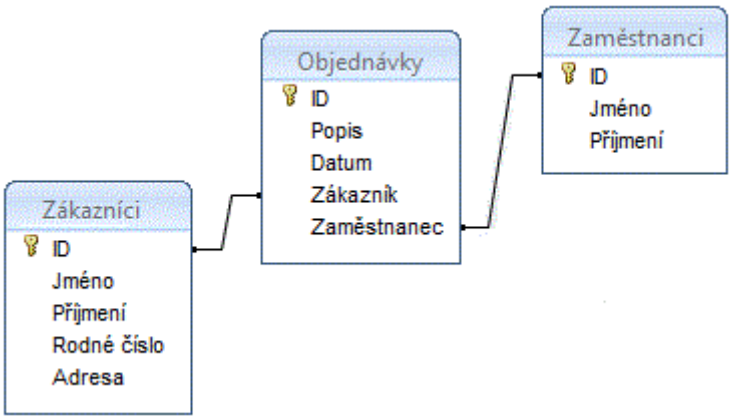
v setu, posun nahoru z potomka na jeho rodiče v jiném setu. Nevýhodou síťové databáze je zejména nepružnost a obtížná změna její struktury.



Obr. 2 - Síťový model

2.3. Relační model dat

Relační databázový model je z uvedených nejmladší a zároveň nejpoužívanější. V současnosti je nejčastěji využíván u komerčních SŘBD. Model má jednoduchou strukturu, data jsou organizována v tabulkách, které se skládají z řádků a sloupců. V těchto tabulkách jsou prováděny všechny databázové operace.



Obr. 3 - Relační model

Databáze dle relačního modelu musí splňovat tyto dvě vlastnosti:

- Databáze je chápana uživatelem jako množina relací a nic jiného.
- V relačním SŘBD jsou k dispozici minimálně operace selekce, projekce a spojení, aniž by se vyžadovaly explicitně předdefinované přístupové cesty pro realizaci těchto operací.

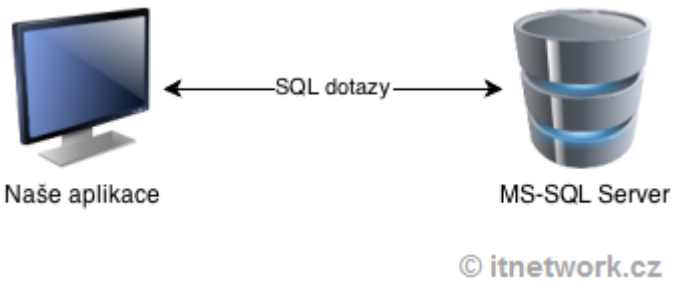
2.4. Objektové databáze

Kromě databází relačních existují i již zmíněné databáze objektové. Ty řeší problém neslučitelnosti objektového a relačního přístupu. Poskytují stejný komfort, jako ORM, ale vnitřně není třeba data převádět do tabulek, ukládají se rovnou jako objekty. Teoreticky neexistuje výkonnostní ani jiný důvod, proč by neměly nahradit databáze relační. V praxi se ale bohužel téměř nepoužívají a můžeme jen doufat, že se to časem změní.

2.5. Připojená aplikace

Přístup připojené aplikace použijeme ve chvíli, kdy potřebujeme data často v reálném čase číst nebo měnit. Pomocí tříd DataReader, Command a Connection posíláme databázi přímo příkazy v jazyce SQL a dostáváme výsledky.

Situaci je znázorněna na obrázku:



2.6. Objektové databáze

Univerzální databázové systémy (database management systems – DBMS) v současné době představují nejrozšířenější prostředek pro uchovávání a manipulaci s daty komerčních aplikací. Zajišťují relativně jednoduchou správu rozsáhlých databází, integritu, efektivní vyhledávání a zpracovávání informací, odolnost proti poruchám a výpadkům, souběžný víceuživatelský přístup a další výhody.

Relační datový model – nejběžnější model používaný v dnešních databázových systémech – omezuje strukturu a vztahy uchovávaných dat na pouhou množinu tabulek nad předdefinovanou množinou základních datových typů. Nespornou výhodou tohoto modelu je jednoduchost a v důsledku toho i snadná standardizace a přenositelnost. Nevýhodou však je rozdílnost schématu reálných dat od vnitřního tabulkového modelu databáze. Veškeré vztahy mezi daty lze reprezentovat pouze tabulkami, což u aplikace s komplikovanějším datovým modelem vede k množství tabulek vzájemně provázaných pomocnými odkazy, tzv. klíči. Tím dochází ke ztrátě přehlednosti, databáze se stává hůře spravovatelnou a budoucí změny v datovém modelu aplikace nutí programátory k

citelným zásahům do jejich tabulkové reprezentace. Relační databáze jsou a zřejmě i nadále zůstanou hlavním prostředkem pro správu dat komerčních aplikací, které jsou charakteristické velkým objemem údajů s jednoduchou strukturou. Řada aplikací, například z oblasti designu, multimédií, geografických systémů apod., však potřebuje takový datový model, který umožní lepší korespondenci mezi složitými reálnými daty a jejich reprezentací v databázovém systému. Tímto modelem je objektový model dat. Objektový model dat v databázových systémech vychází ze známých principů objektově orientovaného modelování a programování. Je však dále obohacen o techniky perzistence, reprezentace vztahů, dotazování, transakčního přístupu apod.

2.7. Základy objektové orientace. Objekty a třídy

Objektově orientovaný model je založen na dekompozici informací z reálného světa na tzv. objekty. Objektem se rozumí každá (i strukturovaná) entita, která je jednoznačně a nezávisle identifikovatelná v rámci určitého kontextu okolního světa. Objekt tak má jednoznačnou identitu, každé dva i jinak datově shodné objekty jsou vzájemně odlišitelné. Identita objektu je určena identifikátorem (object identifier – oid), který je generovaný systémem, unikátní, neměnný po dobu existence objektu, skrytý pro programátora i koncového uživatele. Objekty jsou charakterizovány pomocí tříd. Třída je abstraktní popis objektu, určuje datové složky objektu a operace (nazývané metody), které lze nad objektem provádět. Každý objekt je instancí nějaké třídy, od jedné třídy je možné instanciovat obecně neomezený počet strukturálně shodných objektů. Dále rozlišujeme rozhraní třídy.

2.8. Literály

Kromě objektů se v rámci objektově orientovaného modelu zavádí i pojem literálu. Literál je datová entita určitého datového typu, která však na rozdíl od objektu nemá vlastní identitu. Literály se obvykle vyskytují jako datové atributy objektů. Množina operací nad datovým typem literálu je pevně stanovená, není možné ji měnit. S objekty a literály souvisí pojem proměnlivosti (mutability). Proměnlivost je chápána jako schopnost měnit data při zachování identity – v tomto smyslu jsou objekty proměnlivé, protože je možné měnit hodnoty jejich datových složek a přitom si ponechávají původní identitu. Naproti tomu literály proměnlivé nejsou.

2.9. Operace

Existuje několik základních typů operací nad objekty. Každý objekt má jeden či více konstruktorů. Účelem konstruktoru je inicializovat objekt v okamžiku jeho vytvoření. Dále je součástí každého objektu tzv. destruktor. Destruktor je volán v okamžiku rušení objektu a jeho cílem je provést úklid objektu před jeho odstraněním. Důležitou operací nad objekty je kopírování. Rozlišují se tzv. mělké (shallow) a hluboké (deep) kopie. V případě mělké kopie dojde ke zkopírování atributů objektu, avšak všechny případné odkazy na jiné objekty dále ukazují na stejné objekty jako v originálním objektu. Naproti tomu u hluboké kopie dojde nejen ke zkopírování atributů kopírovaného objektu, ale současně se vytvoří i kopie objektů, na které se původní objekt odkazoval. Dalšími typickými operacemi jsou metody pro zjišťování a přiřazování hodnot atributů, metody provádějící výpočty a manipulace s atributy objektu, metody produkující uživatelský výstup atd.

Objektový datový model jsou data uložené v objektové struktuře. Jde většinou o objektovou mezivrstvu mezi kódem a databází, do které se nasypou data, s kterými pak aplikace pracuje a nezatěžuje dotazy DB server.

Objektově relační datový model

Objektově-relační datový model je klasická tabulková databáze rozšířená o **abstraktní datové typy** (ADT). ADT jsou uživatelem definované typy skládající se ze základních datových typů databáze. Čímž porušuje 1NF??

- Objektové rysy jsou dnes implementovány ve všech velkých SŘBD.
- Objektové typy a jejich metody jsou uloženy spolu s daty v databázi, programátor tedy nemusí vytvářet podobné struktury v každé aplikaci.
- Programátor může přistupovat k množině objektů, jako by se jednalo o jeden objekt.
- Objekty mohou jednoduše reprezentovat vazby, kdy jedna entita se skládá z jiných entit (bez nutnosti použít vazeb).
- Metody jsou spouštěny na serveru – nedochází k neefektivnímu přenosu dat po síti.

Objektové datové typy

mohou obsahovat:

- data – atributy
- operace – metody
- Specifické pro ORŠRBD je to, že se na data můžeme dívat jak z relačního (záznamy, operace) tak objektového pohledu (objekty, metody).

Typy metod

- **Členské metody** – jsou volány nad konkrétním objektem.
- **Statické metody** – jsou volány nad datovým typem.
- **Konstruktor** – pro každý objektový typ je definován implicitní konstruktor.

3.INTEGRITA DATABÁZE

Integrita databáze znamená, že data v ní uložená jsou konzistentní vůči definovaným pravidlům. Lze zadávat pouze data, která vyhovují předem definovaným kritériím (např. musí respektovat datový typ nastavený pro daný sloupec tabulky, či další omezení hodnot přípustných pro daný sloupec). K zajištění integrity slouží integritní omezení. Jedná se o nástroje, které zabrání vložení nesprávných dat či ztrátě nebo poškození stávajících záznamů v průběhu práce s databází. Například je možné zajistit mazání dat, která již ztratila svůj význam - například smažeme-li uživatele, odstraní se i zbytek jeho záznamů v ostatních databázových tabulkách.

3.1. Druhy integritních omezení

- **Entitní integritní omezení** – povinné integritní omezení, které zajišťuje úplnost primárního klíče tabulky (zamezí uložení dat, jež by v těchto polích byla stejná jako v nějakém jiném řádku tabulky)
- **Doménová integritní omezení** – zajišťují dodržování datových typů/domén definovaných u sloupců databázové tabulky
- **Referenční integritní omezení** – zabývají se vztahy dvou tabulek, kde jejich relace je určena vazbou primárního a cizího klíče
- **Aktivní referenční integrita** – definuje činnosti, které databázový systém provede, pokud jsou porušena některá pravidla

3.2. Dodržování integritních omezení

V zásadě existují tři způsoby, jak zajistit dodržování integritních omezení:

- Umístění jednoduchých mechanismů pro dodržování integritních omezení **na straně databázového serveru**
    - jedná se o nejlepší způsob z hlediska ochrany dat
    - uživateli však obvykle přináší delší odezvu systému a nelze vždy zajistit jejich přenositelnost na jiný databázový systém
- Umístění ochranných mechanismů **na straně klienta**
    - pro komfort a nezávislost na databázovém systému je nejlepší volbou nutnost kontrolních mechanismů pro každou operaci může způsobit chyby u aplikací a v případě většího počtu aplikací je potřeba je opravit na více místech

- Samostatné programové moduly **na straně serveru**
    - v moderních databázových systémech jsou pro tento účel implementovány tzv. triggery = samostatné procedury, které lze spouštět automatizovaně před a po operacích manipulujících s daty
    - tento způsob umožňuje implementaci i složitých integritních omezení
    - nevýhody opět přináší provádění na serveru i velmi omezená možnost přenesení na jiný databázový systém

Ideálním řešením je kombinace předchozích v závislosti na konkrétních podmínkách. Kontroly integritních omezení se zpravidla provádějí po každé provedené operaci, což snižuje nároky na server. Není nutno nijak zaznamenávat, které kontroly mají být provedeny později. Složitější integritní omezení však vždy nelze takto ověřit, proto je možné kontrolovat dodržení pravidel až po dokončení celé transakce.

3.3. Vztahy mezi tabulkami

Relace slouží ke svázání dat, která spolu souvisejí a jsou umístěny v různých databázových tabulkách.

V zásadě rozlišujeme čtyři typy vztahů:

- mezi daty v tabulkách není **žádná spojitost**, proto nedefinujeme žádný vztah

- **1:1** (záznamu odpovídá právě jeden záznam v jiné databázové tabulce a naopak)

- **1:N** (přiřazuje jednomu záznamu více záznamů z jiné tabulky)
    - jedná se o nejpoužívanější typ relace, jelikož odpovídá mnoha situacím v reálném životě

- **M:N** (umožňuje několika záznamům z jedné tabulky přiřadit několik záznamů z tabulky druhé)
    - tento vztah bývá z praktických důvodů nejčastěji realizován kombinací dvou vztahů 1:N a 1:M, které ukazují do pomocné, tzv. vazební tabulky složené z kombinace obou použitých klíčů

3.4. Normální formy

Pod pojmem normalizace rozumíme proces zjednodušování a optimalizace navržených struktur databázových tabulek. Hlavním cílem je navrhnout databázové tabulky tak, aby obsahovaly minimální počet redundantních dat. Správnost navržení struktur lze ohodnotit některou z následujících normálních forem.

Nultá normální forma (0NF)

- tabulka obsahuje alespoň jeden sloupec (atribut), který může obsahovat více druhů hodnot

První normální forma (1NF)

- všechny sloupce tabulky nelze dále dělit na části nesoucí nějakou informaci -> prvky musí být atomické
- jeden sloupec neobsahuje složené hodnoty.

Druhá normální forma (2NF)

- tabulka obsahuje pouze sloupce, které jsou závislé na celém klíči

Třetí normální forma (3NF)

- tabulka je ve třetí normální formě, pokud neexistují žádné závislosti mezi neklíčovými sloupci

Čtvrtá normální forma (4NF)

- sloupce v tabulce popisují pouze jeden fakt nebo jednu souvislost

Pátá normální forma (5NF)

- přidáním libovolného nového sloupce by se tabulka rozpadla na více tabulek

3.5. Databázová integrita

Integrita databáze znamená, že databáze vyhovuje zadaným pravidlům – integritním omezením. Tato integritní omezení jsou součástí definice databáze, a za jejich splnění zodpovídá systém řízení báze dat.

Integritní omezení se mohou týkat jednotlivých hodnot vkládaných do polí databáze (například známka z předmětu musí být v rozsahu 1 až 5), či může jít o podmínku na kombinaci hodnot v některých polích jednoho záznamu (například datum narození nesmí být pozdější než datum úmrtí). Integritní omezení se může týkat i celé množiny záznamů daného typu – může jít o požadavek na unikátnost hodnot daného pole či kombinace polí v rámci celé množiny záznamů daného typu, které se v databázi vyskytují (například číslo průkazu v záznamech o osobách).

Velmi často používaným integritním omezením v relačních databázích je tzv. referenční integrita. Jedná se o požadavek, aby pro pole záznamu, jež má obsahovat odkaz na jiný záznam někde v databázi, takový odkazovaný záznam skutečně existoval, tedy aby takový odkaz nevedl „do prázdna“ a nejednalo se o tzv. databázového sirotka.

Integritní omezení

Je třeba zajistit, aby se do databáze dostaly jen takové záznamy, které odpovídají vztahům v reálném světě (např. atribut věk by neměl nabývat záporných hodnot). K ošetření takových případů se používají integritní omezení, která slouží ke stanovení určitého rozmezí hodnot, jakých může konkrétní atribut nabývat. Integritní omezení specifikují, jaká omezení a vnitřní vztahy musí splňovat data v databázi.

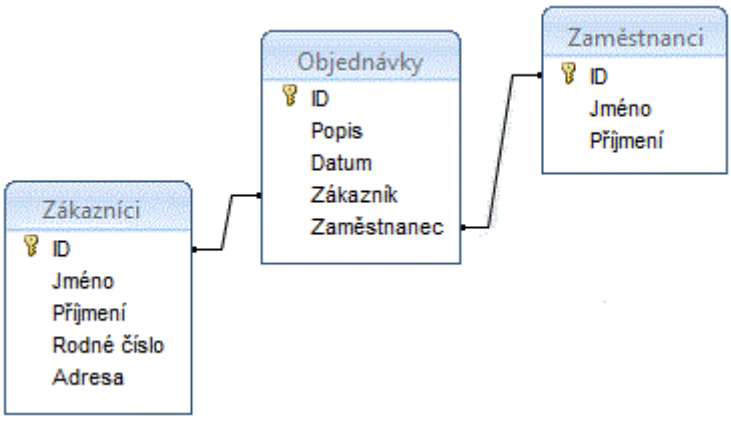
Relace, které vyhovují integritním omezením, jsou přípustné.

Relační databázový systém

- musí plnit tyto tři základní funkce:
    - umožnit definovat typ dat
    - umožnit pracovat s daty
    - musí umožňovat efektivní správu a analýzu uložených dat

- systém by měl obsahovat prostředky umožňující provést požadované činnosti s daty - např. řazení dat, filtrování, výpočty s daty, řídit správu dat
- systém kontroluje zejména přístup k datům, tedy kdo je oprávněn k datům přistupovat a kdo může provádět jejich aktualizaci
- systém řídí sdílení dat mezi více uživateli

4.RELAČNÍ DATABÁZOVÝ MODEL



Obr. 3 - Relační model

Databáze dle relačního modelu musí splňovat tyto dvě vlastnosti:

- Databáze je chápana uživatelem jako množina relací a nic jiného.
- V relačním SŘBD jsou k dispozici minimálně operace selekce, projekce a spojení, aniž by se vyžadovaly explicitně předdefinované přístupové cesty pro realizaci těchto operací.

4.1. 12 pravidel pro relační SŘBD

1. Informační pravidlo:
Všechny informace v relační databázi jsou vyjádřeny explicitně na logické úrovni jediným způsobem - hodnotami v tabulkách.

2. Pravidlo jistoty:
Všechna data v relační databázi jsou zaručeně přístupná kombinací jména tabulky s hodnotami primárního klíče a jménem sloupce.

3. Systematické zpracování nulových hodnot:
Nulové hodnoty jsou plně podporovány relačním SŘBD pro reprezentaci informace, která není definována a to nezávisle na datovém typu.

4. Dynamický on-line katalog založený na relačním modelu:
Popis databáze je vyjádřen na logické úrovni stejným způsobem jako zákaznická data, takže autorizovaný uživatel může aplikovat stejný relační jazyk ke svému dotazu jako uživatel při práci s daty.

5. **Obsáhlý datový podjazyk:**
Relační systém může podporovat několik jazyků a různých módů použitých při provozu terminálu. Nicméně musí být nejméně jeden příkazový jazyk s dobře definovanou syntaxí, který obsáhle podporuje definici dat, definici pohledů, manipulaci s daty jak interaktivně, tak programem, integritní omezení, autorizovaný přístup k databázi, transakční příkazy apod.

6. **Pravidlo vytvoření pohledů:**
Všechny pohledy, které jsou teoreticky možné, jsou také systémem vytvořitelné.

7. **Schopnost vkládání, vytvoření a mazání:**
Schopnost zachování relačních pravidel u základních i odvozených relací je zachována nejen při pohledu na data, ale i při operacích průniku, přidání a mazání dat.

8. **Fyzická datová nezávislost:**
Aplikační programy jsou nezávislé na fyzické datové struktuře.

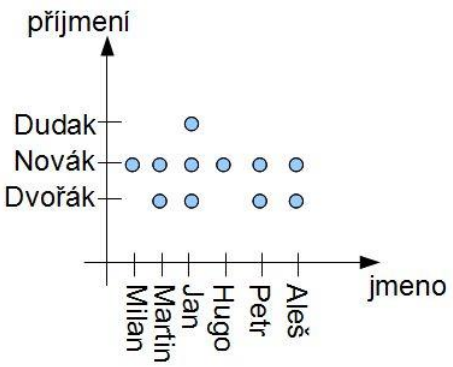
9. **Logická datová nezávislost:**
Aplikační programy jsou nezávislé na změnách v logické struktuře databázového souboru.

10. **Integritní nezávislost:**
Integritní omezení se musí dát definovat prostředky relační databáze nebo jejím jazykem a musí být schopna uložení v katalogu a nikoliv v aplikačním programu.

11. **Nezávislost distribuce:**
Relační SŘBD musí být schopny implementace na jiných počítačových architekturách.

12. **Pravidlo přístupu do databáze:**
Jestliže má relační systém jazyk nízké úrovně, pak tato úroveň nemůže být použita k vytváření integritních omezení a je nutno vyjádřit se v relačním jazyce vyšší úrovně.

4.2. Relační datový model



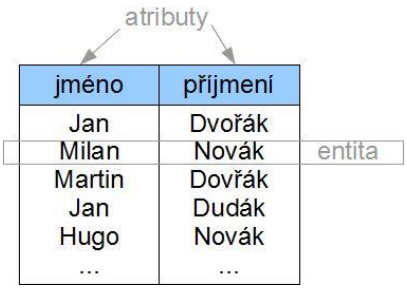
Tabulka s dvěma atributy znázorněná jako relace

Relační datový model je způsob uchování dat v tabulkách. Relační se mu říká proto, že se tabulka se definuje přes relaci.

Relace je tedy v podstatě tabulka. Definována je jako podmnožina kartézského součinu domén.

Relace na obrázku vpravo je tedy podmnožina součinu množin {Dudák, Novák, Dvořák} x {Milan, Martin, Jan, ..., Aleš}

Na rozdíl od **matematické relace** se ta databázová mění v čase (přidáváním a odebíráním prvků relace). Kromě základních množinových operací se u databázové relace setkáme s operaci **selekce** - výběr řádků a **projekce** -výběr sloupců.



Relace z předchozího obrázku zobrazena tabulkou

Doména je množina všech hodnot, kterých může nabývat atribut. Jinak řečeno obor hodnot atributu. V praxi je doména dána integritním omezením (IO).

*Doména atributu Přijmení z obrázků je množina {Dudak, Novák, Dvořák}. *pozn.*

Atribut je vlastnost entity. Z pohledu tabulky jde o sloupec.

Relační schéma můžeme chápat jako strukturu tabulky (atributy a domény).

Příklad pro tabulku (relaci) Učitel:

Atributy: ID, jméno, příjmení, funkce, kancelář
Domény:
D1 - tři písmena z příjmení, tří cifry pořad. čísla
D2 - kalendář jmen
D3 - množina příjmení
D4 - množina funkcí (asistent, vědec, učitel,...)
D5 - A101, A102, ... A160

Relační schéma: Učitel (ID, jméno, příjmení, funkce, kancelář)

Relace: Učitel = {(nov001,lukas,novak,vědec,A135),(kom123,jan,komensky,učitel,A111),...}

Definováno je jako R(A,f), kde A je množina atributů (A1,A2,...An) a funkce f(Ai)=Di přiřazuje atributu doménu.

4.3. Vlastnosti relačního datového modelu

Z definice relace vyplývají tyto jejich tabulkové vlastnosti:

- homogenita sloupců (prvky domény)
- každý údaj (hodnota atributu ve sloupci) je atomickou položkou
- na pořadí řádků a sloupců nezáleží (jsou to množiny prvků/atributů)
- každý řádek tabulky je jednoznačně identifikovatelný hodnotami jednoho nebo několika atributů (primárního klíče)

4.4. Vazby v relačním modelu

Obecně se vazby v relačním modelu realizují pomocí další relace (tabulky). Jedná se o tzv. **vazební tabulku**. Ta **obsahuje** ty atributy relací (tabulek, které se vazby účastní), které jednoznačné identifikují jejich entity - **primární klíče**. Obsahuje-li tabulka atribut, který slouží jako primární klíč v jiné tabulce, pak obsahuje **cizí klíč**. Vazební tabulka tedy obsahuje cizí klíče.

Na obrázku dole máme zobrazenou relaci Učitel s primárním klíčem *idu* a relaci Předmět s primárním klíčem *cp*. K vyjádření vztahu *Učitel UČÍ Předmět* byla vytvořena nová relace Učí, která obsahuje dva cizí klíče (*idu* odkazující na učitele a *cp* odkazující na entitu

předmětu). Nyní tedy můžeme přes vazební tabulku pospojovat učitele s předměty, podle toho kdo co učí.

A proč jsme do tabulky Učitel nepřidali jen atribut předmět? Jednoduše proto, že učitel může učit více předmětů. Mezi učitelem a předmětem je vazba M:N. Takže ani kdybychom přidali do tabulky Předmět atribut učitel, bychom tento vztah nevyřešili (předmět může být učen více učiteli). Pro vztah 1:N by to však šlo a v praxi se to tak i dělá. Takže **vazební tabulka je nutná jen k realizací vztahů M:N**.

Učitel

idu	jméno	příjmení
dvo01	Jan	Dvořák
kov01	Marie	Kovářová
kov02	Martin	Kovadlina
chy01	Jana	Chtrá
mal01	Libuše	Malinová

Učí

idu	cp
dvo01	3
dvo01	1
kov01	2
chy01	1
mal01	5

Předmět

cp	nazev
1	matematika
2	anglický j.
3	fyzika
4	biologie
5	český j.

Ukázka vazební tabulky pro vztah Učí mezi tabulkami Učitel a Předmět. Vztah je M:N, tedy že jeden učitel může učit N předmětů a jeden předmět může být učen M učiteli.

5.ZÁSADY SGL – VYTVÁŘENÍ DOTAZŮ

5.1. Úvod do SQL – popis a vlastnosti

SQL - *Structured Query Language* (Strukturovaný dotazovací jazyk) - je obecný nástroj pro manipulaci, správu a organizování dat uložených v databázi počítače. Je v prvé řadě určen uživatelům, i když jej v mnoha směrech využívají i tvůrci aplikací. Je adaptovatelný pro jakékoliv prostředí.

Název tohoto nástroje je sice stručný, není však výstižný. SQL totiž není pouze dotazovací jazyk, s jeho pomocí je možno také definovat data, tedy strukturu tabulky, naplňovat sloupce tabulky (pole záznamů) daty a definovat vztahy a organizaci mezi položkami dat. Dále umožňuje řízení přístupu k datům, tedy udělování a odebírání přístupových oprávnění na různých úrovních, čímž chrání data před náhodným nebo úmyslným zničením, neautorizovaným čtením nebo manipulaci s nimi, dále také umožňuje sdílené využívání dat a zajišťuje hladký průběh činností, přistupuje-li k datům více uživatelů současně.

SQL je specializovaný programovací jazyk, který se používá ve vhodném prostředí buď uživatelsky nebo interaktivně k okamžitému řešení úloh (nejčastěji dotazy), nebo se jeho příkazy vkládají do hostitelského jazyka. SQL není však plnohodnotným samostatným programovacím jazykem, např. proto, že se v něm ve většině implementací nenachází řídicí programové konstrukce a další požadované prvky, které by měl obsahovat každý obecný programovací jazyk. SQL je tedy standardizovaný nástroj pro práci s relačními databázemi. Nepředstavuje databázový systém, ale různě integrovanou součást systému řízení bází dat.

SQL je především interaktivní dotazovací jazyk - umožňuje získat odpovědi i na velmi komplikované dotazy téměř ihned. Je nástrojem neprocedurálním, s množinovým přístupem k datům a je jazykem standardizovaným, je srozumitelný, protože chápe data v podobě tabulek, což je snadno pochopitelné i uživatelům. Pracuje s relačními databázemi, ve kterých se uživatel dívá na data v podobě soustavy provázaných tabulek. Každá tabulka představuje množinu dat, která je uspořádaná v řádcích (záznamech) a sloupcích (položkách). Na hodnotu dat se uživatel odkazuje jako na prvek v matici.

V převážné většině případů je výsledkem úlohy popsané v SQL nějaká množina dat z jedné nebo více tabulek, tzv. tabulka výsledků, která nemusí být vždy konečným produktem. Může sloužit jako množina vstupních údajů pro další zpracování. Např. pro výtisk etiket nebo vykreslení grafu, atd.

SQL může sloužit jako “společná řeč“, zejména při provozu v sítích, na kterých se používají různé databázové produkty.

Jazyk SQL se používá také k vytváření náhledů (dotazů). Náhledy SQL umožňují vytvořit pro různé uživatele různé pohledy na struktury tabulek a na data. Každý uživatel tak vidí pouze ta data, která má vidět. Přitom data vidí uživatel opět v podobě jednoduché tabulky, i když ve skutečnosti data pocházejí z různých tabulek. Data zobrazovaná v náhledech jsou dynamická, tzn. změní-li se data v tabulkách (databázových souborech), změní se také data, které zobrazuje náhled. Také naopak. Pracuje-li se s aktualizačním náhledem (speciální typ náhledu, který umožňuje nejen data prohlížet, ale i přidávat a aktualizovat) a změní se v něm data, promítnou se změny do patřičných tabulek (databázových souborů).

Klíčovým pojmem v jazyku SQL je ***příkaz***. Každý příkaz začíná klíčovým slovem. Slovo vyjadřuje orientačně, jakou činnost daný příkaz provádí. Za klíčovým slovem následuje jedna nebo více volitelných klauzulí, které blíže specifikují povahu vykonávané činnosti, nebo určují data, s nimiž má příkaz pracovat.

Každá klauzule začíná klíčovým slovem, jako jsou např. FROM nebo WHERE. Některé klauzule jsou povinné, jiné volitelné. Každá implementace SQL používá kromě standardních klauzulí, daných konvencemi ANSI/ISO, své vlastní klauzule, někdy se i činnost standardních klauzulí mírně či více liší.

Jména objektů v jazyku SQL mají ve standardu délku 1-18 znaků, z toho první znak musí být písmeno, a jméno nesmí obsahovat mezery nebo interpunkční znaky. Při udávání jména je někdy potřeba jméno kvalifikovat, vyskytují-li se stejná jména, a není jednoznačně jasné, na který objekt se jméno odkazuje. Kvalifikace jména se provádí pomocí kvalifikátoru ´.´ (tečka). Často se kvalifikace používá se jmény tabulek:

```
<Owner>. <Table_name>
```

nebo při kvalifikaci sloupců, což se v databázových systémech používá velmi často:

```
<Tablename>. <Columnname>
```

obecně se v SQL připouští i kvalif. víceúrovňová:

```
<Owner>. <Table_name>. <Column_name>
```

5.2. Databáze SQL a dotazy

Úvodní do příkazů v SQL databázích

Základem je představit na jednoduchých příkladech základní příkazy jazyka SQL. Informace pak můžeme použít při programování ve Visual Basicu, Delphi, programování webových stránek. Základní úvod co je databáze je zmíněn v článku databáze úvod. Pokud již teorií databázi rozumíte, můžete se podívat jak vytvořit databázi v programu Microsoft Access.

Seznam SQL příkazů

Mezi nejdůležitější příkazy patří: SELECT, INSERT, DELETE, CREATE, FROM, WHERE a mnohé další v následujících kapitolách si na příkladech jednotlivé příkazy představíme. Nejprve si ale uděláme jednoduchou tabulku, na které si příkazy představíme (to aby bylo jasnější, co který ten příkaz dělá)

Vytvoření testovací tabulky

zamestanci : Tabulka

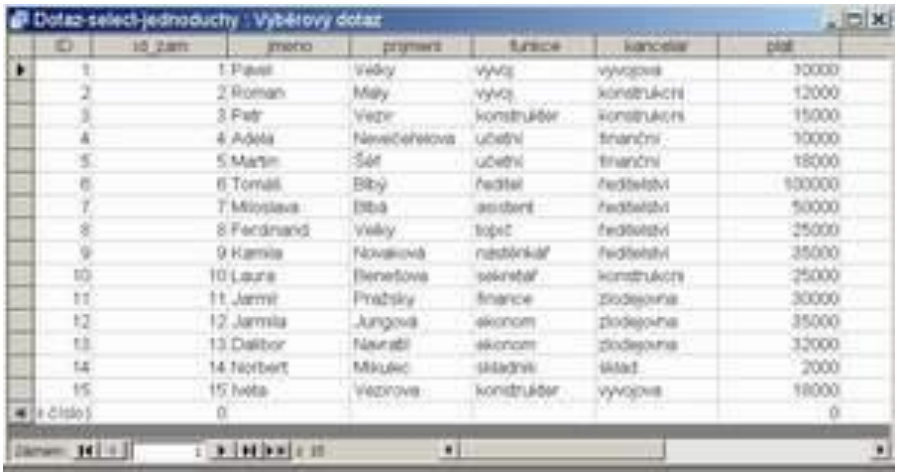
ID	id_zam	jmeno	prijmeni	funkce	kancelar	plat	vek
1	1	Pavel	Velky	vyvoj	vyvojova	10000	20
2	2	Roman	Maly	vyvoj	konstrukcni	12000	25
3	3	Petr	Vezir	konstrukter	konstrukcni	15000	35
4	4	Adela	Nevečeřelova	učetni	finanční	10000	20
5	5	Martin	Šéf	učetni	finanční	18000	23
6	6	Tomáš	Blbý	ředitel	ředitelství	100000	35
7	7	Miloslava	Blbá	asistent	ředitelství	50000	16
8	8	Ferdinand	Velky	topič	ředitelství	25000	66
9	9	Kamila	Novaková	nástěnkář	ředitelství	35000	25
10	10	Laura	Benešova	sekretař	konstrukcni	25000	55
11	11	Jarmil	Pražsky	finance	zlodejovna	30000	36
12	12	Jarmila	Jungová	ekonom	zlodejovna	35000	28
13	13	Dalibor	Navratil	ekonom	zlodejovna	32000	19
14	14	Norbert	Mikulec	skladnik	sklad	2000	48
15	15	Iveta	Vezirova	konstrukter	vyvojova	18000	47
(Číslo)	0					0	0

Záznam: 1 z 15

Abychom si ujasnili pojmy z SQL jazyky, je vhodné si to testovat na nějaké jednoduché tabulce. Poté, co již je plně chápeme, je můžeme nasadit na tabulku s 10.000 řádky. Tuto tabulku můžeme vytvořit v MySQL, FoxPro, Access či nějakém jiném databázovém programu.

Příkaz SELECT

Důležitý příkaz, Použijeme naši vytvořenou tabulku a otestujeme si na ní aplikace tohoto příkazu. Nejjednodušší forma příkazu je tato:




```
SELECT *
FROM employees;
```

Vypíše obsah celé tabulky "zamestananci"

```
SELECT the name
FROM employees;
```

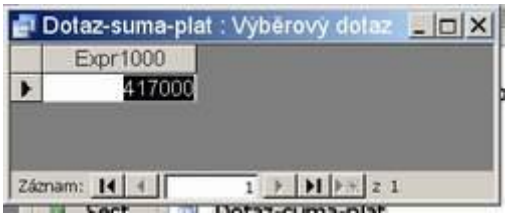
Vypíše obsah sloupce "jmeno" z tabulky "zamestananci"

```
SELECT DISTINCT function
FROM employees;
```

Vypíše obsah sloupce "funkce" z tabulky zamestanci, ale zobrazi jen položky bez duplicit (takže i když vyvojaře máme 4x vypíše se jen jednou)

```
SELECT AVG(plat)
FROM zamestanci;

SELECT SUM (flat)
FROM employees;
```

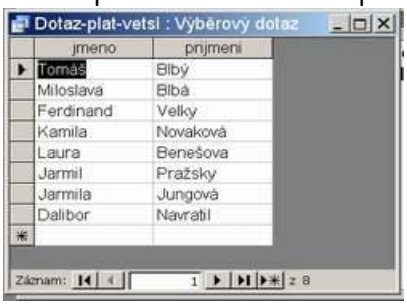


AVG(sloupec) vypočte půměr z daného sloupce, SUM(sloupec) vypočte sumu z hodnot v daném sloupci, případně můžeme i v samotném dotazu provádět výpočty.

```
SELECT SUM (PLAT + 500)
FROM employees;
WHERE
```

WHERE

Až budete mít 10.000 řádků, budete chtít omezit výpočty podle nějakých kritérií. Toho lze docílit příkazem WHERE. Spolu s použitím nějakých matematických symbolů (<,>,<=,>=, ..)




```
SELECT name
FROM employees
WHERE pay> 20000;
```

Vypíše jména zaměstnanců, jejichž plat přesahuje 20 000 Kč

Obecná syntaxe jednoduchého dotazu

SELECT *seznam sloupců*
FROM *tabulka*
WHERE *vyhledávací podmínky*
ORDER BY *sloupce řazení*

Seznam sloupců je buď seznam sloupců oddělených čárkou, nebo znak * pro výběr všech sloupců.

Název sloupce je buď pouze název sloupce nebo **název_tabulky.název_sloupce**. Také je možné použít zápis **tabulka.*** pro výběr všech sloupců z tabulky.

***Tabulka* může být cokoliv z:**

- relace (reálná tabulka v databázi)
- výsledek jiného SELECT dotazu
- pohled
- výsledek operátoru JOIN (virtuální tabulky).

Vyhledávací podmínka je podmínka, které musí být splněna pro každý záznam, který se vypíše ve výsledcích dotazu. Samozřejmě pokud chceme nějaký záznam vypsat, tak musí nejprve existovat ve zdrojové tabulce. Část WHERE je tedy *omezující* podmínka, pokud není uvedena, tak se vypisují všechny řádky z tabulky.

Sloupce řazení je seznam sloupců oddělených čárkou, podle kterých bude výsledná tabulka seřazena, první sloupec je primární kritérium řazení, druhý sloupec je sekundární řazení - pokud nelze rozhodnout podle prvního sloupce, ...

Příklad 1

Zadání: Vyberte z databáze všechny Radky a Tomáše.

Řešení 1

```
SELECT name, surname, nickname FROM person
WHERE name = 'Radek' ORname = 'Tomas'
```

SELECT jmeno, prijmeni, rezdivka **FROM** osoby **WHERE** jmeno='Radek' **OR**jmeno='Tomáš'

Můžeme použít běžných logických operátorů **AND** a **OR**, na většině databázových serverů se dají zapisovat také jako **&&** a **||**. Pro porovnání řetězců se dá použít obyčejné =, je však potřeba myslet na několik věcí. Podle nastavení serveru záleží nebo nezáleží na velikosti písmen, výchozí nastavení (a tedy častější) je, že na velikosti písmen nezáleží. Na serveru Akela.mendelu.cz na velikosti písmen záleží. Dále podle nastavení collation serveru (a databáze) záleží nebo nezáleží na diakritice. Na serveru Akela.mendelu.cz opět na diakritice záleží, čili jméno je nutné psát přesně tak jak je uložené v databázi.

Řešení 2

SELECT jmeno, prijmeni, prezdivka **FROM** osoby **WHERE** jmeno **IN**('Radek','Tomáš')

Alternativa k předchozímu dotazu je použití operátoru **IN**. IN je množinový operátor, který testuje, zda se hodnota nachází v zadané množině, tedy pro každý řádek se vezme hodnota sloupce *jmeno* a testuje se, jestli je v množině ('Radek', 'Tomáš'). Množina může být definovaná buď přímým výčtem hodnot, nebo dalším SQL dotazem.

Příklad 2

Zadání: Vyberte z databáze všechny osoby, které bydlí v Praze.

Řešení – krok 1

```
SELECT id_address, FROM FROM WHERE address mesto = ’Praha’
```

SELECT id_adresy, mesto **FROM** adresy **WHERE** mesto ='Praha'

V prvním kroku zjistíme všechny adresy, které jsou v Praze. Nyní je vhodné připomenout si **schéma databáze**, ze kterého zjistíme, že v tabulce osoby existuje sloupec id_adresy, který se váže na sloupec id_adresy v tabulce adresy. Úvaha je tedy taková, že pokud vybereme všechny osoby, jejichž adresa se nachází ve výsledku výše uvedeného dotazu, tak dostaneme právě ty osoby, jejichž adresa je v Praze.

Řešení – krok 2

```
SELECT name, receive, nickname FROM people
WHERE id_addresses IN ()

SELECT DISTINCT id_address FROM WHERE address
mesto = ’Praha’ ()
```

SELECT jmeno,prijmeni,prezdivka **FROM** osoby **WHERE** id_adresy **IN**(

SELECT DISTINCT id_adresy **FROM** adresy **WHERE** mesto ='Praha'()

Nyní jsou v dotazu dvě konstrukce SELECT. Druhý select je tzv. *pod-dotaz* a je to lehounce upravený dotaz vytvořený v prvním kroku. První úprava se týká toho, že nyní vybírá pouze

sloupec id_adresy. To je zásadní, vzhledem k tomu, že nadřazený SQL dotaz hledá id_adresy, tak pod-dotaz musí vracet pouze id_adresy. Kromě toho operátor IN pracuje pouze se skalární množinou a tedy pod-dotaz (pro definování množiny pro operátor IN) musí vždy vracet právě jeden sloupec. Pokud v pod-dotazu použijete více sloupců tak DB server přivítá chybou:ERROR: subquery has too many columns. Druhá změna spočívá v přidání klíčového slova **DISTINCT**, které zajistí, že vrácené hodnoty budou unikátní (viz následující příklad). V tomto konkrétním případě je zbytečné (protože id_adresy je vždy unikátní), ale vzhledem k tomu, že IN se může chovat velmi prapodivně s pod-dotazem, ve kterém se hodnoty opakují (ona to pak koneckonců není množina), tak jako preventivní univerzální opatření je DISTINCT dobrý.

6.SQL – SLOŽITĚJŠÍ DOTAZY

6.1. Cvičení – dotazy SQL s více tabulkami

Příklad 1

Zadání: napište SQL dotaz, který vybere z databáze všechny osoby, které mají ICQ číslo; vyberte jméno, příjmení a ICQ číslo osoby; seřaďte podle příjmení vzestupně.

Varianta 1
Nejjednodušší - počítá s tím, že v tabulce typy_kontaktu, hodnota id,_typy_kontaktu = 1 odpovídá typu ICQ. Operátor JOIN pracuje vždy se dvěma tabulkami. Spojovací podmínka je vždy rovnost hodnot v nějakých sloupcích. Spojovací podmínka musí obsahovat obě tabulky, které se vyskytují v části JOIN. Spojovací podmínka musí obsahovat sloupce, které zajišťují vazbu mezi tabulkami. V tabulce osoby není žádný odkaz na tabulku kontakty. V tabulce kontakty je jeden odkaz na tabulku osoby - sloupec id_osoby. INNER JOIN vybere z tabulky pouze ty záznamy, které vyhovují spojovací podmínce - tedy pouze osoby, které mají nějaký kontakt (protože je však v dotazu zadaná i vyhledávací podmínka je v tomto konkrétním případě možné použít i LEFT i RIGHT JOIN, je to ovšem jen shoda okolností).

```
SELECT person.name, person.name, contact.contact FROM
     INNER JOIN contacts
          ON contacts.id_osoby = person.id_osoby
WHERE contacts.id_types_contact = '1'
ORDER BY by ASC
```

Varianta 2
Stejná jako předchozí, ale využívá USING. Pokud je spojovací podmínka rovnost stejně pojmenovaných sloupců (id_osoby) lze její zadání zjednodušit.

```
SELECT person.name, person.name, contact.contact FROM
     Persons JOIN contacts USING (id_persons)
WHERE contacts.id_types_contact = '1'
ORDER BY by ASC
```

Varianta 3

Lepší - vybere kontakty podle názvu typu kontaktu, nespoléhá se tedy na nějakou hodnotu id_typu_kontaktu. Název typu kontaktu je v tabulce typy_kontaktu. Je důležité myslet na to, že výsledkem spojení tabulek je tabulka a že operátor JOIN pracuje vždy se dvěma tabulkami.

```
SELECT person.name, person.name, contact.contact FROM
     (INNER JOIN contacts
          ON contacts.id_osoby = person.id_osoby)
               INNER JOIN contact_types
                    ON contact_types.id_type_contact =
                    contacts.id_types_contact
WHERE contact_name.nazev = 'icq'
ORDER BY by ASC
```

SELECT je opět příkaz, který může mít spoustu dalších slov, kombinovat několik tabulek do sebe atd. Ale úplný základ sytaxe je: SELECT, pak následuje seznam sloupců, které chceme získat, případně agregační či jiné funkce, FROM a název tabulky, ze které chceme data získat, WHERE a výraz s omezeními, která musí platit pro žádané řádky. V našem případě chceme, aby sloupec mesto obsahoval přesně hodnotu "Chomutov".

S agregační funkcí to vypadá třeba takto – chceme zjistit kolik řádků, tedy lidí je z Chomutova:

```
SELECT COUNT (*) FROM lide WHERE mesto = "Chomutov";
```

Vrátí nám to jeden řádek v jednom sloupci a bude obsahovat číslo 2.

Podmínky můžou být mnohem složitější s boolean operátory AND a OR, závorkami a dokonce SQL funkcemi, jejichž seznam je opět v dokumentaci. Vyhledání lidí z Chomutova starší 30 let by se udělalo takto:

```
SELECT * FROM lide WHERE mesto = "Chomutov" AND age> 30;
```

Hvězdička znamená “všechny sloupce”.

Napojení dalších tabulek do SELECT dotazu se dělá pomocí JOIN. To téma vyžaduje mít víc než jednu tabulku a nějakou relaci mezi nimi. Například tabulka lide jak to máme, kde každý člověk má unikátní id a pak třeba tabulku nápady se sloupci clovek_id a nápad. Ve sloupci clovek_id by byly cisla lidi, kteří daný nápad vymysleli a ve sloupci nápad by byl text nápadu. Pak bychom mohli udělat dotaz, který by vypsal všechny nápady a ke každému přidal i sloupec s názvem člověka. Detaily jsou ale nad rámec tohoto úvodu:

```
SELECT napady.napad, lide.jmeno FROM napady
LEFT JOIN lide ON lide.id = napady.clovek_id;
```

Výsledkem bude tabulka nápadů, kde v prvním sloupci bude nápad a v druhém jméno člověka, který nápad vymyslel.

Když místo “SELECT sloupce” zadáme “DELETE”, máme dotaz pro smazání řádků. Funguje podobně jako SELECT (je tam podmínka WHERE) ale místo získání výsledků smaže řádky odpovídající podmínce. Pro smazání Adama stačí zadat:

```
DELETE FROM lide WHERE name = ”Adam”;
```

Z těch nejpoužívanějších SQL dotazů ještě UPDATE. Ten provede úpravu řádku. Pokud Kateřina zestárne o jeden rok, provedeme aktualizaci třeba takto:

```
UPDATE people SET age = 30 WHERE name = ”Catherine”;
```

můžeme ale také použít matematický výraz a reference na existující hodnoty sloupců. Takže zvýšit hodnotu sloupce vek o jedna jde i takto:

```
UPDATE people SET age = age + 1 WHERE name = ”Catherine”;
```

To jsou všechny nejčastěji používané SQL dotazy (klauzule). Jednotlivé dotazy mohou mít víc slov pro složitější a přesnější dotazy. Pro jejich pochopení je však nutno studovat víc jednotlivé dokumentace, nebo hledat konkrétní problém. Například když použijeme v SELECT agregační funkci, získáme výsledek agregace celé tabulky. Pokud ale třeba chceme

mít průměry teplot v rocích a máme teploty po měsících, musíme použít GROUP BY a když dáme do Google **group by years datetime mysql**, zjistíme, že potřebujeme DATETIME funkci YEAR:

```
GROUP BY YEAR (record_date)
```

Kromě GROUP BY má SELECT i ORDER BY, kde si zadáme podle jakých sloupců výsledky řadit a jestli sestupně nebo vzestupně. Důležité je někdy pořadí slov. Pro SELECT je opět v **dokumentaci** a vidíme, že nejdřív v dotazu musíme mít GROUP BY … a pak ORDER BY … .

7.GRAFOVÉ DATABÁZE

Graf je datová struktura skládající se z vrcholů a hran. Grafová databáze je systém na ukládání a zpracování dat v podobě grafu. V mnoha případech je modelování domény grafem velmi přirozené – mezi takové domény patří vztahy mezi lidmi, mezi geny a proteiny, mobilní sítě, distribuční sítě různého druhu, neuronové sítě nebo třeba ekologické sítě zachycující interakce organismů.

Mezi grafové databáze se často řadí různé systémy, které spravují data v podobě grafů. Na základě takovéto definice by bylo teoreticky možné pohlížet i na relační databáze, resp. na jejich nadstavby jako na grafové databáze. Relační databáze ovšem neumožňují efektivní uložení a dotazování grafových dat.

7.1. Relační databáze

Průchod grafem v relační databázi vyžaduje joiny, a je tedy neefektivní pro průchod do hloubky. Jeden join za použití indexu vyžaduje logaritmický čas O(log n), bez použití indexu je to O(n log n).

Například vztahy mezi lidmi můžeme modelovat pomocí dvou tabulek: Člověk a Vztah. Pro každý průchod vztahem mezi dvěma lidmi obecně potřebujeme jeden join. Tradiční relační databáze jsou optimalizovány pro jednotky joinů. Jakmile počet joinů přejde do desítek, dostávají se relační databáze do potíží i přes použití indexů.

7.2. Opravdové databáze

Opravdovými grafovými databázemi nazveme pro účel tohoto článku takové databáze, které vyžadují pouze konstantní čas O(1) pro průchod hranou grafu.

Opravdové grafové databáze, podobně jako jiné NoSQL databáze, ukládají data v denormalizované (již „zjoinované“) podobě. U opravdové grafové databáze se tedy platí především u zápisu, zatímco dotazování (čtení) je levnější.

Příkladem opravdové grafové databáze je systém Neo4j, který je ve vývoji již více než deset let a je dostatečně vyspělý pro produkční nasazení, nebo například novější systém OrientDB.

Příkladem databází, které jsou schopné ukládat a dotazovat data v podobě grafů, ale ne nutně efektivně procházet grafy, je většina triplestorů (Sesame, Jena, Virtuoso) nebo FlockDB (projekt Twitteru).

Podobně jako je tomu u většiny NoSQL databází, ani pro grafové databáze neexistuje jednotný

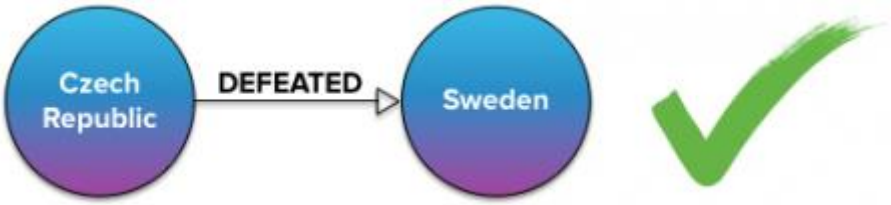
dotazovací jazyk. Nicméně mnoho grafových databází implementuje jazyk na procházení grafů Gremlin vytvořený ve spolupráci s autory Neo4j.

Přechod z relačního světa do světa grafů vyžaduje posun v uvažování a pohledu na data. I když grafy jsou často mnohem intuitivnější než tabulky, postřehl jsem stále opakující se chyby u lidí, kteří s modelováním grafů začínají. V tomto článku se podíváme na jednu z nejčastějších chyb – modelování obousměrných vztahů, a nakonec si ukážeme reálný příklad, ve kterém budeme nadále v seriálu pokračovat.

7.3. Jednosměrné vztahy

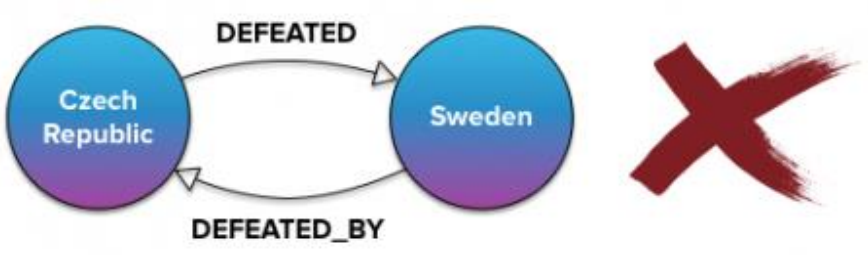
Vztahy v Neo4j musí být nějakého typu, který dává vztahu sémantický význam a také musí mít určený směr. Právě ten vyjadřuje smysl mezi entitami. Jinými slovy, vztah je nejednoznačný bez určení směru vztahu.

Například následující graf ukazuje, že Česká republika porazila (DEFEATED) Švédsko v ledním hokeji. Kdyby se směr vztahu obrátil, Švédové by byli mnohem šťastnější. Bez směru vůbec nevíme, kdo je vítězem, a proto je takový vztah nejednoznačný.



Všimněte si, že z existence tohoto vztahu vyplývá vztah opačném směru, jak je ukázáno níže v dalším grafu. Jedná se o velmi častý případ. Ještě jednou si popíšeme na jiném příkladu, že film Pulp Fiction režíroval (DIRECTED) Quentin Tarantino znamená také, že Quentin Tarantino je režisérem (IS_DIRECTOR_OF) filmu Pulp Fiction. A tímto by mohlo dojít k velmi mnoho párovým vtahům.

Všimněte si, že z existence tohoto vztahu vyplývá vztah v opačném směru, jak je ukázáno níže v dalším grafu. Jedná se o velmi častý případ. Ještě jednou si popíšeme na jiném příkladu, že film Pulp Fiction režíroval (DIRECTED) Quentin Tarantino znamená také, že Quentin Tarantino je režisérem (IS_DIRECTOR_OF) filmu Pulp Fiction. A tímto by mohlo dojít k velmi mnoho párovým vztahům.

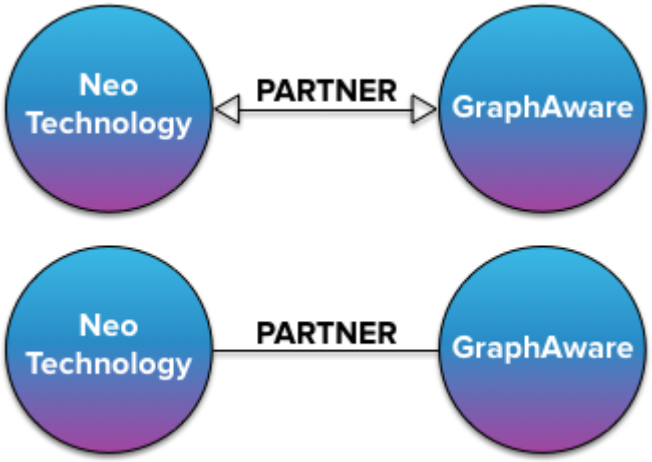


Právě této chyby se lidé často dopouštějí při modelování grafu v Neo4j a vytváří obousměrné vztahy. Vzhledem k tomu, že jeden vztah vyjadřuje druhý (symetrická relace), je to nehospodárné jak z hlediska prostoru, tak z průchodu (traverzování) grafem. Neo4j umožňuje procházet vztahy v obou směrech, takzvaně i proti směru hrany. Navíc díky způsobu, jakým Neo4j data ukládá, nezávisí rychlost průchodu vztahem na jeho směru.

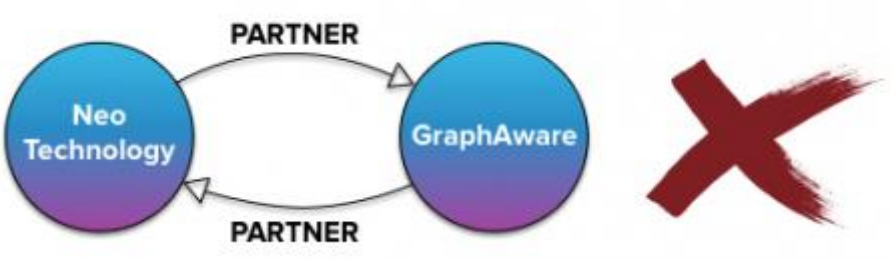
7.4. Obousměrné vztahy

Některé vztahy jsou přirozeně obousměrné. Klasickým příkladem je Facebook nebo vztah přátelství. Vztah je vzájemný – když s někým přátelíte, tak se on přátelí s vámi (možná tedy). V závislosti na tom, jak se díváme na model grafu, bychom o takovém vztahu mohli říci, že je obousměrný neboli neorientovaný.

Příkladem GraphAware a NeoTechnology jsou partnerské společnosti. Jelikož se jedná o vzájemný vztah, mohli bychom modelovat obousměrný neboli neorientovaný vztah.



Ale právě toto není možné v Neo4j – každý vztah musí mít počáteční a koncový uzel. Začátečníci se tak často uchylují k následujícímu modelu, který trpí přesně stejným problémem jako výše zmíněný model ledního hokeje s nadbytečným vztahem.



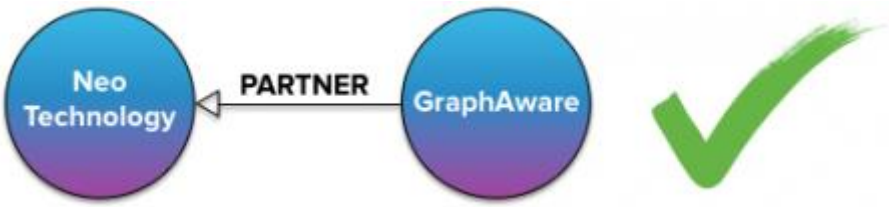
Neo4j API umožňuje vývojářům zcela ignorovat směrovost vztahů při psaní dotazů, pokud si to přejí. Například v jazyce Cypher by vyhledání všech partnerských společností s NeoTechnology mohlo vypadat takto:

```
MATCH (neo) - [: PARTNER] - (partner)
```

Výsledek by byl stejný jako sloučení výsledků z těchto dvou různých dotazů:

```
MATCH (neo) - [: PARTNER] -> (partner) and MATCH (neo)
<- [: PARTNER]
```

Proto je správný (nebo alespoň nejúčinnější) způsob modelování partnerských vztahů pomocí jediného PARTNER vztahu s určením libovolného směru.



7.5. Shrnutí

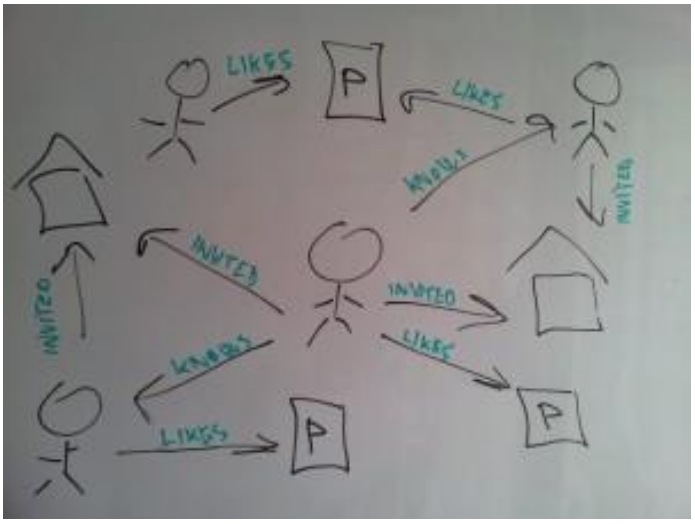
Vztahy v Neo4j lze procházet v obou směrech se stejnou rychlostí. Navíc směrovost může být zcela ignorována. Z tohoto důvodu není třeba vytvářet dva různé vztahy mezi entitami, pokud mají hrany opačného směru stejný význam.

7.6. Metodika modelování grafové databáze

V následujících článcích si ukážeme návrh, implementaci a testování grafové databáze Neo4j na projektu BestPartyToday (jedná se celosvětový pártylist, který nabízí detailní statistiky k akcím). V současné době se v projektu používá databáze MySQL, která obsahuje desítky milionů záznamů, takže nás také čeká přesunutí dat z relačního databázového úložiště do grafového. Ovšem v dnešním příspěvku si pouze namodelujeme podobu grafové databáze z vybraných tabulek a atributů současné struktury MySQL databáze.

O konceptuálním návrhu grafové databáze se pojednává jako o tzv. Whiteboard friendliness. Pro rychlé promítnutí všech myšlenek a postřehů, které byly vzneseny během návrhového brainstormingu, postačí pouze flipchart. Výsledný nákres je velmi jednoduché následně prezentovat ostatním projektovým skupinám (retail department, account managers, sales managers atd.), které nemají technické vzdělání. Pochopení návrhu pro ně už není překážkou, jelikož nákres je možné jednoduše přizpůsobit do podoby reálného života.

7.7. Realizace konceptuálního modelu



Na fotografii finálního návrhu grafové databáze pro projekt BestPartyToday je využito pěti nejvýznamnějších tabulek současné relační databáze, které jsou reprezentovány ikonami znázorňující nodes a směrovými šipkami prorelationships. Skrze entity (nodes a relationships) budeme procházet graf a získávat požadovaná data, která budou zpracována a zobrazena uživateli aplikace. Jejich význam je popsán níže v tabulce. Přehled entit grafového modelu.

8.NOSQL DATABÁZE

NoSQL databáze NoSQL databáze jsou tématem této práce, proto je jim věnována největší část. Nejdříve je třeba říct, co to vlastně NoSQL databáze jsou. Dále je kapitola věnována CAP teorému, na kterém je vysvětleno, že jedním z důvodů, proč existuje tolik NoSQL produktů, je to, že každý z nich může zajistit jen určité vlastnosti na úkor jiných a uživatel se pak musí rozhodnout, která kombinace vlastností je pro něj nejdůležitější a podle toho vybrat produkt. Dále se práce věnuje škálovatelnosti, tedy schopnosti navyšování výkonu databázového 15 serveru, ať již samotným vylepšováním serveru o výkonnější hardware nebo rozprostřením databáze na více serverů. Spolu se škálovatelností je třeba se taky zmínit o shardingu, tedy o technice, která rozděluje databáze na více částí, jež mohou fungovat samostatně a rychleji, než jeden celek. Poté následuje seznámení s nejznámějšími NoSQL produkty, kde jsou následně vybráni tři zástupci, se kterými je provedeno bližší seznámení. Popis Jak již zaznělo v úvodu, NoSQL databáze jsou ty, které nejdou relační cestou řízení dat, ale jinou. Nejsou primárně postavené na tabulkách a nevyužívají SQL pro práci s daty. Jejich doména je vysoká optimalizace pro vyhledávání, na úkor malé funkcionality, která se často omezuje na prosté ukládání dat. Tyto nedostatky, v porovnání s SQL, jsou kompenzovány škálovatelností a výkonem u určitých datových modelů. NoSQL se hodí na ukládání velkého objemu dat, u kterých není třeba uchovávat vzájemné vztahy. Strukturovaná data jsou ovšem povolena. U transakcí nemohou NoSQL databáze nabídnout plnou podporu ACID, nýbrž pouze Eventual consistency (Výsledná shoda). Jde o situaci, kdy díky zanedbání ACID získáme větší dostupnost a také lepší škálovatelnost. Tento přístup bez „silné konzistence“ je vhodný pro NoSQL, které ho taky často aplikují. NoSQL má distribuovanou architekturu, která je odolná vůči chybám. Některá data bývají umístěna na několika serverech a tím selhání jednoho se dá tolerovat. Tyto databáze většinou škálují horizontálně a spravují velké objemy dat, u kterých je důležitější výkon než konzistence.

NoSQL je doslovně kombinací dvou slov: No a SQL. Je to tedy technologie, která stojí proti SQL. Zkratka může být matoucí a mezi lidmi neexistuje jednotný názor na to, co znamená. Nejvíce rozšířený je ovšem ten, který tvrdí, že jde o acronym „Not only SQL.“ Ať už zkratka znamená cokoliv, NoSQL je dnes zastřešující název pro všechny databáze, které nejdou cestou známého RDBMS (relační systém řízení báze dat), ale jdou svojí vlastní, často spojovanou s velkými objemy dat.

8.1. Co je NoSQL?

Především je důležité zdůraznit, co NoSQL není – rozhodně to není NO SQL, tedy trendy odmítání relačních databází. Zkratka NoSQL znamená Not Only SQL, tedy uvědomění si toho, že relační databáze není jediná možnost řešení persistence, že existují i alternativy, které mohou být v některých případech vhodnější.

U rozsáhlejších aplikací pak můžeme dojít k tomu, že na některá data je vhodná relační databáze, na jiná NoSQL databáze a na další data úplně jiná NoSQL databáze. Tento pragmatický přístup, kdy se mixuje použití více databází v jednom projektu, se často označuje jako **polyglot persistence**.

NoSQL databáze vznikaly (a vznikají) jako řešení reálných problémů – není to tedy bláznivý výmysl akademiků odtržených od reality, ale vysoce praktická záležitost. Zrod mnoha NoSQL databází je spjat s projekty, které se musely vypořádat s obrovským množstvím dat – Facebook (**Cassandra**), Google (**BigTable**), Amazon (**Dynamo**), LinkedIN (**Voldemort**) atd.

Použití NoSQL databáze může mít ale smysl i tehdy, pokud máme méně dat (které by v pohodě zvádla relační databáze) – třeba protože nějaká NoSQL databáze nabízí datový model, který je pro naši aplikaci přirozenější.

Ale zpět k úvodní otázce. NoSQL databáze je software pro perzistenci dat, který je alternativou ke klasickým relačním databázím (nic objevného). NoSQL databází existuje mnoho a dají se rozdělit z **mnoha hledisek**. Ve velmi specifických případech může dávat smysl vyvinout i vlastní NoSQL databázi (ale myslete na **NIH**).

8.2. Kdy použít NoSQL

Rozhodnutí, zda použít místo ověřené relační databáze neznámou NoSQL databázi, není jednoduché, a hlavně to není rozhodnutí jen technické, ale i ekonomické.

NoSQL databáze např. často nemají takové možnosti dotazování jako relační databáze. Zkuste se třeba zamyslet nad tím, jak by ovlivnilo vaši aplikaci, kdyby jedinou podporovanou DB operací bylo uložení řádku a jeho načtení podle nějakého ID (nepřeháním!). To vede k tomu, že je na vaši aplikaci přenášena odpovědnost (např. generování sekundárních indexů), kterou jste dosud považovali za samozřejmou součást databázového enginu. Použití NoSQL databáze tak často znamená kompletní redesign celé aplikace, takže hodně učení a práce pro aplikační programátory (kteří nepracují zadarmo).

Většina dnešních programátorů vyrůstala ve světě, kde jedinou možností uložení dat byla relační databáze. Umí tak s ní dobře pracovat (v čemž jim pomáhají léty vyladěné tooly), jsou na ni schopni napasovat jakýkoliv model a už při úvodním seznamování s novým projektem jim v hlavě automaticky naskakují tabulky a vazby mezi nimi. Takže stojí za zvážení, zda se opravdu vyplatí investovat čas (peníze) do učení diametrálně odlišných technologií a zda nemůže být ekonomičtější použít léty ověřené relační databáze namísto nové NoSQL databáze.

Tím nechci říct, že se má absolutně kašlat na vzdělávání a inovaci, ale situace na trhu práce je tristní a ne každý má to štěstí (já ano :-)), že pracuje s bandou nadšených vývojářů, kteří čtou blogy, zajímají se o novinky v oboru a po večerech si zkouší nové technologie. Mnohdy člověk musí pracovat s lidmi, pro které je programování jen způsob obživy, odpracují si svých 8 hodin a tím to pro ně končí. A dělat s takovýmto týmem projekt, který používá NoSQL databázi, může být nákladné.

Náklady spojené s přechodem na NoSQL databázi se ale týkají celého týmu. Také vaši admini si budou muset umět poradit s novým SW v infrastruktuře, naučit se ho spravovat, zálohovat, tunit atp.

Co jsem chtěl říct tímhle zamyšlením? NoSQL je zajímavá alternativa k zavedeným relačním databázím a rozhodně má své místo na trhu a proto doporučuji se o NoSQL databáze zajímat a nějaké si zkusit osahat. Pokud si proti nějaké NoSQL databázi zkusíte napsat jednoduchou aplikaci, neuvěřitelně vám to rozšíří obzory.

Nepoužívejte ale NoSQL databáze jen protože je to cool, protože by to mohlo být lepší nebo protože je používá konkurence. Řiďte se rozumem, ne emocemi. Dobře si spočítejte, co vám NoSQL přinese a jaké budou náklady. Pokud se už pro NoSQL rozhodnete, tak myslete na to, že se nemusíte úplně vzdávat pohodlné relační databáze – v projektu nemusíte používat jednu jedinou databázi…

NoSQL databáze - Systém databází s možností horizontálního i vertikálního škálování. NoSQL databázový systém nevyužívá tabulky, jako je tomu u relačních databází, v tomto případě je cílem jednoduchost vzhledu a možnost horizontálního i vertikálního škálování. NoSQL je optimalizovanou databází, kdy jsou využívány klíče i hodnoty. Struktura ukládání dat je u NoSQL rovněž odlišná, kupříkladu se jedná o struktury stromovou nebo grafovou. NoSQL databázové systémy získávají na stále větší popularitě, především pokud hovoříme o segmentu real-time web. Dosud ale nejsou tolik rozšířené, což je dáno především absencí podpory transakčního modelu ACID či neúplnou standardizací rozhraní. Navíc jsou pro některé firmy NoSQL databáze dosti nákladné.

NoSQL

NoSQL je databázový koncept, ve kterém **datové úložiště** i zpracování dat používají jiné prostředky než tabulková schémata tradiční **relační databáze**. Motivací k tomuto přístupu mohou být jednoduchost designu, horizontální i vertikální **škálovatelnost** a jemnější kontrola dostupnosti. Databáze bez SQL jsou často vysoce optimalizovaná úložiště typu klíč-hodnota (ne vždy). Díky odlišné struktuře ukládání dat (např. stromová, grafová) oproti **RDBMS**, je i **algoritmická složitost** pro různé operace odlišná. Obecně se vhodnost aplikace daného typu databáze liší podle řešeného problému.

Segment NoSQL databází v současnosti významně roste a prospívá především v oblasti **big data** a **real-time webu**. NoSQL systémům se také občas říká „nejen SQL" pro zdůraznění faktu, že často umožňují dotazy v **SQL** (či podobném) jazyce. V kontextu **CAP teorému** NoSQL úložiště často potlačují konzistenci ku prospěchu dostupnosti a tolerance k narušení sítě.

Bariéry k rozsáhlejšímu nasazení těchto úložišť do praxe jsou např. nepřítomnost plnohodnotné podpory transakčního modelu **ACID**, použití (různých) nízkoúrovňových dotazovacích jazyků, nedostatečná standardizace rozhraní a vysoké realizované investice podniků do SQL v minulosti.

9.TRANSAKCE

Transakce je logická jednotka práce sestávající z jednoho nebo více SQL příkazů, které jsou atomické z hlediska zotavení se z chyb. SQL transakce automaticky začíná SQL inicializačním příkazem (SELECT, INSERT). Změny, které realizuje jedna transakce, nejsou viditelné pro ostatní konkurenčně probíhající transakce, pokud daná transakce neskončí.

Transakce může skončit jedním ze čtyř způsobů: – COMMIT končí transakci úspěšně a změny jsou trvale zaznamenány – ROLLBACK přeruší transakci a všechny změny se anulují, databáze se vrátí do stavu před transakcí – v rámci programu - skončí-li program úspěšně, končí úspěšně i SQL transakce – v rámci programu - končí-li program chybou, zruší se transakce SQL.

Řízení přístupu do databáze SQL definuje dva příkazy na řízení přístupu k tabulkám: GRANT a REVOKE Bezpečnostní mechanizmus je založený na – autorizačních identifikátorech – vlastnictví – privilegiích.

9.1. Řešení: Transakce

Transakce lze chápat jako sekvence databázových operací, které splňují ACID:

- **Atomicita** – Všechny operace v rámci transakce se provedou buď všechny nebo žádná.
- **Konzistence** – Před i po vykonání transakce se databáze nachází v konzistentním stavu. Musí být splněna všechna integritní omezení.
- **Izolace** – Jednotlivé transakce jsou izolované. Změny prováděné v průběhu zpracovávání dané transakce nejsou viditelné v ostatních transakcích.
- **Trvalost (durability)** – Po úspěšném ukončení transakce jsou data uložena a nemohou být ztracena.

9.2. Transakce v SQL

V jazyce SQL máme pro práci s transakcemi k dispozici následující příkazy:

BEGIN	spouští transakci ...příkazy v rámci transakce...
COMMIT	uloží a ukončí transakci
BEGIN	...příkazy v rámci transakce...
ROLLBACK	vrátí změny provedené transakcí a ukončí ji

Úrovně izolace

Požadavky ACID jsou relativně silné a jejich zajištění je časově náročné. Proto je možné tyto podmínky zeslabit pomocí tzv. úrovní izolace. Rozeznáváme 4 úrovně (od nejslabší po nejsilnější):

1. READ UNCOMMITTED

Transakci je umožněno číst data zapisovaná jinou transakcí, aniž by tato jiná transakce byla ukončená pomocí COMMIT. Čtení takových dat označujeme jako tzv. **dirty read**. Takto přečtená data mohou být nekonzistentní (čas plyne shora dolů):

2. READ COMMITTED

Na této úrovni izolace již nemůže dojít k **dirty read**. Data, která přečteme, byla vždy zapsána transakcí, která byla úspěšně ukončena pomocí COMMIT. Avšak může dojít k tzv. **non-repeatable read**. Pokud v rámci transakace čteme nějaká data vícekrát, může se stát, že se při opakovaném čtení pokaždé nevrátí stejný výsledek. Mezi čteními dat naší transakcí totiž mohla být úspěšně ukončena souběžná transakce, která zapsala/upravila/smazala nějaká data.

3. REPEATABLE READ

Na této úrovni je zajištěno, že nedojde k non-repeatable read. Data, která jednou přečteme, se již nemohou při dalším čtení změnit. Může však nastat tzv. phantom read. Pokud v rámci transakce čteme nějaká data vícekrát, může se stát, že výsledek opakovaného čtení bude obsahovat nové řádky, které při předchozím čtení ve výsledku nebyly. Mezi čteními dat naší transakcí totiž mohla být úspěšně ukončena souběžná transakce, která zapsala nová data.

4. SERIALIZABLE

Nemůže dojít k žádnému výše uvedenému fenoménu, vynucuje ACID.

10. PROCEDURY A FUNKCE, TRIGGERY A SEKVENCE

Funkce, metody a procedury – jaký je mezi nimi rozdíl?

Všechny tři (funkce, metody i procedury) jsou si poměrně dost podobné – je to POJMENOVANÁ posloupnost příkazů, část programu, kterou můžeme opakovaně volat z jiných částí programu.

10.1. Formální pohled

Z formálního hlediska se řídíme názvosloví daného programovacího jazyka.

- **Funkce** je pojem z funkcionálního programování, vyskytuje se v jazycích jako je JavaScript nebo třeba Haskell.
- Jako **metody** označujeme funkce v objektově orientovaném programování (OOP), kde dochází k propojení datových struktur a funkčního kódu do objektů. Metody nalezneme v jazycích jako Java, Smalltalk a dalších.
- **Procedura** je pojem známý z procedurálních jazyků. Nalezneme je např. v Pascalu, ale také v některých databázových systémech – tzv. uložené procedury.

Na funkce, metody a procedury se ale můžeme dívat z věcného hlediska (podle jejich vlastností a smyslu) a odchýlit se od terminologie konkrétního programovacího jazyka.

Funkce

Funkce v programování má velmi blízko k funkcím, jak je chápeme v matematice. Funkce má určitý definiční obor (typ vstupních parametrů) a určitý obor hodnot (typ návratové hodnoty).

Příklad jednoduché funkce v JavaScriptu:

```
Function sum (a, b) {return a + b};
```

V objektových jazycích jako Java máme sice metody a ne funkce, to nám ale nebrání v nich funkce psát:

```
Public static int sum (int a, int b) {return a + b};
```

Přestože sečti() – sum() je formálně metoda (veřejná a statická), ve skutečnosti se jedná o funkci a třída zde plní pouze úlohu jmenného prostoru (společně s názvem balíčku) a nevytváříme její instance (resp. nemusíme). Jedná se o návrhový vzor Knihovní třída.

V Javě najdeme takové funkce např. ve třídě java.lang.Math. Následující funkce vrací větší ze dvou čísel:

```
Int x = java.lang.Math.max (a, b);
```

Procedura

Procedura je zvláštním případem funkce – NEMÁ návratovou hodnotu a nemusí mít ani vstupní parametry. Používají se často při dávkovém zpracování – např. každou hodinu zavoláme proceduru, která zpracuje objednávky, které se nashromáždily v databázi, a předá je do jiného systému.

Příklad velmi jednoduché procedury v PostgreSQL, která sečte u dosud nesečtených řádků hodnoty sloupců a a b a výsledek uloží do sloupce c:

```
CREATE OR REPLACE FUNCTION add ()
RETURNS void AS
$ BODY $
UPDATE table of summaries
SET c = a + b
WHERE c is NULL
$ BODY $
LANGUAGE sql VOLATILE;
```

Tato procedura je napsaná v jazyce SQL – kromě toho můžeme psát plpgsql, který nám umožní tvořit i velmi složité procedury, případně můžeme použít jazyk C či jiný.

Výše uvedená procedura (formálně funkce) nemá žádné vstupní parametry ani návratovou hodnotu, je to prostě jen posloupnost příkazů jazyka, kterou jsme si nějak pojmenovali a uložili.

Zajímavostí je, že procedury můžou mít parametry – a to nejen klasické vstupní, ale i výstupní (nebo vstupně/výstupní). Procedura s výstupním parametrem nám slouží podobně jako funkce – přestože nemá klasickou návratovou hodnotu. Proceduře můžeme např. předat nějaká data a ona nám je upraví.

Takové procedury si můžeme simulovat i v jazyce Java. Mějme nějakou přepravku (strukturu):

```
Public class Person {public String name;
Public String Surname; Public String fullname};
```

A proceduru:

```
Public class StoredProcedures {public void reportNameName
(Person o) {o.celeName = o.name + ”” + oName}};
```

Proceduře předáme data (osobu) a dojde k jejich požadované úpravě. Ovšem pokud programujete tímto stylem v Javě, pravděpodobně děláte něco špatně a připravujete se o hlavní výhody OOP.

10.2. Metoda

Metody jsou součástí třídy a (pokud nejsou statické) jsou úzce spjaté s danou instancí třídy (objektem). Nejsou to tedy funkce pracující s nějakými globálními proměnnými a daty, ale mají přístup k proměnným dané instance (v tomto případě osoby).

Předchozí příklad můžeme přepsat „objektovějším“ způsobem:

```
Public class Person {public String name; Public
String Surname; Public String getCeléName ()
{return name + ”” + surname}};
```

Třídu s uloženými procedurami vůbec nepotřebujeme a požadovanou funkcionalitu přesuneme blíže k datům (do třídy Osoba). Všimněte si, že ani NEMUSÍME ukládat vypočtenou hodnotu celéJméno – vypočteme ji až ve chvíli, kdy ji někdo bude potřebovat, tzn. zavolá metodu get CeléJméno().

Co je trigger?

(Trigger je procedura, která je automaticky spouštěna, když se něco stane. Mohu ho definovat na DML (modifikace) i DDL (vytváření) operace. A navíc mohu říct, jestli se má provést před, nebo po dané operaci. (BEFORE, AFTER) Zajímavostí je, že trigger se může

jmenovat stejně jako tabulka (avšak není to doporučováno) a že lze definovat libovolný počet triggerů na stejnou tabulku i událost. Avšak pozor: nelze určit pořadí triggerů, ve kterém budou volány a triggery nesmí na sobě vzájemně záviset. V triggerech se také nesmí využívat rekurzivní volání. V těle triggeru lze provádět libovolné SQL dotazy - INSERT, UPDATE, DELETE. Avšak pokud chcete použít dotaz SELECT, musíte ho využít v kombinaci s klíčovým slovem INTO, které zajistí uložení načtených dat do lokálních proměnných, nebo kurzoru.

Typy triggerů

- DML triggery na tabulkách
- INSTEAD OF triggery na pohledech (views)
- Systémové triggery na DATABASE or SCHEMA: DATABASE - triggery se spustí na každou událost pro všechny uživatele, SCHEMA - triggery se spustí na každou událost pro zvoleného uživatele

Trigger je nástroj, který zajišťuje automatické provedení programu v jazyce SQL při vložení, zrušení nebo změně záznamu v určité tabulce. Trigger může například zajistit, že:

- před vložením nového záznamu do tabulky bude provedena kontrola jeho obsahu a doplněny hodnoty některých sloupců;
- po smazání záznamu se provedou následné akce;
- při změně hodnoty určitého sloupce v záznamu se porovná stará a nová hodnota a na základě jejich vztahu se provedou další akce.

Triggery zjednodušují tvorbu aplikací, protože přenášejí část práce databázové aplikace na server. Umožňují centralizované definování pravidel platných pro informační systém. Existuje-li například v podnikovém informačním systému tabulka zaměstnanců, lze pomocí triggerů popsat, jaké všechny akce musí být provedeny při přijetí nebo propuštění zaměstnance, změně platu nebo přeřazení do jiného oddělení. Tyto akce se naprogramují na jednom místě, ale budou sloužit všem aplikacím, které manipulují s tabulkou zaměstnanců. Dodržení pravidel pro údržbu evidence zaměstnanců pak bude zajišťovat server automaticky a konzistence dat bude zajištěna bez ohledu na možné chyby v měnících se aplikacích.

Trigger je pojmenovaným objektem patřícím do databázové aplikace. Má tyto vlastnosti:

- je svázán s určitou tabulkou a reaguje na změny v této tabulce;

- reaguje na právě jednu z SQL akcí **INSERT**, **DELETE** nebo **UPDATE**, triggery reagující na UPDATE mohou navíc **specifikovat množinu sloupců**, jejichž změna je spouští; trigger se spouští také prováděním obdobných akcí při práci s formuláři;

- provádí se buď před (BEFORE) provedením výše specifikované akce, nebo po ní (AFTER);

- pokud akce ovlivňuje více než jeden záznam, pak se může spouštět buď pro každý ovlivněný (tj. vložený, zrušený nebo změněný) záznam zvlášť, nebo pro všechny záznamy najednou;

- může obsahovat podmínku, která se vyhodnotí před provedením triggeru a pokud není splněna, trigger nebude spuštěn;

- specifikuje akci (zapsanou v jazyce SQL), která bude automaticky provedena, jakmile jsou splněny podmínky pro spuštění triggeru.

Při provedení konkrétní akce mohou být splněny podmínky pro spuštění více než jednoho triggeru. V takovém případě jsou spuštěny po řadě všechny. V současné podobně návrhu normy SQL 3 není zahrnut způsob, jak by uživatel v textu triggerů mohl definovat pořadí jejich spouštění. Je pravděpodobné, že před definitivní redakcí normy bude tento způsob specifikován a poté bude přidán i do 602SQL.

Provádění triggerů může měnit obsah databáze a tím spouštět další triggery. Spouštění triggerů se může do sebe libovolně zanořovat.

Při provádění triggerů se nekontrolují práva (triggery se provádějí s maximálními právy). Aby díky tomu nemohlo dojít k neoprávněnému přístupu k datům, může triggery vytvářet pouze uživatel obsazený do **role** Author v aplikaci, do níž trigger patří. V **zamčené aplikaci** není tedy možné triggery vytvářet.

INSERT a UPDATE triggery se nespouštějí při importu dat do tabulky. DELETE triggery se nespouštějí při smazání tabulky jako objektu (při mazání záznamů samozřejmě ano). Spouštění triggerů při akcích vyvolaných aktivní referenční integritou závisí na nastavení **příznaku kompatibility** SQLOPT_NO_REFINT_TRIGGERS (4096)

Pro účely ladění může být výhodné **trasovat** spouštění triggerů (situace TRACE_TRIGGER_EXEC), triggery lze také **debugovat** stejně jako procedury.

Triggery a transakce

Trigger se vždy provádí jako součást **transakce**, v níž je spuštěn příkaz, který trigger spustil. **Chyba**, která případně nastane při provádění triggeru, má stejné důsledky, jako chyba ve spouštěcím příkazu - trigger nelze odvolat bez odvolání tohoto příkazu, příkaz nelze odvolat bez odvolání změn provedených triggerem.

Chyby **kategorie** "rollback exception condition", pokud se vyskytnou uvnitř triggeru, ignorují vnitřní deklarace handlerů (tj. handlery v triggeru a procedurách volaných tímto triggerem). Zachytit je lze pouze handlerem pro spouštěcí příkaz.

Uvnitř triggerů se nesmí provádět žádné explicitní **transakční příkazy**, v opačném případě nastane chybový **sqlstate** 2D000 (SQ_INVAL_TRANS_TERM).

11. ANALYTICKÉ NÁSTROJE OLAP

11.1. OLAP – informace pod kontrolou

Kteří zákazníci přinášejí 80 % zisku? Jak se změnila úspěšnost udržení zákazníků po zavedení věrnostního programu? Jak různé události ve firmě (kampaně, zavádění nových produktů, změny firemních procesů apod.) a na trhu (aktivity konkurence, změny ekonomických podmínek apod.) ovlivňují kolísání prodeje? Jaké je srovnání s minulým obdobím? Tyto a podobné otázky si klade každý realistický manažer. Každý den se snaží pochopit chování firmy na základě analýzy údajů, které má k dispozici. V dobře fungujících společnostech jsou součástí vybraných oddělení analytici, kteří tuto roli přebírají a manažerům poskytují pouze zdůvodněná doporučení. Všichni tito lidé potřebují mít připravenou infrastrukturu a nástroje, které jim umožňují tuto pro firmu velmi potřebnou činnost vykonávat.

11.2. Co a jak zjednodušuje práci?

Cesta k úspěšné analýze dat začíná už při jejich přípravě. Prvním důležitým krokem je shromáždění dat z podnikových systémů do formy vhodné pro analýzu a reportování. Již při této přípravě je nutné dbát na správné sjednocení různorodých dat a ošetření jejich kvality. Musíme vždy důvěryhodně zdokumentovat vazby na zdrojové systémy, aby výsledky byly vždy věrohodné a doložitelné - pro případ řešení chyb, které se v původních datech vyskytují. Výsledná data se ukládají do relační databáze, kde jsou k dispozici v rozsahu potřebném k operativním analýzám. Takto relačně připravená data slouží k vytváření detailních reportů o výkonnostních parametrech společnosti a jednodušším analýzám založeným na detailních datech. V následujícím zjednodušeném příkladu si však představme požadavek na jiný typ informací: Firma s celostátní působností prodává různé kategorie výrobků. Manažer člení jednotlivé pobočky do regionů. Kategorie prodávaných výrobků mají tři úrovně - kategorie, podkategorie, konkrétní výrobek. Firma má logicky definováno, že prodej se řídí plánem. Reálné plnění plánu je pak dáno součtem za všechny výrobky a všechny pobočky.

11.3. Jak tyto informace získat?

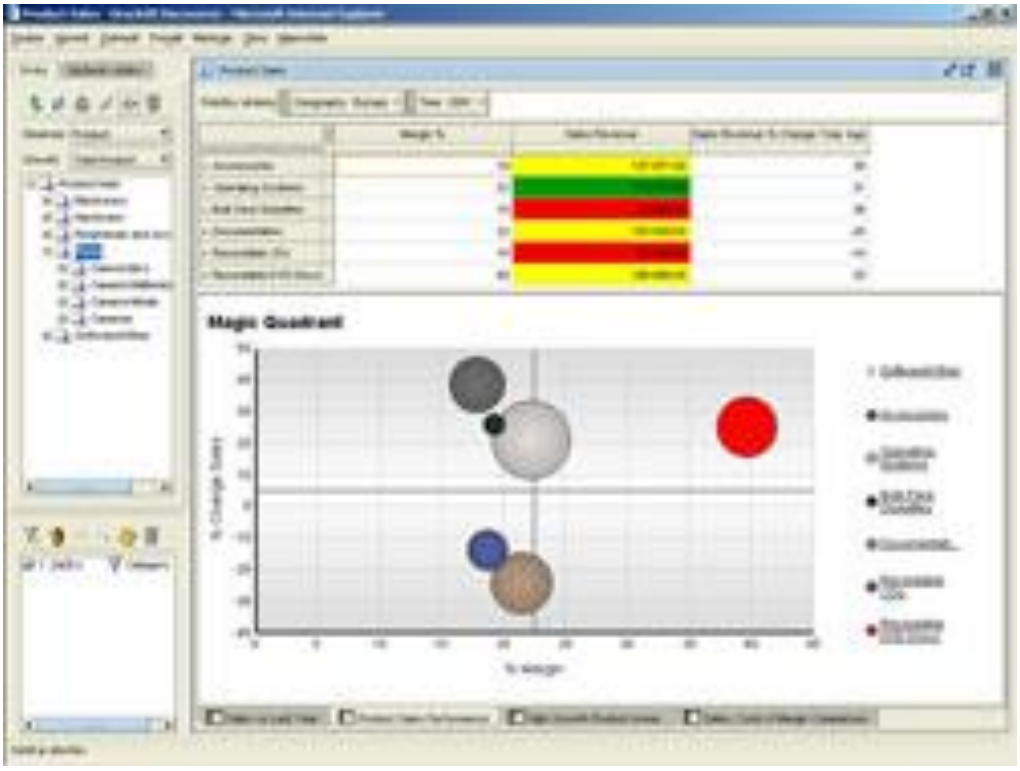
Pro tyto potřeby slouží pokročilé analýzy, které jsou označovány jako multidimenzionální. Manažer může sledovat měřitelné parametry firmy v souvislostech, různých úhlech a různých mírách detailu. K takové analýze je nutná technologie OLAP (on-line analytical processing), která zajistí uložení a předpočítání dat takovým způsobem, že následné

dotazy trvají přiměřenou dobu. Technologie OLAP přináší možnost pracovat s daty na sumární úrovni, zde identifikovat problém či zajímavou oblast a tzv. drilováním postupovat k takové úrovni detailu, která je pro naše rozhodování zajímavá. Typickou oblastí, na které se výhody technologie OLAP demonstrují, je analýza prodeje.

Obr. 1: OLAP v nástroji Oracle Business Intelligence – Discoverer

Manažer může prostřednictvím nástrojů pro práci s OLAPem kontrolovat, jak se vyvíjí realita proti plánu. Pokud plán není plněn, trendová křivka včas tuto informaci ukazuje, a konec roku nemusí skončit fiaskem. Sumární čísla je možné snadno rozdrilovat o úroveň níže, pak jsou vidět sumární plnění plánu v jednotlivých regionech. Analýza může ukázat, že problém spočívá pouze v jediném z nich, ostatní plán plní. Dalším drilováním se dostane na úroveň poboček, kde identifikuje pět takových, které v problematickém regionu výrazně zaostávají. Systém umožní rozdrilování na úroveň kategorií a subkategorií až na jednotlivé výrobky, které se v daném místě nedaří prodávat. Na základě takové informace pak lze navrhnout a provádět korektivní kroky, které situaci zlepší. Z příkladu je patrné, že data při použití technologie OLAP jsou vlastně uložena ve stromové struktuře - kostce. Zde jsou pro každou úroveň hloubky tohoto stromu předpočítány hodnoty pro příslušný podstrom. Taková hierarchická struktura je definovatelná na většině podnikových dat. Výrazně zpřehledňuje a zjednodušuje analýzu při jejich vyhodnocování. Cenou za toto zjednodušení je čas a místo potřebné k vypočítání a uložení zmíněného stromu. Je-li však kostka naplněna, dokáže přesně odpovídat na velmi komplikované dotazy, zejména provádět srovnání časových řezů omezených složitými podmínkami. Typický komplikovaný dotaz zní: "Kterých deset zákazníků zaznamenalo

nejvyšší nárůst obratu s mojí firmou ve srovnání s minulým rokem, a to v okresech, které patří mezi 20 % mých nejméně ziskových?" Výsledek pak může sloužit pro nadstandardní péči o tyto zákazníky. S technologií OLAP je možno dotazovat data z mnoha úhlů pohledu - mnoha dimenzí - a tyto úhly libovolně parametrizovat. Jak jsme si ukázali, pro tyto typy dotazů není vhodné ukládat data v relační podobě.



Obr. 2: Analýza prodeje v nástroji Oracle Business Intelligence - Discoverer

11.4. Analytické nástroje

Až do tohoto okamžiku jsme popisovali způsoby a technologie uložení dat. K práci s nimi (provádění analýz) jsou zapotřebí vhodné nástroje. Volba vhodného analytického nástroje je pro efektivní získávání informací naprosto klíčovou záležitostí. Je nutné zohlednit zejména zkušenosti analytiků, kteří ve firmě již dnes pracují s prostředky, které mají k dispozici - velmi často jsou používány nástroje z rodiny Microsoft Office. Je proto důležité, aby zvolené řešení umožňovalo dostatečnou spolupráci s nástroji, se kterými jsou lidé zvyklí pracovat - to vede k nižším nákladům na adoptování efektivnějšího způsobu práce s informacemi. Zároveň musí být také zajištěna bezpečnost přístupu k informacím, automatizovaná distribuce reportů, kvalitní grafický výstup pro prezentování, případně integrace s firemním intranetem.

Hlavní jsou lidé

Přes záplavu technologií však nesmíme zapomenout na lidi. Jedněmi z nejcennějších lidí, které firma může získat, jsou zkušení analytici. Protože pouze zkušený analytik může vyčíst ze záplavy dat podstatné informace a z nich formulovat doporučení pro management.

11.5. OLAP kostka

Prohledávání do hloubky je úplný algoritmus (projde každý uzel). Jeho princip spočívá v tom, že ex OLAP krychle (online analytical processing) je způsob organizace dat, který rozšiřuje dvojrozměrně tabulkové uspořádání tak, že každá datová dimenze je uložena v jedné ose kostky. Tím překonává některá omezení relačních databází.

Uspořádání dat do vektorů kostek umožňuje k nim zpětně přistupovat z různých hledisek (dimenzí) stejným způsobem. Odpadá tím na výkonnost systému náročné spojování mnoha tabulek RDBMS. Nicméně fyzické ukládání dat do kostek neumožňuje rychlou editaci, v takovém případě je třeba přepracovat celou kostku. Kostka je tvořena hodnotami, které jsou kategorizovány do dimenzí. Struktura je implementována relačními tabulkami ve hvězdicovém schématu či schématu sněhové vločky. Jedná se typicky o rodič-potomek (parent-child) strukturu, kde rodičovské prvky reprezentují konsolidaci potomků a zároveň ony samy mohou být agregovány do svých rodičovských prvků.

Základní operace s datovými kostkami

Pro analytické potřeby se provádí následující operace s datovými kostkami, které mají za účel zpracování a projekci dat tak, aby usnadnily jejich pochopení.

Krájení kostky: omezení jedné nebo více dimenzí na podmnožinu o jednom prvku.

Kostkování: omezení jedné nebo více dimenzí na podmnožinu o dvou a více prvcích.

Roll up a drill down: jedná se o navigaci datovou hierarchií směrem nahoru a dolů.

Pivotování: otáčení kostky za účelem získání jiné perspektivy na vztahy dat.

Agregace: Konsolidace podle vztahů určených vzorci.

12. SPECIFIKA DATABÁZOVÝCH SYSTÉMŮ. TECHNOLOGIE PŘÍSTUPU K DATABÁZÍM. GEOGRAFICKÉ INFORMAČNÍ SYSTÉMY.

Databáze (neboli datová základna, též databanka) je systém souborů s pevnou strukturou záznamů. Tyto soubory jsou mezi sebou navzájem propojeny pomocí klíčů. V širším smyslu jsou součástí databáze i softwarové prostředky, které umožňují manipulaci s uloženými daty a přístup k nim. Tento software se v české odborné literatuře nazývá systém řízení báze dat (SŘBD). Běžně se označením databáze – v závislosti na kontextu – myslí jak uložená data, tak i software (SŘBD).

12.1. Přehled přístupu k datům v technologii ASP.NET

Visual Studio 2010

Webové aplikace běžně přistupují ke zdrojům dat pro ukládání a načítání dynamických dat. Můžete psát kód pro přístup k datům pomocí tříd z oboru názvů System.Data (obecně označované jako technologie ADO.NET) a z oboru názvů System.Xml. Tento přístup byl běžný v předchozích verzích technologie ASP.NET.

Avšak technologie ASP.NET také umožňuje provádět vázání dat deklarativně. Pro většinu běžných scénářů není zapotřebí psát žádný kód, tyto mohou být:

- Výběr a zobrazení dat.
- Řazení, stránkování a také cachování dat.
- Aktualizace, vkládání a mazání dat.
- Filtrování dat pomocí parametrů za běhu.
- Vytvoření scénářů předloha-podrobnosti (master-detail) pomocí parametrů.

Technologie ASP.NET obsahuje několik typů serverových ovládacích prvků, které jsou součástí modelu deklarativního vázání dat, včetně ovládacích prvků zdroje dat, ovládacích prvků vázání dat a rozšiřujícího ovládacího prvku pro dotazy. Tyto ovládací prvky zvládají základní úlohy, které jsou vyžadovány modelem bezstavového webu s cílem zobrazit a aktualizovat data na webových stránkách technologie ASP.NET. Ovládací prvky umožňují přidat na stránku funkcionalitu vázání dat, aniž by bylo nutné porozumět podrobnostem životního cyklu požadavku stránky.

13. DATABÁZOVÉ OBJEKTY

Pojem „databáze“ je často zjednodušován na to, co je ve skutečnosti databázový systém (databázový stroj) nebo též systém řízení báze dat. Ten neobsahuje pouze tabulky – ty jsou jedny z mnoha tzv. databázových objektů (někdy též databázových entit). Pokročilejší databázové systémy dále obsahují:

- **pohledy** neboli views – SQL příkazy, pojmenované a uložené v databázovém systému. Lze z nich vybírat (aplikovat na ně příkaz SELECT) jako na ostatní tabulky.
- **indexy** pro každou tabulku. Klíče jsou definovány nad jednotlivými sloupci tabulek (jeden klíč jich může zahrnovat i více) a jejich funkce je vést si v tabulkách rychlé LUT (look-up tables – „pořadníky“) na sloupce, nad nimiž byly definovány, vyloučit duplicitu v záznamech nebo zajišťovat fulltextové vyhledávání.
- spouště neboli **Trigger**– mechanizmus nad jednotlivými řádkami tabulky (případně samotnou tabulkou), který se vyvolá po změně, odstranění nebo přidání řádky, případně smazání tabulky a provede předprogramovanou akci (například kontrolu integrity dat, doplnění hodnot…)
- **uživatelem definované procedury a funkce** – některé databázové stroje podporují ukládání pojmenovaných kusů kódu, které provedou v databázi nad danými tabulkami určitou sekvenci příkazů (procedury) nebo navíc vrátí nějaký výsledek (uživatelské funkce). Mohou mít parametry, které se většinou dělí na vstupní (IN), výstupní (OUT) a vstupně-výstupní (INOUT).
- **události**, též (počeštěně) „eventy“ – de facto procedury, spouštěné v určitý (uživatelem definovaný) datum a čas nebo opakovaně s definovatelnou periodou. Mohou sloužit k údržbě, promazávání dočasných dat či kontrolování referenční integrity.
- **formuláře** – některé databázové systémy jako např. Microsoft Access umožňuje uživatelům vytvářet vstupní formuláře pro vizuálně přívětivé zadávání hodnot. Uživatel si může např. nadefinovat rozložení jednotlivých vstupních polí z dané tabulky, popisky atd.
- **sestavy** nebo též reporty – podobně jako u formulářů sestavy umožňují uživateli definovat layout s políčky dané tabulky, do kterého se při použití doplní aktuální hodnoty. Používají se pro výstup dat (tisk, prezentaci nebo pouhé zobrazení). Sestavy mohou být např. doplněny o filtry, které vyfiltrují jen kýžené záznamy.
- **uživatelská oprávnění** – u lepších databázových systémů je samozřejmostí nabídnout možnosti, jak oddělit jednotlivé úrovně přístupu k ostatním objektům databáze jejich uživatelům. Možností bývají desítky, s rozlišením na jednotlivé typy příkazů, které ten který uživatel bude nebo nebude mít oprávnění spustit.
- **partitioning** – způsob, jak rozdělit data v tabulce na více pevných disků a tím rozložit zátěž na ni kladenou

- **procesy** – databázové stroje umí podat přehled o procesech, které jejich služeb aktuálně využívají.
- **proměnné nastavení** – typicky desítky proměnných, které lze přenastavovat a tím ovlivňovat chování a výkon databázového stroje jako takového.
- **collation** – MySQL má pokročilé možnosti pro nastavení několika desítek znakových sad a porovnávání, souhrnně nazývané collation. Nastavení collation může být provedeno na jednotlivé textové sloupce, celé tabulky i celé databáze (s kaskádovitou dědičností). Collation ovlivňuje i řazení, například hodnota utf8_czech_ci zajistí správné řazení podle češtiny (tedy včetně diakritiky a včetně ch).
- **vizuální E-R schéma** – (v MySQL INFORMATION.SCHEMA). Vizuální reprezentace vztahů (relací) na sobě závislých polí (cizích klíčů) mezi tabulkám.

13.1. Základní pojmy databázové technologie

Úvod k databázovým technologiím

S příchodem mikropočítačů a jejich vstupem do života každého z nás se zdá, že vzrůstá zájem o počítačově zpracovávaná data, a to ne pouze ve smyslu něco si vypočítat, ale především ve smyslu něco se dozvědět. Vysvětlení je zcela pochopitelné, uvážíme-li, že i ty nejdokonalejší počítačové hry časem omrzí a že programovat v Basicu výpočty typu řešení soustavy dvou rovnic o dvou neznámých nepřináší až tak velké uspokojení.

Člověk by si rád uspořádal a vyhledával některé informace, které často používá a které se v průběhu používání mění. Za předpokladu vhodných prostředků pro ukládání takových informací ve formě dat by byl ochoten prosedět několik večerů u klávesnice a příslušná data si sám vyrobit, koupit nebo vyměnit s jiným podobným nadšencem. Příkladem může být knihovna, úřad, zpracovávání letenek, městské muzeum, nemocnice, podnik apod.

Hotový objekt popsaného typu můžeme nazvat informačním systémem (IS). IS je tedy soubor prvků ve vzájemných informačních a procesních vztazích (informační procesy), které zpracovávají data a zabezpečují komunikaci informací mezi prvky. Z uživatelského hlediska by měl mít IS takové vlastnosti, aby manipulace s ním byla co nejjednodušší (vstup dat, formulace dotazů, použití aplikačních programů), a současně, aby funkce, které poskytuje, byly dostatečně silné a rychlé. Předložit uživateli pružný „prázdný“ IS lze dnes poměrně rychle, avšak s dostatečnou mírou znalostí.

Naším cílem bude ukázat si tzv. databázovou technologii zpracování dat. Databázová technologie je soubor pojmů, prostředků a technik sloužící pro vytváření informačních systémů (IS). Na nejzákladnější úrovni si můžeme představit architekturu IS s databází takto: data jsou organizována v databázi (DB), jsou řízena balíkem programů, který se nazývá systém řízení bází dat (SŘBD). Databáze a SŘBD tvoří tzv. databázový systém (DBS).

V naší terminologii můžeme jednoduše psát DBS = DB + SŘBD. IS využívá data z DBS buď přímo, nebo je zpracovává aplikačními programy.

13.2. Geografický informační systém

(GIS; anglicky Geographic information system) je geografický informační systém, který umožňuje ukládat, spravovat a analyzovat prostorová data.

Většina objektů a jevů reálného světa se vyskytuje na některém místě zemského povrchu (např. strom, dům, řeka) nebo má vztah k některému místu na zemském povrchu (občan má někde trvalé bydliště, výrobek byl vyroben v určité továrně). Zároveň se tyto objekty vyskytují v daném prostoru společně s mnoha dalšími objekty a navzájem se ovlivňují. Proto znalost umístění a vzájemných prostorových souvislostí mezi objekty je velmi významná a může sehrát důležitou roli v řadě oborů lidské činnosti, od návrhu umístění jaderné elektrárny až po návrh obchodní sítě a vyhodnocování její úspěšnosti.

Prakticky to znamená, že v našich datech v počítači musíme mít zaznamenáno obojí současně, tj. jak vlastní údaje o objektu, tak údaje o jeho poloze. Tomuto typu dat říkáme geografická (nebo prostorová) data a počítačovému systému, který umožňuje ukládat a využívat taková data, říkáme geografický informační systém, zkráceně GIS.

Základní komponenty geografických informačních systémů

Základní komponenty geografických informačních systémů jak uvádí Vít Voženílek ve své knížce Geografické informační systém I. Pojetí, historie, základní komponenty (1998)[6] jsou:

- Hardware
- Software
- Data
- Obsluhující personál

Pro efektivní práci systémů je nezbytná jejich vyváženost. Hardware, neboli technické vybavení, představuje technickou základnu geografických informačních systémů. Software, neboli programové vybavení, představuje soubor programů vykonávající veškeré operace systému. Data jsou klíčovým prvkem každého geografického informačního systému. GIS je z pohledu organizační struktury skutečným systémem. Jeho fungování je souhrnem činností, které zabezpečují jednotlivé funkce systému.

13.3. Geografická data

Dimenze geoobjektů

Základním dělením geoobjektů je dělení podle počtu dimenzí. Reálné objekty na zemském povrchu jsou vždy trojrozměrné. Do prostředí GIS se však převádí (transformují) dle potřebné úrovně zjednodušení.

- 0D geoobjekty – bezrozměrné objekty, body, definované pouze svou polohou. Příkladem může být například autobusová zastávka v GISu modelujícím dopravu nebo GSM vysílač v GISu mobilního operátora modelující pokrytí signálem.
- 1D geoobjekty – jednorozměrné obj., úseky čar (hran, linií), s konečnou délkou a nulovou plochou. Pomocí 1D geoobjektů se nejčastěji modelují silnice, řeky apod.
- 2D geoobjekty – dvojrozměrné obj., mnohoúhelníky (plochy, polygony), s konečným obvodem a konečnou plochou.
- 3D geoobjekty – trojrozměrné obj., geometrické těleso. V GISech se používají výjimečně, ve specifických případech. Třetí rozměr je v GISech nejčastěji modelován pomocí tzv. digitálního modelu terénu (DMT, anglicky DEM) útvarem "Povrch" (surface) - spojenými topologickými plochami (2,5D).

Základem každého GIS jsou geografická data. Geografická data jsou informace o zemském povrchu a objektech, které se na něm nachází. Data si lze představit jako určité vrstvy informací, které se nazývají témata. Každé téma reprezentuje určitý prvek (např. silnice, jezera, města a podobně). Geografická data mohou být v GIS organizována dvěma základními modely:

13.4. Sběr dat

Získávání dat a přesun informací do systému zabere hodně času. Existuje celá řada metod používaných k zadávání dat do GIS, které jsou uloženy v digitálním formátu. Exitující data vytištěné na papíře nebo PET filmu mohou být digitalizovány nebo skenovány k výrobě digitálních dat. Digitizér produkuje vektorová data jako body, linie a polygony. Skenováním map v podobě rastru, který by mohly být dále zpracováván k výrobě vektorových dat.

Sbíraná data mohou přímo vstoupit do GIS použitím techniky zvané souřadnicová geometrie. Pozice z globálního družicového navigačního systému (GNSS) lze také shromažďovat a následně importovat do GIS. Současným trendem ve sběru dat umožňujícím uživatelům využívat počítače v terénu s možností upravovat živá data pomocí připojení k bezdrátové síti nebo i úpravy bez připojení k internetu. To eliminuje potřebu importovat a aktualizovat data v kanceláři poté, co byla shromážděna prací v terénu. To zahrnuje schopnost začlenit pozic získaných pomocí laserového dálkoměru.

Nové technologie umožňují uživatelům vytvářet mapy, jakož i analýzy přímo v terénu, což činí projekty účinnější a mapování přesnější. Dálkově snímané data hrají také důležitou roli při shromažďování údajů a skládají se ze senzorů připojených k platformě. Senzory zahrnují kamery, digitální skenery a LIDAR, zatímco platformy se obvykle skládají z letadel a satelitů. V Anglii v polovině roku 1990, hybridní balónky zvané Helikites první propagoval použití kompaktních digitálních fotoaparátů vzdušných výsadkových jako Geo-informační systémy. Software letadla měřící s přesností na 0,4 mm byl použit k propojení fotografií s měřením ze země. Helikitesy jsou levné a shromažďují přesnější údaje, než letadla. V poslední době vývoj miniaturních bezpilotních dronů. Například dron Aeryon Scout byla použit k mapování 50 akrového areálu s hustotou vzorkování každý 1 palec (2,54 cm) za pouhých 12 minut.

13.5. Multidimenzionální kostka

Základní stavební jednotkou v multidimenzionální databázi je kostka (cube, datová kostka, multidimenzionální kostka, hyperkostka). Kostka v multidimenzionální databázi se skládá ze sady dimenzí a měr. Dimenze (rozměr) kostky jsou kategorie, vůči kterým chceme data agregovat a analyzovat. Dimenze vznikají z tabulek relačních databází. Typickými dimenzemi v multidimenzionálních databázích jsou čas, poloha, výrobek. Dimenze se může skládat z řady úrovní, které dále zpřesňují údaje. Míry kostky jsou kvantitativní údaje, které chceme analyzovat. Tak jako dimenze jsou i míry odvozeny z tabulek relačních databází. Běžnými mírami jsou prodeje, výdaje, ceny, ale téměř každý kvantitativní údaj může být mírou multidimenzionální kostky.

13.6. Uložení dat v OLAP databázi

Zdrojová data z relační databáze mohou být uložena jednou ze tří základních metod: Datové kostky obsahují souhrnná data, které jsou základem multidimenzionální analýzy. Komplexní multidimenzionální aplikace (např. informetrie) obsahuje data související s různými kontexty, které ale mohou některé dimenze sdílet. Je proto srozumitelnější vytvářet jednu datovou kostku pro jeden kontext než používat datovou kostku zahrnující všechny kontexty. Datová kostka pozůstává jenom z těch dimenzionálních atributů, které sdílejí všechny její číselné atributy. To znamená, že dimenzionální atributy tvoří základ datové kostky. Pokud chceme získat informace z několika datových kostek, mohou být tyto atributy propojeny na úrovni jedné nebo více společných dimenzí. Navigaci mezi datovými kostkami uživatel nevidí.

Tabulky dimenzí popisují specifické vlastnosti dimenzí. Pro každý dimenzionální atribut může existovat nanejvýš jedna tabulka dimenze. Na rozdíl od původního hvězdicového modelu, v našem modelu mohou být dimenze datové kostky bez tabulek dimenzí. Na

schématické úrovni se tabulka dimenze skládá ze jmen atributů, přičemž na úrovni příkladů pozůstává z hodnot těchto atributů.

V informetrii musejí být často souhrnná data analyzována na základě komplexních kritérií, které závisí na vlastnostech dimenzí. Aby bylo možné tato kritéria specifikovat, měla by být informace použita ve více tabulkách dimenzí. Z toho vyplývá, že sémantické vztahy mezi tabulkami dimenzí se vytvářejí na základě sdílených hodnot některých atributů v dimenzionálních tabulkách.