VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA OSTRAVA

Databáze

Učební texty předmětu Výpočetní technika - část Databáze

Upraveno pro distanční výuku

doc. Dr. Vladimír Homola, Ph.D.
Ing. Jarmila Drozdová, Ph.D.

Ostrava 2020

Tyto výukové materiály byly doplněny o podrobnější výklad některých termínů a postupů, které jsou běžně prezentovány při přímé výuce. Navíc byly vloženy kapitoly, na které je při prezenční výuce pouze odkazováno, a které jsou následně předmětem úloh na cvičeních.

K publikaci ISBN 978-80-248-4265-3

OBSAH

1 Úvod

1.1 Vymezení problematiky

Pojem *Databáze* (také Báze dat) je dnes naprosto běžně a suverénně používán. Přitom jen málokdo by zřejmě dovedl přesně vyjádřit, co si vlastně pod tímto pojmem představuje - a pokud ano, představy různých jedinců by byly vesměs různé. Proto nejprve ke genezi vlastního pojmu Databáze resp. Báze dat.

Především samotný pojem *data*. Latinské slovo *data* je množné číslo od slova *datum* (=*dáno* od slova *dare*, dávat). Slovo *data* začalo být používáno jako množné podstatné jméno, ale v průběhu doby se ustálilo nejčastější používání v jednotném čísle jako hromadné jméno (podobně jako nádobí, žactvo, stádo). Další změna v chápání pojmu data přišla díky změně sociálních a ekonomických podmínek při nástupu nových výrobních způsobů až k počátkům průmyslové revoluce. Data začala být chápána jako údaje, především číselné.

Nárůst počtu člověčích jedinců po období demografických krizí implikoval kvantitativní nárůst dat - když pro nic jiného, tak pro výběr daní. To ovšem způsobilo nárůst počtu speciálních jedinců tato data zpracovávající, a tedy zvýšené náklady na jejich vyplácení. Ale takové zvýšení nákladů se - tehdy i dnes - příjemcům výsledků zpracování dat hrubě nelíbilo. Vymysleli tedy mechanizaci tohoto procesu a vyrobili nejprve záznamníky na uchovatelné medium a jejich čtečky (nejprve papírové proužky a kartičky), pak nástroje na jejich zpracování (třídění, výběr a základní početní operace). Již v roce 1890 pro sčítání lidu v USA vynalezl Herman Hollerith počítací a třídící stroj, který zpracovával děrované kartičky o velikosti dolarových bankovek. Mimochodem jím založená firma Electric Tabulating Systems se později přejmenovala na Computing - Tabulating - Recording Company a ještě později na IBM.

Vymýšlení pokračovalo přidáváním komplikovanějších operací, zejména rozhodovacích. Vznikly tedy nástroje pro cyklické opakování jednotlivých kroků zpracování dat a pro větvení této posloupnosti podle stanovených podmínek. Cyklus vymýšlení posloupnosti operací (software) a technologické zdokonalování nástrojů pro jejich provádění (hardware) pokračuje známým způsobem do dneška.

Z hlediska databázové problematiky je však podstatné, že v jisté historické fázi se ukázala potřeba nejen data *fyzicky* zaznamenat a zpracovat, ale pro tyto účely data *logicky* uspořádat. Tato logická schémata byla nejprve aplikována na jednotlivé skupiny stejnorodých dat samostatně (osobní data jednotlivých zaměstnanců; data o denní výrobě jednotlivých oddělení; data o ziscích z prodeje jednotlivých výrobků atd) - z dnešního hlediska na jednotlivé datové soubory. Brzy však bylo zřejmé, že nestačí mít samostatně jednotlivé skupiny dat. Mezi těmito skupinami totiž mohou existovat zcela logicky nějaké závislosti: výrobek v nějakém oddělení vyrobil nějaký zaměstnanec. Přibyla tudíž data týkající se nikoliv reálných dat, ale popisující právě *vztahy* mezi reálnými daty. Dále se ukázalo, že i slovní pokyny pro zpracování reálných dat (např. Vyber tržby za výrobky z 1. června seřazené podle jednotlivých oddělení) lze celkem jednoduše formalizovat a takto je ukládat jako další komponenty potřebné pro celý proces zpracování. Všechny tyto tři části (reálná data, vztahy mezi daty, pokyny pro jejich zpracování) patří k sobě a začaly tvořit základ databází resp. bází dat.

Shora zmíněné jednoduché pokyny pro zpracování dat pro profesionálně vedené databáze určitě nedostačují. Pro jejich komplexní zpracování byly definovány specificky zaměřené programovací jazyky a sestaveny překladače z těchto jazyků (Cobol, xBase aj.). Každý z těchto jazyků však už je postaven pro konkrétní logickou organizaci dat v databázi. Je tedy třeba se zmínit nejen o některých z těchto logických organizací dat, ale i o širší problematice báze dat.

1.2 Demonstrační témata

Při výkladu databázové problematiky je třeba uvádět jednoduché a názorné příklady dokreslující popisovaný pojem. V těchto výukových materiálech byla zvolena dvě témata:

- V dílech I a II je použito téma z hydrogeologie. Z hypotetických hydrogeologických vrtů se čerpá voda, která je následně laboratorně analyzována na přítomnost nežádoucích látek. Ačkoliv je téma na první pohled odborné, čtenáři stačí v prvním

přiblížení představa díry do země, ze které se nějakým čerpadlem dopravuje nějakým výkonem na povrch voda. Z jejich vzorků se následně zjišťuje její chemická povaha - např. jestli a jak moc je slaná.

- V dílu III, který je věnován praktické tvorbě a následnému zpracování v prostředí databázových programů, je zvoleno geoturistické téma: cestovní kancelář organizuje pro své klienty tematické zájezdy do geovědně atraktivních lokalit. Toto nosné téma je uvedeno ve třech stupních geneze. Pro první je uveden nejen logický postup při vytváření struktury databáze, ale je k disposici i vzorové řešení. Detailně jsou témata popsána přímo v uvedeném dílu těchto textů.

Díl I: Databáze obecně

2 Organizace dat

Organizace dat je jisté uspořádání dat, které má za účel umožnit efektivní zpracování dat potřebných pro aplikace. Zahrnuje postupy a metody, jak data na médiích ukládat a jak je hledat. Tyto postupy a metody jsou soustředěny v programech, které požadované akce provádí. Programy mají tři úrovně:

- Základní systémová úroveň, kterou realizují komponenty operačního systému pro všechny uživatelské programy jednotně. Zápis a hledání dat podléhá fyzickým charakteristikám jednotlivých médií a proto se tato úroveň nazývá fyzická úroveň (dat, záznamu, organizace apod.).
- Úroveň báze dat. Jde o maximálně obecné, avšak většinou na jediný typ databázové struktury zaměřené programové systémy. Poskytují především možnost definice struktury záznamu (z jakých datových polí se záznam skládá, označení, zda jde o pole klíčové nebo hodnotové ap.), aktualizace dat (co do množství, kvality i struktury), a konečně získávání informací na základě zadaných specifikací. Základní funkcí bází dat je záznam a čtení takto nadefinovaných struktur na fyzické úrovni (tj. komunikace s programy systémové úrovně). Dále poskytují možnost provádění vazeb mezi jednotlivými soubory a výběr dílčích datových polí z takto provázaných souborů. Tato úroveň se nazývá logická úroveň.
- Uživatelská úroveň. Báze dat jsou zcela postačující systémy pro požadavky uživatelů na zpracování dat. Pro svou obecnost však kladou na běžné uživatele značné nároky na zvládnutí formalizovaného způsobu jejich ovládání. Proto báze dat poskytují prostředky, jak vytvořit rozhraní mezi obecnou bází dat a specializovanou potřebou toho kterého uživatele. Těmito prostředky jsou většinou programovací jazyky různých syntaxí, ale vždy se zajištěnou možností použít kteroukoliv z funkcí báze dat. Databázoví specialisté ve spolupráci s odborníky různých oblastí pak pomocí daného programovacího jazyka vytváří uživatelské, účelově orientované programy, které sice neposkytují možnost kdykoliv použít jakoukoliv funkci báze dat, ale zato uživatel pracuje v prostředí své odbornosti, ve kterém se dobře orientuje především terminologicky.

Poznámka: *Pojmy Pole, Záznam, Soubor apod. jsou podrobněji rozebrány v následujících odstavcích.*

2.1 Koncepce báze dat

Tato publikace je zaměřena na úroveň báze dat a případné vztahy k úrovni systémové tak, aby byly jasně vidět aspekty zvláště bází nejrůznějších geo-dat v moderních systémech. K vytváření bází dat vedly potřeby nesmírně dynamického rozvoje nových technologií umožňujících koncentraci a zpracování dat kvantity a kvality dříve nevídané (viz např. už jen družicové informace z nejrůznějších oblastí: geografie, geofyzika, strukturní geologie, environmentální inženýrství apod.).

2.1.1 Klasické metody

Klasické metody zpracování dat většinou nelze na data zmíněného rozsahu a provázanosti aplikovat. Především mají principiální nevýhody, a s růstem množství dat jsou stále méně efektivní. Jsou založeny na následujících principech:

- Data jsou uložena v souborech (z hlediska operačního systému). Soubor se skládá ze záznamů tvořených jedním nebo více fyzickými bloky.
- Záznam je členěn (pouze logicky, nikoliv fyzicky) na jednotlivá datová pole. Často má jedno nebo více polí identifikační význam a tvoří pak klíč záznamu. V takovém případě bývají soubory uspořádány ve vzrůstající posloupnosti hodnot klíčů a při každé změně klíčového pole a přidání nebo vypuštění záznamu je nutno soubor fyzicky přeorganizovat.

Organizace souborů (= jejich struktura) je dána potřebou konkrétních programů, které je používají. Programátor takových programů nejprve zjistí, jaká data jsou k disposici, jaká data na jejich základě je nutno odvodit, a podle toho navrhne strukturu souborů, která je z jeho subjektivního hlediska optimální.

V každém takto vytvořeném programu je tedy přesně udáno, které soubory se mají pro zpracování použít, jaká je jejich přesná interní struktura, a jaká data z nich mají tvořit požadované výsledné informace. Takový klasický způsob zpracování dat má mnoho nevýhod; mezi nejvýraznější patří:

- Během jednotlivých kroků při zpracování dat vzniká množství různých souborů. Je obtížné udržet přehled o jménech, obsahu, formě, době platnosti ap. všech existujících souborů.
- Velké množství souborů vede k tzv. redundanci dat: tatáž data jsou uložena na několika různých místech. Při aktualizaci takového údaje je nutno provést aktualizaci na všech místech jeho výskytu, což je náročné jednak časově a jednak organizačně.
- Velké obtíže nastávají při potřebě použít soubory se strukturou definovanou na míru dané aplikaci v jiných programech, které vyžadují strukturu jinou. Programy pro přeorganizování dat (zvláště pro změnu logické struktury) se svou složitostí mohou blížit samotným zpracovávajícím aplikacím.
- Jestliže je pro nějaký soubor opodstatněné vést dva nebo více nezávislých klíčů, je zapotřebí při přechodu od zpracování podle jednoho ke zpracování podle druhého provést jeho fyzické přeorganizování.

Právě při fyzickém přeorganizovávání objemných souborů může vznikat několik dočasných souborů, které mohou zahltit kapacitu i velkých médií.

2.1.2 Báze dat

Koncepce bází dat se snaží tyto nevýhody odstraňovat. Používají přitom následující metody:

1. Sdružují a provazují data souborů.
2. Oddělují popis logické struktury dat od dat samotných a od zpracovávajících programů.
3. Přistupují k jednotlivým údajům zvláštní programovou vrstvou, nikoliv přímo jednotlivými uživatelskými programy.
4. Poskytují možnost vyhodnotit uložená data jakýmkoliv způsobem. K uloženým datům umožňují současný přístup více uživatelů s jejich plnou vzájemnou ochrannou.

ad 1): Sdružování a provazování dat souborů znamená odstranění redundantních dat. Umožňuje spojování logicky navazujících souborů, ve kterých místo násobného přímého výskytu nějaké datové hodnoty udržuje ukazovátka (pointery, adresy) na jedinečné místo uložení této hodnoty.

ad 2): Oddělení popisu struktury od vlastních dat znamená, že kromě vlastních uložených dat jsou ukládány také informace o místě a způsobu jejich uložení. Tento popis může být fyzicky přítomen ve stejném souboru jako data, nebo v souboru samostatném. Jestliže se zpracovávající programy opírají o tyto popisy struktury, není zapotřebí programy měnit při reorganizaci vlastních dat.

ad 3): K vlastním datům nepřistupují aplikace přímo, ale žádají o jejich dodání obecně přístupné programové komponenty systému řízení báze dat. Provádí to většinou voláním podprogramů s parametry, kterými jsou popisy požadovaných dat získané z (od dat odděleného) popisu struktury.

ad 4): Zvláště v síťovém prostředí a při práci s citlivými informacemi je zapotřebí zajistit bezpečnost informací. Báze dat k tomu používají na jedné straně prověřování oprávněnosti přístupu pomocí seznamu uživatelů a jejich přístupových hesel a práv, na druhé straně mechanismus zamykání souborů, záznamů a polí dat.

Pro umožnění shora uvedeného musí být zajištěna

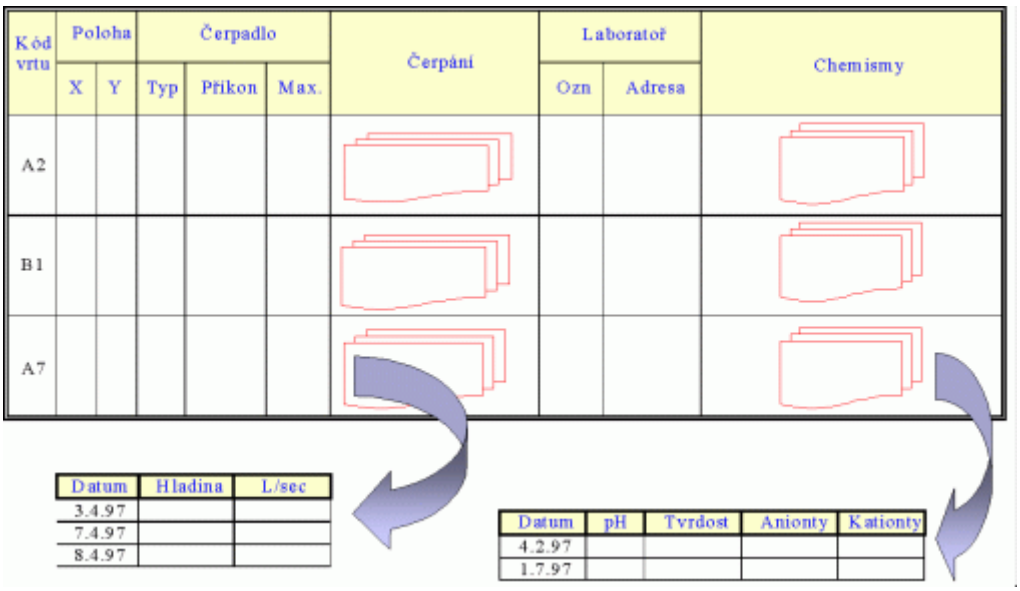
- fyzická nezávislost dat: fyzické uložení (např. délka bloku) může být změněno, aniž by bylo nutno přepisovat uživatelské aplikace,
- logická nezávislost dat: může být přidána položka nebo rozšířena logická struktura, aniž by to mělo vliv na existující aplikace,
- programová komunikace: musí být k disposici vhodné programové moduly systémů řízení báze dat pro dodávání dat do aplikací,
- uživatelská komunikace: musí být k disposici vhodný komunikační prostředek pro uživatele - nepočítačové odborníky; tím může být uživatelská aplikace, ale preferuje se obecný dotazovací jazyk nezávislý na použitém systému báze dat.

2.2 Soubor - záznam - pole

Z uživatelského hlediska je základní logickou jednotkou (datové) pole. Obsahuje hodnotu některého typu elementární informace, fyzicky proto bývá uloženo na jednom nebo více bytech. Pole je charakterizováno atributy (typ pole, jeho délka, poloha desetinné tečky nebo čárky apod.).

Jestliže více polí tvoří logický celek, tvoří tato pole segment. Segment sám nemusí být tvořen pouze poli, ale libovolnou posloupností polí a segmentů, je-li to potřebné nebo vhodné z hlediska logické struktury.

Segmenty, které k sobě logicky patří, tvoří záznam. Záznamy, které k sobě logicky patří, tvoří soubor. Soubory tvoří bázi dat.



Předchozí obrázek znázorňuje soubor s informacemi o podzemních vodách. Záznam je tvořen informacemi o jednom vrtu. Má šest segmentů: Kód vrtu, Poloha, Čerpadlo, ..., Chemismy. První segment je tvořen jediným polem. Segmenty Čerpání a Chemismy mají charakter násobného, opakujícího se segmentu, tj. mohou být přítomny vícekrát. Je samozřejmé, že - až na řídké výjimky - je počet záznamů v souboru vždy proměnný, stejně jako počet násobných segmentů v záznamu. Omezení shora je dáno především kapacitou média.

Některé typy organizace dat na systémové úrovni kladou další omezení na maximální počet záznamů v souboru (např. v systémech s pevným přidělením místa pro soubor) - musí být definován před vytvořením souboru. Některé systémy báze dat kladou další omezení na (maximální) počet opakování segmentů - rovněž musí být definováno před vytvořením souboru.

2.3 Popis a výskyt

Příklad z předchozího odstavce uvádí pohled na "naplněný" soubor. Z tohoto pohledu je zřejmá nejen logická struktura souboru, ale i jeho obsah (kdyby bylo hodně místa na papíře). Pro potřeby vytváření a zpracování bází dat je však zapotřebí od sebe oddělit popis struktury (typy polí, segmentů ...) a výskyt dat (záznamu, segmentu ...).

Popis struktury stanovuje počet polí, jejich typ a pojmenování, komposici segmentů ap. Popis každého segmentu se tedy vyskytuje jen jednou. Avšak segment, který je popsán ve struktuře, se - zvláště u násobných segmentů - nemusí ve skutečném datovém záznamu vyskytnout ani jednou, nebo naopak se může vyskytnout několikrát, přitom u různých záznamů v různém počtu. To se týká např. segmentu Čerpání: po založení záznamu tento segment asi ještě zápis o čerpání mít nebude; ale čas od času přibude další záznam o čerpání tak, jak to stanoví provozní podmínky.

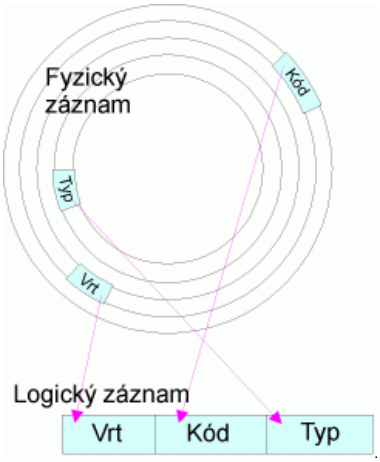
Je zřejmé, že popis struktury je zapotřebí formalizovat, a to jak na úrovni uživatele (=člověka), tak na úrovni aplikace (=programu). Různé systémy báze dat přijímají popisy struktur v různé syntaxi. Např. v hierarchickém modelu bývá používán popis, obdobný popisu souboru v jazyku Cobol. Zjednodušený příklad zápisu je následující:

```
01 Vrt.
  02 Kód-vrtu pic X(5).
  02 Poloha.
    03 X pic N(8).
    03 Y pic N(8).
  02 Čerpadlo.
    03 Typ pic X(10).
    03 Příkon pic N(4).
    03 Maximum pic N(6).
  02 Čerpání occurs (=opakuje se).
    03 Datum pic D(8).
    03 Hladina pic N(8,2).
    03 L-Sec pic N(6,1).
  ...
```

2.4 Vztah mezi logickými a fyzickými záznamy

V předchozím odstavci je podána ukázka logického záznamu. Tak uživatel požaduje, aby se mu záznam jevil. Při klasickém způsobu zpracování (a pohříchu mnohde ještě přetrvávajícímu klasickému myšlení) je od logického k fyzickému záznamu velmi blízko - třebas až na úroveň totožnosti. V bázi dat je ovšem logický záznam budován až při požadavku uživatele.

Takové požadavky mohou být v různých situacích velmi rozmanité a v době vytváření databází ani nemusí být známy. Protože je požadováno jen jedno fyzické uložení jednoho údaje, nemusí být jednotlivá pole logického záznamu dokonce ani v jednom jediném fyzickém souboru: např. shora uvedený logický záznam může být komponován z údajů geologa (vrty a jejich poloha), údajů odběratele (čerpadla a čerpání) a údajů chemika (výsledky chemických analýz).



Logické záznamy vytváří na přání uživatele aplikační programy (dotazovací jazyky) výběrem a organizací z fyzického záznamu. Logický záznam tedy může obsahovat

- části fyzického záznamu,
- celý fyzický záznam,
- více fyzických záznamů nebo jejich částí, přičemž tyto fyzické záznamy mohou a nemusí být součástí jednoho nebo více souborů.

Rozlišuje se tedy struktura organizace dat (vnější, uživatelský popis dat sloužící k vytváření logických záznamů), a struktura uložení dat (vnitřní, systémový popis dat fyzických záznamů).

Organizace dat v bázi dat musí umožňovat právě spolupráci mezi uložením a organizací.

2.5 Lineární a nelineární záznamy

Lineární záznam je takový, jehož pole nejsou vzájemně podřízena. Například záznamy o čerpání vod z vrtů obsahující kód vrtu, datum, hladinu a sekundové množství jsou klasickým příkladem lineárních záznamů. Jednou z jejich charakteristik je např. to, že při návrhu struktury nezáleží na pořadí polí v záznamu.

Nelineární záznam je takový záznam, v němž může existovat vztah nadřazenosti a podřízenosti polím. K nejvýznamnějším a nejpoužívanějším nelineárním strukturám patří shora zmíněná hierarchická struktura. Příklad tamtéž uvedený lze graficky znázornit např. schématem obvyklým v teorii grafů, kde z hlediska této teorie jde o strom.

Obecně lze říci, že u nelineárních záznamů záleží na pořadí polí v záznamu. Konkrétně u hierarchické struktury lze zavést zřejmý pojem úroveň pole jako počet nadřízených polí. Pak při návrhu struktury nezáleží na pořadí polí stejné úrovně podřízené stejnému poli. Všechna ostatní pořadí jsou pak evidentně významná.

Možnost vytvářet nelineární záznamy je důležitou vlastností bází dat. Ukazuje se tak opět význam rozlišení fyzických a logických záznamů.

3 Datové modely hierarchický a relační

Tato kapitola by logicky měla být uvedena až před kapitoly o podrobném popisu relačního modelu, na který se publikace soustřeďuje. Rámcové informace o alespoň dvou modelech organizace databází jsou uvedeny na tomto místě z důvodu lepšího pochopení následujících kapitol o metodách uložení dat a hledání v nich.

3.1 Hierarchický datový model

Model vychází z použité hierarchické struktury dat tak, jak byla kdysi implementována např. jazykem Cobol pro zobrazení hodnot dat a jejich vzájemných vztahů (nadřízenosti a podřízenosti). Tento model se neopírá o matematickou teorii, i když přejímá část terminologie z teorie grafů. Přesto nalezl v praxi široké uplatnění.

> ***Poznámka:*** *Cobol (= Common Business Oriented Language), jeden z nejlepších programovacích jazyků pro hierarchické databázové aplikace, je nabízen např. jako komponenta edice Microsoft Visual Studio 2017 - zde od firmy Micro Focus, Premier Partner firmy Microsoft.*

Na hierarchickém modelu byla a je založena řada systémů řízení báze dat, např. IMS firmy IBM, český DBS kdysi používaný v řadě EC apod.

Hierarchická struktura je taková, kde záznamy jsou v hierarchickém vztahu nadřazenosti a podřízenosti. Přitom se používá "rodinná" terminologie - rodič a potomek - ve zřejmém významu.

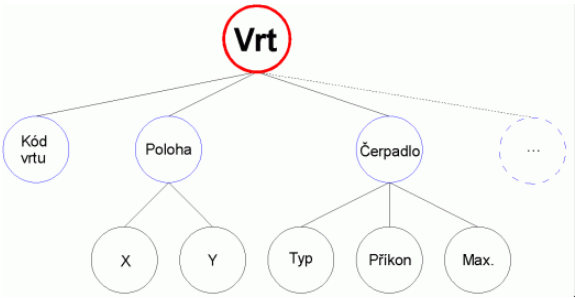
V hierarchické struktuře má každý potomek jediného rodiče, existuje jediný rodič, který není potomkem a potomek v jednom vztahu může být rodičem v jiném vztahu.

Pokud je zapotřebí popsat, na kterém místě hierarchické struktury se nějaký záznam nalézá, používá se k tomu tzv. přístupová cesta. To je možno díky popsaným vlastnostem hierarchické struktury, které zaručují, že od kořene lze dojít k danému záznamu jediným způsobem.

Často se vyžaduje (většinou z ryze praktických důvodů např. sběru dat), aby v každém záznamu existoval klíč. V takovém případě lze přístupovou cestu popsat jednoduše jako posloupnost klíčů počínaje klíčem kořenového záznamu přes klíče všech nadřízených až po klíč daného záznamu včetně.

Zmíněné pojmy z teorie grafů se při popisu hierarchických struktur využívají v tomto smyslu:

- záznam = uzel grafu
- vztah rodič - potomek = hrana grafu
- rodič a potomek = incidenční uzly hrany
- hierarchická struktura = souvislý graf, který je stromem
- báze dat hierarchického modelu = graf, který je les (tj. množina disjunktních stromů)
- přístupová cesta k záznamu = cesta v grafu od kořene k danému uzlu.



Nevýhodou hierarchických systémů je velmi obtížná implementace odkazů. Při případných realizacích se sice rozšiřují možnosti, snižuje redundance dat, ale současně dochází k nutnosti promíchávat otázky uložení na médiu s otázkami struktury modelu, ke znepřehlednění a zvláště ke snížení abstrakce při práci s daty.

3.2 Relační datový model

Jedním z nejjednodušších zápisů dat je zápis do klasické tabulky. Takto převážně vznikají zápisy dat např. v terénu, obecně v neautomatizovaných částech systému. Charakteristický pro tento zápis je její členění do sloupců, z nich každý má nadpis. To je velmi podstatné, protože nadpis ve smyslu identifikace údajů automaticky indukuje také typ údajů v daném sloupci. Sloupce jsou tedy "homogenní co do typu". Klasickým příkladem je výňatek z komplexní petrologické databáze (Global Data Base in Sedimentary Petrology, Géodiffusion, Paris 1991):

Skupina	Spec.	Ref.kód	SiO2	TiO2	Al2O3	Hornina
BVJ	A	392	49,88	1,85	14,27	Bazalt
BVJ	B	392	47,63	3,24	12,20	Bazalt
BVJ	C	392	46,93	4,38	12,29	Bazalt
BVJ	D	392	73,75	1,11	12,93	Sklo
NN	A	50	75,11	0,36	9,40	Ryolit
NN	B	50	67,79	0,54	14,50	Ryolit
NN	C	50	46,60	1,97	9,90	Pikritický bazalt
BYY	G	425	46,80	2,31	15,78	3-fenokryst.bazalt
BYY	L	425	45,56	3,59	13,31	Bazalt

Skupina	Ref.kód	Reference
BVJ	392	Richardson ...
NN	50	Mathias ...
BYY	425	DeWaard ...

Skupina	Spec.	Prvek	Konc.
BVJ	A	Cu	750
BVJ	A	Cr	120
BVJ	A	Ni	355
BVJ	D	Cu	68
BVJ	G	Ba	110

Pevným počtem n sloupců tvoří data v řádcích uspořádané n-tice. Každý prvek n-tice, nazývaný (datové) pole - Data Field, je toho typu, jaký odpovídá typu sloupce. Je tedy prvkem (konečné nebo nekonečné) jednoznačně určené množiny D_i (např. množiny všech datumů, množiny všech racionálních čísel, množiny {Ano;Ne} apod.).

4 Metody uložení dat

Problematika uložení dat je problematikou proto, že se data ukládají na lineárním fyzickém mediu. Paměť počítače je lineární posloupnost paměťových prvků adresovaných od nuly. Magnetická páska je lineárním mediem už z fyzikální podstaty. Disky a diskety s různou technickou konstrukcí jsou jednotným vzorcem linearizovány na shodnou (nanejvýš různě dlouhou) posloupnost paměťových elementů. Data z klávesnice jsou linearizována reálným časem (jak postupně v čase přichází "do počítače") apod. Datové struktury však lineární zdaleka být nemusí.

V celé této kapitole se nadále pod pojmem paměť bude rozumět jakékoliv shora naznačené linearizované medium schopné "zapamatovat si" data.

4.1 Postupné uložení

Toto uložení je charakteristické tím, že při uložení záznamů využívá paměť od počátku (= od adresy nula) a bez mezer. Volná zůstávají paměťová místa s nejvyššími adresami v případě, že objem dat je menší než kapacita paměti.

Další rozlišovací úrovní postupného uložení dat je jejich uspořádání v paměti.

4.1.1 Sekvenční uložení

Sekvenční uložení je takové, při němž se záznamy ukládají v pořadí, jak v čase přicházejí - tvoří vstupní sekvenci. Pokud je zapotřebí nějaký záznam najít, je nutno projít všechny záznamy od počátku.

4.1.2 Sériové uložení

Při sériovém uložení řídí pořadí záznamů hodnota nějakého klíče - posloupnost klíčů tvoří uspořádanou sérii. Záznam s nejmenší (největší) hodnotou je prvním záznamem, záznam s největší (nejmenší) hodnotou je poslední. Řazení dat je vzestupné (sestupné).

Vyhledávání záznamů se v sériovém uložení velmi zjednoduší. Největší nevýhodou je nutnost fyzicky reorganizovat záznamy, když dojde požadavek na přidání nebo vypuštění záznamů (nebo ke změně hodnoty klíče záznamu).

4.1.3 Ostatní postupná uložení

Z ostatních typů se občas používá řazení podle četnosti vyhledávání. Vyhledávání je pak celkově rychlejší než při sekvenčním uložení. Je však podmíněno jednak znalostí této četnosti (nutnost experimentů), jednak stálosti této četnosti v čase.

4.2 Rozptýlené uložení

Zásadní rozdíl oproti postupnému uložení je v tom, že mezi jednotlivými záznamy mohou být mezery a že jsou uloženy bez ohledu na nějaké uspořádání.

Při zápisu dat rozptýleného uložení se používají dvě metody:

a. Je jedno, kam bude zapisovaný záznam uložen. Proto se uloží do kteréhokoliv volného, délkově vyhovujícího místa v paměti. Tato metoda - má-li být rozumně rychlá a efektivně využívat paměť - používá pomocnou evidenci volného místa v paměti; z

této evidence lze kdykoliv zjistit adresy a velikosti volných míst. Při zápisu se pak pomocí této evidence volí optimální strategie ukládání - např. se nejprve hledá přesně stejně veliké volné místo, jako má zapisovaný záznam.

b. Místo, kam bude záznam zapsán, se určí na základě obsahu záznamu. V naprosté většině se k tomu používá hodnot klíčů. Existuje tedy zobrazení množiny hodnot klíčů do množiny adres. Toto zobrazení nemusí být prosté. Funkce taková zobrazení definující se nazývají ***Hash - funkce***.

Příklad: Nechť každý z 13 vrtů v terénu má číselný kód od 1 do 13 (vrty jsou tedy "očíslovány"). Nechť délka záznamu obsahující údaje o každém vrtu je 67. (První závěr: pro uložení těchto dat je zapotřebí minimálně 13 x 67 = **871** elementů paměti, tzn. např. adresový prostor <0,870>). Funkce ***A***

```
A = A (v) = (v-1) * 67
```

kde ***v*** je kód vrtu, je ***hash - funkce***, která každému záznamu s daným číslem vrtu přiřadí adresu uložení, která je z intervalu <0,804> (804 = 871 - 67). Tato funkce je ***prostá*** (dva různé vrty jsou umístěny na dvě různá místa). Funkce není zobrazením množiny kódů <1,13> ***na*** interval <0,804>, ale jen ***do*** tohoto intervalu (obrazy jsou jen počáteční adresy záznamů).

4.3 Kombinované uložení

4.3.1 Uložení s oblastí přeplnění

Tato metoda kombinuje sériové a rozptýlené uložení. Při primárním vytvoření je soubor vytvářen sériově (tj. ve vzrůstajícím nebo klesajícím pořadí klíčů) s ohledem na Hash - funkci. Nově přidávané záznamy jsou - většinou rozptýleným způsobem - zapisovány do tzv. zóny přeplnění (overflow area).

V případě, že v souboru může existovat větší či menší počet záznamů se stejnou hodnotou klíče, modifikuje se metoda tak, že jsou vytvářeny tzv. skupiny záznamů (záznamy se stejným klíčem), přičemž se předpokládá stejnoměrné rozdělení četnosti výskytu. Proto se každé skupině vyhradí stejné místo. Záznamy se stejným klíčem jsou zapisovány sekvenčně v prostoru paměti vyhrazené skupině a jednotlivé skupiny jsou zapisovány sériově v prostoru paměti vyhrazené skupinám. Dojde-li během zápisu záznamu do skupiny k vyčerpání místa určeného skupině, zapisují se tyto záznamy do zóny přeplnění.

Viz rovněž metody nepřímé adresace pro hledání v datech.

4.3.2 Uložení s indexy

V moderních databázových (zvláště relačních) systémech jde o jedno z nejužívanějších uložení. Zásadně se týká záznamů, pro něž je definován alespoň jeden klíč.

Jde v podstatě o tzv. uložení s odkazy. Kromě záznamů jsou ukládány také odkazy na ně. Tyto odkazy však nejsou většinou ukládány přímo v datovém souboru, ale v samostatných souborech.

4.3.3 Indexová tabulka

Při tomto způsobu uložení jsou data záznamy ukládány sekvenčně do souboru dat. Zároveň je pro každý záznam dat vytvořen záznam indexu. Záznam indexu má dvě pole: prvním je hodnota klíče daného záznamu, druhým je adresa v souboru dat, počínaje kterou byl tento záznam uložen. Takto vytvořený záznam indexu je zapsán do souboru indexů. Záznamy týkající se jednoho klíče jsou souhrnně označovány jako indexová tabulka. Každá indexová tabulka může být uložena v samostatném souboru, nebo mohou být všechny indexové tabulky uloženy v jediném souboru (nebo některé tabulky v jednom, některé v jiném souboru apod.). Tyto soubory se nazývají indexové soubory.

Soubor indexů mívá nejrůznější organizaci (snad kromě sekvenční). Nejčastěji to bývá struktura s odkazy nebo jiné rozptýlené uložení, méně často uložení sériové.

Zvláště pro velké objemy dat je tento způsob uložení jedním z nejefektivnějších z hlediska využitého paměťového prostoru i (při vhodně zvolené metodě hledání v indexových souborech) při zpracování přímým způsobem.

Efektivita při ukládání spočívá jak v časovém hledisku (data jsou ukládána bez jakékoliv další reorganizace souboru vždy přímo na konec souboru), tak v hledisku hospodárnosti (v datovém souboru nejsou nevyužitá místa). Nutnost zpracovat současně i indexové soubory však je daleko menším zatížením, protože jednak jsou záznamy indexového souboru krátké (pouze dvě pole oproti např. desítkám polí vlastního datového souboru), jednak mohou být organizována zcela odlišně od vlastního souboru dat. Navíc, protože jsou poměrně málo objemné, mohou být celé umístěny v operační paměti, čímž se jak vytváření, tak zpracování dále nesmírně urychlí.

4.4 Uložení s odkazy

I když to není vždy pevným pravidlem, jde nejčastěji o sekvenční uložení záznamů, přičemž se při ukládání provádí dodatečné akce.

Odkazem se rozumí takový údaj v záznamu, který popisuje, kde se nachází jiný záznam; pro jednoduchost bude nejprve popsána situace, kdy tímto jiným záznamem je (logicky) další záznam. Odkaz je tedy datové pole, které však na rozdíl od běžných polí nenaplňuje uživatel, ale obslužný program.

> ***Poznámka**: Protože vztah před - za je binární operací porovnávání, musí být pro záznamy takovým způsobem ukládané definován výraz, nad jehož hodnotami se porovnávání provádí. Takový výraz je (viz výše) klíčem záznamu.*

Odkazem může být např.

- adresa v paměti (např. disková adresa)
- pořadové číslo záznamu (program si pořadové číslo převede na adresu v paměti)
- symbolický odkaz (program musí mít k disposici prostou hash-funkci)

4.4.1 Odkaz NIL

Při popisované organizaci uložení existuje jedna hodnota odkazu (většinou označovaná NIL), která "neukazuje nikam". Tato hodnota je volena tak, že jí nemůže nabýt žádná reálná adresa, pořadové číslo nebo odkaz. Touto hodnotou musí být např. obsazen odkaz logicky posledního záznamu, protože za ním "už nic není".

Při popisu struktury s odkazy se používá pro vyznačení vztahu mezi dvěma bezprostředně souvisejícími záznamy zřejmá terminologie: předchůdce - následovník, nadřízený - podřízený, rodič - dítě. Poslední termíny (angl. parent record - child record) se používá snad nejčastěji.

Záznamy se zapisují tak, jak přichází; z tohoto hlediska jde o sekvenční organizaci. Při ukládání záznamů se při této organizaci postupuje následovně:

- První příchozí záznam se zapíše jako první, přičemž pole odkazu na další se naplní hodnotou NIL. V tomto okamžiku je totiž první záznam také záznamem logicky posledním, za ním "už není nic".
- Druhý příchozí záznam se zapíše jako druhý. Nyní však mohou nastat dvě situace:
- Klíč druhého záznamu je větší nebo roven klíči prvního záznamu. Jinak řečeno, druhý záznam je logicky za prvním záznamem. Proto je nutno pole odkazu na následníka v prvním záznamu (doposud NIL) přepsat odkazem na druhý záznam, a pole odkazu na následníka ve druhém záznamu obsadit hodnotou NIL (druhý záznam se stává záznamem posledním).
- Klíč druhého záznamu je menší než klíč prvního záznamu. V tom případě první záznam zůstává logicky posledním záznamem (v poli odkazu na následníka zůstává NIL), kdežto druhý záznam je prvním logickým záznamem, je před prvním záznamem, proto pole odkazu na následníka druhého záznamu se obsadí odkazem na první záznam.
- Třetí příchozí záznam se zapíše jako třetí. Teď nastane jedna ze třech situací: třetí záznam je před prvním, mezi prvním a druhým, nebo za druhým. Upraví se tedy pole odkazů na následníka jen ve třetím záznamu, v prvním a třetím, nebo ve druhém a třetím.

Takovým způsobem se postupuje dále. Je zřejmé, že při zápisu každého dalšího záznamu se musí najít v již zapsaných záznamech dva klíče, mezi které klíč nově příchozího záznamu patří, tam stávající řetěz odkazů "rozpojit" a (pomocí odkazů) tam vložit nově příchozí záznam.

Velmi jednoduché je však logické vypuštění záznamu. Obsah pole odkazu na následníka vypouštěného záznamu se prostě přepíše do pole odkazu předchůdce.

4.4.2 Odkaz na předchůdce

Výše byl popsán způsob vytváření odkazu na následovníky. Zcela stejně se může vytvořit druhé pole odkazu v záznamu, odkaz na předchůdce. V tomto případě je hodnota NIL hodnotou odkazu na předchůdce logicky prvního záznamu.

4.4.3 Odkaz na souseda

Sousedem záznamu **A** je takový záznam **B**, který má stejný klíč jako záznam **A**. Stejně jako odkazy na předchůdce a následovníka mohou být v záznamech pole odkazů na sousedy. Z hlediska popisované metody je dále možno rozdělit sousedy na levé a pravé. Levý soused je ten, který je zapsán dříve, pravý soused je zapsán později. Tedy "nejlevější" soused má odkaz na levého souseda obsazený hodnotou NIL, "nejpravější" soused má hodnotou NIL obsazený odkaz na pravého souseda.

Tato metoda již ve fázi ukládání používá metod pro hledání; vlastní ukládání tedy probíhá daleko pomaleji než např. prostý sekvenční zápis. Je to však na podporu pozdějšího zpracování souboru, které se stává daleko efektivnějším. Protože se soubor vytváří jen jednou, ale zpracovává násobně, je tato metoda často využívána.

***Poznámka**: Soubory s indexy jsou takovým případem souboru s odkazy, kdy odkazy spolu s hodnotami klíčů jsou uloženy v samostatném souboru.*

5 Hledání v datech

Geo - data se vyznačují značným množstvím, klíč logického záznamu se může konstruovat z datových polí více souborů ap. Hledání v takových datech je kritickým momentem zpracování. Proto efektivita zpracování dat přímo závisí na efektivitě hledání v datech.

Hledáním rozumíme dodání takového záznamu ze souboru (souborů) dat, který splňuje dané kritérium. Tímto kritériem může být jakýkoliv logický výraz, vyhodnotitelný pro každý záznam. Záznam, pro nějž po vyhodnocení odevzdá tento logický výraz hodnotu logické 1, dané kritérium splňuje.

Jako kritérium efektivity různých metod hledání se používá průměrný počet vyhodnocení kritéria, jehož splnění je vyžadováno. Toto vyhodnocení vždy pracuje s hodnotami polí záznamu, proto jedno vyhodnocení je přímo spojeno s jedním vyzdvižením dat záznamu (všech nebo jen některých; velmi často jsou rychlosti v obou případech stejné!). Protože vyzdvižení dat záznamu je spojeno s přístupem k mediu - a to je časově velmi náročná operace - potvrzuje to jen tvrzení o hledání v datech jako o kritickém místě procesu zpracování dat.

V dalším bude symbolem N označován počet všech záznamů, symbolem Z_i počet kroků nutný pro vyhledání a vyhodnocení i-tého záznamu, symbolem Z_p průměrný počet nutných vyhodnocení, Z (= Z_p . N) počet všech vyhodnocení.

5.1 Sekvenční hledání

Při sekvenčním hledání se probírají záznamy postupně od začátku jeden za druhým tak dlouho, dokud není nalezen záznam vyhovující vyhledávacímu kritériu.

Nechť p_i je pravděpodobnost, že i-tý záznam bude nalezen v jednom kroku. Sekvenční hledání se používá tam, kde nejsou známy bližší podrobnosti o způsobu uložení; proto musíme předpokládat rovnoměrné rozložení, kde ovšem je $p_i = 1/N$. Je tedy

$$Z_p = Z_1.p_1 + \ldots + Z_N.p_N$$

$$Z_p = 1/N \; . \; (Z_1 + \ldots + Z_N)$$

Při sekvenčním hledání je $Z_i = i$, tedy

$$Z_p = 1/N \; . \; (1 + \ldots + N) = (N + 1)/2$$

Při sekvenčním hledání je tedy k nalezení požadovaného záznamu nutno prohlédnout průměrně polovinu souboru.

5.2 Sériové hledání

Sériové hledání se uplatňuje v souborech se sériovým způsobem uložení. Záznamy jsou tedy řazeny v uspořádané posloupnosti (sérii) hodnot klíčů. V metodách sériového hledání je vždy vyhledávacím kritériem hodnota klíče, podle kterého je soubor řazen.

5.2.1 Sériové hledání s pravidelnými skoky

Tato metoda (myšleně) rozděluje soubor na bloky o shodném počtu záznamů. Nejprve je (sekvenčně) nalezen blok, ve kterém se záznam nachází, a poté (sekvenčně) hledaný záznam.

Nechť je tedy počet bloků m, počet záznamů v každém bloku s. Je tedy N = m . s. Nechť j označuje číslo bloku, j je z <1,m>; dále označme h číslo záznamu v bloku, h je z <1,s>. Pro dosažení j-tého bloku je zapotřebí j kroků, pro dosažení h-tého záznamu je zapotřebí 0 až s-1 kroků (nula proto, že hledaný záznam může být prvním v bloku a už tedy netřeba hledat dále; s-1 proto, že s-tý už je první v dalším bloku). Je pak

$$Z = s \,.\, (1 + \ldots + m) + m \,.\, (1 + \ldots + s-1) Z = s \,.\, m.(m+1)/2 + m \,.\, (s-1).s/2$$

Protože je s = N/m, je

$$Z = N \,.\, (N + m^2) \,/\, (m)$$

a tedy

$$Z_p = Z/N = (N + m^2) \,/\, (m) = f\ (m)$$

Pro kontrolu: je zřejmé, že uvedená metoda je pro jednozáznamový blok totožná se shora popsaným sekvenčním hledáním: prohledávají se postupně všechny záznamy počínaje prvním, a to po jednom. Je-li m = 1, je

$$Z_p = (N + 1^2) \,/\, (1) = (N + 1) \,/\, 2$$

což je shoda s výsledkem odvozeným pro sekvenční hledání.

Dále pro všechna m>1 je

$$(N + m^2) \,/\, (m) < (N + 1) \,/\, 2$$

z čehož plyne, že průměrný počet kroků při sériovém hledání s pravidelnými kroky je vždy lepší než sekvenční hledání. Už např. pro m=2 je průměrný počet kroků zhruba poloviční.

Protože průměrný počet kroků je funkcí m, lze zjistit, pro jaké m_0 má tato funkce minimum (tj. pro jaké velikosti bloků je hledání optimální). Musí být

$$df \,/\, dm = 0$$

a protože

$$Z_p = f\ (m) = (N + m^2) \,/\, (m)$$

je po provedení derivace a zjištění m_0

$$m_0 = \sqrt{(N)}$$

a tedy po dosazení rovněž

$$Z_p = \sqrt{(N)}$$

Např. pro soubor o 10 000 záznamech je optimální délka bloku 100 a průměrný počet kroků pro nalezení hledaného záznamu rovněž 100. Pro srovnání: při sekvenčním hledání to je 5 000!

5.2.2 Sériové hledání půlením intervalu

Metoda je někdy označována také jako binární hledání, je logicky totožná se stejně označovanou metodou numerické matematiky. Protože jde o sériové hledání, lze aplikovat pouze na soubory se sériovým uložením, tedy seřazené podle hodnot klíče. Metoda je popsána pro vzestupné řazení, pro sestupné postačí zaměnit relace.

Princip metody je následující:

Existuje záznam, který rozděluje soubor na dvě, co do počtu prvků (až na jeden) stejné části. Všechny záznamy před tímto záznamem mají hodnoty klíče menší nebo rovnu, za tímto záznamem větší nebo rovnu. Vybere se tedy tento "prostřední" záznam a jeho klíč se porovná s hledaným klíčem. Jsou-li totožné, je záznam nalezen. Je-li prostřední klíč větší než hledaný, hledá se stejným způsobem v první polovině souboru. Je-li prostřední klíč menší než hledaný, hledá se stejným způsobem ve druhé polovině souboru.

Metoda končí ve dvou případech: hledaný záznam je nalezen, nebo není co půlit - v tom případě hledaný klíč nemá žádný záznam souboru.

Pro odvození průměrného počtu kroků uvažme toto: existuje jediný záznam dosažitelný právě jedním krokem (prostřední); existují dva záznamy, dosažitelné právě dvěma kroky (prostřední v dolní a hodní polovině souboru); čtyři dosažitelné třemi kroky ... obecně $2^{(j-1)}$ dosažitelné j kroky.

Počet kroků, kterými je každý prvek dosažitelný, determinuje rozklad množiny záznamů na třídy. Do třídy j patří $2^{(j-1)}$ záznamů. Součet kroků pro každou třídu je tedy

$$Z_j = j \cdot 2^{(j-1)}$$

Uvažujme nejprve, že ideální počet záznamů souboru je

$$N = 2^k - 1$$

(tj. pro každé půlení existuje přesně prostřední prvek) a označme

$$a = \log_2 (N-1)$$

Pak pro celkový počet kroků je (matematické odvození je vynecháno)

$$Z = 2^a \cdot (a-1) + 1$$

a tedy - protože $Z_p = Z/N$ -

$$Zp = (N+1)/N \cdot \log2 (N+1) - 1$$

tj. přibližně

$$Z_p = \log_2 (N)$$

Tento vztah byl odvozen pro ideální binární hledání ($N = 2^k - 1$), což je v praxi málokdy splněno. Porušení této podmínky však - zvláště pro větší počet záznamů - nevede k podstatnému zhoršení rychlosti vyhledávání.

Tato metoda je tedy ze všech zatím uvedených nejrychlejší.

5.3 Přímý přístup k datům

Pod přímým přístupem k datům se rozumí přímé čtení záznamu ze známé adresy. V těchto metodách je tedy $Z_p=1$, každý požadovaný záznam se přečte hned napoprvé.

Metody přímého přístupu k datům evidentně vyžadují znalost adresy pro každý záznam. To neznamená, že v každém okamžiku jsou známy adresy všech záznamů najednou. Znamená to, že v okamžiku potřeby daného záznamu existuje možnost adresu zjistit.

U těchto metod se obdobně jako u metod sériového hledání předpokládá existence klíče a vzestupné nebo sestupné uspořádání jeho hodnot.

5.3.1 Přímé adresování

Při této metodě je adresou přímo klíč nebo jeho lineární transformace.

Příklad: Nechť každý z 13 vrtů v terénu má číselný kód od 1 do 13 (vrty jsou tedy "očíslovány"). Nechť délka záznamu obsahující údaje o každém vrtu je 67. Klíčem vrtu je jeho číslo -v- a z něj lze odvodit adresu záznamu výrazem A = (v-1) * 67.

Metoda přímého adresování je vhodná v případě, že

- klíč je numerický
- ke každé hodnotě klíče z nějakého uzavřeného intervalu existuje záznam
- délka záznamu je u všech záznamů stejná

5.3.2 Nepřímé adresování

Nejčastěji však data v souborech nesplňují ani přibližně požadavky odůvodňující přímé adresování. Je to např. tehdy, když

- posloupnost klíčů ani teoreticky neobsahuje všechny hodnoty jediného uzavřeného intervalu, ale mezi hodnotami jsou "velké mezery".
- klíč je koncipován tak, že teoreticky umožňuje použít všechny hodnoty z nějakého uzavřeného intervalu, ale je známo, že všech skutečných záznamů bude mnohem méně.
- klíčem je výraz jiného typu než numerického a nemůže tedy být přímo použit k adresování.

V uvedených a podobných případech je nutno mít k disposici funkci, která transformuje hodnotu klíče na adresu. Adresa není tedy určena přímo, ale zprostředkovaně klíčem; proto se tyto metody nazývají metody nepřímého adresování, a protože používají (již při výkladu metod zápisu zmíněnou) Hash - funkci, nazývají se také Hash - metodami.

Zpravidla není možno sestrojit funkci A = A (k) tak, aby to bylo prosté zobrazení (celé) množiny klíčů na rovnoměrně rozložené adresy v množině použitelných adres. Na druhé straně by funkce A neměla být (z časových důvodů při jejím vyhodnocování) příliš složitá. Proto se užívají takové funkce, které některé adresy ponechávají neobsazené a jiné adresy jsou obrazem většího počtu klíčů.

Nechť funkce A (k) zobrazuje množinu klíčů na M různých adres. Pro uložení celého je však zapotřebí N různých adres. Paměť může být tím rovnoměrněji rozdělena, čím větší je poměr M/N. Ve skutečnosti se však pro uložení všech N záznamů nevyužije všech M adres. Některé zůstávají nevyužité, některé jsou použity vícekrát. Je-li počet adres použitých alespoň jednou označen L, pak se hodnota l = L/M označuje jako koeficient zaplnění (loading factor).

Použijme termín skupina záznamů pro ty záznamy, jejichž klíč je transformován na stejnou adresu. Rozměr skupiny záznamů je počet záznamů skupiny (ideální je samozřejmě rozměr všech skupin rovný jedné). Zvolí-li se při vytváření souboru velký rozměr skupiny, zmenší se pravděpodobnost nutnosti zápisu do zóny přeplnění (a tím zkrátí čas hledání v této oblasti), ale zvětší se čas při hledání ve skupině. Zvolí-li se naopak při vytváření souboru malý rozměr skupiny, zvýší se pravděpodobnost nutnosti zápisu do zóny přeplnění a tím se zvětší doba hledání v této zóně.

Je zřejmé, že základem optimální volby rozměru skupiny záznamů je kvalifikovaný odhad množiny skutečných klíčů, které budou do souboru skutečně zapisovány. Protože tento odhad se liší od aplikace k aplikaci, bývá u obecných systémů rozměr skupiny i velikost oblasti přeplnění volitelným parametrem souboru.

Funkcí A = A (k) existuje celá řada a z hlediska zaměření tohoto článku nemá smysl je zde uvádět. Účinnost vyhledávacích metod na nich založených ovlivňují následující parametry:

- rozměr skupiny záznamů
- hustota zaplnění skupin záznamů
- vlastní algoritmus transformace klíče záznamu na adresu počátku skupiny záznamů
- způsob zpracování oblasti přeplnění.

5.4 Hledání v datech s indexy

Tyto metody hledání mohou být aplikovány pouze na soubory dat, ke kterým byly vytvořeny indexové tabulky (viz odstavec o metodách uložení s indexy). Protože indexové tabulky jsou vytvářeny na základě hodnot klíčů, je možno hledat záznam pouze podle kritéria, kterým je hodnota klíče.

> ***Poznámka**: Protože vlastní data jsou uložena sekvenčně, je principiálně možno ke každému stávajícímu sekvenčnímu souboru vybudovat indexovou tabulku založenou na nějakém klíči, popř. vybudovat více indexových tabulek s různými přístupovými klíči postupně v čase.*

Při této metodě se hledání ve vlastním souboru dat převádí na hledání dané hodnoty klíče v indexové tabulce. Po nalezení záznamu indexové tabulky je z pole adresy získána adresa datového záznamu v souboru dat, a tento záznam se pak přímo získá jediným čtením.

Pro hodnocení metod hledání v datech s indexy platí vše, co bylo uvedeno dříve, ovšem aplikováno na soubor indexů. Organizace uložení v souboru indexů tedy určuje efektivitu používání souborů s indexy. Protože z dosud hodnocených metod hledání je nejméně efektivní sekvenční metoda, téměř nikdy se nepoužívá ani sekvenční organizace uložení, ani sekvenční hledání.

Velmi často se pro indexové tabulky používá uložení s odkazy. Záznam pak má 6 polí: čtyři pole s odkazy (na předchůdce, následovníka, a na levého a pravého souseda), jedno pole pro hodnotu klíče a jedno pro hodnotu adresy záznamu. Indexová tabulka pak má tvar stromu (z teorie grafů). Každý záznam je pak jedním uzlem v tomto grafu, hrany grafu jsou obrazem odkazů. Výhodou je možnost ukládat v každém uzlu ne kompletní hodnoty klíče, ale jen část hodnoty klíče náležející dané úrovni uzlu v grafu.

> ***Poznámka:** Právě takovým způsobem jsou organizovány indexové soubory v relačním databázovém systému Microsoft Fox, jednom z nejvýkonnějších databázových systémů v prostředí počítačů třídy IBM/PC.*

6 Teoretické základy relačních databází

6.1 Kartézský součin množin

Teoretickým základem relačních databází je teorie množin a množinové operace. Pro účely těchto výukových textů a s ohledem na cílový obor studia bude podán jen maximálně zhuštěný výběr pojmů a jejich vztahů.

Kapitola se snaží o pokud možno jednoduché přiblížení teoretického základu toho, co je běžně označováno jako "tabulka dat relační databáze". Protože jde o kapitolu publikace pojednávající o databázích v digitálním prostředí, soustřeďuje se na konečné množiny a rovněž definice některých pojmů přizpůsobuje praktickým databázovým aplikacím. Používá sice běžný matematický aparát známý z teorie množin, autor však doufá, že připojené komentáře vysvětlí dostatečně podstatu problému i těm, kteří matematiku nemají moc rádi.

6.1.1 Zavedení pojmu

Definice: *Kartézský součin* množin A, B (značení: A × B) je množina všech uspořádaných dvojic takových, že první prvek dvojice je prvkem množiny A a druhý prvek dvojice prvkem množiny B:

$$A \times B = \{ [a,b] \mid a \in A \wedge b \in B \}$$

Obdobně kartézský součin množin A, B, C (značení: A × B × C) je množina všech uspořádaných trojic takových, že první prvek trojice je prvkem množiny A, druhý prvek trojice je prvkem množiny B a třetí prvek trojice je prvkem množiny C:

$$A \times B \times C = \{ [a,b,c] \mid a \in A \wedge b \in B \wedge c \in C \}$$

atd. Je-li jedna z množin prázdná, je i kartézský součin prázdná množina.

Označení: Součin A × A označujeme také $\mathbf{A^2}$, M × M × M × M × M označujeme také $\mathbf{M^5}$ atd.

Jsou-li množiny A a B konečné s počty prvků n_A a n_B, je i jejich kartézský součin A × B konečná množina; počet jejich prvků (=počet uspořádaných dvojic) je roven $n_A \times n_B$.

6.1.2 Komentář

Kartézský součin *dvou* množin A a B je tedy množina *dvojic* obsahující kombinace "každý z A s každým z B". Je-li například množina A tříprvková množina obsahující tři barvy:

A = {červené, zelené, žluté}

a množina B dvouprvková množina obsahující dva kusy ovoce:

B = {jablko, hruška}

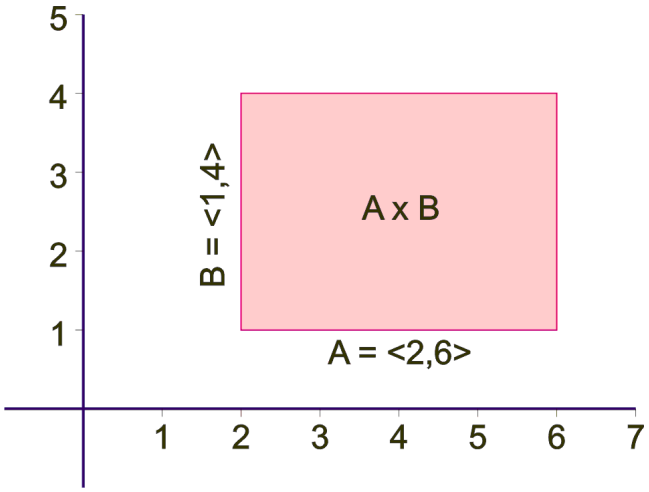
pak kartézským součinem A × B je množina šesti (3 x 2 = 6) dvojic - každá barva s každým ovocem:

A × **B** = { [červené,jablko], [červená,hruška], [zelené,jablko], [zelená,hruška], [žluté,jablko], [žlutá,hruška] }

Tento zápis je však poněkud nepřehledný. Pro přehlednější zobrazení kartézského součinu se pro množiny s malým počtem prvků může použít tabulka, např.

	jablko	**hruška**
červená	červené jablko	červená hruška
zelená	zelené jablko	zelená hruška
žlutá	žluté jablko	žlutá hruška

I v některých případech nekonečných množin lze s výhodou kartézský součin dvou množin znázornit graficky. Jsou-li např. A a B uzavřené intervaly reálných čísel, A=<2,6> a B=<1,4>, pak znázornění kartézského součinu A × B může být např. následující:



Problematičtější je znázornění kartézského součinu tří nebo více množin - zde by se jednalo o tří- a vícerozměrné útvary znázorňované v rovině papíru obtížně. Proto lze u konečných množin použít i tabulkového znázornění tohoto typu:

barva ∈ **A**	**ovoce** ∈ **B**
červené	jablko
červená	hruška
zelené	jablko
zelená	hruška
žluté	jablko
žlutá	hruška

Kartézský součin tří množin je pak tabulka se třemi sloupci, čtyř množin se čtyřmi sloupci atd. Důležité je, že a) v každém sloupci jsou prvky jen jedné konkrétní množiny, b) první sloupec obsahuje prvky pocházející z první množiny, druhý sloupec z druhé množiny atd.

6.2 Relace

6.2.1 Zavedení pojmu

Definice: *Binární relace* R v množinách A a B je libovolná podmnožina kartézského součinu A × B:

$$R \subseteq A \times B$$

Označení: Je-li [a,b] ∈ R, píšeme: aRb a čteme: prvek a je v relaci R s prvkem b, nebo: prvku a je v relaci R přiřazen prvek b. Je-li naopak [a,b] ∉ R, píšeme $a\bar{R}b$ a čteme: prvek a není v relaci R s prvkem b.

Označení: Je-li R ⊆ A × A, pak R nazýváme *relací v množině* A.

Definice: *Ternární relace* R v množinách A, B a C je libovolná podmnožina kartézského součinu A × B × C:

$$R \subseteq A \times B \times C$$

Definice: Obecně pak *n-ární relace* R v množinách A_1, A_2, ..., A_n je libovolná podmnožina kartézského součinu $A_1 \times A_2 \times ... \times A_n$:

$$R \subseteq A_1 \times A_2 \times ... \times A_n$$

6.2.2 Komentář

Definice říká: je-li kartézský součin např. A × B tvořen *všemi* kombinacemi prvků z A a B, pak relace je tvořena jen *některými* kombinacemi prvků z A a B, např.

barva ∈ A	**ovoce ∈ B**
červené	jablko
zelené	jablko
zelená	hruška

nebo

barva ∈ A	**ovoce ∈ B**
červené	jablko
žlutá	hruška

Omezme se nyní jen na binární relaci R a na jedinou množinu A. Nechť v množině

A = {3, 5, 7, 9}

je dána relace

R = { [3,5], [3,7], [3,9], [5,7], [5,9], [7,9] }

Je tedy např. 3R7, 7R9, ale $9\bar{R}3$.

Jsou-li množiny A a B konečné, lze pro znázornění relací použít několika způsobů. Nejčastěji používané jsou dva: maticový a tabulkový. Maticovým zápisem relace R z předchozího příkladu je následující matice 4x4 (nad matici resp. před matici byly pro přehlednost přidány nadpisy sloupců resp. řádků); v ní hodnota **0** značí, že prvky v relaci nejsou, hodnota **1** značí, že prvky v relaci jsou:

(a↓) R (b→)	**3**	**5**	**7**	**9**
3	0	1	1	1
5	0	0	1	1
7	0	0	0	1
9	0	0	0	0

Tabulkovým zápisem relace R z předchozího příkladu je následující tabulka:

a ∈ A	b ∈ B
3	5
3	7
3	9
5	7
5	9
7	9

Maticové zobrazení n-árních relací pro větší n je velmi nepraktické a nepřehledné. Proto se relace s konečným (často i značným) počtem prvků zobrazují výhradně jako n-sloupcové tabulky - pokud se samozřejmě nedají vyjádřit jinak, např. symboly výrokového počtu apod.

6.3 Relační databáze

Na shora zavedeném pojmu relace jsou konstruovány tzv. **relační databáze**. Nechť například množiny **D**, **C** a **N** značí po řadě množinu všech datumů, množinu všech řetězců (řetězec = posloupnost jednoho nebo více písmen, cifer a jiných znaků) a množinu všech racionálních čísel. Mějme relaci **R** definovanou jako podmnožinu kartézského součinu **K** = D × N × C × C. Množina **K** je (teoreticky nekonečná) množina všech uspořádaných čtveřic, kde první prvek čtveřice je datum, druhý racionální číslo, třetí je řetěz a čtvrtý je rovněž řetěz. Množinu **R** vytvořme tak, že vybereme jen některé čtveřice. Z hlediska praktického použití jakákoliv náhodná čtveřice, např.

[12/04/1543, 0, blabla, gaga]

asi nebude příliš zajímavá. Ovšem čtveřice

[01/01/1992, 9200, Novák, řidič]

už může vypovídat o jisté reálné situaci: prvního ledna 92 byl přijat Novák jako řidič s platem 9200 Kč. Tabulkový zápis relace R pak může vypadat např. takto:

Datum ∈ D	Plat ∈ N	Jméno ∈ C	Profese ∈ C
01/01/1992	9 200	Novák	řidič
15/06/1973	14 500	Novotný	vedoucí
01/11/1990	8 300	Nováček	vrátný
...	...	...	...
15/06/1978	17 800	Bratka	analytik

Každá čtveřice v tabulce podává jisté informace o jednom konkrétním pracovníkovi. Všechny čtveřice tvořící tuto tabulku pak podávají informace o všech pracovnících nějaké organizační jednotky.

Relační databáze tedy může být chápána jako obdélníkové schéma tvořené řádky a sloupci, které se řídí následujícími pravidly:

- Řádků je libovolný počet (i třebas nula - relace jako množina může být prázdná). Pro praktické databázové aplikace je však oproti definici relace vhodné, aby byla připuštěna možnost násobného výskytu stejných řádků.
- Sloupců je libovolný počet (nejméně však jeden - kartézský součin v "minimálním" případě je alespoň A^1).
- Sloupce mají "nadpisy" identifikující množiny, ze kterých pochází prvky daného sloupce. Podle definice nic nebrání tomu, aby se v kartézském součinu nevyskytovala jedna množina vícekrát (viz v příkladu shora množina C). Pak však musí být "nadpisem" rozlišeno, na jakém místě v kartézském součinu se množina vyskytuje - je totiž nepraktické označení "množina C třetího sloupce" apod. V příkladu shora nadpis "Jméno" identifikuje množinu C na třetí a nadpis "Profese" na čtvrté pozici kartézského součinu D x N x C x C.

***Poznámka**: Důsledkem teoretického základu relační databáze je to, že v jednom sloupci se nemohou vyskytovat dva prvky z různých množin.*

Příklad shora demonstrující pojem relační databáze je jedním z nejjednodušších. Obecně jsou totiž množiny tvořící kartézský součin skutečně libovolné množiny - množina obrázků, množina psychických stavů, množina vůní apod. Nic

tedy nebrání např. tomu, aby jedna z nich byla množina uspořádaných n-tic (tedy nějaká relace). Může to být např. relace z kartézského součinu L x S x P, kde **L** je množina všech kalendářních let, **S** množina všech škol a **P** množina všech prospěchů. Označme tuto relaci (=množinu uspořádaných trojic) písmenem A. Konkrétně může být A rovno

Rok ∈ L	Škola ∈ S	Prospěch ∈ P
1972	ZŠ Dlouhá	Vyznamenání
1975	SVVŠ Příčná	Uspěl
1983	ZŠ Dubí	Vyhověl
...	...	...
1982	VUML	Výtečný

Tabulka zaměstnanců pak může být obrazem následující relace:

Datum ∈ D	Plat ∈ N	Jméno ∈ C	Absolvent ∈ A	Profese ∈ C
01/01/1992	9 200	Novák	Rok ∈ L / Škola ∈ S / Prospěch ∈ P 1972 / ZŠ Dlouhá / Dostatečný 1982 / VUML / Výtečný	řidič
15/06/1973	14 500	Novotný	Rok ∈ L / Škola ∈ S / Prospěch ∈ P 1963 / ZŠ Dubí / Vyhověl 1967 / SPŠ Strojní / Vyznamenání 1970 / ČVUT / Výtečný	vedoucí
...	...	...	...	...
15/06/1978	17 800	Bratka	Rok ∈ L / Škola ∈ S / Prospěch ∈ P 1972 / ZŠ Dlouhá / Vyznamenání 1975 / SVVŠ Příčná / Uspěl	analytik

6.4 Databázové názvosloví

V počítačových databázových aplikacích se většinou používá uživatelsky zaměřená terminologie na rozdíl od matematické terminologie podané shora. V následujícím textu jsou **tučně** označeny termíny používané v relačních databázových aplikacích.

- **Tabulka relační databáze** - konkrétní relace v konkrétním kartézském součinu.
- **Záznam** - konkrétní prvek relace, tj. uspořádaná n-tice, také **řádek** tabulky relační databáze.
- **Hodnota** - konkrétní prvek uspořádané n-tice, tj. konkrétní prvek záznamu, řádku.
- **Datový typ** - hovorové označení některých konkrétních (reálných i abstraktních) množin figurujících v kartézském součinu (např. *datumový* pro množinu všech kalendářních datumů, *textový* pro množinu všech znakových řetězců apod.). Z hlediska uživatele databází klíčové místo. Databázové systémy poskytují totiž velmi omezený počet množin (většinou hardwarově determinovaných); jen z nich může uživatel umísťovat hodnoty ve svých tabulkách.
- **Typ hodnoty** - stejné jako datový typ; používá se pro zdůraznění toho, že hodnota pochází z určité konkrétní množiny.

7 Relační model - pokročilé

7.1 Vlastnosti binárních relací

Odstavec se týká pouze relací tvaru $R \subseteq A \times A$, tj. relací v množině A. Tyto relace mohou mít některé vlastnosti (např. že pro žádný prvek $a \in A$ neobsahují dvojici [a, a]). V následující tabulce jsou definovány některé základní typy relací podle svých vlastností:

***Poznámka**: Tento odstavec je vložen pro úplnost, zájemce jen o databáze ho může přeskočit. Nicméně je dobré si ho přečíst proto, že ukazuje teoretický základ některých zcela běžně používaných symbolů.*

7.1.1 Zavedení pojmu

Definice: V následující tabulce jsou definovány některé základní typy relací podle svých vlastností; relace R je vždy v těchto případech relací v množině A:

Relace R je ...	**... právě když platí:**
reflexivní	$\forall x \in A : xRx$
symetrická	$\forall x, y \in A : xRy \rightarrow yRx$
tranzitivní	$\forall x, y, z \in A : xRy \wedge yRz \rightarrow xRz$
areflexivní	$\forall x, y \in A : xRy \rightarrow x \neq y$
antisymetrická	$\forall x, y \in A : xRy \wedge yRx \rightarrow x = y$
ekvivalence	R je reflexivní, symetrická, tranzitivní
(neostré) uspořádání	R je reflexivní, antisymetrická, tranzitivní
ostré uspořádání	R je areflexivní, tranzitivní

7.1.2 Komentář

Kartézský součin A x A je množina kombinací každého prvku z A se všemi ostatními prvky A, ale i sám se sebou. Binární relace R v A je pak množina jen některých takových kombinací. Komentujme jen některé vlastnosti.

Relace R je **reflexivní**, jestliže v množině některých takových kombinací jsou určitě všechny kombinace všech prvků A samy se sebou (a třeba ještě některé jiné kombinace).

Naopak relace R je **areflexivní**, jestliže v množině některých takových kombinací určitě není žádná kombinace nějakého prvku A sama se sebou. Definice to říká takto: je-li v relaci nějaká kombinace [a,b], pak je určitě a různé od b (protože kdyby b bylo stejné jako a, je tam kombinace [a,a] a to bylo vyloučeno).

Relace R je **symetrická**, jestliže v množině některých takových kombinací platí toto: je-li tam [a,b], určitě je tam taky [b,a].

Naopak relace R je **antisymetrická**, jestliže v množině některých takových kombinací platí toto: je-li tam [a,b], určitě tam není [b,a] a naopak. Definice to říká takto: Je-li tam [a,b] i [b,a], pak určitě a i b jsou stejné prvky.

Zjišťování, zda nějaká daná relace má nějakou konkrétní vlastnost, znamená ověření podle definice v hořejší tabulce.

> Důležitý příklad: Nechť je dána relace R v {3, 5, 7, 9} - viz příklad shora. Tato relace je areflexivní (pro všechna [x,y] ∈ R je x ≠ y) a tranzitivní (3R5 ∧ 5R7 → 3R7; 3R7 ∧ 7R9 → 3R9; 5R7 ∧ 7R9 → 5R9). Relace je tedy **ostré uspořádání**; taková relace se často místo obecného R značí "<". Je tedy 3<5, 3<7, 3<9, 5<7, 5<9, 7<9.

7.2 Rozšíření definice relačního modelu

Množina všech uspořádaných n-tic $<a_1, a_2, \ldots, a_n>$, kde A_i jsou libovolné množiny, a_i je z A_i a n je přirozené číslo, je z teorie množin známa jako kartézský součin $K = A_1 \times A_2 \times \ldots \times A_n$ a každá podmnožina R z K je známa jako n-ární relace v K. Je tedy možno pohlížet na každou tabulku, která má "sloupce homogenní co do typu", jako na n-ární relaci - viz kapitola výše.

V databázové terminologii se jednotlivé množiny A_i nazývají (datové) **domény** (data domain).

Pro určení relace R pro potřeby modelu báze dat je zapotřebí zadat

- konečnou množinu atributů F - což jsou jména polí s případnými dalšími specifikacemi (šířka sloupce, počet desetinných míst ap.) využitelných pro uživatelskou i programovou identifikaci
- domény A_i, tj. množiny možných hodnot každého pole
- podmnožinu kartézského součinu domén, tj. vlastní relaci (z hlediska tabulky je tím určen počet polí a pořadí sloupců).

Relaci je tedy možno definovat jako trojici R = <F, D, T>, kde

- F je konečná množina jmen atributů
- D je zobrazení, přiřazující každému f z F doménu D(f) atributu f. Domény jsou libovolné neprázdné množiny (konečné nebo nekonečné); je-li f, g z F, f různé od g, nemusí být D(f) různé od D(g).
- T je konečná podmnožina kartézského součinu X [D(f)] všech domén atributů, f je z F.

Tabulky dat, které reprezentují relace, mají - jak již bylo uvedeno - následující vlastnosti:

1. Každému prvku relace odpovídá jeden řádek tabulky
2. Žádné dva řádky nejsou identické
3. Sloupec s atributem f z F v záhlaví obsahuje jen hodnoty z domény D(f).

Okolnost, že v praxi často nebývá splněna vlastnost ad 2), se řeší "očíslováním řádků"; přidá se jeden sloupec, jehož doména je podmnožina přirozených čísel.

7.3 První normální forma

Přestože domény mohou být libovolné množiny, používá se z čistě praktických důvodů většinou jen množin, jejichž prvky jsou

- číselné hodnoty (vzhledem ke zpracování v počítačovém prostředí výhradně racionální)
- datumové hodnoty (uspořádané trojice <d,m,r> přirozených čísel s příslušně definovaným oborem a operacemi)
- textové hodnoty (posloupnosti prvků z ze Z, kde Z je konečná neprázdná množina)
- binární hodnoty (prvky dvouprvkové množiny {Ano, Ne})
- relace.

Domény obsahující jen prvky analogické prvním čtyřem vyjmenovaným se nazývají jednoduché. Jestliže relace obsahuje pouze jednoduché domény, nazývá se relace v první normální formě. Proces převedení relace do první normální formy se nazývá normalizace.

Relace, která není v první normální formě, je dána např. následující tabulkou:

Vrt	X	Y	Čerpání			Laboratoř
			Datum	Hladina	l/sec	
A12	1202	485	12.01.92	35	17	KHS OV
			17.01.92	38	17	
			19.01.92	36	19	
			20.01.92	37	18	
A18	1203	477	Datum	Hladina	l/sec	OHS KI
			13.01.92	48	26	
			16.01.92	52	29	

Normalizace této relace může vést k jediné relaci, která již v první normální formě je:

Vrt	X	Y	Datum	Hladina	l/sec	Laboratoř
A12	1202	485	12.1.92	35	17	KHS OV
A12	1202	485	17.1.92	38	17	KHS OV
A12	1202	485	19.1.92	36	19	KHS OV
A12	1202	485	20.1.92	37	18	KHS OV
A18	1203	477	13.1.92	48	26	OHS KI
A18	1203	477	16.1.92	52	29	OHS KI

Jde o poměrně jednoduchý příklad; je však nutno upozornit na to, že procesem normalizace zvláště u složitě logicky strukturovaných relací mohou často vznikat redundantní údaje. I v tomto příkladu se redundanci dat nevyhneme: souřadnice jednoho vrtu jsou uloženy na několika různých místech. Pokud např. se posléze zjistí, že souřadnice X vrtu A12 byla zaznamenána chybně, je nutno hodnotu opravit ne na jediném, ale na několika různých místech. To je přesně ta situace, kterých by v reálné praxi mělo být co nejméně.

Některé atributy nebo jejich spojení lze v případě potřeby použít jako klíče. Označme je jako možné klíče. Klíče se používají jednak pro vyhledávání, jednak pro uspořádání. Jestliže se některý možný klíč skutečně pro daný účel použije, stává se po dobu použití primárním klíčem. Je-li klíč tvořen jediným atributem, označuje se jako jednoduchý klíč; je-li tvořen spojením dvou a více atributů, nazývá se spojený klíč. Je-li klíč vytvořen pomocí operací definovaných na hodnotách polí a na konstantách, nazývá se obecný klíč.

Pomocí klíčů lze nahradit jednu nenormalizovanou relaci více normalizovanými relacemi se stejným datovým obsahem, jak to ukazuje následující obrázek. Klíčem (a to jednoduchým) je v tomto případě atribut Vrt.

VRTY

Vrt	X	Y	Laboratoř
A12	1202	485	KHS OV
A18	1203	477	OHS KI

ČERPÁNÍ

Vrt	Datum	Hladina	l/sec
A12	12.01.92	35,00	17
A12	17.01.92	38,00	17
A12	19.01.92	36,00	19
A12	20.01.92	37,00	18
A18	13.01.92	48,00	26
A18	16.01.92	52,00	29

Pro formalizaci zápisu struktury se často používá notace, která je základem některých dotazovacích jazyků. V této notaci se zapíše struktura relací následovně:

```
VRTY (^VRT, X, Y, LABORATOŘ) ČERPÁNÍ (^VRT, ^DATUM, HLADINA, L/SEC)
```

Identifikátor relace je uveden před kulatými závorkami; uvnitř nich jsou uvedeny jednotlivé atributy. Je-li před některým z nich uveden znak ^, je tím označen klíč.

Nenormalizovaná relace se pak v této notaci zapíše následovně:

```
VRTY (^VRT, X, Y, ČERPÁNÍ (^DATUM, HLADINA, L/SEC), LABORATOŘ)
```

Porovnáním zápisů obou struktur lze odhalit jeden z možných postupů tvorby klíčů při procesu normalizace, kterým vzniká více relací: každá hierarchicky vnořená relace předřadí svému vlastnímu klíči (klíčům) klíč (klíče) relace bezprostředně hierarchicky nadřazené.

7.4 Operace s relacemi

V relačním modelu je báze dat definována jako n-ární relace; takových bází dat se tedy týkají všechny vlastnosti, které lze odvodit matematickým aparátem pro relace. Tento aparát jako obecný aparát matematický je pro účely počítačového zpracování velmi dobře algoritmicky propracován. S relacemi jako takovými se pracuje poměrně jednoduše; výhoda relačních modelů databází spočívají právě v jednoduchosti práce s relacemi.

Pro relace se zavádí především operace; ze základních operací uveďme projekci a spojení jako nejčastěji vyžadované operace.

Nechť P a Q jsou relace. Operaci x nazýváme projekcí tehdy, je-li P x Q relace, která vznikne z P vynecháním atributů P, které nejsou současně atributy Q, a následným vynecháním shodných řádků. Tato operace se používá pro zbavení se nepodstatných nebo v danou chvíli nedůležitých informací.

Příklad:

```
VRTY x (^VRT, LABORATOŘ) = (^VRT, LABORATOŘ)
```

Nechť P a Q jsou relace, které mají alespoň jeden shodný atribut. Operaci + nazýváme spojením tehdy, je-li P + Q relací, která vznikne z P přidáním atributů Q, které nejsou současně atributy P, a následným vynecháním shodných řádků. Spojení vytvoří relaci z hodnot atributů obou relací, které odpovídají hodnotám shodného (shodných) atributů.

Příklad:

```
VRTY + (^VRT, ^DATUM, L/SEC) = (^VRT, X, Y, LABORATOŘ, ^DATUM, L/SEC)
```

Tyto a další operace vytváří algebru relací. Kromě operací lze vytvářet další relace také relačním kalkulem. Přitom se využívá symboliky používané obdobně v jiných partiích matematiky:

Symbolika	Význam
P.x	Množina dat atributu x relace P
P (Q.x, R.y, ...)	Relace P nad hodnotami dat atributu x relace Q, atributu y relace R
p:v	Operátor výběru: platný, nabývá-li p hodnoty v
∃ , ∀	Existenční a všeobecný kvantifikátor
∨, ∧, ¬	Logické operátory nebo, a současně, negace

Pomocí zavedené symboliky a relačního kalkulu můžeme zavést např. relaci U, která popisuje množinu takových čerpacích měření všech vrtů, kde vydatnost přesahuje 0.5:

```
U (^VRTY.VRT, ^ČERPÁNÍ.DATUM, ČERPÁNÍ.L/SEC) : ČERPÁNÍ.L/SEC > 0.5
```

7.5 Normální formy

V předchozích odstavcích byl zaveden termín první normální forma. Příklady osvětlily způsob převodu relace na normalizovaný tvar. Při porovnání výsledků dvou různých normalizací téže relace (do jedné a do dvou relací) a ve smyslu zavedených operací nad relacemi je vidět, že druhý způsob spočívá ve vytvoření dvou projekcí na podmnožiny atributů se stejným informačním významem a ekvivalentním datovým obsahem. Stanovení vhodné (logické) reprezentace dané relace bude cílem následujících odstavců.

Další výklad je dokreslen následujícím příkladem. Nechť je dáno několik vrtů v několika lokalitách; vzorky vod byly podrobeny chemické analýze a výsledky byly shrnuty do relace

```
ANALÝZA (VRT, LOKALITA, LÁTKA, MNOŽSTVÍ)
```

Je-li tedy

```
<V12, Poruba, pH, 7>
```

z relace ANALÝZA, pak to znamená, že z vrtu V12, který je v Porubě, byl analyzován vzorek na pH a byla zjištěna jeho hodnota rovna 7.

Z příkladu je vidět, že

- kódem vrtu je jednoznačně určeno jeho umístění, jinak řečeno, ze zadaného kódu vrtu lze odvodit jeho umístění: VRT -> LOKALITA
- analyzované množství je jednoznačně určeno kódem vrtu a kódem látky, jinak řečeno, ze zadaných kódů vrtů a látky lze odvodit hodnotu: VRT, LÁTKA -> MNOŽSTVÍ
- naproti tomu ze samotné lokality nelze jednoznačně odvodit žádný jiný údaj, není tedy např. LOKALITA -> MNOŽSTVÍ.

Rozborem dalších možností lze odvodit, že jediným klíčem relace je {VRT, LÁTKA}; pouze tento klíč jednoznačně zpřístupňuje všechny údaje v řádku.

Mezi atributy relace se tedy mohou vyskytovat vazby, které jsou dány významem těchto atributů v reálném světě, jejich sémantikou. Formálně jsou tyto vazby označovány jako závislost.

Závislost lze definovat takto:

Nechť je dána relace R = <A, D, T>, nechť je B z A, C z A. Množinu atributů C v R nazveme **závislou** na množině atributů B v R, jestliže pro libovolné n z T, v z T platí: je-li n[b] z B pro všechna b z B, je n[c] = v[c] pro všechna c z C. Skutečnost, že C je závislá na B, je symbolicky označena B -> C.

Pomocí závislostí atributů je možno zachytit a formalizovat část sémantického významu atributů. O tom, zda v relaci platí závislost mezi atributy, se rozhodne podle tabulky relace. Často je však možné vyjít ze vztahů mezi atributy v popisovaném reálném světě. Takovým způsobem lze definovat např. klíč v relaci.

Nechť K je podmnožina A. K je pro R[A] **klíčem** relace R, jestliže (a) K -> A a dále (b) žádná vlastní podmnožina K nemá předchozí vlastnost.

Některé problémy mohou nastat při aktualizaci dat. Jestliže se v předchozím příkladu přestane provádět analýza pH ve vrtu V13, odstraní se z relace řádek pro V1 Tím se ovšem ztratí informace o existenci a umístění vrtu V13 (který ovšem reálně existuje dál). Pokud někde vznikne další vrt, nemůže se do relace přidat, dokud není známo, co a v jakém množství bylo analyzováno. Přejmenuje-li se lokalita, je třeba zaměnit starý údaj za nový na mnoha řádcích.

Tyto problémy jsou důsledkem toho, že LOKALITA závisí nejen na celém klíči {VRT, LÁTKA}, ale i na jeho části {VRT}. Základní myšlenkou, která z toho vyplývá, je vytvoření různých projekcí této relace a uchovávat je samostatně.

V uvedeném příkladu to znamená vytvořit projekce P1 a P2 relace ANALÝZA na množiny {VRT, LOKALITA} a {VRT, LÁTKA, MNOŽSTVÍ}.

Použitá metoda rozkladu se opírá o pojem závislosti. Každý krok rozkladového algoritmu záleží v nahrazení jedné relace dvěma jejími projekcemi, přičemž nesmí dojít ke ztrátě informace.

Lze dokázat, že z platnosti závislosti B -> C v relaci R plyne rozložitelnost (bez ztráty informace) R na své projekce R [B∪C] a R [B∪C'], kde C' je doplněk množiny C do A-B.

Definujme nejprve úplnou závislost takto:

Nechť R [A] je relace, nechť je B z A, C z A. C **úplně závisí** na B v R, jestliže C -> B a pro žádnou vlastní podmnožinu B1 z B není B1 -> C.

Nyní lze definovat druhou normální formu:

Relace R [A] je relace ve **druhé normální formě**, jestliže (a) R [A] je v první normální formě a dále (b) každý atribut a z A, jenž nepatří žádnému klíči R, úplně závisí na každém klíči K v R.

Shora uvedená relace ANALÝZA není ve druhé normální formě, protože má atribut LOKALITA, který nepatří žádnému klíči, funkčně závisí na jediném klíči {VRT, LÁTKA}, ale nezávisí na něm úplně (závisí totiž jen na jeho části VRT). Ovšem obě projekce P1 a P2 relace ANALÝZA už ve druhé normální formě jsou.

Nicméně i s relacemi ve druhé normální formě mohou nastat problémy. Uvažujme relaci VRTY (VRT, LOKALITA, SPAD), kde SPAD je jednotkový spad v dané lokalitě:

Jediným klíčem je {VRT}. Jestliže se však uzavře poslední vrt v Nové Vsi, ztratí se informace o spadu v dané lokalitě. Problém je zřejmě v existující závislostí LOKALITA -> SPAD; žádný z těchto atributů nepatří ke klíči relace. Tyto a obdobné obtíže se je možno odstranit přechodem k projekcím (zde na {VRT, LOKALITA}) a relacím ve třetí normální formě:

Řekneme, že relace R je ve **třetí normální formě**, jestliže (a) R je ve druhé normální formě a dále (b) množina všech atributů, které nepatří žádnému klíči (tj. množina neprimárních atributů) je nezávislá: žádný neprimární atribut nezávisí na některém z ostatních neprimárních atributů.

Uvedenou relaci VRTY lze projekcí převést na dvě relace UMÍSTĚNÍ (VRT, LOKALITA) a ZNEČIŠTĚNÍ (LOKALITA, SPAD). Toto rozdělení je konec konců i logické, protože spad závisí spíše na lokalitě jako takové než na samotném vrtu.

Elementy relace, která je ve třetí normální formě, mají následující strukturu: existují pro ně hodnoty klíčů (zpravidla klíče jediného), které tento element plně identifikují, a dále hodnoty atributů, které jistým způsobem elementy popisují. Tyto "popisné" hodnoty neprimárních (= neklíčových) atributů jsou na sobě nezávislé v tom smyslu, že žádná z nich není funkčně určena kombinací některých ostatních.

7.6 Návrh logické struktury

Shora uvedené normální formy mají důležitý praktický význam.

Především na úrovni řízení báze dat (= úroveň programů) je evidentně zpracování relací ve vyšší normální formě daleko jednodušší a tedy rychlejší než relací v nižší normální formě (popř. zcela nenormalizované). Přestože pro koncového uživatele je to irelevantní, je na místě připomenout, že jednodušší zpracování vyžaduje i jednodušší programování a je známo, že čím jednodušší je program, tím méně potenciálních chyb obsahuje.

Daleko důležitější je však otázka praktického zpracování dat uživatelem počínaje aktualizací a konče čerpáním informací. Údržba koncovým uživatelem dat, která jsou nenormalizovaná, je bez rizika porušení integrity dat nemožná bez obslužných programových nadstaveb, specificky programovaných pouze na tuto datovou strukturu. Toto riziko se - byť v menší míře - vyskytuje i v relacích ve druhé normální formě: přidání dalšího vrtu do relace VRTY předchozího odstavce znamená, že uživatel musí zadat s kódem vrtu a lokality také přesně stejnou hodnotu spadu jako u předchozích řádků téhož vrtu! Teprve relace ve třetí normální formě plně odstraňují (až na zřejmé klíče) redundanci dat.

Protože dnes mají koncoví uživatelé obecných systémů řízení báze dat možnost plně definovat, plnit a aktualizovat své vlastní báze dat, je vhodné závěrem uvést obecný postup normalizace již ve fázi návrhu logické struktury:

- Nenormalizovaný tvar → první normální forma
  - rozložení všech datových struktur, které nejsou dvourozměrné, na dvourozměrné
- První normální forma → druhá normální forma
  - odstranění neúplných závislostí neprimárních atributů na potenciálních klíčích
- Druhá normální forma → třetí normální forma
  - odstranění závislostí neprimárních atributů na sobě

Návrh logické struktury báze dat založené na relačním modelu tedy spočívá v identifikaci všech atributů postihujících existující objekty, a vzájemných vztahů mezi těmito atributy. Na základě této identifikace je třeba - s jistou dávkou opatrnosti - rozhodnout o klíčích relací.

Relační systémy jsou založeny na formalizovaných pojmech používajících matematický aparát. Lze vytvořit analytické nástroje pro automatizaci shora popsaných identifikačních činností a návrhu výsledných relací ve třetí normální formě. Ukázalo se však, že tyto nástroje jsou - zvláště díky moderním propracovaným systémům řízení bází dat - pro koncové uživatele vcelku zbytečné.

Díl II: Dotazovací jazyk

8 Jazyk SQL

Problematika uložení a zpracování dat - jak je možno vidět ze všech předchozích kapitol - je nesmírně rozsáhlá: od fyzického uložení dat po logické struktury, od sekvenčního přístupu po přístup přímý atd.

U koncového uživatele - nepočítačového odborníka však není možno očekávat, že se nejprve stane odborníkem přes data. Proto je pochopitelná snaha vytvářet pro tyto uživatele jednak formalizované, jednak pokud možno normalizované nástroje pro zpracování dat v počítačovém prostředí jednotným způsobem s tím, že uživatel není zatěžován otázkami uložení, organizace apod.

Pokusů o vytvoření jednotného alespoň dotazovacího prostředí bylo učiněno několik. Faktem zůstává, že - zvláště při velkých objemech dat - jistý stupeň přehledu uživatele o datové problematice je vyžadován u všech takových prostředí. Čím vyšší stupeň přehledu o datech uživatel má, tím optimálněji je schopen zajistit jejich sběr a zpracování, a tím mohutnějším nástrojem se pak takový prostředek pro něj stává.

Jedním z poměrně zdařilých a v mnoha různých (i operačních) systémech implementovaným nástrojem pro zpracování dat je SQL (z plného anglického označení Structured Query Language; toto označení se většinou ani nepřekládá a používá se jen SQL).

Oproti jiným podobným obecným nástrojům má SQL nejen funkci dotazovací (tj. pro čerpání dat z existujících databází), ale i funkci úplného zpracovatele dat: vytváření nových databází a tabulek o definované struktuře, plnění daty, změny dat ap. Jak už je patrno z názvu, jde v principu o formalizovaný "počítačový" jazyk definovaný svými jednotlivými příkazy. Přestože obsahuje příkazy pro tvorbu komplexních programů, s ohledem na zaměření těchto výukových textů bude otázka programování zcela pominuta. V dalším budou zmíněny jen příkazy použitelné relativně jednoduchým způsobem pro jednotlivé, přesně definované akce.

> ***Poznámka:*** *Anglický termín "Query" znamená "Dotaz", SQL tedy doslova znamená "Strukturovaný dotazovací jazyk". Jazyk je ovšem tvořen příkazy, jejichž vydáváním uživatel přikazuje bázi dat provedení konkrétní akce s konkrétními parametry. Přesto se rozmohla podivná praxe označovat i každý jednotlivý příkaz jazyka termínem "Dotaz". Z toho pak plynou takové kuriózní nesmysly jako např. v české mutaci programu Access: VYTVÁŘECÍ DOTAZ. To uživatele staví do role poníženého prosebníka, který se dotazuje: Databáze databáze, byla bys tak hodná a vytvořila mně novou tabulku? A databáze třeba odpoví: teď ne, je horko a nechce se mně :-)*

Množinu příkazů, kterou se tyto texty zabývají, lze rozložit na třídy podle jejich funkce. K označování těchto tříd bývá zvykem používat následující označení:

1. DDL - Data Definition Language - jazyk pro definici dat, přesněji příkazy pro práci se strukturou dat. V těchto textech jsou některé z nich zmíněny zvláště pro ukázku práce s datovými typy.
2. DML - Data Manipulation Language - jazyk pro manipulaci s daty, přesněji příkazy pro změnu dat. V těchto textech jsou uvedeny stručné ukázky pro představu způsobu aktualizace dat.
3. DCL - Data Control Language - jazyk pro řízení dat, přesněji příkazy pro práci s přístupovými právy, řízením transakcí apod. Touto skupinou se tyto texty nezabývají.

> ***Poznámka:*** *Na rozdíl od většiny publikací je zde příkaz SELECT vyčleněn ze třídy DML a diskutován samostatně. Jako jediný totiž (tedy spolu se zde neuváděným příkazem TRANSFORM) po svém provedení předává uživateli, který ho vydal, data ve formě množiny záznamů (= record set). A jako jediný vlastně přichází v úvahu pro použití absolventy našich oborů v praxi - těžko si lze představit majitele firmy, který by nechal svého např. hlavního geologa příkazy SQL měnit a rušit data své firemní databáze.*

V současné době většina implementací SQL pracuje nad relačními databázemi (tj. mající podobu tabulky - viz příslušná kapitola shora). Odtud také zástupná klíčová slova typu "table". Protože však existuje poměrně snadná

(protože mechanická) konverze hierarchické (ne síťové) struktury na relační, některé SQL mají možnost pracovat i na hierarchických organizacích datových bází.

V této kapitole je popsána logika a nástin použití základních příkazů SQL. Protože jde o poměrně širokou problematiku, u které je předpoklad plného využití, je nutno odkázat na příslušné referenční manuály. Proto také jsou v této kapitole jednotlivé příkazy uváděny již na konkrétních příkladech a nejsou popisovány ani syntakticky úplně. Všechny příklady jsou předkládány na části datového modelu uvedeného v předchozích kapitolách. Jde o data, jejichž strukturu a částečně obsah ukazují následující schémata:

VRTY

KOD	MISTO	X	Y	Z	PRUMER	HLOUBKA	TYP_CERP	PRIKON	VYKON
A12	Janov	485	1202	322	20	860	C-7a	2500	26
A18	Popice	477	1203	305	20	920	C-7a	3000	32
C22	Cerkev	481	1195	242	15	750	Brum	2000	11
S11	Malkov	480	1199	298	22	800	AVG	2500	29
A45	Borek	479	1207	331	15	840	SL1T	1800	17
B02	Mostek	475	1196	240	25	980	Brum	2000	11
B17	Svatava	483	1205	362	20	870	SL1T	1800	17
B18	Budov	486	1208	377	15	940	SL1T	1800	17

SOUVRSTVI

KOD	NAZEV
O	Ostravské souvrství
K	Karvinské souvrství

VRSTVY

SOUVRSTVI	KOD	STRED_MOC	PLOCHA
O	R02	42	6500
O	R05	17	7100
O	R07	23	6100
O	R10	11	5900

VZORKY

SOUVRSTVI	VRSTVA	VRT	HLOUBKA
O	R02	A18	560
O	R02	A18	598
O	R02	A18	623
O	R05	A18	744
O	R05	A18	768
O	R07	A18	977
O	R02	B02	512
O	R05	B02	715
O	R05	B02	733
O	R05	B02	749

ČERPÁNÍ

VRT	DATUM	HLADINA	MNOZSTVI
A12	33615	35	17
A12	33620	38	17
A12	33622	36	19
A12	33623	37	18
A18	33616	48	26
A18	33619	52	29

Poznámka: *z technických důvodů nejsou ve shora uvedených tabulkách šířky sloupců opticky ekvivalentní jejich skutečným hodnotám používaným v příkladech.*

8.1 Vymezení dat

Protože uspořádání dat je chápáno "tabulkově", jsou zavedeny ve zřejmém smyslu tyto pojmy: jméno sloupce, šířka sloupce, typ sloupce. Konkrétně typ sloupce je typem každého z jednotlivých údajů, které sloupec obsahuje (viz doména a její atributy v kapitole o relačních databázích).

SQL pracuje obecně s typy dat a jejich kódováním dle následujících odstavců. Různé implementace mohou zavádět další typy údajů; níže uvedené čtyři typy však postačují pro většinu aplikací běžných aplikací. Podrobnosti o konkrétní implementaci je pak třeba hledat v dokumentaci konkrétního databázového prostředí. Podrobněji o typech dat v prostředí Microsoft viz kapitola "Datové typy při provozu databází v prostředí Microsoft" níže.

> **Důležitá poznámka:** Pro skutečně fundované zvládnutí nejen databázové, ale obecně informatické problematiky je zcela nezbytný přehled o interním uložení dat v digitálním prostředí a o práci s nimi. Úvodní informace z této oblasti lze najít např. v [6].

8.1.1 Numerická data

Jde o číselná data v běžném pojetí. Pro větší variabilitu použití jsou dále dělena na dva základní pod-typy: čísla celá a ta čísla racionální, která mají konečnou "desetinnou" část. I tyto pod-typy ještě mohou být jemněji děleny podle velikosti paměti potřebné pro uložení jejich hodnot; tím je de facto dán rozsah jejich možných hodnot.

8.1.2 Znaková (textová) data

Jakékoliv texty, obsahující jakékoliv znaky podle kódových tabulek znaků. Charakteristikou je jejich šířka - maximální počet znaků (nikoliv počet potřebných bytů paměti).

8.1.3 Datumová resp. i časová data

Kalendářní datum bývá zavedeno jako samostatný typ proto, že jsou na něm definovány přirozeným způsobem operace přičítání čísla, odečítání čísla a odečítání dvou datumů, avšak tyto operace nejsou "běžné" aritmetické operace (např. přechod přes konec měsíce nebo roku). Potřebná paměť je nejčastěji 8 bytů.

8.1.4 Logická data

Dvouhodnotová data: ANO a NE. Používají se jako indikativní ukazatele apod. Potřebná paměť je nejčastěji 1 byte.

8.2 Datové typy při provozu databází v prostředí Microsoft

Zcela zásadní je v konkrétním databázovém systému přesné určení množiny množin, které poskytují své prvky (= hodnoty) do n-tic (= záznamů) libovolné relace (= tabulky), kterou lze v systému vytvořit a udržovat.

Firma Microsoft se významně podílela na budování jak databázových prostředí obecně, tak konkrétních implementací. Ve své řadě systémů známých jako nejrůznější "Wokna" a "Oficy", provozovaných nejčastěji na strojích řady PC bázovaných procesorem (a matematickým koprocesorem) Intel, staví na hardwarových typech s pochopitelných důvodů: instrukce procesoru zpracovávající hardwarové typy dat jsou evidentně nejrychlejším a nejoptimálnějším nástrojem pro použití v databázových programech.

Na druhé straně se stejná firma musela zabývat přístupem k datům jiných systémů. Jejich autoři však stáli před stejným problémem, a to naštěstí vedlo k jisté unifikaci v realizacích datových modelů a tedy i datových typů.

Obecným nástrojem pro správu databází a čerpání informací se postupem času stal shora uvedený dotazovací jazyk SQL (Structured Query Language). Pomocí příkazů jazyka lze tabulky databází především vytvořit a upravit. Příkazy tedy musí mít možnost označit typ dat konkrétního sloupce případně jeho další atributy. Tak např. příkaz

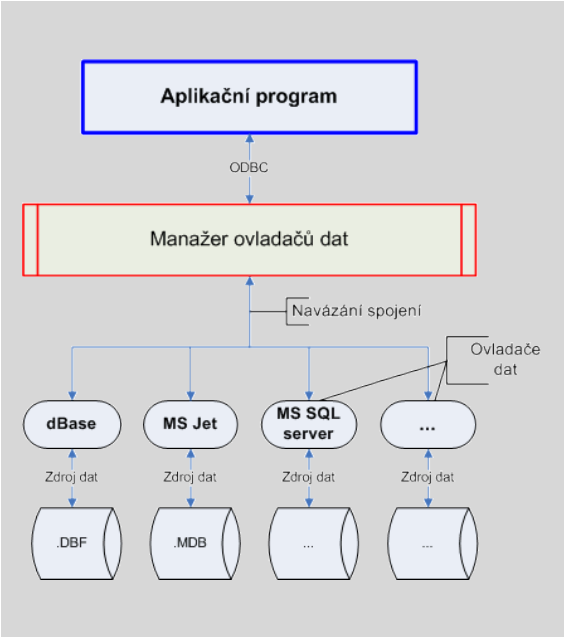
```
create table VYDAJE (DATUM date, CENA numeric, NAKOUPENO text(30))
```

vytváří tabulku VYDAJE se třemi sloupci (DATUM, CENA, NAKOUPENO), přičemž ve sloupci DATUM se budou uchovávat datumové, ve sloupci CENA číselné a ve sloupci NAKOUPENO textové údaje. Typy dat jsou v příkazu označeny klíčovými slovy **date**, **numeric**, **text**.

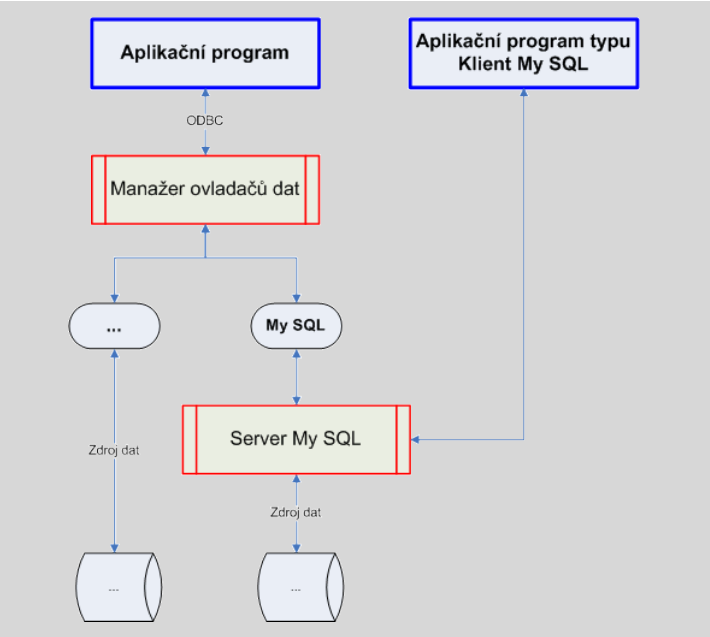
V jednom (operačním) systému se běžně může uchovávat několik databází (= zdrojů dat) různých formátů. Je to dáno historicky (např. dBase - formát .DBF), požadavky na maximální zabezpečení (servery na úrovni operačního systému) apod. Příkazy jsou vydávány jádru konkrétního databázového systému, který je interpretuje, pomocí driverů - řadičů příslušných ovladačů dat (viz obrázek vlevo níže). Na obrázku vpravo je pak znázorněn případ, kdy je možno požadovat data i od běžících serverů, ne nutně téhož (operačního) systému.

Pro obecné použití jazyka a pro přenositelnost je žádoucí, aby pokud možno všichni poskytovatelé dat přijímali tentýž tvar příkazů včetně klíčových slov. Při použití klíčových slov ve významu typů hodnot je naopak nežádoucí, aby stejná klíčová slova označovala jinou přijímanou množinu hodnot nebo jiná omezení na ni kladenou.

Přístup k datům v systémech s databázovým jádrem



Přístup k datům v systémech i se servery SQL - příklad pro MySQL



V současné době - z pohledu uživatelů systémů Microsoftu - lze pozorovat několik jemně se odlišujících verzí jazyka SQL.

- Především je to ta verze, která by měla být závazná pro všechny interprety - verze popsaná americkou ANSI normou a označovaná jako ANSI SQL. Dále bude označována jako *A-SQL*.
- Dále je to verze, kterou Microsoft používá pro své vlastní speciální databáze, obhospodařované "databázovým tryskovým strojem" - Database Jet Engine, a kterou nazývá Jet SQL. Dále bude označována jako *J-SQL*.
- Konečně je to verze, kterou - opět Microsoft - používá pro zpracování dat v rámci svého SQL serveru a kterou nazývá MS Server SQL. Dále bude označována jako *S-SQL*.
- Z jiných serverů než Microsoftu uveďme server MySQL hojně v Česku používaný, protože jeho základní verze je zadarmo. Verze jeho SQL bude označována jako *M-SQL*.

***Pozn.**: Databázový program Access z Microsoft Office používá implicitně J-SQL.*

8.2.1 Obecné typy

Do databází různých systémů lze čerpat hodnoty z následujících - většinou instrukcemi (ko-)procesoru zpracovatelných - množin [v hranatých závorkách jsou tučně uvedena klíčová slova typu použitelná v dané verzi SQL; je-li jich více, jsou to synonyma].

***Poznámka**: Verze M-SQL má hodnoty některých typů mírně odlišné od ostatních (např. minimální datum je 1.1.1000 místo jinak uváděného 1.1.100).*

8.2.1.1 Bajt

Množina 256 hodnot celých čísel z intervalu <0,255>. Každá hodnota obsazuje 1 byte. [A-SQL: nemá. J-SQL: **byte**. S-SQL: **tinyint**. M-SQL: **tinyint unsigned** - dvě klíčová slova.]

8.2.1.2 Celé číslo

Množina 65 536 hodnot celých čísel z intervalu <-32 768, +32 767>. Každá hodnota obsazuje 2 byty. [A-SQL: **smallint**. J-SQL: **smallint**, **short**. S-SQL: **smallint**. M-SQL: **smallint**.]

8.2.1.3 Dlouhé celé číslo

Množina 4 294 967 296 hodnot celých čísel z intervalu <-2 147 483 648, +2 147 483 647>. Každá hodnota obsazuje 4 byty. [A-SQL: **integer**. J-SQL: **integer**, **long**. S-SQL: **integer**. M-SQL: **integer**, **int**.]

8.2.1.4 Racionální číslo v jednoduché přesnosti

Množina 4 294 967 296 hodnot racionálních čísel z intervalu <-3.402823 x 10^{38}, +3.402823 x 10^{38}>. Každá hodnota obsazuje 4 byty. Nejmenší kladné číslo různé od nuly je 1.401298 x 10^{-45}. [A-SQL: **real**. J-SQL: **real**, **single**. S-SQL: **real**. M-SQL: **float**.]

8.2.1.5 Racionální číslo ve dvojnásobné přesnosti

Množina zhruba 1.845 x 10^{19} hodnot racionálních čísel z intervalu <-1.79769313486231 x 10^{308}, +1.79769313486232 x 10^{308}>. Každá hodnota obsazuje 8 bytů. Nejmenší kladné číslo různé od nuly je 4.94065645841247 x 10^{-324}. [A-SQL: **float**. J-SQL: **float**, **double**. S-SQL: **float**. M-SQL: **double**, **real**.]

8.2.1.6 Text

Množina textových řetězců, z nichž každý obsahuje žádný, jeden až 255 znaků. Jeden znak je přitom uložen pod svým kódem dle kódové tabulky znaků. [A-SQL: nemá. J-SQL: **text**. S-SQL: **text**. M-SQL: **tinytext**.]

8.2.1.7 Memo

Množina textových řetězců, z nichž každý obsahuje žádný, jeden až 65 534 znaků. Jeden znak je přitom uložen pod svým jednobytovým kódem dle kódové tabulky znaků. Na rozdíl od předchozího typu Text může Memo obsahovat i znaky pro odřádkování. [A-SQL: nemá. J-SQL: **memo**. S-SQL: nemá. M-SQL: **text**.]

8.2.1.8 Date

Množina kalendářních datumů počínaje 1.1. roku 100 a konče 31.12. roku 9999. Každé takové datum je pak representováno hodnotou typu Double (viz výše) - tedy racionálním číslem ve dvojnásobné přesnosti, přičemž celá část takového čísla je pořadové číslo dne na časové ose počínaje 30.12.1899 (to je den s pořadovým číslem 0). Každý následující den má tedy pořadové číslo o 1 větší než den předcházející. Desetinná část racionálního čísla vyjadřuje zlomek dne, tedy čas v daném dni (např. 0.75 = 3/4 nějakého dne je 6 hodin večer). Dvojnásobnou přesností uložení je dána i přesnost zaznamenání časového údaje - kolem roku 2000 to je zhruba 1 x 10^{-7} vteřiny. [A-SQL: **date**. J-SQL: **date**, **datetime**. S-SQL: **datetime**. M-SQL: **date**.]

Raritou je ve firmě Microsoft tento fakt: počátek datumů v programu Access této firmy je skutečně 30.12.1899. Ovšem počátkem datumů v programu Excel téže firmy je 31.12.1899 (tj. o jeden den víc), přičemž konverze dat mezi oběma programy probíhá v pořádku - prostě odečtou nebo přičtou jedničku. Proč si raději nedali tu práci se sjednocením, to snad neví ani kolega Gates.

8.2.1.9 Logical

Množina dvou hodnot obvykle označovaných {Ano, Ne}, {True, False}, {0, 1}, {Yes, No} apod. Teoreticky stačí pro uchování jedné hodnoty této množiny jeden bit, z praktických důvodů se často používá jeden byte s tím, že nulový byte většinou reprezentuje hodnotu Ne, všechny ostatní hodnoty Ano. [A-SQL: nemá. J-SQL: **bit**, **logical**, **yesno**. S-SQL: **bit**. M-SQL: **bit**.]

8.2.2 Další typy prostředí Windows

Do databází lze čerpat hodnoty i z dalších množin. Ty však už mohou být systémově závislé: zatímco hodnota některé množiny je v jednom (zde operačním) systému podporována, ve druhém ne - nebo je podporována jinak. Níže jsou uvedeny ty množiny, které podporují systémy řady Windows firmy Microsoft.

8.2.2.1 Objekt OLE

Množina objektů OLE (Object Linking and Embedding), každý o velikosti žádný (= null, nothing), jeden až zhruba 2 mld. bytů. Pod pojmem OLE se rozumí unifikované prostředí poskytující objektově orientované služby. OLE objekt je tedy objekt využívající služeb tohoto prostředí - např. obrázky, dokumenty apod. Je zřejmé, že služby jsou vázané na konkrétní operační systém a databáze s OLE objekty nejsou obecně mezi systémy přenositelné. [A-SQL: nemá. J-SQL: **image**, **longbinary, oleobject**. S-SQL: **image**.]

8.2.2.2 Binární

Množina (binárních) hodnot na jednom až 510 bytech. Tyto hodnoty nejsou při vstupu a výstupu nijak upravovány (např. překladem podle kódové tabulky): jak vstoupí, tak také vystoupí. [A-SQL: **bit**. J-SQL: **binary**. S-SQL: **binary**. M-SQL: nemá.]

8.2.2.3 Automatické číslo

Množina 2 147 483 647 celých čísel počínaje 1. Z této množiny databázový systém přiděluje hodnoty do databáze jednak automaticky, jednak jedinečně. [A-SQL: nemá. J-SQL: **counter**. S-SQL: nemá. M-SQL: nemá.]

8.2.2.4 Měna

Množina hodnot z intervalu <-922 337 203 685 477.5808, +922 337 203 685 477.5807> na 8 bytech. Jde o desetinná čísla vždy se čtyřmi desetinnými místy. [A-SQL: nemá. J-SQL: **money**, **currency**. S-SQL: **money**. M-SQL: do jisté míry lze použít **bigint**.]

8.2.2.5 Identifikační číslo

Množina celých kladných čísel s binárním uložením na 16 bytech. Jedná se o jedinečný identifikátor každého objektu, který se registruje do prostředí Windows firmy Microsoft, známé GUID (Globally Unique IDentifier). Je generován algoritmem zaručujícím jeho jedinečnost. [A-SQL: nemá. J-SQL: **guid**. S-SQL: nemá. M-SQL: nemá.]

8.3 Tvar a zápis jmen

Pro označování (a rozlišování) datových polí, tabulek, dotazů ap. se používá **name**. Záměrně uvádíme nejprve originální anglický termín; vychází zvláště z objektového programování, na němž je dnes založena naprostá většina software a datových modelů, se kterými software pracuje. Do češtiny je samozřejmě překládán jako **jméno**, ale je to trochu složitější.

Prapůvod má tento pojem v počátcích programování, kdy vyvstala potřeba určení konkrétního místa v paměti místo číselnou adresou nějakým vypovídajícím symbolickým označením. V intencích tehdejších programovacích jazyků se pro takové označení začal používat (a dodnes používá) pojem **identifikátor**, definovaný takto:

```
Identifikátor je posloupnost písmen a číslic začínající písmenem.
```

Tedy např. Alfa, Beta12 - ale nikoliv 12Beta. Definice nepřipouští skutečně nic jiného, zvláště mezeru uvnitř identifikátoru, pomlčku apod. Protože celý vývoj programování se odehrával především v anglicky mluvících zemích, nemůže být znakem identifikátoru znak s diakritikou (to v angličtině není písmeno).

Identifikátory se tedy začaly hojně používat pro to, co dalo tomuto pojmu vzniknout - pro identifikaci. Postupně se jím začaly označovat nejen adresy paměti, ale i složitější logické konstrukce a jejich části. Tam vznikl obecnější pojem, **jméno**; stanovilo se, že bude mít tvar identifikátoru. Jméno se dnes používá jednak k symbolickému označování, jednak k jedinečné identifikaci konkrétního objektu v množině stejných typů (např. konkrétního datového pole v množině datových polí tabulky).

Striktní definice identifikátoru začala být omezující v okamžiku, kdy se připustila tvorba identifikátoru (tedy jmen v uvedeném smyslu) laiky, tedy obecně komukoliv. První rozšíření definice spočívalo v tom, že jako číslice se začal

připouštět i znak _ (podtržítko). *Uvnitř* identifikátoru se tedy lze zapsat podtržítko a tím se alespoň opticky oddělí dvě nebo více "částí" identifikátoru.

Nicméně laických uživatelů toho, co se dnes obecně označuje "výpočetní technika", přibývalo a dnes tvoří naprostou většinu. Nutno vidět, že oni tu techniku kupují a jsou tedy za nimi obrovské peníze. A právě jim se muselo nějak vyjít vstříc ohledně pojmenování, která si oni určí s ohledem na čitelnost v jejich jazycích.

Proto došlo k rozšíření pravidla pro zápis jména. Jméno má především tvar identifikátoru. Obsahuje-li navíc kromě číslic a anglických písmen např. mezeru nebo písmena **v národních abecedách**, pak je nutno takovou posloupnost znaků uzavřít do hranatých závorek.

Jméno sloupce tedy může být

```
DenNakupu
Den_Nakupu
[Den Nákupu]
```

Konvenci uzavírání znaků jména do hranatých závorek je možno v mnoha např. databázových systémech používat i nadbytečně - tedy je psát i v případě, že by tam být nemusely:

```
[DenNakupu]
```

To používá např. i editor dotazů programu Access: často se setkáme se situací, kdy v našem zápise hranaté závorky nepoužijeme (protože tam být nemusí), ale editor je tam ve finále přidá. Z hlediska funkčnosti to nevadí, ale může to znepřehlednit čitelnost zápisu našich konstrukcí.

8.4 Hodnota NULL

Při zpracování dat umístěných v datových polích je třeba mít nástroj na zjištění takového stavu datového pole, kdy datové pole neobsahuje žádnou hodnotu. To řeší relační databáze zavedením označení **NULL** - nikoliv nula, ale ve smyslu prázdný, neplatný.

Na tuto hodnotu lze obsah datového pole testovat, nikoliv však operátorem **=** (rovnítko), ale operátorem **is** (dvě písmena **i** a **s** vedle sebe, bez mezery, 3. os. j.č. od to be), na velikosti písmen nezáleží. Správně je tedy

```
ZAJEZD.CENA is NULL
```

zatímco nesprávně je

```
ZAJEZD.CENA = NULL
```

přesněji: syntakticky to nesprávně není, ale vždy to bude vyhodnoceno jako NEPRAVDA.

Pro informaci o nenaplnění datového pole lze rovněž použít funkci **IsNull**:

```
IsNull (ZÁJEZDY.CENA)
```

Výsledkem volání je logická hodnota True (Ano, Pravda), jestliže datové pole, které je parametrem, je prázdné, nenaplněné nějakou reálnou hodnotou.

Výsledkem volání je logická hodnota False (Ne, Nepravda), jestliže datové pole, které je parametrem, není prázdné, tedy obsahuje nějakou reálnou hodnotou příslušného typu.

Funkci **IsNull** stejně jako operátor **is** lze v dotazech použít především v podmínkách při výběru řádků do výsledku zpracovávaného dotazu aj.

8.5 Příkazy třídy DDL

Poznámka: *V této kapitole jsou použity identifikátory typů dat (date, integer ...) podle kapitoly "Datové typy při provozu databází v prostředí Microsoft" uvedené výše. Byly zvoleny identifikátory používané v Microsoft Jet Database; při více synonymech pak ty, které je možno použít i v jiných mutacích než MS Jet.*

8.5.1 Vytvoření nové tabulky

Vytvořením nové tabulky se ve většině systémů rozumí založení nového datového úložiště s daným jménem, který obsahuje dvě části: jednak popis jednotlivých sloupců tabulky, jednak vlastní data tabulky organizovaná do řádků. Bezprostředně po vytvoření nové tabulky je druhá část prázdná (zatím nejsou vložena žádná data).

Vytvoření tabulky ČERPÁNÍ provede následující příkaz:

```
create table ČERPÁNÍ (VRT text(5), DATUM date, HLADINA integer, MNOŽSTVÍ integer)
```

Po klíčových slovech CREATE TABLE se tedy uvede jméno tabulky a za ním, uzavřený v kulatých závorkách, seznam popisů jednotlivých sloupců tabulky (=polí řádků). Každý popis obsahuje jméno sloupce následované klíčovým slovem ve smyslu typu event. s uvedením maximální šířky.

Příkaz

```
create table ČERPÁNÍ (VRT text(5) primary key, ...)
```

provede totéž, ale navíc pro tabulku definuje tzv. primární přístupový klíč, jehož hodnotami budou kódy vrtů zařazovaných řádků dat.

8.5.2 Úprava struktury tabulky

Úpravou struktury se rozumí přidání nebo vypuštění sloupce tabulky. Bez bližších komentářů jsou uvedeny příklady charakteristického tvaru:

```
alter table ČERPÁNÍ add column VYDATNOST real
```

přidá do tabulky ČERPÁNÍ nový sloupec VYDADNOST.

```
alter table ČERPÁNÍ alter column VYDATNOST float
```

změní v tabulce ČERPÁNÍ typ dat ve sloupci VYDATNOST.

```
alter table ČERPÁNÍ drop column VYDATNOST
```

vypustí sloupec VYDATNOST z tabulky ČERPÁNÍ.

8.6 Příkazy třídy DML

8.6.1 Naplnění (doplnění) tabulky daty

Do existující tabulky (např. právě vytvořené) je možno přidávat data. Do shora vytvořené tabulky ČERPÁNÍ se přidají nové řádky příkazem např.

```
insert into ČERPÁNÍ values ('V12', #12/02/97#, -122, 78)
```

Příkaz přidá na konec tabulky nový řádek dat a naplní ho hodnotami uvedenými jako seznam v kulatých závorkách. Data musí být uvedena v takovém množství, kolik je polí, a musí následovat v tom pořadí, jak byla pole uvedena při vytvoření tabulky. Datum se předpokládá v americkém formátu - zde tedy nikoliv 12. února, ale 2. prosince.

8.6.2 Změna údajů v tabulce

Aktualizaci existujících dat v existující tabulce provádí příkaz UPDATE. Např. zvýšení množství čerpání o 0.1 u všech řádků s hladinou pod 100 provede příkaz

```
update ČERPÁNÍ set MNOŽSTVÍ = MNOŽSTVÍ+0.1 where HLADINA < 100
```

Vynulování hladiny a množství u některých řádků provede příkaz

```
update ČERPÁNÍ set MNOŽSTVÍ = 0, HLADINA = 0 where VRT = 'V12'
```

Výrazem za klíčovým slovem WHERE se určuje množina řádků, pro které se má náhrada (=aktualizace) provést - zde tedy pro všechny údaje vrtu V12.

8.6.3 Vypuštění řádků tabulky

Otázka vypouštění řádků je poněkud složitější. Příkaz

```
delete from ČERPÁNÍ where HLADINA < 100 and MNOŽSTVÍ > 1
```

doslova přeložen vypustí z tabulky ČERPÁNÍ řádky, kde je hladina pod 100 a množství nad 1. Problém spočívá v nebezpečnosti akce. Pokud (např. omylem) by byly vypuštěny jiné řádky než uživatel skutečně vypustit chtěl, obtížně by se zajišťovala náprava.

Některé implementace SQL provedou nikoliv fyzické vypuštění, ale pouze označení řádků za neplatné. Toto označení lze především příkazem typu RECALL odstranit, takže se žádné řádky neztratí. Dále: při jakémkoliv dalším zpracování se takto označené řádky nebudou vůbec brát v úvahu. Při výběrech dat, při aktualizaci atd. to bude vypadat tak, jako by tam tyto řádky skutečně nebyly.

Fyzické zrušení (vypuštění, odstranění) řádků z databáze pak provede speciální, k tomu určený příkaz, který většinou závisí na implementaci. Je-li např. SQL implementováno v prostředí xBase, je tímto příkazem PACK.

8.7 Čerpání údajů z databází

Tento odstavec je stěžejním odstavcem celé kapitoly. Zatímco organizaci databází (jejich zakládání a aktualizaci) - zvláště informačních systémů větších rozsahů - provádí většinou specializovaní nebo alespoň zvlášť určení pracovníci, čerpat z databází by měli mít potřebu všichni.

K získávání informací z existujících bází dat slouží v SQL dva příkazy - SELECT a TRANSFORM. Zde se soustředíme jen na příkaz SELECT. Jde o velmi silný, poměrně obecný a tedy poněkud složitější příkaz. Proto bude vysvětlen v jednotlivých krocích.

Jako příklad budeme uvažovat uvedenou část modelové databáze geologických souvrství. Konkrétně půjde o tabulky VZORKY, VRSTVY, SOUVRSTVÍ.

Obecně lze říci, že příkaz SELECT vybere určené údaje z určených databází (tabulek) a vytvoří tabulku novou. Může jít o tabulku fyzickou (tj. soubor na médiu) nebo častěji tabulku dočasnou (která po použití zanikne).

8.7.1 Specifikace zdroje datového pole

Při současném zpracování dvou nebo více datových zdrojů může nastat případ, že dvě datová pole v *různých* datových zdrojích jsou pojmenována *stejným* jménem. Je tedy třeba rozlišit, ze kterého datového zdroje dané datové pole pochází.

V dotazech to je řešeno následovně. Nechť např. v nějaké tabulce NÁKUP je datové pole CENA a v nějaké jiné tabulce ZBOŽÍ je rovněž datové pole CENA. Použití pole CENA z tabulky NÁKUP zajistí konstrukce tvaru

```
NÁKUP.CENA
```

kdežto použití pole CENA z tabulky ZBOŽÍ zajistí konstrukce tvaru

```
ZBOŽÍ.CENA
```

Obecně tedy jde o konstrukci tvaru

```
ZDROJ.POLE
```

která se v případě dvou a více zdrojů u stejně pojmenovaných polí použít musí, kdežto v jiných případech použít nemusí. Část "ZDROJ." bývá označována jako **specifikace zdroje datového pole**.

Specifikaci zdroje datového pole je možno použít i v případě *jediného* datového zdroje. V něm se dvě stejně pojmenovaná datová pole vyskytovat nemohou, a tedy použití specifikace datového pole je jaksi nadbytečné. Tyto

"nadbytečné" specifikace např. generuje editor dotazů programu Access do příkazů SQL - a tím je mimochodem činí méně přehlednými, než by mohly pro jediný datový zdroj být.

8.7.2 Alias

Pro účely jednoho každého *dotazu* je možno každý datový zdroj (i kdyby byl jen jeden) dočasně pojmenovat jinak - lze mu přidělit tzv. **alias** (= zástupné jméno: Jan Novák alias Černá Ruka). Pokud se např. tabulce ÚČASTNÍCI přidělí alias LIDÉ, pak použití pole JMÉNO zajistí místo ÚČASTNÍCI.JMÉNO konstrukce tvaru

```
LIDÉ.JMÉNO
```

Je třeba znovu zdůraznit, že jde o dočasné jiné pojmenování datového zdroje, nikoliv o přejmenování. Alias se uplatní jen při provádění toho dotazu, kde je zaveden; po ukončení tohoto dotazu přestává existovat. Pro vytvoření jména pro alias platí stejná pravidla jako pro jména tabulek, dotazů ap. Protože se uplatní jen v rámci konkrétního dotazu, bývá zvykem z důvodu přehlednosti jméno vytvářet co nejkratší; je-li to jediný znak, musí to být písmeno.

Přidělení alias se zajistí v textu příkazu SQL v klauzuli *from* následovně:

```
... from ÚČASTNÍCI LIDÉ ...
```

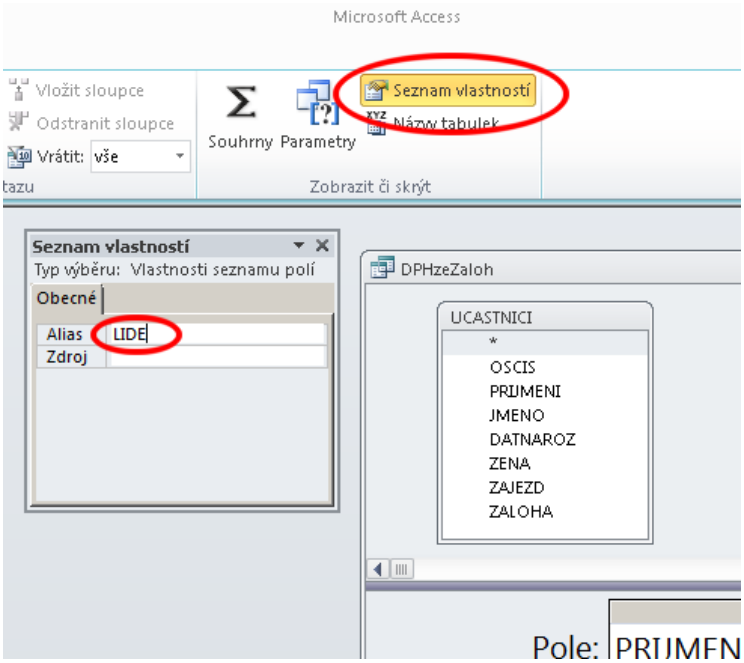
Mnoho interpretů SQL přijímá i lépe vnímaný tvar

```
... from ÚČASTNÍCI as LIDÉ ...
```

V této souvislosti předběhneme (viz odstavec *Dotazy pro laiky* v kapitole *Dotazy* níže). Přidělení alias se zajistí v návrhovém prostředí programu Access následovně (příklad pro přidělení alias LIDÉ tabulce ÚČASTNÍCI):

- Aktivuje se příslušný datový zdroj v návrhovém prostředí dotazu (zde tabulka ÚČASTNÍCI).
- Stiskne se tlačítko Seznam vlastností na kartě Zobrazit či skrýt.
- Tím se aktivuje formulář Seznam vlastností.
- V textovém poli Alias se přepíše původní jméno požadovaným alias (zde LIDÉ).
- Formulář je pak možno zavřít.

Postup znázorňuje následující obrázek:



Poznámka: Shora bylo zmíněno, že datovému zdroji je *možno* přidělit alias. Existují případy, kde je to nejen možno, ale i *nutno*. Takové případy však překračují záměr této publikace.

8.7.3 Údaje z jedné tabulky

Mějme tedy tabulku VZORKY, která má pole SOUVRSTVI, VRSTVY, VRT, HLOUBKA. Následně jsou uvedeny nejjednodušší tvary příkazu SELECT.

```
select * from VZORKY
```

Vybere všechny sloupce ze všech řádků tabulky VZORKY.

```
select VRT, HLOUBKA from VZORKY
```

Vybere sloupce VRT a HLOUBKA ze všech řádků tabulky VZORKY.

```
select distinct VRT, HLOUBKA from VZORKY
```

Vybere sloupce VRT a HLOUBKA z tabulky VZORKY. Klíčové slovo DISTINCT zajišťuje, že ve vytvořené tabulce nebudou žádné dva řádky stejné - řádků tedy bude nanejvýš stejný počet jako ve zdrojové tabulce VZORKY.

```
select VRT, HLOUBKA*100/2.54 as PALCE from VZORKY
```

Vybere sloupce VRT a HLOUBKA ze všech řádků tabulky VZORKY. Na výstupu však druhý sloupec nebude obsahovat původní údaje o hloubce, ale údaje přepočtené podle uvedeného výrazu (přepočet na palce). Tento druhý sloupec výstupu také nebude mít název HLOUBKA, ale PALCE. Příklad demonstruje použití výrazů pro výpočet hodnot výstupu.

Poznámka: *Konstrukci <výraz> AS <jméno> je možno použít kdykoliv kdekoliv při určování, jaké hodnoty mají tvořit sloupec výstupní tabulky, a jak se tento sloupec má jmenovat. Tato skutečnost již nebude dále zdůrazňována a většinou bude používáno pouze jméno sloupce zdrojové tabulky.*

```
select VRT, HLOUBKA from VZORKY into VÝBĚR
```

Vybere sloupce VRT a HLOUBKA ze všech řádků tabulky VZORKY, a z těchto údajů vytvoří novou tabulku VÝBĚR. Tabulka VÝBĚR bude mít dva sloupce VRT a HLOUBKA (v tomto pořadí) s atributy i hodnotami dat přejatými z tabulky VZORKY. Počet řádků obou tabulek bude shodný.

Poznámka: *Konstrukci INTO <výstup> je možno použít, kdykoliv je zapotřebí vytvořit novou tabulku na základě dat v tabulkách již existujících. Tato skutečnost již nebude dále zdůrazňována a klauzule INTO bude většinou vynechána (tj. výstup bude směrován na standardní výstup, kterým je většinou obrazovka terminálu nebo počítače).*

```
select VRT, HLOUBKA from VZORKY into VÝBĚR where HLOUBKA > 100
```

Vybere sloupce VRT a HLOUBKA ze všech řádků tabulky VZORKY, a z těchto údajů vytvoří novou tabulku VÝBĚR. Tabulka VÝBĚR bude mít dva sloupce VRT a HLOUBKA (v tomto pořadí) s atributy i hodnotami dat přejatými z tabulky VZORKY. V tabulce VÝBĚR však budou zařazeny jen údaje z těch řádků tabulky VZORKY, ve kterých je HLOUBKA větší než 100.

Poznámka*: Konstrukci WHERE <podmínka> je možno použít kdykoliv kdekoliv při určování, jaké řádky zdrojové tabulky se mají zpracovávat při vytváření řádků výstupní tabulky. Tato skutečnost již nebude dále zdůrazňována a většinou bude používáno všech řádků zdrojové tabulky.*

```
select VRT, HLOUBKA from VZORKY into table VÝBĚR order by VRT asc, HLOUBKA desc
```

Vybere sloupce VRT a HLOUBKA ze všech řádků tabulky VZORKY, a z těchto údajů vytvoří novou tabulku VÝBĚR. Tabulka VÝBĚR bude mít dva sloupce VRT a HLOUBKA (v tomto pořadí) s atributy i hodnotami dat přejatými z tabulky VZORKY. Řádky v tabulce VÝBĚR budou seřazeny (asc - ascending = vzestupně) podle kódů vrtů (A18 bude před B11). Pokud dva řádky budou mít shodný kód vrtu, bude rozhodovat hloubka: [V18,140] bude před [V18,120], protože u jména HLOUBKA je uvedeno klíčové slovo desc - descending = sestupně.

Poznámka*: klíčové slovo asc se nemusí uvádět. Není-li uvedeno ani asc, ani desc, platí asc.*

```
select VRT, min (HLOUBKA) as DOLE, max (HLOUBKA) as NAHORE from VZORKY group by VRT
```

Jedna z velmi silných možností příkazu SELECT. Příkaz v tomto tvaru provádí nejprve myšlené seskupování ("grupování") řádků zdrojové tabulky VZORKY podle údajů ze sloupce VRT - v každé takové skupině jsou ty řádky tabulky VZORKY, které mají stejnou hodnotu ve sloupci VRT. Na výstupu se pak vytvoří tolik řádků, kolik je skupin (tj. kolik různých kódů vrtů je ve vstupní tabulce).

Z každé takto vytvořené skupiny se pošle na výstup jeden řádek. Výstupní tabulka bude mít tři sloupce. První (se jménem VRT) bude obsahovat kód vrtu společný všem řádkům skupiny. Druhý (se jménem DOLE) bude obsahovat nejmenší z hodnot HLOUBKA v této skupině. Analogicky třetí sloupec (se jménem NAHORE) bude obsahovat největší z hodnot HLOUBKA v této skupině.

***Poznámka**: Kromě funkcí MIN a MAX je možno při seskupování používat funkce AVG (average = průměr), COUNT (count = počet) a SUM (sum = součet). Podrobněji o agregačních funkcích odstavec "Seskupování záznamů" níže.*

Je možno použít seskupování podle více kritérií. Například klauzule

```
... group by SOUVRSTVI, VRT ...
```

vytvoří tolik skupin, kolik je ve vstupní tabulce různých dvojic [SOUVRSTVI,VRT].

***Poznámka:** Konstrukci GROUP BY <sloupec> je možno použít, kdykoliv nastane potřeba na výstup zařadit ne jednoduché údaje ze vstupní tabulky, ale údaje nějakým způsobem agregované. Tato skutečnost již nebude dále zdůrazňována a většinou bude používáno neseskupených dat.*

8.7.4 Údaje ze dvou tabulek

Výstupní tabulku lze vytvořit na základě údajů ze dvou a více zdrojových tabulek. Nejjednodušší je případ dvou tabulek.

```
select VZORKY.VRT, VZORKY.HLOUBKA, SOUVRSTVI.NAZEV from VZORKY, SOUVRSTVI
```

Tento základní tvar příkazu SELECT obsahující dva zdrojové soubory pracuje následovně: především je vytvořena kombinace všech řádků (se všemi sloupcovými údaji) jedné zdrojové tabulky a všech řádků (se všemi sloupcovými údaji) druhé zdrojové tabulky - tedy jejich kartézský součin.

***Poznámka**: mají-li obě vstupní tabulky po pouhém tisící řádků, má tato kombinovaná tabulka řádků milion.*

Dále: uvedená výstupní tabulka je vytvořena ze všech těchto zkombinovaných řádků s tím, že bude mít tři sloupce; v prvním budou data ze sloupce VRT té části, která vznikla z tabulky VZORKY; ve druhém budou data ze sloupce HLOUBKA té části, která rovněž vznikla z tabulky VZORKY; konečně ve třetím budou data ze sloupce NAZEV té části, která vznikla z tabulky SOUVRSTVI.

Toto je přesný popis vzniku tabulky. Je zřejmé, že právě v uvedeném příkladu se většinou ve výstupní tabulce ocitnou nesmyslné kombinace údajů. Jestliže např. vrt A18 zasahuje jen do ostravského souvrství (tj. logická je jen kombinace [O,A18]), přesto se ve výstupní tabulce objeví i kombinace [K,A18] a případné další.

Je proto zapotřebí vybrat jen ty kombinace, které vyplývají z dat vzorků. To provede tvar příkazu SELECT s již dříve uvedenou klauzulí WHERE:

```
select VZORKY.VRT, VZORKY.HLOUBKA, SOUVRSTVI.NAZEV from VZORKY, SOUVRSTVI where
VZORKY.SOUVRSTVI = SOUVRSTVI.KOD
```

Nyní jsou do výstupní tabulky ze všech možných kombinací zařazeny jen ty, kde kód SOUVRSTVÍ ve VZORKU je rovno KÓDU v SOUVRSTVÍ. Tento mechanismus je možno považovat za vytvoření "filtru" mezi VZORKY a SOUVRSTVÍ-mi. "Odfiltrují" se všechny řádky, které nesplňují podmínku uvedenou za WHERE, jinak zde: ke každému řádku ze VZORKŮ se připojí ten řádek ze SOUVRSTVÍ, který má shodný kód s kódem souvrství ve VZORCÍCH.

Klauzuli WHERE je možno dále rozšířit o "běžné" podmínky výběru, např.

```
select VZORKY.VRT, VZORKY.HLOUBKA, SOUVRSTVI.NAZEV from VZORKY, SOUVRSTVI where
VZORKY.SOUVRSTVI = SOUVRSTVI.KOD and HLOUBKA > 100
```

Funkce je zcela totožná jako shora až na to, že ve výstupní tabulce budou zahrnuty jen ty řádky, kde je větší hloubka než 100.

***Poznámka 1**: Pokud je v příkazu select uvedena klauzule where (tj. podmínka výběru při vstupu do zpracování, pak většina interpretů ji respektuje již v první fázi zpracování. Pokud se čerpá ze dvou zdrojů majících po tisíci řádcích, pak se nevytvoří fyzicky milion kombinací, ale jen nezbytně potřebný počet.*

***Poznámka 2**: zcela podle potřeby je možno použít klauzulí GROUP BY, ORDER BY a INTO TABLE ve smyslu popsaném shora.*

Uvedený příklad na čerpání dat ze dvou datových zdrojů používal mechanismu tzv. kartézského součinu dvou množin. Nevýhodou však je, že výsledek takového příkazu nelze použít pro aktualizaci dat zdrojových tabulek.

Čerpat data ze dvou tabulek lze však také pomocí mechanismu připojení (angl. join). Protože zápis příkazu SQL je lineární (zleva doprava), pak při použití dvou datových zdrojů je jeden první (vlevo od druhého) a jeden druhý (vpravo od prvního). Jedna z tabulek připojuje druhou na základě nějaké podmínky:

```
select VZORKY.VRT, VZORKY.HLOUBKA, SOUVRSTVI.NAZEV from VZORKY left join SOUVRSTVI
on VZORKY.SOUVRSTVI = SOUVRSTVI.KOD where HLOUBKA > 100
```

Výsledek bude na první pohled stejný jako v předchozím případě. V situaci, kdy si tabulka VZORKY **zleva** připojuje tabulku SOUVRSTVÍ, objeví se na výstupu i ty vzorky, pro které případně ve druhé tabulce neexistuje souvrství. Takové vzorky by se v předchozím případě neobjevily. Kdyby si naopak tabulka VORKY **zprava** (right join) připojovala tabulku souvrství, objevily by se na výstupu všechna souvrství a jejich event. vzorky, a to i ta souvrství, pro něž vzorky neexistují. Konečně kdyby byly oba datové zdroje **vnitřně** (inner join) připojeny, byl by výsledek datově stejný jako v předchozím případě. V čem je však podstatný rozdíl: pomocí výsledku druhého příkazu (s join) jde na rozdíl od předchozího příkladu v přesně stanovených případech data aktualizovat. Viz také odstavec "Join" níže.

8.7.5 Údaje z více tabulek

Pro tři a více tabulek platí zřejmá analogie se dvěma tabulkami. Nejprve se - v případě kartézského součinu - vytvoří myšleně "mezi-tabulka" tvořená ze všech možných kombinací všech řádků všech zdrojových tabulek. Výstupní tabulka bude mít tolik sloupců, kolik je určeno výčtem za klíčovým slovem SELECT. Na výstup se dostanou jen ty řádky, které splní podmínku uvedenou za WHERE.

Tedy např. pro čtyři tabulky příkaz

```
select VZORKY.VRT, VZORKY.HLOUBKA, VRTY.X, VRTY.Y, VRSTVY.PLOCHA, SOUVRSTVI.NAZEV
from VZORKY, VRTY, VRSTVY, SOUVRSTVI where VZORKY.VRT = VRTY.KOD and VZORKY.VRSTVA =
VRSTVY.KOD and VRSTVY.SOUVRSTVI = SOUVRSTVI.KOD
```

ponechá ve výstupní tabulce jen ty řádky, které mají shodné "vazební" prvky uvedené na každé straně rovnítek.

***Poznámka**: i u tohoto tvaru příkazu SELECT je možno zcela podle potřeby použít klauzulí GROUP BY, ORDER BY a INTO TABLE ve smyslu popsaném shora.*

Ve druhém případě - s klauzulí join - vytváří každá konstrukce typu

```
... (A left join B on A.X = B.Y) ...
```

nový datový zdroj, který (nebo na který) je možno připojit další datový zdroj, tím vznikne další nový datový zdroj, na který lze připojit ... atd.

8.7.6 Poddotazy

Síla příkazu SELECT se násobí možností použít uvnitř jednoho příkazu SELECT jiný příkaz SELECT - tzv. poddotaz. Vychází se z toho, že výsledek příkazu SELECT je sám o sobě relací, tj. množinou (jednotlivých hodnot daného typu, dvojic hodnot, trojic atd). Tuto množinu lze využít např. při výpočtu výsledných údajů nebo při určení datových zdrojů. Velmi často se použije při definování podmínek, za kterých se řádek ve výstupní tabulce vytvoří. V tomto případě se uplatní operátory IN, ANY, ALL, jak je ukázáno dále.

```
select VRT, HLOUBKA   from VZORKY   where VRT in (select distinct VRT from ČERPÁNÍ
where MNOŽSTVÍ > 10)
```

Tento příkaz SELECT obsahuje druhý příkaz SELECT (vždy uzavřený v kulatých závorkách) použitý v klauzuli WHERE; "vnitřní" příkaz se provede nejdříve a vytvoří (dočasný) seznam kódů vrtů, kde se čerpalo (alespoň jednou) větší množství než 10. Následuje tvorba výsledné tabulky obsahující sloupce VRT a HLOUBKA z tabulky VZORKY již popsaným způsobem. Do výsledné tabulky se však umístí jen ty řádky, které vyhovují podmínce uvedené v klauzuli WHERE - zde ty, kde se kód VRT vyskytuje v (in) seznamu vytvořeném oním "vnitřním" příkazem SELECT (tedy v seznamu vrtů, v nichž bylo alespoň jednou čerpáno větší množství než 10).

***Poznámka:** kromě IN lze použít i negaci NOT IN.*

```
select VRT, HLOUBKA from VZORKY where exists (select * from ČERPÁNÍ where VZORKY.VRT = ČERPÁNÍ.VRT)
```

Při vytváření výsledné tabulky obsahující sloupce VRT a HLOUBKA z tabulky VZORKY se pro každý řádek nejprve zjistí, zda je splněna podmínka daná klauzulí WHERE; tj. zda existuje (exists) alespoň jeden řádek tabulky vytvořené "vnitřním" příkazem SELECT. Ten vybere z tabulky ČERPÁNÍ všechny řádky mající kód vrtu shodný s kódem právě zařazovaného řádku z tabulky VZORKY. Pokud existuje, řádek se zařadí; pokud neexistuje, řádek se nezařadí. Výsledná tabulka tedy bude obsahovat údaje jen o těch vrtech, ze kterých bylo čerpáno.

Poznámka: *kromě EXISTS lze použít i negaci NOT EXISTS.*

```
select VRT, HLOUBKA from VZORKY where HLOUBKA+10 > ALL (select HLADINA from ČERPÁNÍ where ČERPÁNÍ.DATUM >= #01/01/1997# )
```

Při vytváření výsledné tabulky obsahující sloupce VRT a HLOUBKA z tabulky VZORKY se pro každý řádek nejprve zjistí, zda je splněna podmínka daná klauzulí WHERE; tj. hloubka vrtu je alespoň o 10 větší než všechny hladiny při čerpáních počínaje rokem 1997.

```
select VRT, HLOUBKA from VZORKY where HLOUBKA+10 > ANY (select HLADINA from ČERPÁNÍ where ČERPÁNÍ.DATUM >= #01/01/1997# )
```

Při vytváření výsledné tabulky obsahující sloupce VRT a HLOUBKA z tabulky VZORKY se pro každý řádek nejprve zjistí, zda je splněna podmínka daná klauzulí WHERE; tj. hloubka vrtu je alespoň o 10 větší než alespoň jedna hladina při čerpáních počínaje rokem 1997.

Díl III: Databáze prakticky

Tyto texty jsou základem pro výuku i v jiných národních jazycích, zvláště EN a DE. Aby nebylo nutno zásadně přepracovávat cvičné databáze, jsou v níže uváděných příkladech jména (tabulek, datových polí, dotazů apod.) respektována z hlediska definice, tj. jakožto identifikátoru:

Identifikátor = posloupnost písmen a číslic začínající písmenem

V jiných abecedách však podivný znak typu "U nad kterým je O" rozhodně není písmenem, proto jsou všechna jména ve cvičných databázích i v příkladech tohoto textu odvolávající se na ně uváděna **BEZ DIAKRITIKY**. Jména dále striktně respektují výše uvedenou definici identifikátoru (tj. např. bez mezer) z důvodu nekonfliktního zápisu příkazů SQL. K tomu viz však odstavec *Tvar a zápis jmen*; možnosti tam popsané nejsou v následujících kapitolách záměrně využity.

9 Výuková témata

Protože se předmět Databáze vyučuje pro geo-vědní obory, jsou témata volena z této oblasti. Nosné téma je zaměřeno na geoparky v České republice a turistické aktivity směrem k nim. V několika stupních složitosti formuluje jednak popis situace, jednak otázky, na které by budovaný databázový informační systém měl umět odpovědět.

Témata jsou zaměřena na *výuku* databázové problematiky; rozhodně proto nelze zkoumat jejich skutečné uplatnění v reálném provozu. Jsou formulována právě pro demonstraci jednotlivých postupů a vlastností jak databázové problematiky jako takové, tak i databázových programů. Postupují od nejjednodušších po komplikovanější; jejich formulace je volena tak, aby při studiu databází nezatěžovala složitostí problému a přitom popisovala celkem uvěřitelnou situaci.

9.1 Geoparky - výukové téma I

Provozujete cestovní kancelář **{GeoParkTour}** nabízející v prvním červencovém týdnu několik týdenních zájezdů do geoparků a chráněných krajinných oblastí. Cílem každého zájezdu je daná lokalita s ubytováním a stravováním hotelového typu v blízké městské aglomeraci. Odtud vyjíždějí účastníci každý den na exkurze do cílové lokality s předem daným programem.

Každý zájezd má pro jednoho účastníka stanovenou pevnou jednotnou cenu. Ta zahrnuje dopravu, ubytování, stravování, pojištění a další výdaje každého jednotlivce. 10% z této částky je zisk cestovní kanceláře. Na druhé straně musí vaše kancelář počítat pro každý zájezd s jistými celkovými režijními výdaji nezávislými na počtu účastníků.

Vytvořte databázi, ve které budete sledovat obsazenost jednotlivých zájezdů jednotlivými účastníky. Ti budou předem u vás skládat zálohu minimálně ve výši čtvrtiny ceny zájezdu. Zajistěte, aby databáze byla schopna odpovědět na následující otázky:

***Metodická poznámka**: Z textu zadání vyplývá, že jedna osoba nemůže být účastníkem více zájezdů.*

- 1. Kolik účastníků se přihlásilo na jednotlivé zájezdy?
- 2. Kolik je vybráno celkem na zálohách?
- 3. Kolik ještě doplatí účastníci v jednotlivých zájezdech?
- 4. Kolik roků je nejstaršímu přihlášenému muži?
- 5. Který zájezd je ztrátový?

9.2 Geoparky - výukové téma II

Provozujete cestovní kancelář **{GeoParkTour}** mající řadu klientů. Někteří z nich zvolí z vaší prázdninové nabídky, kdy pořádáte každý sudý týden jeden zájezd do různých geoparků a chráněných krajinných oblastí. Cílem každého zájezdu je daná lokalita s ubytováním a stravováním hotelového typu v blízké městské aglomeraci. Odtud vyjíždějí účastníci každý den na exkurze do cílové lokality s předem daným programem.

Každý zájezd má pro jednoho účastníka stanovenou pevnou jednotnou cenu. Ta zahrnuje dopravu, ubytování, stravování, pojištění a další výdaje každého jednotlivce. 10% z této částky je zisk cestovní kanceláře. Na druhé straně musí vaše kancelář počítat pro každý zájezd s jistými celkovými režijními výdaji nezávislými na počtu účastníků.

Vytvořte databázi, ve které budete sledovat obsazenost jednotlivých zájezdů jednotlivými účastníky. Ti budou předem u vás skládat zálohu minimálně ve výši čtvrtiny ceny zájezdu. Zajistěte, aby databáze byla schopna odpovědět na následující otázky:

***Metodická poznámka**: Z textu zadání vyplývá, že - na rozdíl od předchozího tématu - může být jedna osoba účastníkem více zájezdů a termíny zájezdů se nepřekrývají.*

- 1. Kolik účastníků se přihlásilo na jednotlivé zájezdy?
- 2. Kolik účastníků je přihlášeno na jediný zájezd?
- 3. Kdo jmenovitě je přihlášen na více zájezdů?
- 4. Kdo jmenovitě je přihlášen na nejvíce zájezdů?
- 5. Na který zájezd se nikdo nepřihlásil?
- 6. Který zájezd je ztrátový?

9.3 Geoparky - výukové téma III

Provozujete cestovní kancelář **{GeoParkTour}** mající řadu klientů. Někteří z nich zvolí z vaší prázdninové nabídky, kdy pořádáte každý sudý týden jeden zájezd do různých geoparků a chráněných krajinných oblastí. Cílem každého zájezdu je daná lokalita s ubytováním a stravováním hotelového typu v blízké městské aglomeraci. Odtud vyjíždějí účastníci každý den na exkurze do cílové lokality s předem daným programem. V případě zájmu mohou účastníci některé exkurze vyměnit za fakultativní jednodenní výlety do geologicky nebo přírodně zajímavých míst. Každý den je přitom nabízen nanejvýš jeden výlet.

Každý zájezd má pro každého účastníka stanovenou jednotnou cenu za ubytování, stravování a exkurze, která však nezahrnuje případné fakultativní výlety - ty jsou zpoplatněny samostatně. Na druhé straně musí vaše kancelář počítat pro každý zájezd s jistými celkovými režijními výdaji nezávislými na počtu účastníků.

Vytvořte databázi, ve které budete sledovat obsazenost jednotlivých zájezdů jednotlivými účastníky. Ti budou předem u vás skládat zálohu minimálně ve výši čtvrtiny ceny zájezdu. Zajistěte, aby databáze byla schopna odpovědět na následující otázky:

> ***Metodická poznámka**: Z textu zadání vyplývá, že jedna osoba může být účastníkem více zájezdů a v rámci jednoho zájezdu absolvovat i několik výletů; termíny zájezdů se přitom nepřekrývají.*

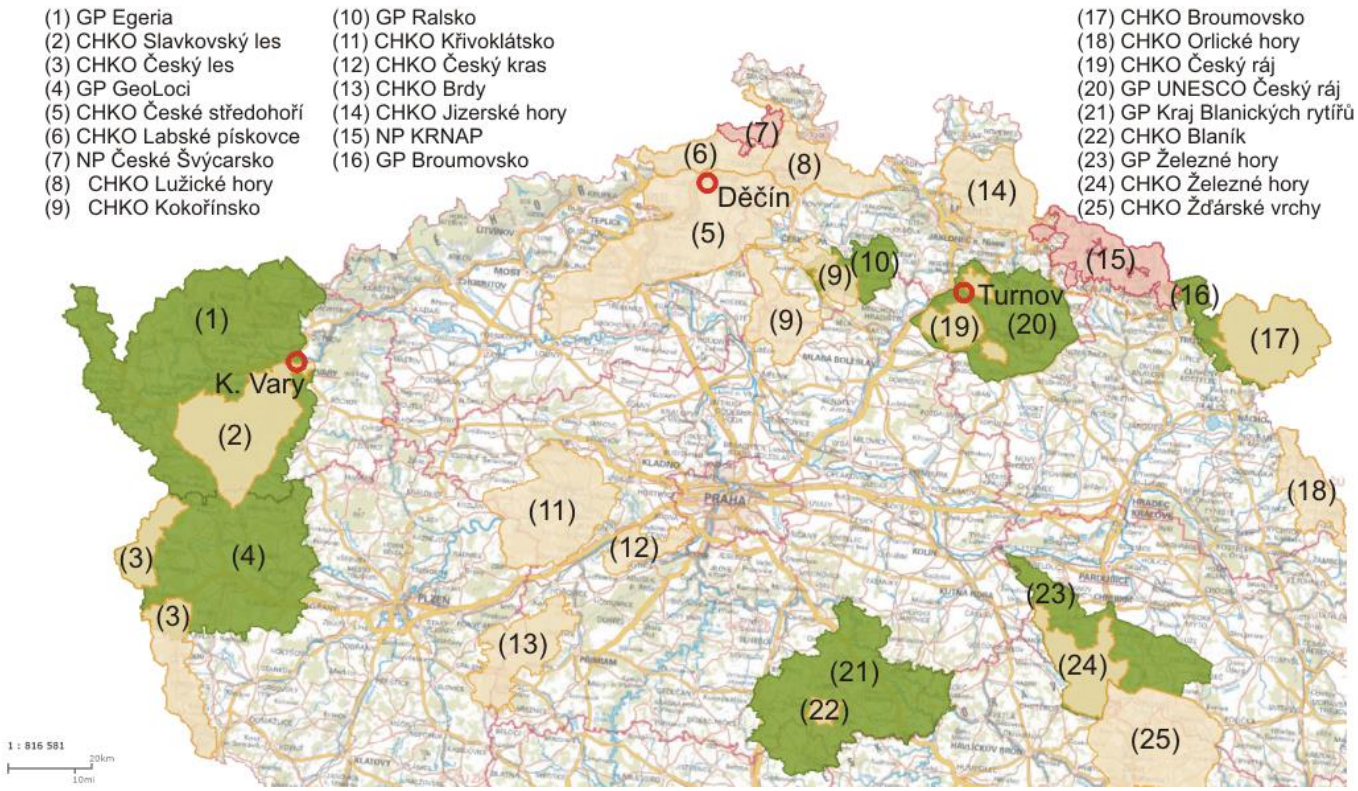
- 1. Kolik účastníků se přihlásilo na jednotlivé zájezdy?
- 2. Kolik je vybráno celkem na zálohách?
- 3. Kolik zaplatí účastníci v jednotlivých zájezdech navíc za výlety?
- 4. Který výlet kterým účastníkům nelze ze zdravotních důvodů doporučit?
- 5. Který nabízený výlet je nejžádanější?
- 6. Na které nabízené výlety se nikdo nepřihlásil?
- 7. Cíl kterého výletu je od místa ubytování vzdušnou čarou nejdále?

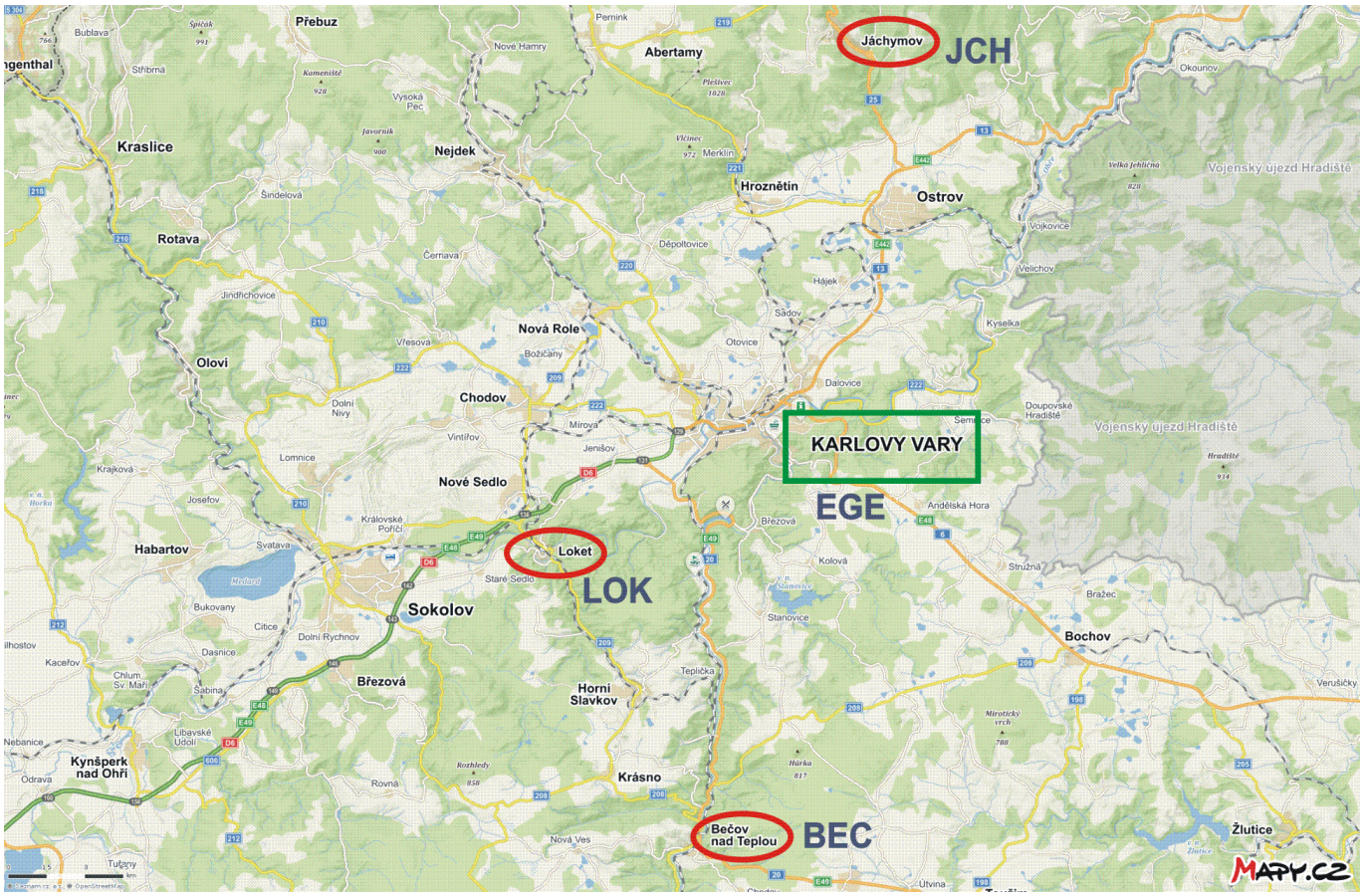
9.4 Geoparky - konkretizace lokalit

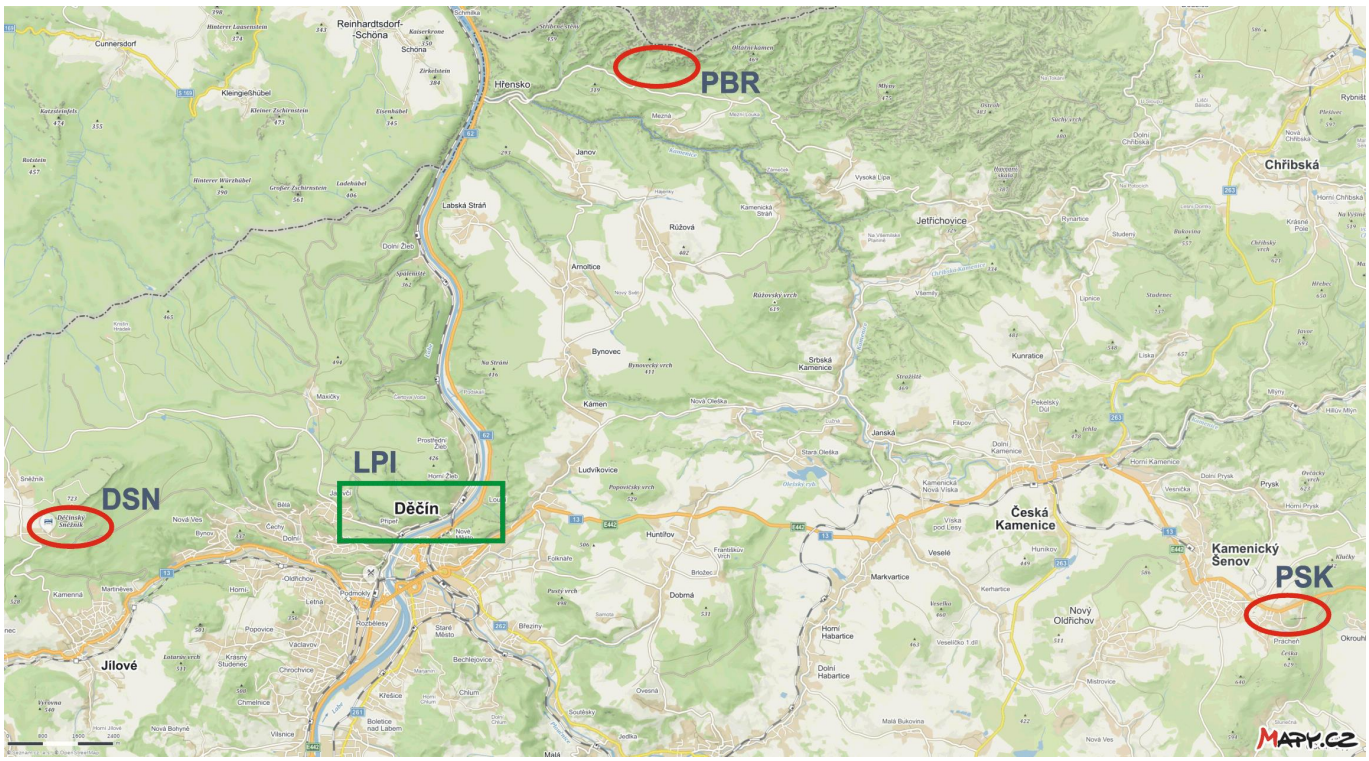
Pro názornost demonstračních témat bylo zvoleno několik lokalit jednak jako cílů zájezdů, jednak jako výletů pořádaných z nich:

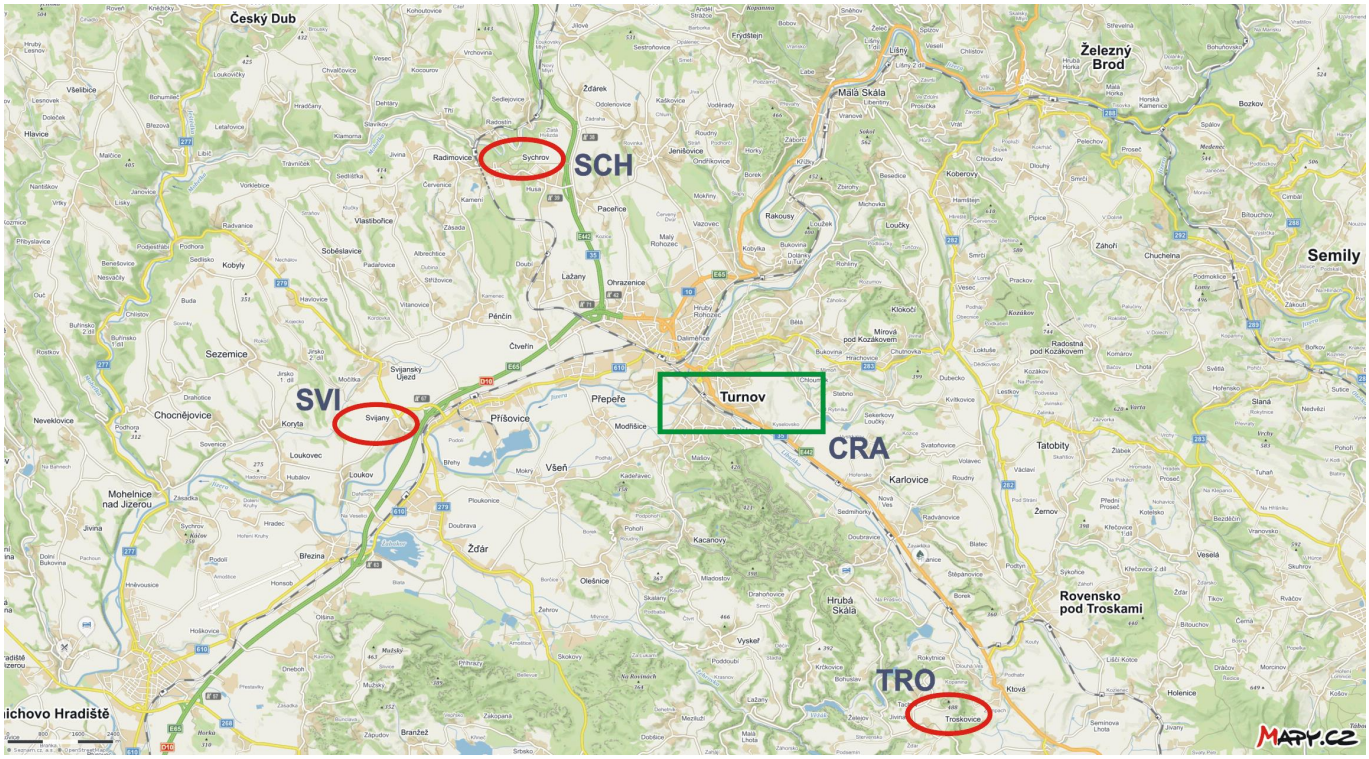
- EGE - Geopark Egeria, ubytování v Karlových Varech, výlety:
  - BEC - Bečov nad Teplou
  - JCH - Jáchymov
  - LOK - Loket
- LPI - CHKO Labské Pískovce, ubytování v Děčíně, výlety:
  - DSN - Děčínský Sněžník
  - PBR - Pravčická brána
  - PSK - Panská skála
- CRA - Geopark Český Ráj, ubytování v Turnově, výlety:
  - SCH - Sychrov
  - SVI - Svijany
  - TRO - Trosky
- ZVR - CHKO Žďárské vrchy, ubytování ve Škrdlovicích, výlety:
  - HBR - Havlíčkův Brod
  - KOS - Kušumberk a Luže
  - ZNS - Žďár nad Sázavou

Situaci mohou zobrazit např. následující výřezy map (© ČUZK, AOPK ČR 2016, MAPY.CZ 2018):


(1) GP Egeria
(2) CHKO Slavkovský les
(3) CHKO Český les
(4) GP GeoLoci
(5) CHKO České středohoří
(6) CHKO Labské pískovce
(7) NP České Švýcarsko
(8) CHKO Lužické hory
(9) CHKO Kokořínsko
(10) GP Ralsko
(11) CHKO Křivoklátsko
(12) CHKO Český kras
(13) CHKO Brdy
(14) CHKO Jizerské hory
(15) NP KRNAP
(16) GP Broumovsko
(17) CHKO Broumovsko
(18) CHKO Orlické hory
(19) CHKO Český ráj
(20) GP UNESCO Český ráj
(21) GP Kraj Blanických rytířů
(22) CHKO Blaník
(23) GP Železné hory
(24) CHKO Železné hory
(25) CHKO Žďárské vrchy
Děčín
K. Vary
Turnov
PRAHA
1 : 816 581


Jáchymov
JCH
KARLOVY VARY
EGE
Loket
LOK
Bečov nad Teplou
BEC
Přebuz
Abertamy
Kraslice
Nejdek
Hroznětín
Ostrov
Rotava
Nová Role
Olovi
Chodov
Nové Sedlo
Habartov
Sokolov
Bochov
Březová
Horní Slavkov
Krásno
Kynšperk nad Ohří
Žlutice
Vojenský újezd Hradiště
MAPY.CZ


PBR
LPI
Děčín
DSN
PSK
Česká Kamenice
Kamenický Šenov
Jílové
Hřensko
Chřibská
Jetřichovice
Labská Stráň
Janov
Růžová
Arnoltice
Bynovec
Srbská Kamenice
Kamen
Ludvíkovice
Huntířov
Dobrná
Markvartice
Nový Oldřichov
Dolní Habartice
Horní Habartice
Veselé
Kunratice
Janská
Nová Oleška
Stará Oleška
Folknáře
Březiny
Reinhardtsdorf-Schöna
Schöna
Cunnersdorf
Kleingießhübel
Mapy.cz


SCH
SVI
Svijany
Turnov
CRA
TRO
Troskovice
Český Dub
Železný Brod
Semily
Mohelnice nad Jizerou
Sezemice
Chocnějovice
Loukovec
Příšovice
Přepeře
Všeň
Žďár
Karlovice
Tatobity
Rovensko pod Troskami
Hrubá Skála
Malá Skála
Radostná pod Kozákovem
Mapy.cz

10 Analýza úlohy

V této kapitole jsou použity informace podané na obecné úrovni v kapitolách předchozích. Protože níže bude popsán praktický postup při analýze a budování konkrétní relační databázové aplikace, je vhodné některé dříve obecně zmíněné pojmy konkretizovat - na první pohled nelogicky - právě zde, aby čtenář nemusel listovat celým textem.

10.1 Indexy a klíče, vazby mezi tabulkami

Shora v teoretické části těchto učebních textů, v kapitole *Metody uložení dat*, byly zmíněny pojmy *klíč* a *index*. V oblasti relačních databází jde o základní nástroje jednak pro realizaci logických vazeb mezi tabulkami (z pohledu autora databáze), jednak pro optimalizaci přístupu k datům (z pohledu autora implementace databázového systému). Protože v literatuře jsou tyto pojmy chápány různě, upřesníme dále jejich postavení v popisovaných úlohách.

Databázový *index* je datová struktura v databázovém pravěku původně zamýšlená jen pro interní optimalizace databázových činností, např. při ukládání dat na medium nebo vyhledávání. Index dané tabulky je v nejjednodušším případě kopie jednoho nebo více sloupců této tabulky. Je-li index vytvořen z několika sloupců, bývá označován jako *spojený index*. Ve složitějších případech může jít o kopie ne celých sloupců, ale jen dat z těch sloupců, které splňují nějakou podmínku (někdy pro upřesnění nazývaný *parciální index* nebo *filtrovaný index*). Kvalitní databáze poskytují možnost vytvářet indexy nejen z pouhých hodnot ve sloupci (ve sloupcích), ale obecně z hodnot výrazů obsahujících operace a volání funkcí - a ty nejkvalitnější i volání uživatelem definovaných funkcí.

Pod pojmem *indexová tabulka* je možno si představit dvousloupcovou tabulku, kde první sloupec obsahuje index, druhý odkazy na fyzické umístění toho řádku tabulky, z něhož pochází konkrétní hodnota indexu. Protože indexové tabulky lze vytvářet seřazené podle indexu, byly od počátku databází využívány především pro vyhledávání a pro přístup k datům v určitém pořadí.

Poznámka: *Doporučená představa indexové tabulky je pro běžného zájemce o databázovou problematiku asi nejstravitelnější. Interně ve většině případů nejde o tabulky, ale o instance objektových tříd modelujících např. stromové struktury s odkazy známé z teorie grafů apod.*

Položením teoretických základů relačních databází opírajících se o teorii množin se indexové tabulky začaly využívat také při zajištění logických vazeb mezi tabulkami. Bylo však třeba pro některá využití indexů vyžadovat některé vlastnosti - index v obecném pojetí může tvořit z tabulky cokoliv.

Pojem *klíč* se začal používat pro takový index, jehož hodnoty jednoznačně identifikují konkrétní záznam (= řádek tabulky relační databáze). Je zřejmé, že v tom případě nemohou existovat dvě stejné hodnoty indexu, index musí být jedinečný. Dále: existuje možnost, že by jedna nebo více hodnot indexu byla prázdná (angl. *null*). To by znamenalo, že sice některé indexy jednoznačně identifikují některý záznam, ale na některé záznamy by nebylo odkazováno. V praxi je taková situace pro vytváření logických vazeb mezi tabulkami nežádoucí (uživatel by ztratil informaci o některých datech), proto se od klíče vyžaduje i *neprázdnost* (the key is *not null*).

Avšak i jedinečných a neprázdných klíčů může být více. Z hlediska efektivity zpracování se používají co nejkratší klíče. Proto se definuje *kandidátní klíč* jako klíč, který

- obsahuje jedinečné hodnoty,
- neobsahuje prázdné (null) hodnoty,
- obsahuje nejmenší počet polí, který ještě zaručují jedinečnost,
- jednoznačně identifikují každý záznam v tabulce.

Autor databáze pak může jeden z kandidátních klíčů prohlásit za *primární* klíč. Obvykle volí takový, který se v kontextu úlohy jeví jako nejvhodnější referenční klíč pro danou tabulku. Příkladem může být osobní číslo zaměstnance, IČO organizace, kód zájezdu apod.

Nechť nyní je nějaká tabulka A opatřena kandidátním klíčem. Nechť v jiné tabulce B existuje index, jehož hodnoty jednoznačně identifikují záznam (= řádek) v tabulce A jejím kandidátním klíčem. Protože takový index v tabulce B obsahuje klíčové hodnoty v tabulce A, nazývá se index v tabulce B *cizím* klíčem. Jinak řečeno, cizí klíč je definovaný v tabulce B, ale odkazuje svými hodnotami na kandidátní klíč tabulky A. Právě dvojice (cizí klíč, kandidátní klíč) definují *vztah*, relaci, vazbu (*relationship*, relation, binding) mezi dvěma tabulkami.

Tabulka B, obsahující cizí klíč, je označována jako *odkazující* (child, dceřiná) tabulka. Tabulka A, obsahující kandidátní klíč, je označována jako *odkazovaná* (parent, rodičovská) tabulka.

> ***Poznámka:*** *Pokud je kandidátní klíč v odkazované tabulce A prohlášen primárním klíčem, dovedou mnohé databázové systémy zajistit řadu dalších funkcí, např. automatickou kontrolu cizího klíče při přidávání záznamu do odkazující tabulky a jiné.*

V odkazující tabulce může několik řádků odkazovat na stejný řádek v odkazované tabulce. Takový vztah se označuje jako vztah 1 : M (one to many).

Tento teoretický výklad je použit níže při definici struktur jednotlivých tabulek a logických vazeb mezi nimi.

10.2 Příklad analýzy - Téma I

Databázové aplikace tvoří v informačních systémech jejich softwarovou podporu. V procesu tvorby databázových aplikací lze rozpoznat několik na sebe navazujících fází:

1. Shromáždění poznatků o struktuře pracoviště, kde bude systém nasazen, o vazbách mezi organizačními složkami, o činnostech zde vykonávaných, o toku materiálu a dokladů, o přibližných počtech jak živého tak neživého inventáře pracoviště - prostě o situaci, jaká na daném pracovišti panuje.
2. Definice otázek, na které uživatelé jednotlivých částí budovaného informačního systému očekávají odpovědi.
3. Výčet dat, která jsou nutná pro zodpovězení definovaných otázek. V této fázi se již konkretizuje typ dat s ohledem na jejich charakter a rozsah.
4. Rozdělení dat do jednotlivých logických celků - v případě relačních databází tedy do jednotlivých tabulek, stanovení jmen jednotlivých komponent (tabulek, sloupců). Protože v naprosté většině budou data jednotlivých tabulek v nějakém logickém vztahu, stanoví se v této fázi, které datové položky které tabulky budou tento vztah realizovat a jak vztah (vztahy) vlastně budou vypadat.
5. Shromáždění datových podkladů, na kterých bude vytvořená databázová aplikace testována.
6. Vytvoření datových zdrojů a jejich naplnění testovacími daty.

7. Vytvoření programové podpory (samostatných programů, procedur, funkcí) ve zvoleném programovacím jazyku nebo zvolených programovacích jazycích.
8. Testování vytvořené databázové aplikace na připravených testovacích datech, posléze její naplnění "ostrými" daty a předání do zkušebního provozu.

První a druhá fáze bývá často označována jako *předprojektová příprava*, třetí a čtvrtá fáze jako *analýza úlohy*. Je vidět, že shora uvedený popis demonstračních témat je podán jako výsledek předprojektové přípravy.

Analýza úlohy je jednou z klíčových činností při tvorbě databázové aplikace. Protože jde o problematiku poměrně náročnou a přesahující záměr těchto učebních textů, omezíme se jen na schematický popis analýzy prvního, nejjednoduššího tématu (GeoParky I).

Především tedy výčet dat. Ze zadání vyplývá, že se bude pracovat minimálně se dvěma subjekty: s klientem cestovní kanceláře a se zájezdem. Dále: jedna osoba sice nemůže být dle zadání účastníkem více zájezdů, ale naopak jeden zájezd může mít více účastníků. Každý účastník tedy bude odkazovat na svůj zájezd, několik účastníků na stejný zájezd. Kdyby údaje o účastnících a údaje o zájezdech byly "spolu" (ve stejné tabulce), musely by se údaje o zájezdech totožně opakovat několikrát - totiž tolikrát, kolik účastníků na ně jede. Tvořily by tedy redundantní (opakované, nadbytečné) údaje, což je např. z hlediska oprav dat naprosto nevhodné. Budou proto minimálně dvě tabulky: jedna s účastníky (ÚČASTNÍCI), jedna se zájezdy (ZÁJEZDY).

Při opětném pročítání popisu situace už lze nejen vyjmenovávat data, ale přímo je zařazovat do konkrétní tabulky. V následujícím textu budou data a tabulka uvedena v hranatých závorkách [a].

Provozujete cestovní kancelář nabízející v prvním červencovém týdnu několik týdenních zájezdů **[termín odjezdu a příjezdu - ZÁJEZDY]** do geoparků a chráněných krajinných oblastí **[popis - ZÁJEZDY]**. Cílem každého zájezdu je daná lokalita **[lokalita - ZÁJEZDY]** s ubytováním a stravováním hotelového typu v blízké městské aglomeraci. Odtud vyjíždějí účastníci každý den na exkurze do cílové lokality s předem daným programem.

Každý zájezd má pro jednoho účastníka stanovenou pevnou jednotnou cenu **[cena zájezdu - ZÁJEZDY]**. Ta zahrnuje dopravu, ubytování, stravování, pojištění a další výdaje každého jednotlivce. 10% z této částky je zisk cestovní kanceláře. Na druhé straně musí vaše kancelář počítat pro každý zájezd s jistými celkovými režijními výdaji **[režie - ZÁJEZDY]** nezávislými na počtu účastníků.

Vytvořte databázi, ve které budete sledovat obsazenost jednotlivých zájezdů jednotlivými účastníky. Ti budou předem u vás skládat zálohu **[záloha - ÚČASTNÍCI]** minimálně ve výši čtvrtiny ceny zájezdu **[cena zájezdu - ZÁJEZDY]**. Zajistěte, aby databáze byla schopna odpovědět na následující otázky:

- 1. Kolik účastníků se přihlásilo na jednotlivé zájezdy? Odpovědí je počet účastníků, netřeba další data.
- 2. Kolik je vybráno celkem na zálohách? Odpovědí je součet záloh, netřeba další data.
- 3. Kolik ještě doplatí účastníci v jednotlivých zájezdech? Odpovědí je rozdíl součtu cen [ZÁJEZDY] a záloh [ÚČASTNÍCI], netřeba další data.
- 4. Kolik roků je nejstaršímu přihlášenému muži? Pro odpověď jsou třeba další data: **[datum narození - ÚČASTNÍCI]** a **[pohlaví - ÚČASTNÍCI]**.
- 5. Který zájezd je ztrátový? Odpovědí je rozdíl součtu zisků a režie, netřeba další data.

Mezi tabulkami jistě existuje logický vztah. Vždyť např. pro každého účastníka je nutno stanovit, kolik má ještě doplatit: cena zájezdu je v [ZÁJEZDY], již zaplacená záloha v [ÚČASTNÍCI]. Při procházení tabulky účastníků bude proto zapotřebí ke každému řádku s účastníkem logicky připojit právě ten řádek se zájezdem, na který účastník jede. Ergo musí být v obou tabulkách nějaký (společný) údaj, který řádek účastníka ztotožní s řádkem zájezdu.

Při stanovení takového vazebního údaje je třeba mít na paměti, že se bude v tabulce [ÚČASTNÍCI] vyskytovat několikrát (několik účastníků jede na stejný zájezd), kdežto v tabulce [ZÁJEZDY] jen jednou (kdyby byl u více zájezdů stejný, nedalo by se u účastníků určit, na který z nich vlastně jede). Protože v tabulce účastníků se vyskytne několikrát, neměl by být příliš dlouhý - neměl by to být např. cíl zájezdu v plném znění. Bývá zvykem, že se každému zájezdu přiřadí nějaký krátký jednoznačný kód, např. textový nebo číselný. Tento kód pak bude v tabulce [ZÁJEZDY] jednoznačně zájezd identifikovat, zatímco v tabulce účastníků bude sdělovat, který zájezd daný účastník absolvuje.

Příklad takového "okódování" je vidět shora v odstavci *Geoparky - konkretizace lokalit*. Oproti textu popisujícímu situaci je možno - analogicky ke kódu zájezdu - přidat kód účastníka. Na většině pracovišť se stejně používá známé

osobní číslo nebo jiný ekvivalent. Že by v tabulce účastníků mělo být i jejich jméno a příjmení (odděleně, viz řazení), je nasnadě.

Nyní je zapotřebí určit datové typy. U některých dat je to jednoznačné, např. datum odjezdu. Pokud jde o data již v praxi používaná, je vhodné se držet právě tohoto používaného tvaru a významu (např. PSČ). Mnoho dat v budované databázové aplikaci je však specifických právě pro tuto aplikaci a autor volí podle svého uvážení. Například kód zájezdu: text (VARY) nebo číslo (285)? A když číslo, celé (285 = Vary) nebo desetinné (258.1 = Vary s polopenzí, 285.2 = Vary s plnou penzí)?

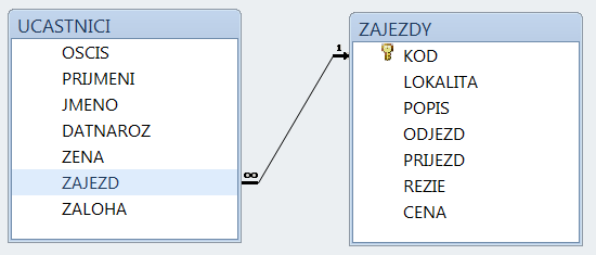
10.3 Struktura tabulek pro Téma I

Výsledkem předchozích úvah může být např. tato struktura tabulek:

Téma I	ÚČASTNÍCI	Název pole	Datový typ	Velikost pole	Popis
		OSCIS	text	5	Osobní číslo
		PRIJMENI	text	31	Příjmení
		JMENO	text	25	Jméno
		DATNAROZ	Datum a čas	8	Datum narození
		ZENA	ano/ne	1	Žena?
		ZAJEZD	text	3	Kód zájezdu
		ZALOHA	dvojitá přesnost	8	Zaplacená záloha

Téma I	ZÁJEZDY	Název pole	Datový typ	Velikost pole	Popis
		KOD	text	3	Kód zájezdu
		LOKALITA	text	63	Lokalita ubytování
		POPIS	text	255	Popis zájezdu
		ODJEZD	Datum a čas	8	Datum odjezdu
		PRIJEZD	Datum a čas	8	Datum příjezdu
		REZIE	dlouhé celé číslo	4	Režijní částka cestovky
		CENA	dlouhé celé číslo	4	Cena pro účastníka

Zbývá stanovit logický vztah mezi tabulkami. Ten je však z předchozích úvah zřejmý. Primárním klíčem v odkazované tabulce ZÁJEZDY bude logicky datové pole KÓD, cizím klíčem v odkazující tabulce ÚČASTNÍCI bude pole ZÁJEZD:



10.4 Cvičné soubory pro téma I

Pro výuku je připraven především databázový soubor obsahující pouze data v přiměřeném množství ve struktuře dle předchozí analýzy. Protože tato data už formují tabulky s konkrétní strukturou a konkrétními datovými typy, studenti je použijí až po analýze zadání, off-line návrhu tabulek a diskusi k možnostem. Databázi je pak možno stáhnout - např. do DOKUMENT-ů – jako GEODATA1. Dále je připraven databázový soubor obsahující možné kompletní řešení zmíněného tématu. Tento soubor si studenti stáhnou a otevřou samozřejmě až po vyčerpání svých vlastních intelektuálních schopností samostatně úlohu vyřešit. Soubor jsou dostupný jako GEOPARK1.

Pokud se studenti hlásí jako uživatelé Student na učebně VŠB TU Ostrava, J-409, pak po přihlášení mají cvičnou databázi v dokumentech uživatele Student (t.j. ve "svých" dokumentech) v adresáři **Databáze**.

11 Databázové programy a jejich prostředí

11.1 Databázové programy

Přístup k databázím - jak ke struktuře, tak k datům - lze zajistit dvěma způsoby: buď profesionálním programováním vlastních aplikací, nebo profesionálně vytvořeným uživatelským prostředím. Zde se budeme zabývat druhým způsobem: uživatelské prostředí je realizováno spuštěním konkrétního programu, který patří do třídy tzv. *databázových programů*.

Je zřejmé, že databázové programy jsou vytvořeny pro jednoznačně určené

- operační systémy (např. MS-Dos, Unix),
- grafická rozhraní operačního systému (např. Windows),
- databázové systémy (např. Paradox, xBase).

Běžný uživatel očekává, že mu databázový program umožní (minimálně pro osobní uživatelovy databáze)

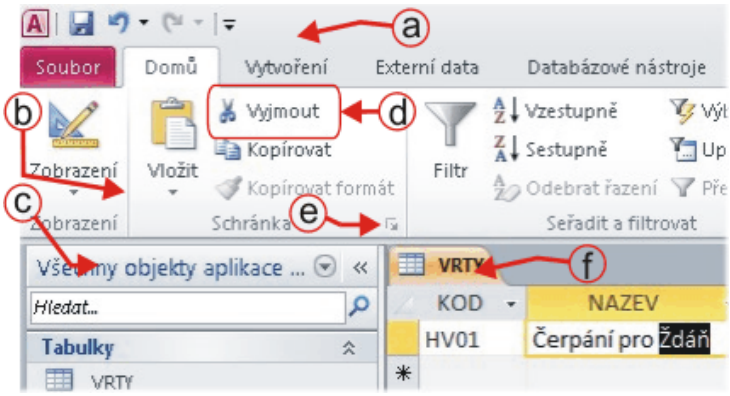
- vytvořit databázi a v ní jednotlivé tabulky včetně klíčů a vazeb mezi nimi, dotazy a další komponenty;
- měnit strukturu jak databáze jako takové, tak jejich jednotlivých komponent;
- vkládat a aktualizovat data s alespoň minimální kontrolou konzistence jak jednotlivých hodnot, tak vzhledem k vazbám mezi daty;
- využívat případně pro práci s daty specifická grafická prostředí (známá např. jako formuláře);
- čerpat informace prostřednictvím definovaných příkazů dotazovacího jazyka (ne nutně definovaných tímto uživatelem);
- data získaná pro informaci uživatele zobrazovat jak na zobrazovacím (displej), tak na trvalém (tiskárna) zařízení.

Tyto funkce - kromě řady dalších - splňuje databázový program dodávaný v rámci Microsoft Office a nazvaný Access. Protože MS Office jsou v praxi široce rozšířené, mají studenti během studia možnost je běžně využívat. Je třeba zmínit, že databázový program Access je začleněn jen do vyšších verzí Office; univerzita však disponuje pro pedagogy i studenty licencemi těch nejvyšších verzí.

Praktické postupy při tvorbě a zpracování databází budou demonstrovány právě v prostředí programu Access. Snahou autorů těchto textů však bylo vybírat takové postupy a taková rozhraní, které čtenáři naleznou ve více méně stejné podobě i funkčnosti ve většině jiných (nejen MS) databázových programů.

11.2 Konvence

V těchto textech je popisován postup v prostředí databázového programu Access, edice Microsoft Office 2010. Tomu odpovídá i terminologie:



Na předchozím obrázku je označeno:

a. Hlavní karty - dříve Hlavní menu, zde aktivní hlavní karta Domů.
b. Skupina příkazů - zde skupina příkazů Schránka.
c. Navigační panel otevřené databáze.
d. Příkaz - zde příkaz Vyjmout.
e. Tlačítko pro Nástroje skupiny.
f. Záložka - zde záložka dat tabulky VRTY.

Ovládací prvek z bodu e. předchozího seznamu bude v textu zobrazováno symbolem ◢. Pro další ovládací prvky je použita standardní terminologie resp. zobrazení (tlačítko, zaškrtávací pole ☐ resp. ☑, rozvíjecí seznam aj.).

Pro vyznačení posloupnosti aktivací návazných ovládacích prvků je použit zápis

`Hlavní karta / Skupina příkazů / Příkaz`

Značí tedy zápis

`Domů / Schránka / Vyjmout`

(viz situaci na předchozím obrázku) požadavek na aktivaci hlavní karty **Domů**, tam vyhledání skupiny **Schránka**, a v ní použít příkaz **Vyjmout**.

Výjimkou je položka **Soubor** hlavního menu, kde na druhém místě je uvedena položka levého panelu a na třetím místě skupina parametrů.

Některé příkazy ve skupině mají pod ikonou nebo vedle textu ještě tlačítko ▼. Pokud to nevyplyne z kontextu, bude výslovně řečeno, zda se má stisknout ikona nebo toto tlačítko pod (vedle) ní.

11.3 Nastavení prostředí

Při práci s databází je v grafickém uživatelském prostředí zapotřebí vidět co nejvíce informací o její struktuře, obsahu, rozložení atd. Jde o takové vizuální informace, které pomáhají uživateli v sestavení databáze, nastavení vlastností a ve zefektivnění činností obecně. Po instalaci je prostředí programu Access nastaveno z hlediska našich lokálních zvyklostí celkem rozumně, není třeba tedy žádných podstatných změn.

Protože však podstatnou součástí výuky tvoří zápis a provádění příkazů SQL je pouze vhodné (pro bystrooké nikoliv nutné) změnit velikost písma v editačním okně SQL. Jde o posloupnost

`Soubor / Možnosti / Návrhář objektů / Návrh dotazu`

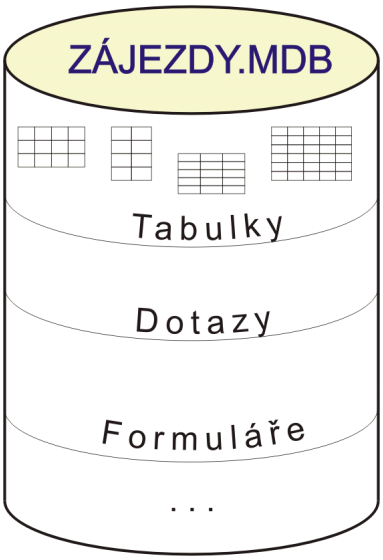
a tam změna velikosti písma - podle zrakové kondice uživatele většinou stačí 12 nebo 14 bodů.

12 Struktura databáze

12.1 Databázový soubor a jeho struktura

Ačkoliv v databázovém prostředí Microsoftu je možno zpracovávat databáze několika různých organizačních struktur, pro seznámení se s databázovou problematikou se jeví jako nejvhodnější organizace označovaná jako Microsoft Jet Database. Jde o relační databázi, jejíž základní (případně všechny) komponenty jsou umístěny v jediném souboru s příponami MDB resp. od verze Office 2007 i ACCDB. Její logickou strukturu lze znázornit např. takto:

Struktura souboru Microsoft Jet. Pouze v části **Tabulky** jsou fyzická data, v ostatních logických částech databáze jsou návrhy (např. dotazů, formulářů).



12.2 Kroky při vytváření databáze

Z předchozího schématu vyplývá základní postup autora databáze:

a. Nejprve je nutno vytvořit soubor databáze jako takový. Databáze bude zpočátku prázdná, v žádné části nebude nic.
b. Má-li databáze být užitečná, musí v ní být nějaká data. Ta mohou být jen v tabulkách - je tedy zapotřebí vytvořit alespoň jednu tabulku s daty.

Z toho však hned vyplývá další: data se mohou zadávat jen do *existující* tabulky. Tabulka začne existovat, až někdo databázi sdělí její strukturu: kolik má sloupců, jak pojmenovaných a jakého typu budou data v jednotlivých sloupcích umísťovaná. Proto tedy druhý krok spočívá ve dvou činnostech (v tomto pořadí):

c. Vytvoří se tabulka definováním její struktury a jejího jména.
d. Vytvořenou tabulku je možno začít plnit daty.

Definice struktury tabulky spočívá v zadání následujících vlastností pro každý sloupec tabulky:

i. Jméno sloupce. Musí být v rámci tabulky jedinečné (tj. v tabulce nemohou existovat dva sloupce se stejným jménem).
ii. Typ dat ve sloupci. Z důvodu přenositelnosti se doporučuje omezit na datové typy Číslo, Datum a čas, Text, a Ano/Ne.
iii. Datový typ Číslo vyžaduje zadání podtypu, datový typ Text zadání max. počtu znaků.

Důležitá poznámka: *Uvedeným postupem je skutečně nutno projít. Při práci s daty (např. při zadávání dat nebo jejich opravě) má v daný okamžik tabulka pevně danou strukturu, kterou při práci s daty obecně nelze změnit. Kdykoliv však je možno práci s daty ukončit, změnit strukturu tabulky (např. přidat sloupec, rozšířit textový sloupec apod.) a následně pokračovat v práci s daty. To je zásadní fakt, který je např. rozevlátým uživatelům Excelu zcela proti mysli. Ti jsou zvyklý při práci frc sem jednu hodnotu, prásk jinam další hodnotu - teď. Kolik času však následně stráví hledáním hodnot umístěných podle*

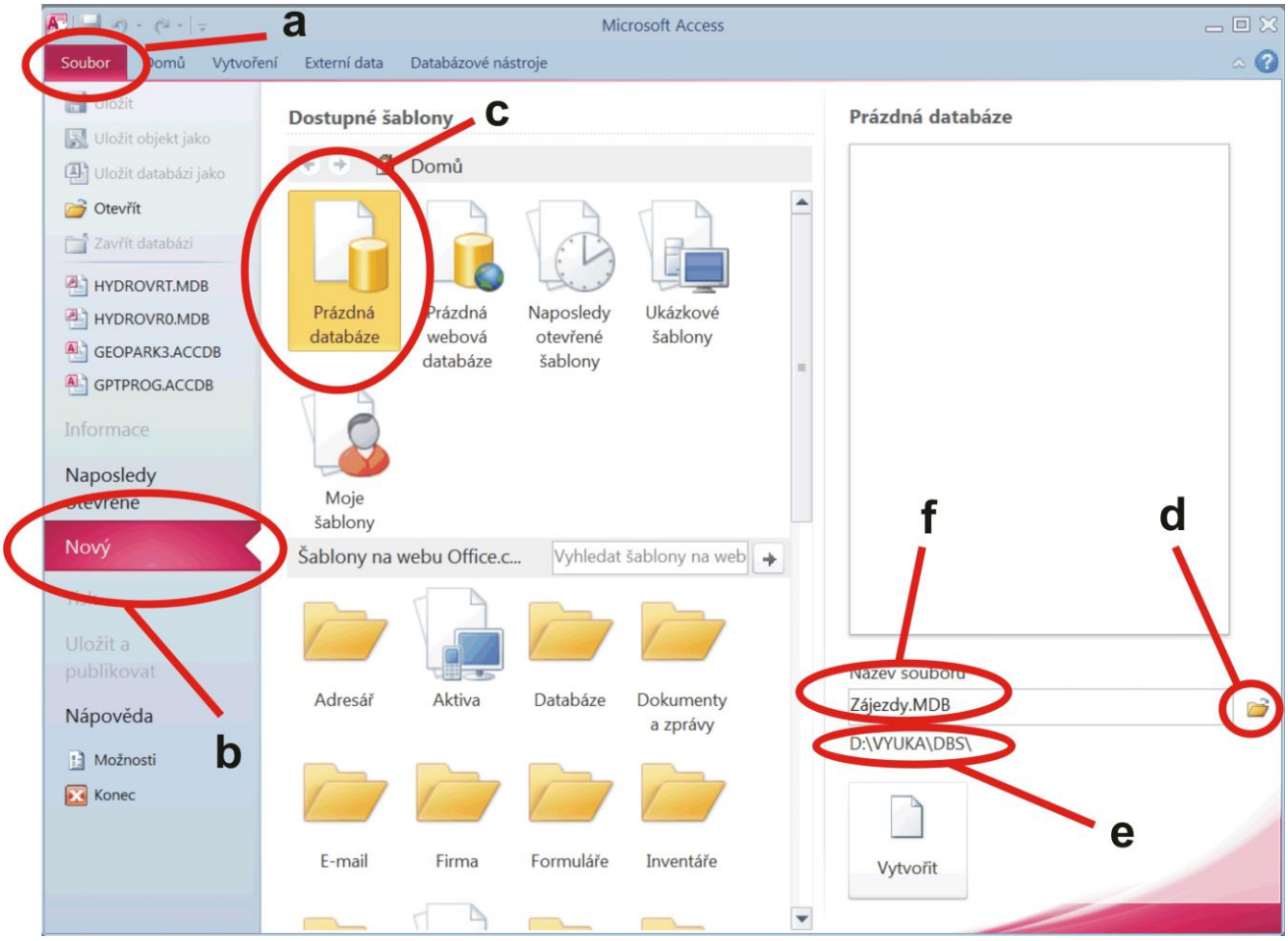
aktuálního mentálního stavu své osobnosti, na to teď nemyslí. Prostě pevná struktura tabulek databáze je svazuje v rozletu a to si nechají líbit. Proto databáze používají snad jen z donucení ☺

13 Vytvoření databáze prakticky

Postupy uvedené v tomto odstavci respektují strukturu tabulek ze shora uvedeného Tématu I.

13.1 Vytvoření nové prázdné databáze

V prostředí programu Access je postup při vytvoření nové prázdné databáze dán následujícím schématem:

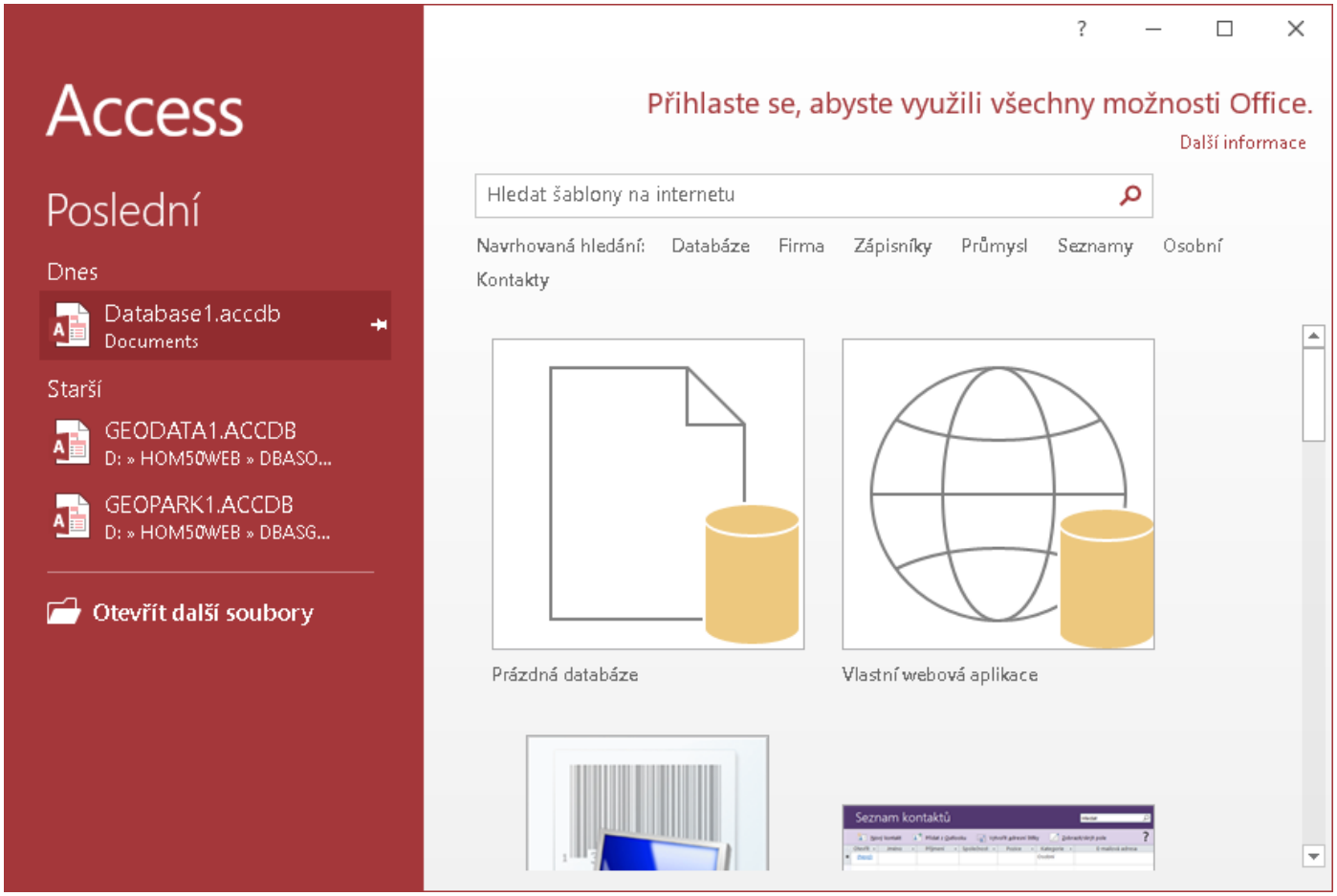


Spočívá tedy v posloupnosti kroků:

a. Aktivace hlavního menu Soubor.
b. Aktivace nabídky Nový.
c. Aktivace nabídky Prázdná databáze.
d. Stisknutím tlačítka s ikonou složky výběr cílové složky. V předloženém dialogovém okně lze současně zadat i typ databáze (Mdb, Accdb) i označení souboru.
e. Kontrola správného umístění souboru.
f. Případné zadání označení souboru, pokud nebylo učiněno v kroku d.

Po stisknutí tlačítka Vytvořit databázový program nejen databázi s požadovaným označením v požadovaném umístění vytvoří, ale rovněž ji otevře a předloží uživateli k dalšímu zpracování.

Jak již bylo řečeno shora, příklady jsou zobrazeny v prostředí programu Access v MS Office 2010. V novějších verzích se mohou mírně odlišovat; vždy je však zachována stejná logika. Např. v prostředí verze Office 2016 vypadá úvodní obrazovka např. takto:



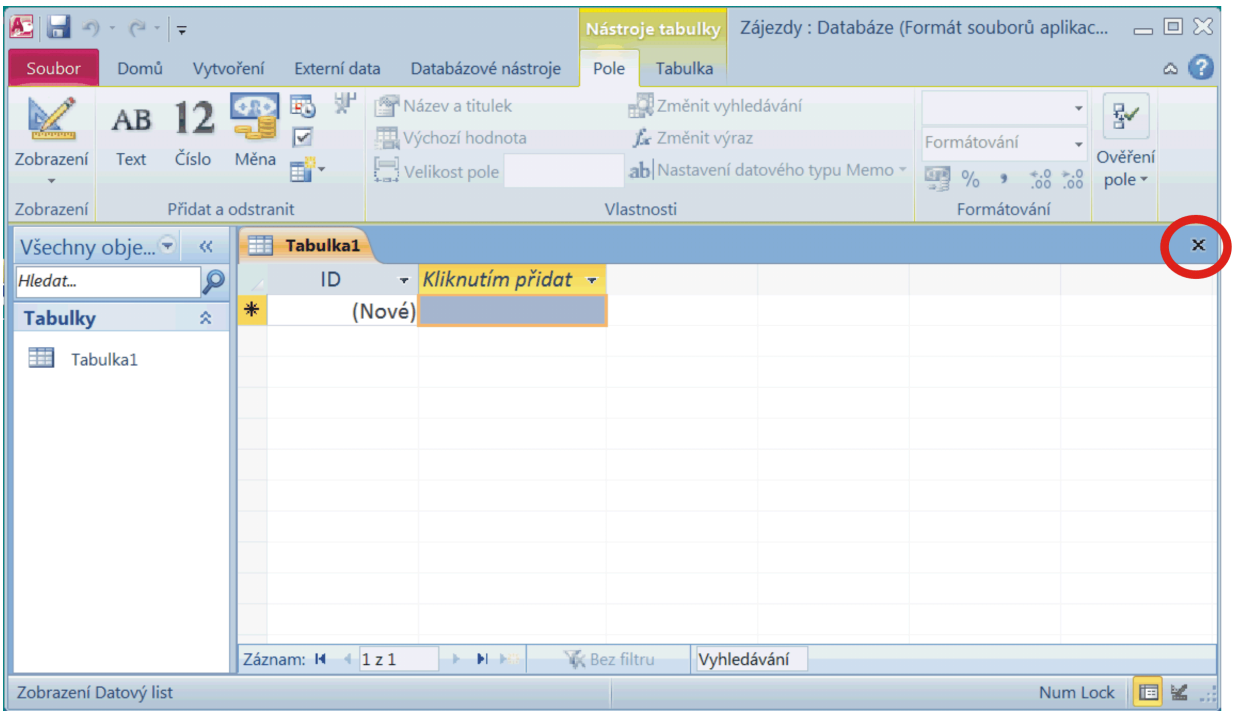
Dotaz na umístění a označení souboru pak následuje po stisknutí *Prázdná databáze*.

13.2 Vytvoření nové tabulky po vytvoření databáze

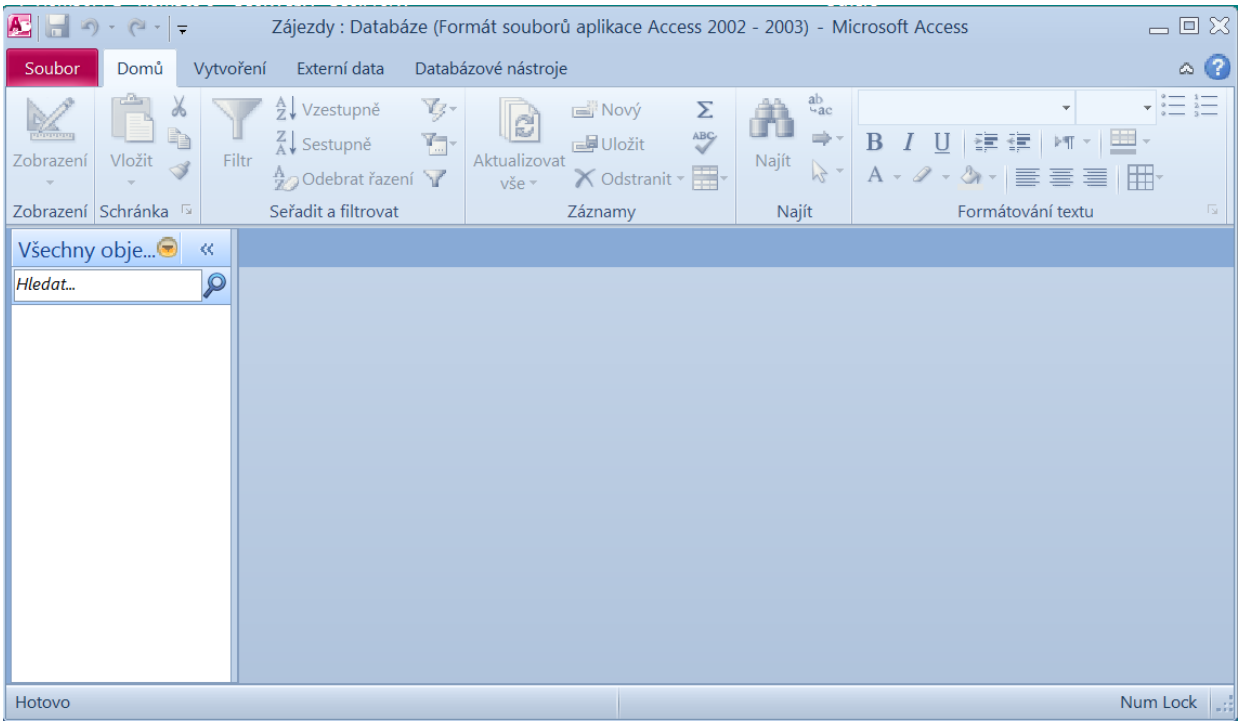
Autoři programu Access zřejmě uvažovali takto: když uživatel právě nechal vytvořit novou prázdnou databázi a když databáze bez dat (tj. bez tabulek) nemá smysl, předložíme mu rovnou prostředí umožňující tabulku vytvořit - a to přímým zadáváním dat z klávesnice. K tomu několik poznámek:

1. Samotná úvaha je nepříliš bystrá. Proč by uživatel musel bezprostředně po vytvoření nové databáze zadávat data první tabulky ručně? Proč by tabulku nemohl vytvořit importem z jiných zdrojů, zvláště když je dat více - a navíc je uživatel výše zmíněný zavilý Excelista?
2. Prostředí, které zde autoři programu Access předkládají uživateli pro vytvoření první tabulky, je (zvláště pro studenty začátečníky) to nejméně vhodné, a to především z didaktických důvodů. Stírá ostrou hranici mezi prací se strukturou tabulek a prací s jejich daty, a uživatele spíše zmate.
3. Takové prostředí se ve většině databázových programů vůbec nevyskytuje. To je další důvod, proč studenty nezatěžovat něčím atypickým.

Ukončit práci se zmíněným nabízeným prostředím lze klasicky dle následujícího obrázku:



Takovým postupem se nic nevytvoří (a uživatel nemusí hledat jak se toho zbavit). Prostředí prázdné databáze je pak připraveno:

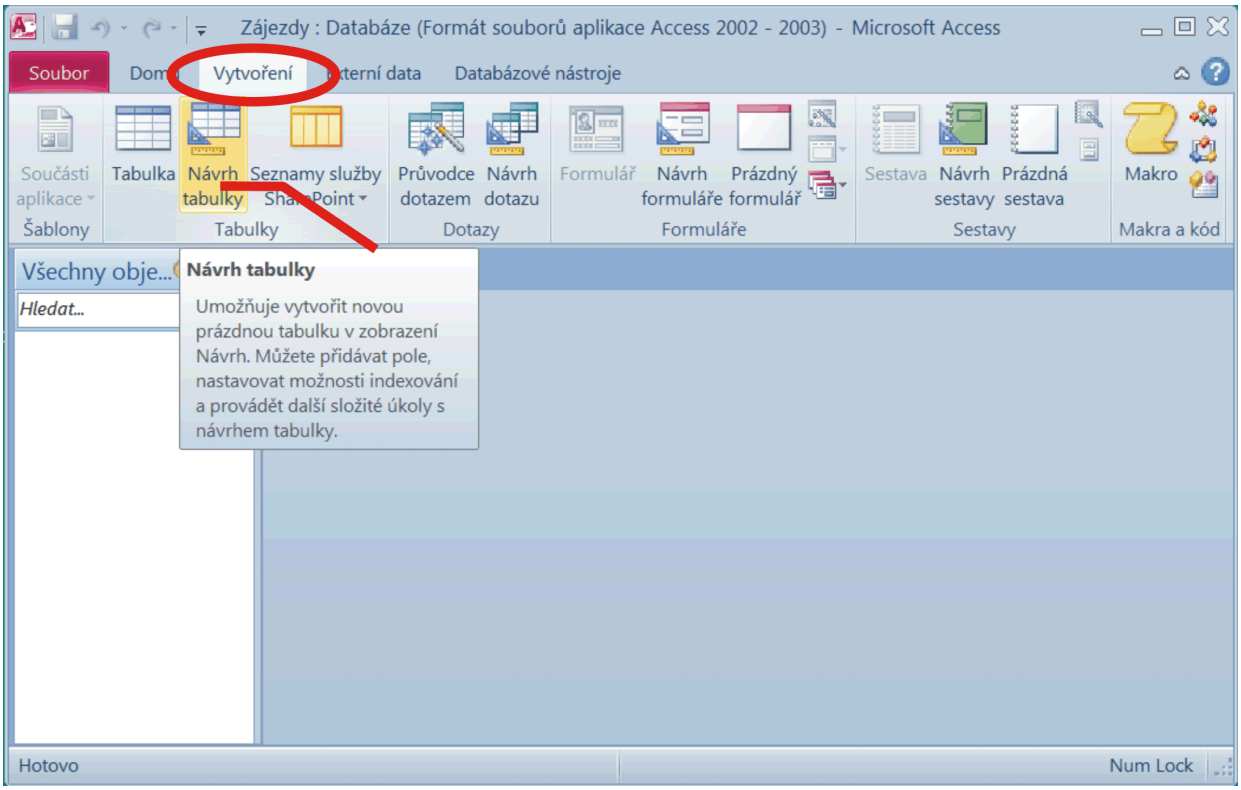


a lze začít využívat standardního postupu, obvyklého ve většině databázových systémů - viz následující odstavce.

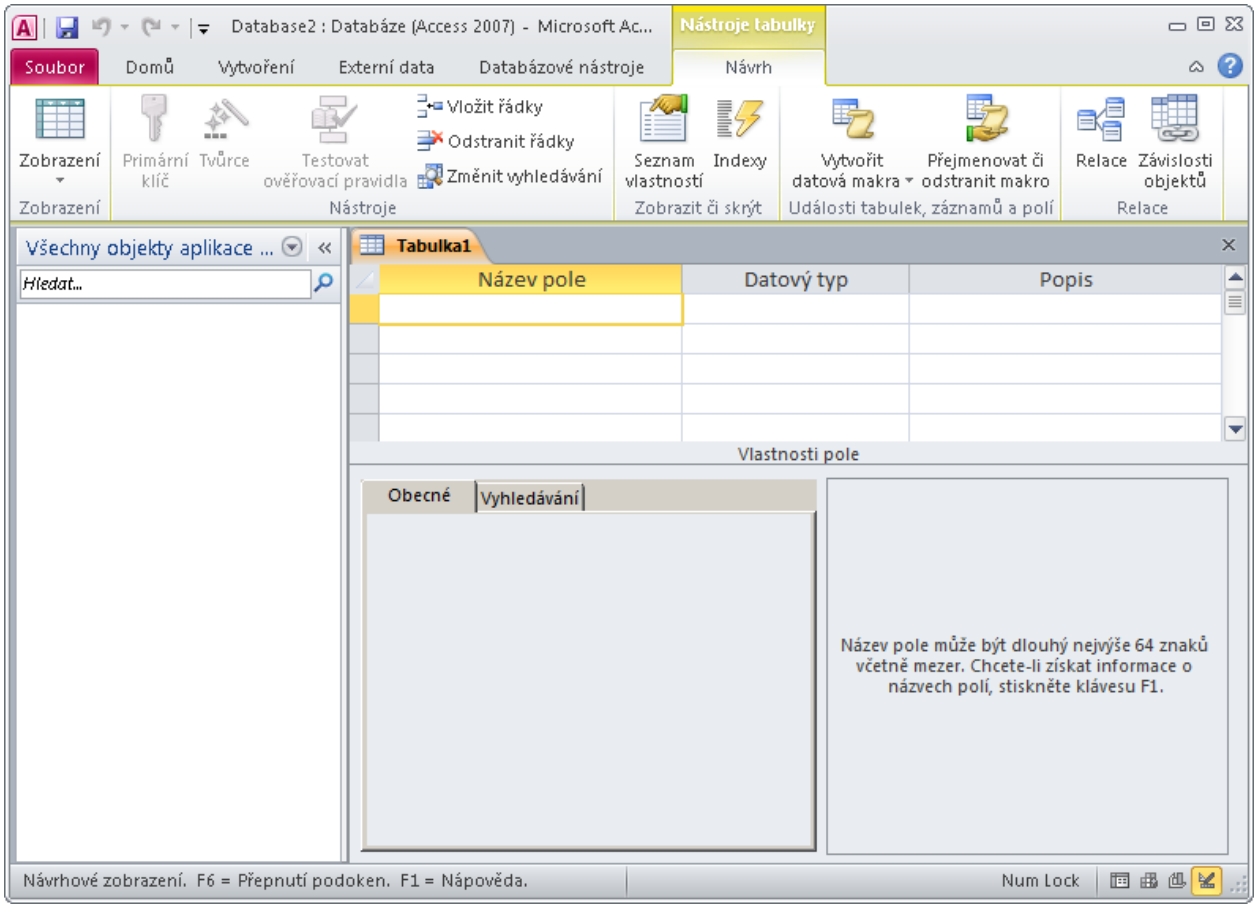
13.3 Vytvoření nové tabulky ve standardním návrhovém prostředí

Jak bylo shora zmíněno, spočívá vytvoření nové tabulky v zadání její struktury: vyjmenování sloupců tabulky a přidělení jejich vlastností. Posloupností

Vytvoření / Tabulky / Návrh tabulky

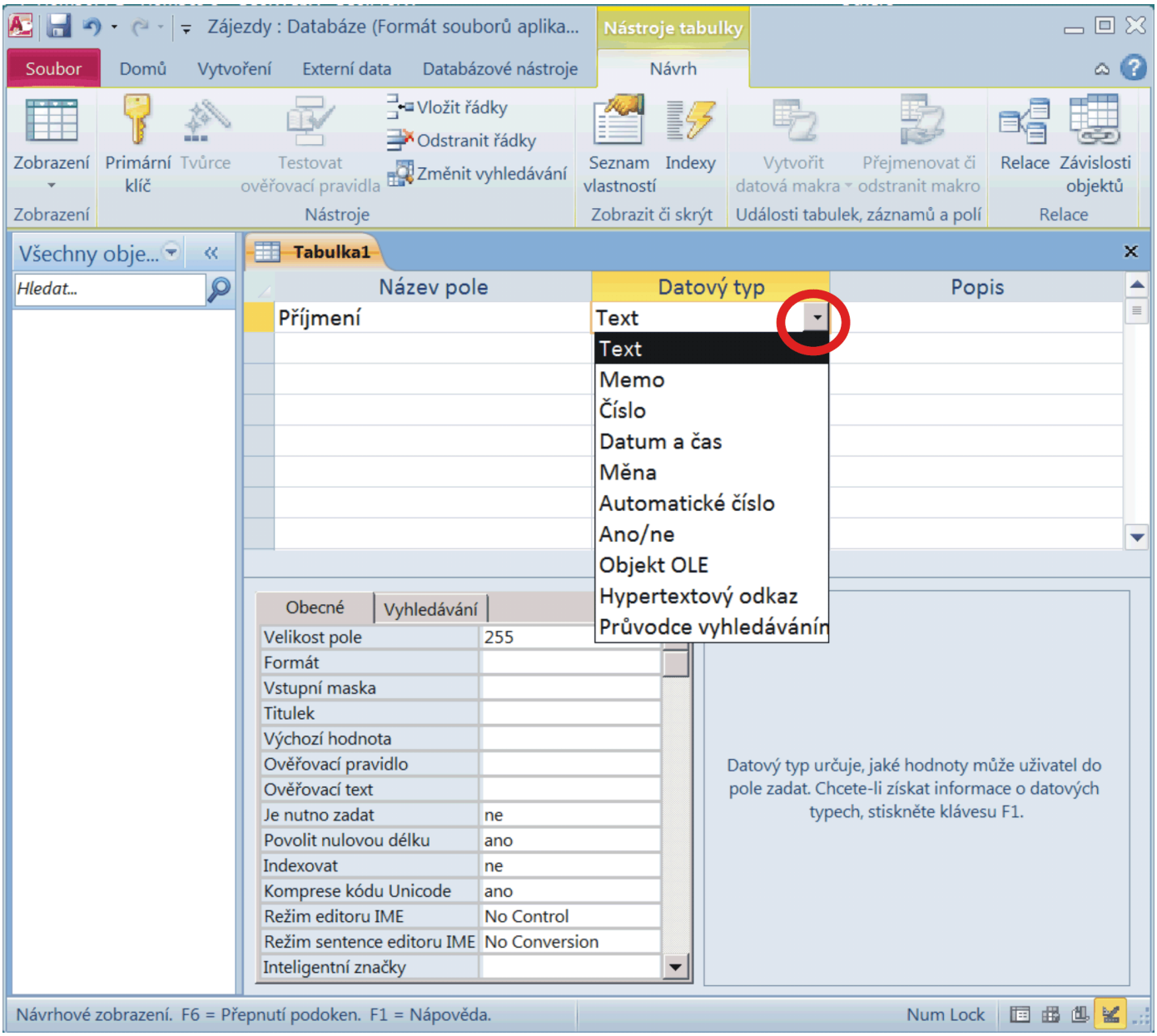


je uživateli předloženo obvyklé prostředí - v terminologii programu Access označované jako *Návrhové zobrazení*:



Jednotlivé *řádky* zde slouží k zadávání informací o jednotlivých *sloupcích* vytvářené tabulky. Za povšimnutí stojí z angličtiny doslova přeložený nadpis *Název pole*. Anglická databázová terminologie (historicky) používá označení *Database Record* (databázový záznam) pro řádek datové tabulky, a ten má *Data Fields* (datová pole).

Po zadání názvu pole (= jména sloupce vytvářené tabulky) je jako datový typ zobrazen Text. Toto počáteční přiřazení datového typu je dáno nastavením programu Access a uživatel programu Access ho může změnit podle svých potřeb. V každém případě lze datový typ definovaného sloupce vybrat z nabídky:

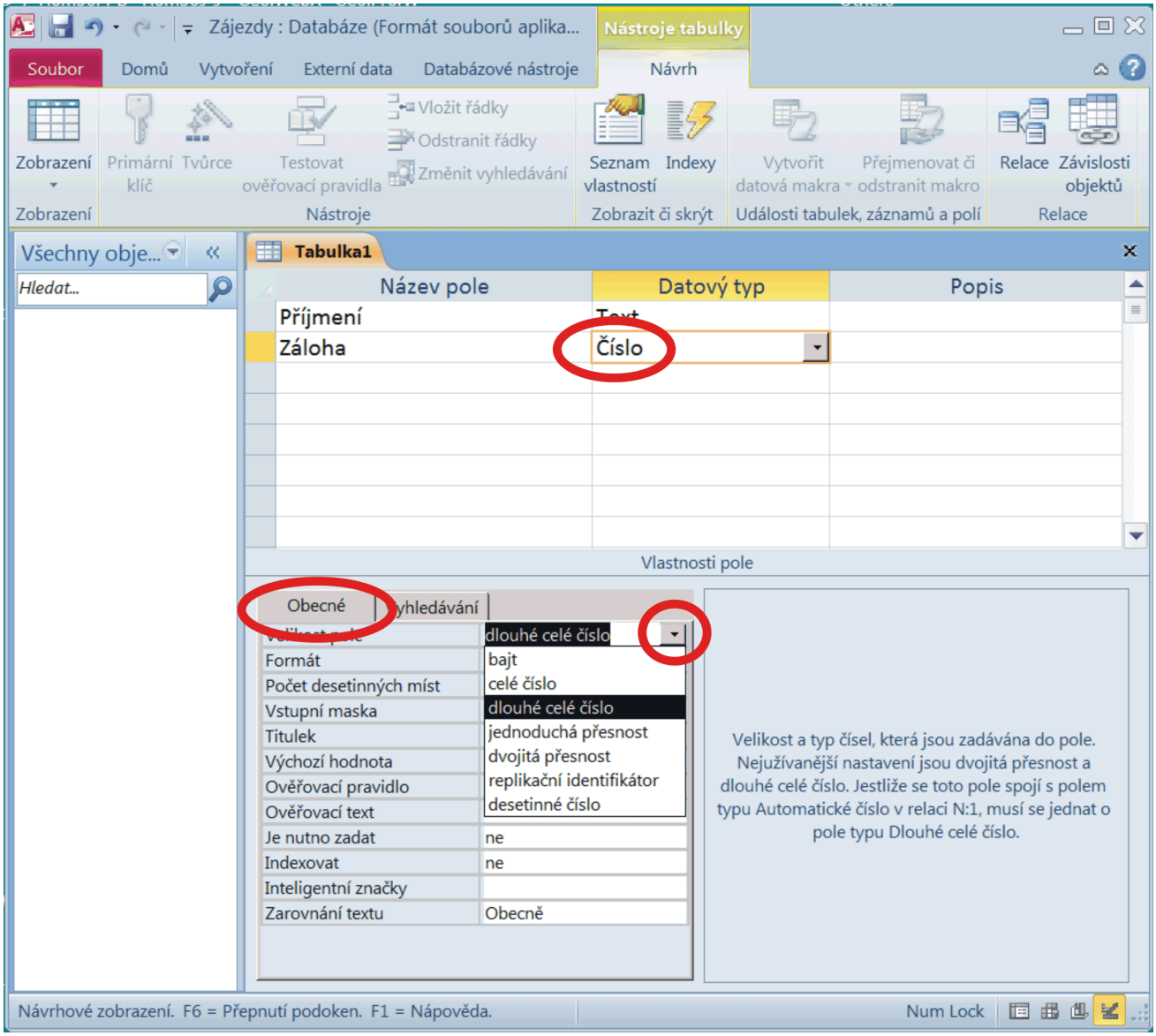


Pokud autor požaduje pro právě definované datové pole jeho začlenění do primárního klíče, použije tlačítko

`Návrh / Nástroje / Primární klíč`

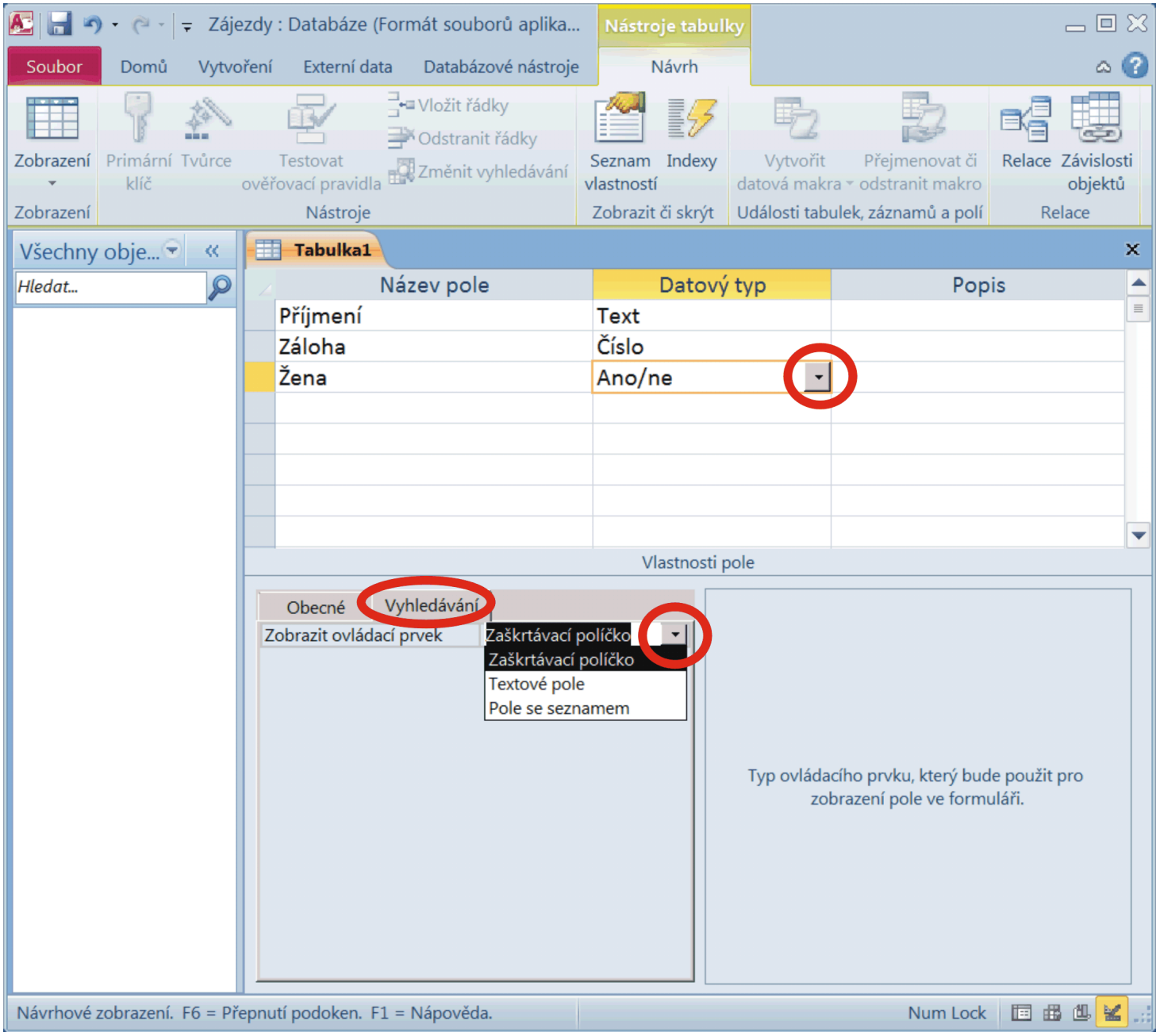
V těchto učebních textech se důsledně držíme práce pouze se čtyřmi datovými typy: **Text**, **Číslo**, **Datum a čas** a **Ano/ne**. Těmito datovými typy totiž disponují všechny databázové systémy a v případě přechodu k práci pod jiným databázovým systémem by neměl být problém aplikovat zde nabyté dovednosti tam.

Už po zadání názvu datového pole jsou ve spodní části návrhového zobrazení nabízeny další vlastnosti, kterými lze ovlivnit vzhled resp. chování dat při práci s nimi. Asi nejpodstatnější vlastnost je zde uváděna jako první - Velikost pole. Uplatní se u textových typů dat (určuje největší počet znaků, který lze do pole zadat), a u číselných dat. Číselný datový typ popišme podrobněji:

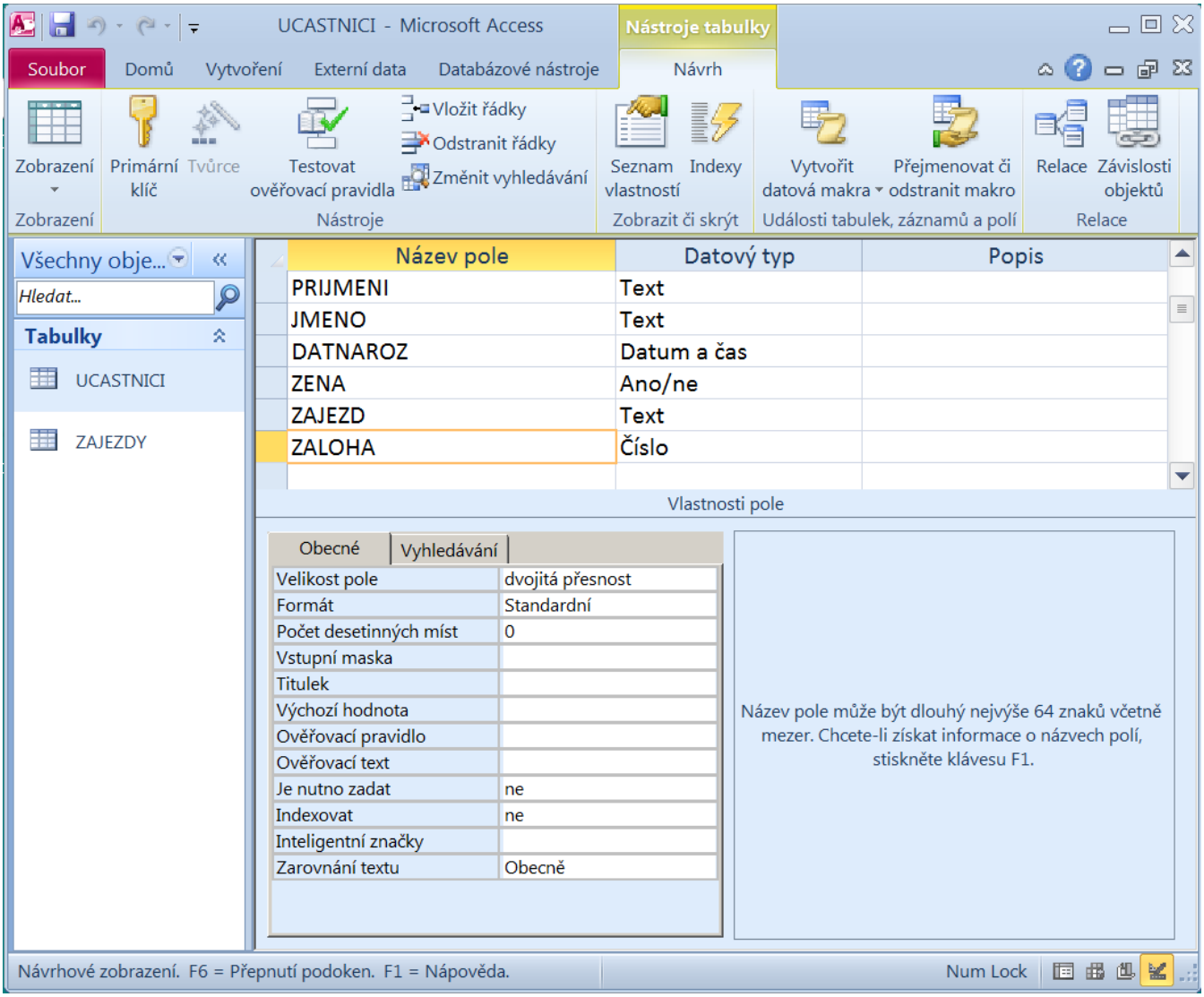


Autor tabulky z nabídky skutečně určuje "velikost pole": kolik bytů "na šířku" bude mít daný sloupec tabulky. Parametry jednotlivých číselných podtypů jsou popsány shora v teoretické části. V těchto učebních textech se důsledně držíme práce pouze s pěti číselnými podtypy: **bajt** (byte), **celé číslo** (integer), **dlouhé celé číslo** (long integer), **jednoduchá přesnost** (single precision) a **dvojitá přesnost** (double precision).

Zvláštní pozornost zaslouží datový typ Ano/ne. Jako jednu z vlastností mu totiž lze nastavit způsob pozdějšího zobrazení v prostředí tzv. datového listu. Může jít především o textové vyjádření, nebo - častější a oblíbené - zaškrtávací políčko (check box):



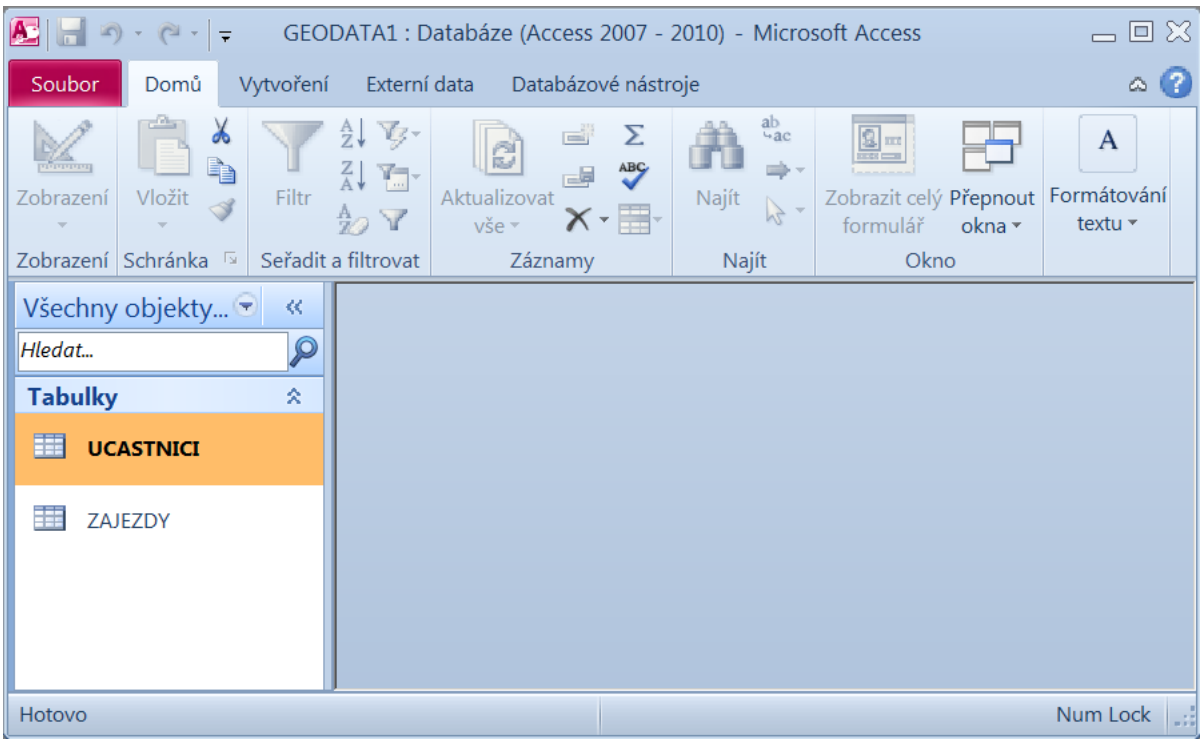
Na závěr návrhové zobrazení pro práci se strukturou tabulky ÚČASTNÍCI z databázového souboru GEODATA1.ACCDB:



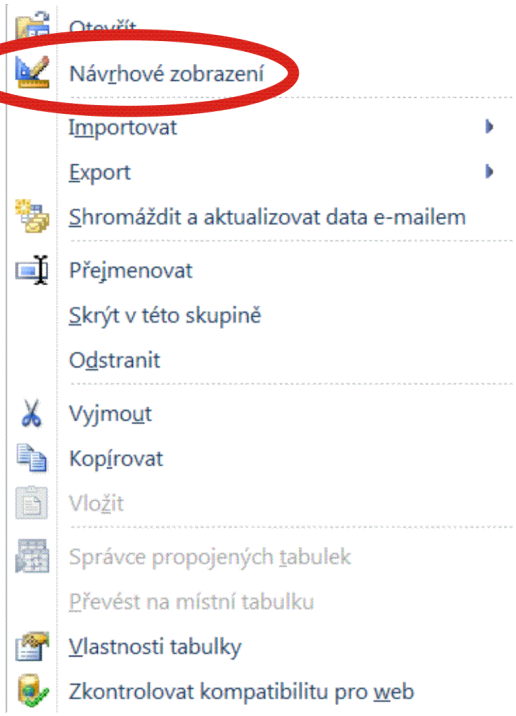
Důležitá poznámka: Vlastnosti, které autor databáze nastavuje jednotlivým datovým polím ve spodní části "Vlastnosti pole" návrhového zobrazení, se v naprosté většině nepřenáší do jiných databázových systémů (např. při exportu). Projeví se jen při práci s touto databází pomocí databázového programu Access. Výjimkou jsou v části "Vlastnosti" podtypy datového typu "Číslo".

13.4 Změna struktury tabulky

Právě popsané návrhové zobrazení se použije nejen při vytváření nové tabulky, ale vždy při potřebě změny struktury existující tabulky - přidání sloupců, změna šířky sloupce, přidání indexů apod. Návrhové zobrazení se strukturou konkrétní tabulky se aktivuje takto (ukázka je pro tabulku ÚČASTNÍCI):



a. Pravý klik na požadovanou tabulku zobrazí kontextové menu.
b. Z něj se vybere položka *Návrhové zobrazení*:



13.5 Vytvoření a úprava relace

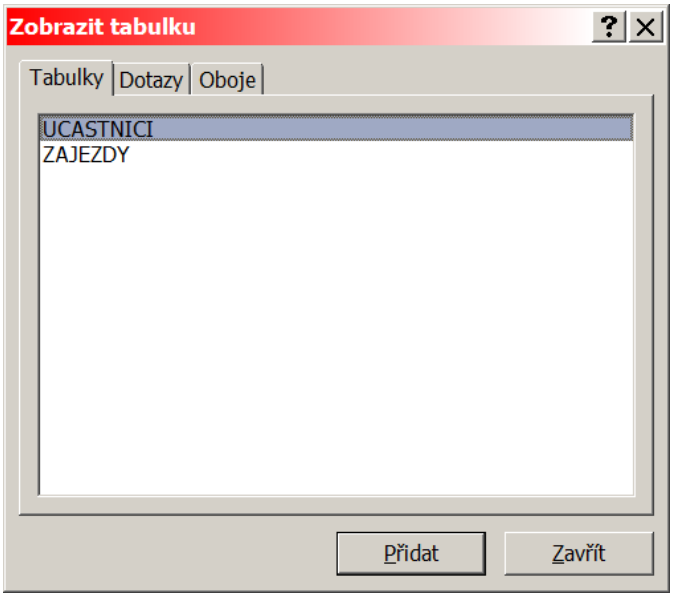
Relace (definice vazby, vztahu) mezi tabulkami se definuje resp. upravuje v návrhovém zobrazení relací. To se aktivuje postupem

Databázové nástroje / Relace / Relace

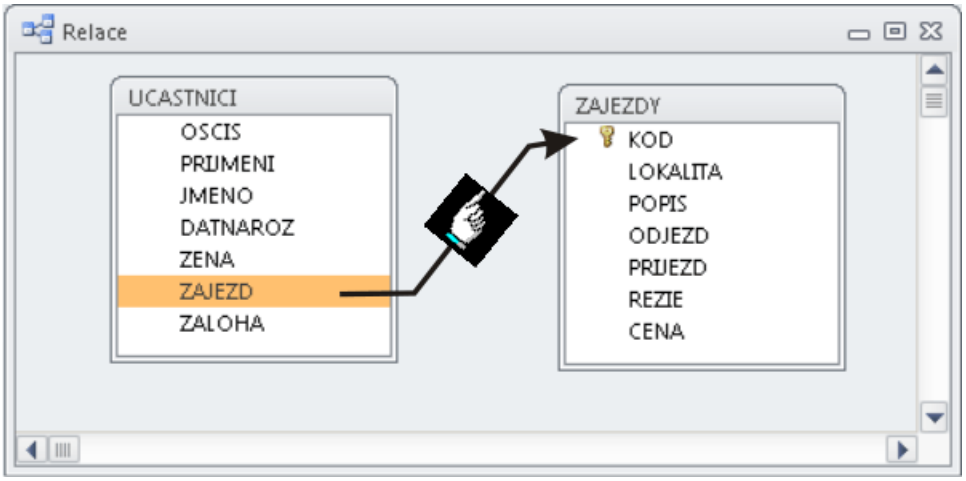
Celý postup při vytvoření relace bude předveden na relaci vycházející z tabulky ÚČASTNÍCI a směrované do tabulky ZÁJEZDY, přičemž vazebními poli jsou ZÁJEZD na straně ÚČASTNÍKŮ a KÓD na straně ZÁJEZDŮ.

13.5.1 Vytvoření relace

Při první aktivaci návrhového prostředí předloží databázový program uživateli dialog *Zobrazit tabulku*. V něm je úkolem návrháře přidat do návrhového prostředí ty datové zdroje, které se budou účastnit vytvářené relace. V popisovaném příkladu se vazby budou účastnit obě tabulky ÚČASTNÍCI i ZÁJEZDY, postupně se tedy označí a tlačítkem *Přidat* přidají obě. Dialog se poté zavře.

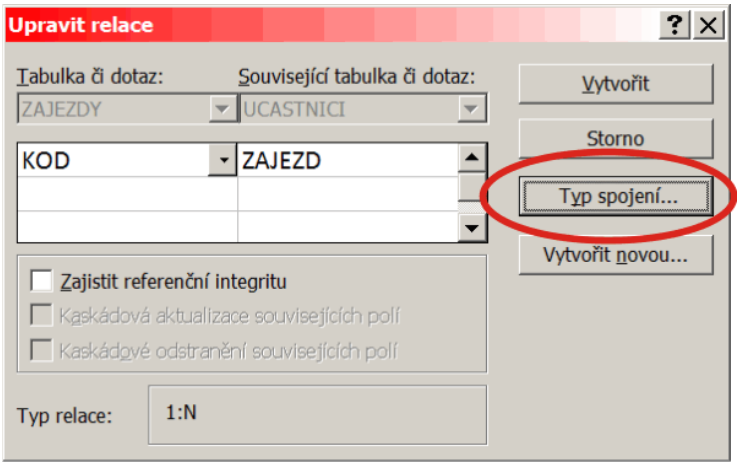


V návrhovém prostředí jsou nyní zobrazeny vybrané datové zdroje. Nyní se vazba celkem intuitivně vytvoří tažením myší (drag and drop): uchopí se položka ZÁJEZD a přetáhne se do položky KÓD:

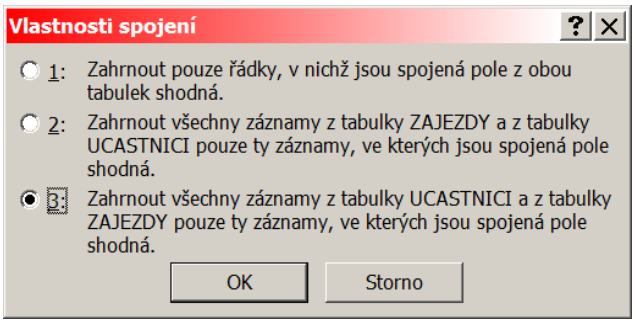


Jakmile je v položce KÓD uvolněno tlačítko myši, předloží databázový program další dialog, pomocí kterého se kompletně definují vlastnosti vytvářené relace.

Jako první se volí tzv. *typ spojení*:

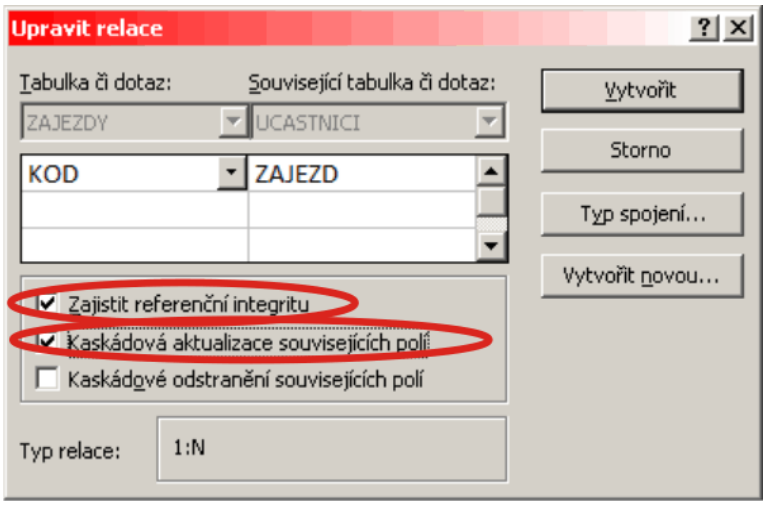


V předloženém formuláři se určí vzájemný vztah mezi datovými zdroji (v tomto případě tabulkami): která z nich je odkazující a která odkazovaná (nabídky 2 a 3), případně jestli je jejich vztah "rovnocenný" (nabídka 1). Vždy je nutno velmi pečlivě uvážit, k jakému vztahu se skutečně směřuje; souvislost mezi směrem tažení (z které tabulky do které) zde často není. V diskutovaném tématu jde o nabídku 3; skutečně požadujeme procházet všechny účastníky a ke každému přiřadit ten "jeho" zájezd:

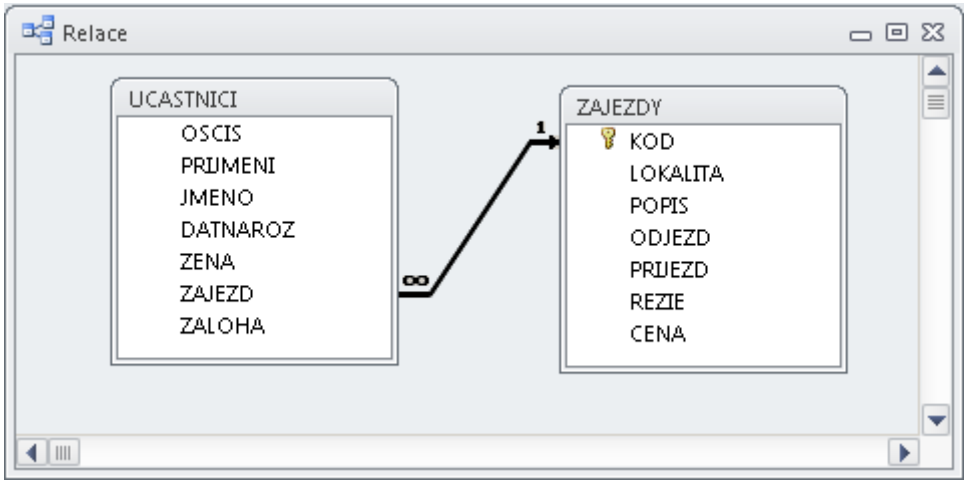


V dalším kroku volíme, zda požadujeme nebo nepožadujeme některé systémem zajišťované mechanismy.

- Zajištění referenční integrity: databázový systém kontroluje, zda se v odkazované tabulce vyskytuje hodnota s klíčem, který je rovný cizímu klíči v odkazující tabulce.
- Kaskádová aktualizace souvisejících polí: databázový systém automaticky změní hodnoty cizích klíčů v odkazující tabulce při změně klíče v odkazované tabulce.
- Kaskádové odstranění souvisejících polí: databázový systém automaticky odstraní všechny záznamy v odkazující tabulce mající stejný cizí klíč jako klíč v odstraněném řádku odkazované tabulky.



Po stisknutí tlačítka Vytvořit je vytvoření relace s danými vlastnostmi hotovo:



Ve schématu značí znak ∞ u šipky skutečnost, že na straně odkazující tabulky může být libovolný počet záznamů se stejným cizím klíčem, kdežto znak 1 na straně odkazované tabulky je takový záznam jediný. Takový typ relace bývá označován jako 1:N.

13.5.2 Úprava relace

Zobrazit formulář pro úpravu relace lze při otevřeném návrhovém prostředí relací

Databázové nástroje / Relace / Relace

buď dvoj-klikem na spojovací čáru v návrhovém prostředí pro relace, nebo čáru myší pomocí kliknutí vybrat a pak

Nástroje pro relace / Návrh / Nástroje / Upravit relace

Relaci lze odstranit tak, že se nejprve kliknutím vybere příslušná spojovací čára a ta se poté klávesou Delete smaže.

Formulář pro úpravu i odstranění lze rovněž vybrat z kontextového menu spojovací čáry, které se zobrazí pravým kliknutím na spojovací čáru.

14 Přímá práce s daty na uživatelské úrovni

V předchozí kapitole byla popsána práce se strukturou tabulek a vazbami mezi nimi. Málokterý zaměstnavatel však povolí svým běžným zaměstnancům (účetním, personalistům apod.) pracovat se strukturou svých dat už s ohledem na možnou destrukci firemní databáze při nekvalifikovaném přístupu. Zajisté však bude po nich požadovat údržbu dat - minimálně těch, které spadají do okruhu jejich profesí. Právě základním činnostem nad daty je věnována tato kapitola.

Stejně - jako již několikrát - zdůrazňujeme, že různé databázové programy poskytují různé možnosti. V těchto textech jsou popsány činnosti a prostředí vyskytující se ve více či méně stejné podobě a funkčnosti ve většině z dostupných programů.

14.1 Datový list

Základním prostředím, ve kterém uživatel pracuje s daty, je (terminologií programu Access) *datový list*. Pokud správce databáze některé činnosti uživatelům neomezí, provádí uživatel v prostředí datového listu zejména

- vkládání nových záznamů,
- vypouštění nepotřebných nebo neplatných záznamů,

- změnu hodnot datových polí v některých záznamech.

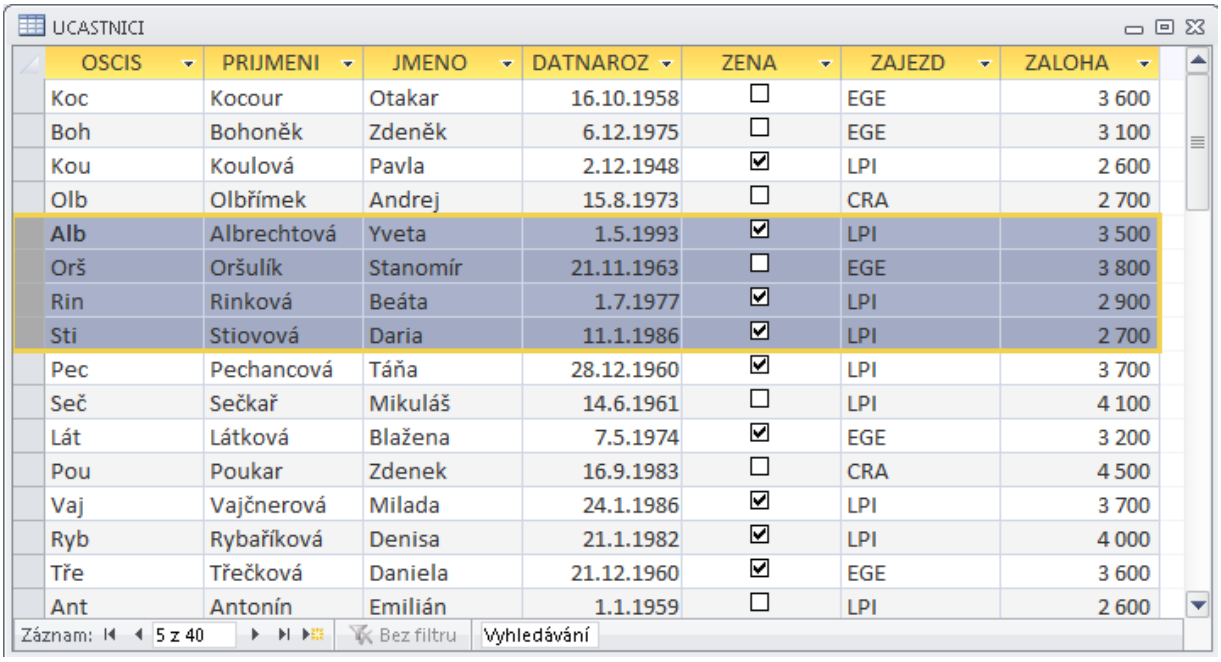
Pro hledání záznamů, se kterými je zapotřebí pracovat - i pro pouhé prohlížení dat a vyvozování informací z nich - nabízí databázové programy mechanismy datových listů pro

- zobrazení záznamů v požadovaném pořadí (nepřesně označované jako řazení - data sorting),
- zobrazení jen záznamů splňujících požadovaná kritéria (tzv. filtrování dat),
- zobrazení jen některých datových polí v požadovaném pořadí.

Datový list otevře databázový program např. dvojklikem na ikonu požadované tabulky v levém panelu databáze. Lze použít i pravý klik na požadovanou tabulku a následným výběrem položky *Otevřít* z rozvinutého kontextového menu. Pro demonstrační tabulku ÚČASTNÍCI má datový list následující podobu:

V prostředí datového listu má každý řádek zdroj v jednom záznamu, sloupce jsou tedy tvořeny datovými poli jednotlivých záznamů. Na začátku každého řádku je vloženo tlačítko (bez textu), stejně tak v záhlaví sloupců jsou tlačítka (s textem rovným názvu příp. nadpisu datového pole). Při práci v prostředí datového listu se uživatel orientuje především pomocí tzv. aktuálního záznamu. Je to ten jediný záznam, s jehož datovými poli se v daný okamžik pracuje. Opticky je označen zvýrazněním tlačítka na začátku příslušného řádku. Ve spodní části datového listu, v jeho stavovém řádku, je zobrazeno pořadové číslo záznamu (počítáno shora od jedné v rámci *zobrazených* záznamů), a rovněž celkový počet v datovém listu zobrazených záznamů. Uživatel mění polohu aktuálního řádku kliknutím na kterékoliv jeho datové pole, kurzorovými klávesami, klávesami Page Up a Page Down, případně tlačítky se šipkami ve stavovém řádku.

K označení (výběru) jednoho nebo více po sobě jdoucích řádků slouží tlačítka na začátku řádků. Stisknutím jednoho se záznam vybere a stává se aktuálním záznamem. Stisknutím a tažením nahoru resp. dolu se spojitě vybírá více záznamů, přičemž první z nich, jehož tlačítko se stiskne, se stává aktuálním záznamem. Vybrané záznamy lze především odstranit (např. klávesa Delete), nebo klasicky zkopírovat a přes schránku přidat jejich kopii jako nové záznamy tabulky (v tom případě vždy na konec bez ohledu polohu aktuálního řádku):



UCASTNICI

OSCIS	PRIJMENI	JMENO	DATNAROZ	ZENA	ZAJEZD	ZALOHA
Koc	Kocour	Otakar	16.10.1958	☐	EGE	3 600
Boh	Bohoněk	Zdeněk	6.12.1975	☐	EGE	3 100
Kou	Koulová	Pavla	2.12.1948	☑	LPI	2 600
Olb	Olbřímek	Andrej	15.8.1973	☐	CRA	2 700
Alb	Albrechtová	Yveta	1.5.1993	☑	LPI	3 500
Orš	Oršulík	Stanomír	21.11.1963	☐	EGE	3 800
Rin	Rinková	Beáta	1.7.1977	☑	LPI	2 900
Sti	Stiovová	Daria	11.1.1986	☑	LPI	2 700
Pec	Pechancová	Táňa	28.12.1960	☑	LPI	3 700
Seč	Sečkař	Mikuláš	14.6.1961	☐	LPI	4 100
Lát	Látková	Blažena	7.5.1974	☑	EGE	3 200
Pou	Poukar	Zdenek	16.9.1983	☐	CRA	4 500
Vaj	Vajčnerová	Milada	24.1.1986	☑	LPI	3 700
Ryb	Rybaříková	Denisa	21.1.1982	☑	LPI	4 000
Tře	Třečková	Daniela	21.12.1960	☑	EGE	3 600
Ant	Antonín	Emilián	1.1.1959	☐	LPI	2 600

Záznam: 5 z 40 Bez filtru Vyhledávání

14.2 Řazení

Jednou z nejčastěji požadovaných funkcí databázových programů je zobrazit uživateli data v uživatelem stanoveném pořadí. Řadit data lze dle jednoho nebo více kritérií. Jako první popišme požadavek na řazení podle jednoho kritéria - např. podle příjmení, posléze podle více kritérií.

14.2.1 Řazení podle jednoho kritéria

Uživatel nejprve označí ten (jediný) sloupec = datové pole, podle něhož požaduje zobrazit seřazené záznamy, a to stisknutím tlačítka se jménem (event. nadpisem) sloupce:

UCASTNICI

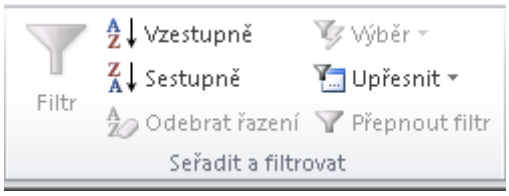
OSCIS	PRIJMENI	JMENO	DATNAROZ	ZENA	ZAJEZD	ZALOHA
Koc	Kocour	Otakar	16.10.1958	☐	EGE	3 600
Boh	Bohoněk	Zdeněk	6.12.1975	☐	EGE	3 100
Kou	Koulová	Pavla	2.12.1948	☑	LPI	2 600
Olb	Olbřímek	Andrej	15.8.1973	☐	CRA	2 700
Alb	Albrechtová	Yveta	1.5.1993	☑	LPI	3 500
Orš	Oršulík	Stanomír	21.11.1963	☐	EGE	3 800
Rin	Rinková	Beáta	1.7.1977	☑	LPI	2 900
Sti	Stiovová	Daria	11.1.1986	☑	LPI	2 700
Pec	Pechancová	Táňa	28.12.1960	☑	LPI	3 700
Seč	Sečkař	Mikuláš	14.6.1961	☐	LPI	4 100
Lát	Látková	Blažena	7.5.1974	☑	EGE	3 200
Pou	Poukar	Zdenek	16.9.1983	☐	CRA	4 500
Vaj	Vajčnerová	Milada	24.1.1986	☑	LPI	3 700
Ryb	Rybaříková	Denisa	21.1.1982	☑	LPI	4 000
Tře	Třečková	Daniela	21.12.1960	☑	EGE	3 600
Ant	Antonín	Emilián	1.1.1959	☐	LPI	2 600

Záznam: 1 z 40 Bez filtru Vyhledávání

Poté zvolí způsob řazení

Domů / Seřadit a filtrovat / Vzestupně či Sestupně

tedy z karty



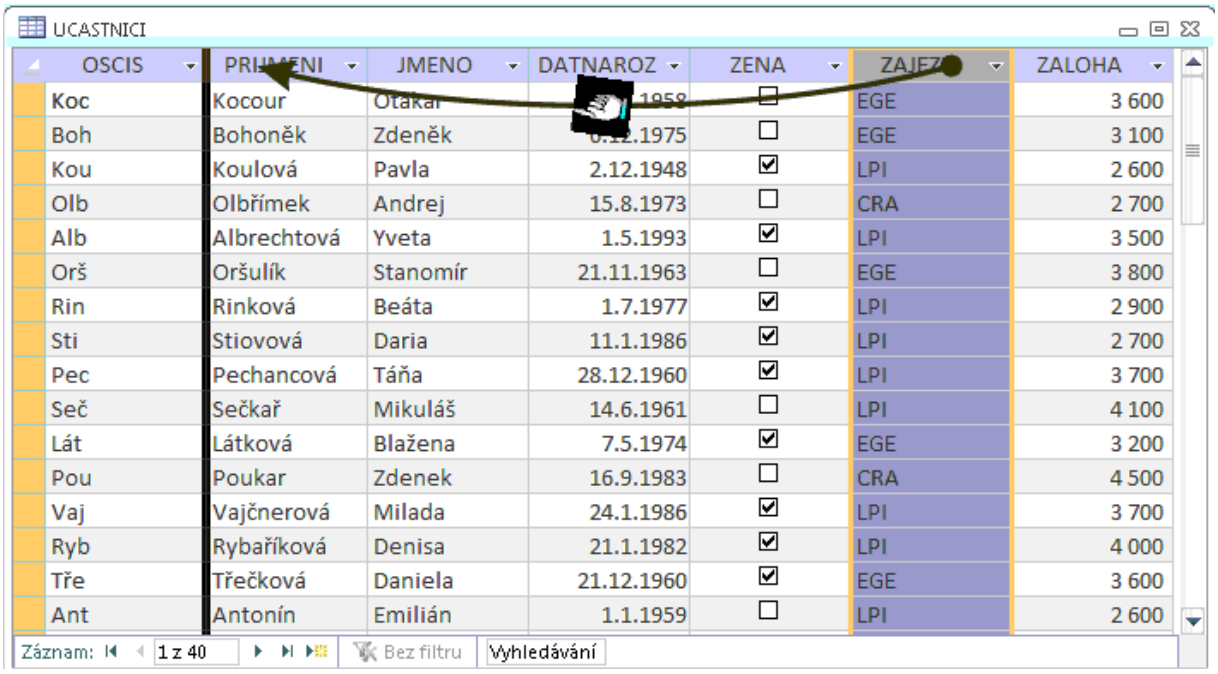
Výsledkem je takové zobrazení záznamů v datovém listu, které respektuje požadovaný způsob řazení (zde tedy dle příjmení "podle abecedy"; pro jednoduchost netřeba uvádět výsledek).

14.2.2 Řazení podle více kritérií

Zde je situace složitější už pro vlastní chápání mechanismu řazení "podle více kritérií". *Neřadí se* postupně podle prvního, druhého - až n-tého kritéria (!). *Nejprve* se seřadí podle hodnot *prvního* kritéria. Pak ve skupinách záznamů se *stejným prvním* kritériem se řadí podle *druhého kritéria*. Následně ve skupinách záznamů se *stejným prvním i druhým* kritériem se řadí podle třetího kritéria ... atd. Uveďme příklad pro řazení podle ZÁJEZDU a PŘÍJMENÍ (v tomto pořadí).

V prostředí datového listu je zapotřebí dát databázovému programu najevo náš požadavek nejen na počet řadících kriterií, ale i na posloupnost jejich důležitosti. Evidentně se počet kriterií zadá počtem označených sloupců (zde dvou). Ovšem pořadí důležitosti je možno zadat několika způsoby. Databázový program Access nabízí v prostředí datového listu tu - v našich končinách - intuitivní možnost: zleva doprava. Ovšem jednak nelze zvolit několik sloupců (= datových polí) nespojitě - stejně jako např. v programu Excel "s CTRL", ale nelze tím stanovit pořadí důležitosti. Proto v prostředí datového listu programu Access je zavedena možnost změny pořadí sloupců (=datových polí) dočasně podle požadavků uživatele. Protože pak jde o sloupce "vedle sebe", lze jich označit současně více.

Změna pořadí sloupců: Tlačítkem v záhlaví sloupce se označí celý sloupec. Následně se - opět tlačítkem v záhlaví označeného sloupce - tažením sloupec umístí na požadovanou pozici:



Po přemístění pak datový list respektuje nové pořadí sloupců:

UCASTNICI

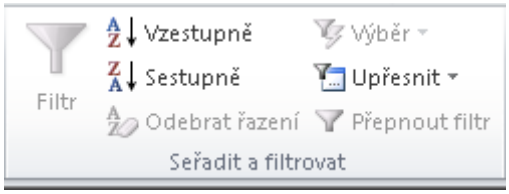
OSCIS	ZAJEZD	PRIJMENI	JMENO	DATNAROZ	ZENA	ZALOHA
Koc	EGE	Kocour	Otakar	16.10.1958	☐	3 600
Boh	EGE	Bohoněk	Zdeněk	6.12.1975	☐	3 100
Kou	LPI	Koulová	Pavla	2.12.1948	☑	2 600
Olb	CRA	Olbřímek	Andrej	15.8.1973	☐	2 700
Alb	LPI	Albrechtová	Yveta	1.5.1993	☑	3 500
Orš	EGE	Oršulík	Stanomír	21.11.1963	☐	3 800
Rin	LPI	Rinková	Beáta	1.7.1977	☑	2 900
Sti	LPI	Stiovová	Daria	11.1.1986	☑	2 700
Pec	LPI	Pechancová	Táňa	28.12.1960	☑	3 700
Seč	LPI	Sečkař	Mikuláš	14.6.1961	☐	4 100
Lát	EGE	Látková	Blažena	7.5.1974	☑	3 200
Pou	CRA	Poukar	Zdenek	16.9.1983	☐	4 500
Vaj	LPI	Vajčnerová	Milada	24.1.1986	☑	3 700
Ryb	LPI	Rybaříková	Denisa	21.1.1982	☑	4 000
Tře	EGE	Třečková	Daniela	21.12.1960	☑	3 600
Ant	LPI	Antonín	Emilián	1.1.1959	☐	2 600

Záznam: 1 z 40 | Bez filtru | Vyhledávání

Nyní (protože sloupce již jsou vedle sebe) je lze označit současně a stejně jako shora v případě jediného kritéria zvolit požadovaný způsob řazení:

Domů / Seřadit a filtrovat / Vzestupně či Sestupně

a volit z karty



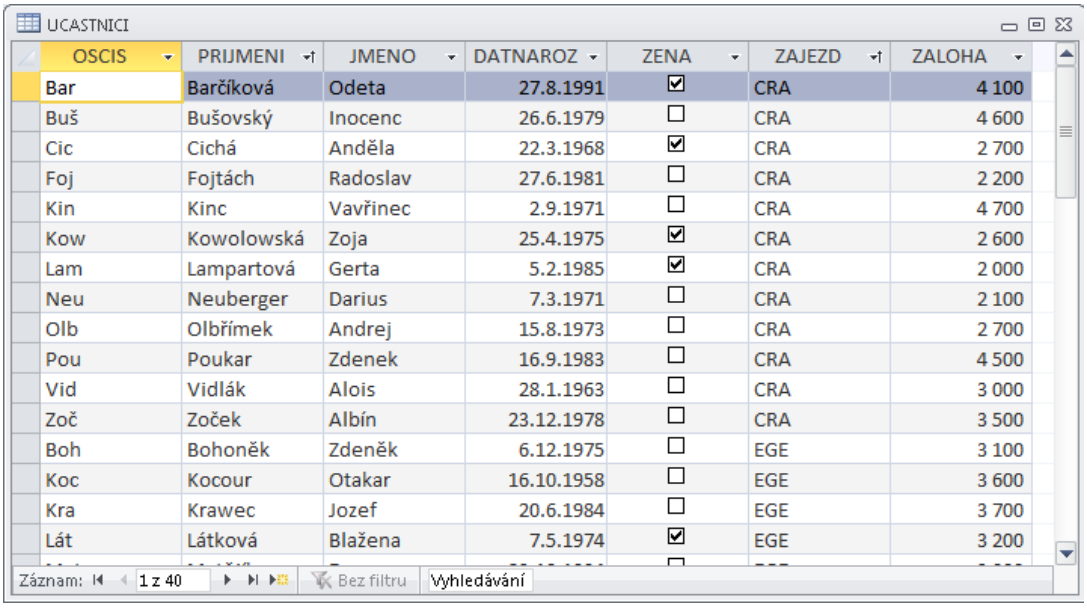
Výsledkem je takové zobrazení záznamů v datovém listu, které respektuje požadovaný způsob řazení (zde tedy dle zájezdu "podle abecedy" a ve skupinách řádků se stejným zájezdem dle příjmení opět "podle abecedy"):

UCASTNICI

OSCIS	ZAJEZD	PRIJMENI	JMENO	DATNAROZ	ZENA	ZALOHA
Bar	CRA	Barčíková	Odeta	27.8.1991	☑	4 100
Buš	CRA	Bušovský	Inocenc	26.6.1979	☐	4 600
Cic	CRA	Cichá	Anděla	22.3.1968	☑	2 700
Foj	CRA	Fojtách	Radoslav	27.6.1981	☐	2 200
Kin	CRA	Kinc	Vavřinec	2.9.1971	☐	4 700
Kow	CRA	Kowolowská	Zoja	25.4.1975	☑	2 600
Lam	CRA	Lampartová	Gerta	5.2.1985	☑	2 000
Neu	CRA	Neuberger	Darius	7.3.1971	☐	2 100
Olb	CRA	Olbřímek	Andrej	15.8.1973	☐	2 700
Pou	CRA	Poukar	Zdenek	16.9.1983	☐	4 500
Vid	CRA	Vidlák	Alois	28.1.1963	☐	3 000
Zoč	CRA	Zoček	Albín	23.12.1978	☐	3 500
Boh	EGE	Bohoněk	Zdeněk	6.12.1975	☐	3 100
Koc	EGE	Kocour	Otakar	16.10.1958	☐	3 600
Kra	EGE	Krawec	Jozef	20.6.1984	☐	3 700
Lát	EGE	Látková	Blažena	7.5.1974	☑	3 200

Záznam: 1 z 40 | Bez filtru | Vyhledávání

Formálně - pro zachování původního pořadí sloupců - je možno přemístěný sloupec vrátit zpět. Použitý způsob řazení tím není dotčen:



UCASTNICI

OSCIS	PRIJMENI	JMENO	DATNAROZ	ZENA	ZAJEZD	ZALOHA
Bar	Barčíková	Odeta	27.8.1991	☑	CRA	4 100
Buš	Bušovský	Inocenc	26.6.1979	☐	CRA	4 600
Cic	Cichá	Anděla	22.3.1968	☑	CRA	2 700
Foj	Fojtách	Radoslav	27.6.1981	☐	CRA	2 200
Kin	Kinc	Vavřinec	2.9.1971	☐	CRA	4 700
Kow	Kowolowská	Zoja	25.4.1975	☑	CRA	2 600
Lam	Lampartová	Gerta	5.2.1985	☑	CRA	2 000
Neu	Neuberger	Darius	7.3.1971	☐	CRA	2 100
Olb	Olbřímek	Andrej	15.8.1973	☐	CRA	2 700
Pou	Poukar	Zdenek	16.9.1983	☐	CRA	4 500
Vid	Vidlák	Alois	28.1.1963	☐	CRA	3 000
Zoč	Zoček	Albín	23.12.1978	☐	CRA	3 500
Boh	Bohoněk	Zdeněk	6.12.1975	☐	EGE	3 100
Koc	Kocour	Otakar	16.10.1958	☐	EGE	3 600
Kra	Krawec	Jozef	20.6.1984	☐	EGE	3 700
Lát	Látková	Blažena	7.5.1974	☑	EGE	3 200

Záznam: 1 z 40 | Bez filtru | Vyhledávání

14.3 Filtrování

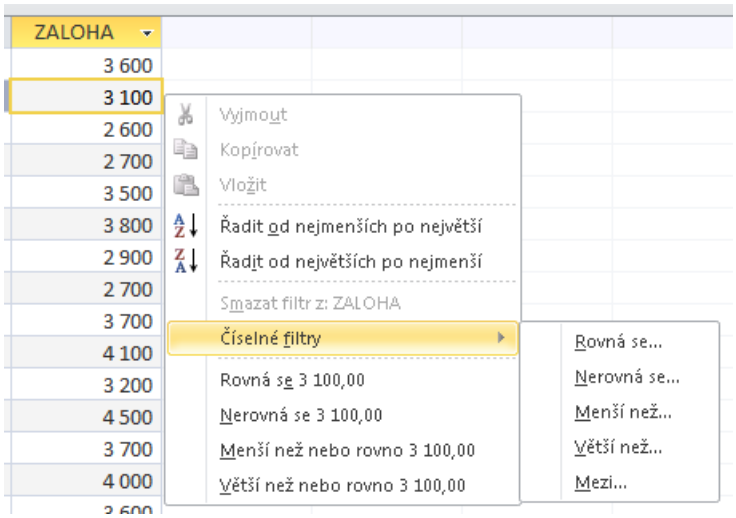
Filtrováním se rozumí výběr jen takových řádků tabulky, které splňují určitá kritéria, určité podmínky - tyto řádky se "odfiltrují" do dalšího zpracování; ostatní v tabulce zůstanou, ale do následného zpracování nebudou pojaty.

Požadavek uživatele na filtrování lze zadat několika způsoby: od nejjednodušších (většinou postačují akce myší) až po nejkomplexnější (pomocí tzv. dotazů). Zde popíšeme jednoduchý, celkem intuitivní způsob *pravým* tlačítkem myši. Jak známo, konvence grafického uživatelského rozhraní doporučují rezervovat pravý klik na nějaký objekt pro zobrazení kontextového menu daného objektu. V prostředí datového listu jsou těmito objekty jednotlivé datové hodnoty, na základě nichž se sestavují podmínky filtrování. Protože se však podmínky pro různé datové typy formulují různými způsobem, musí být i kontextová menu podřízena datovému typu. Zde popíšeme formulaci jednoduchých podmínek pro číselné, textové a datumové hodnoty.

Ve všech případech je na počátky *pravý klik* na konkrétní (popř. libovolnou) hodnotu v konkrétním sloupci.

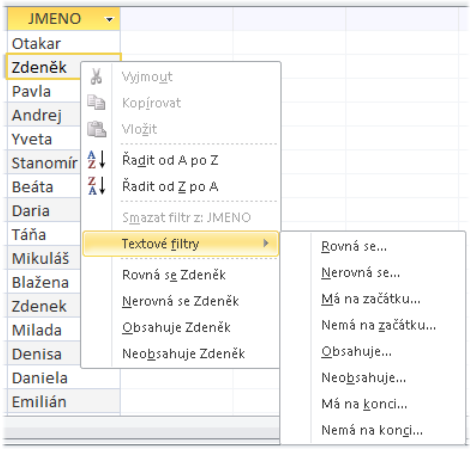
14.3.1 Filtrování dle číselných hodnot

Pravý klik na číselnou hodnotu (v ukázce na hodnotu *3100*) zobrazí kontextové menu pro číselná pole. Ve spodní části nabízí podmínky pro tu konkrétní hodnotu, na kterou byl pravý klik učiněn. Zde tedy záleží na tom, na kterou hodnotu bylo kliknuto. V kontextovém menu je dále položka Číselné filtry, nabízející formulaci podmínky pro libovolné číselné hodnoty v daném sloupci - zde tedy nezáleží na hodnotě, na kterou bylo kliknuto.



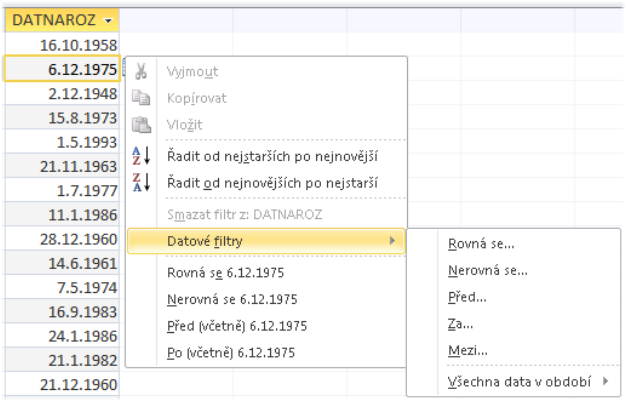
14.3.2 Filtrování dle textových hodnot

Pravý klik na textovou hodnotu (v ukázce na hodnotu *Zdeněk*) zobrazí kontextové menu pro textová pole. Ve spodní části nabízí podmínky pro tu konkrétní hodnotu, na kterou byl pravý klik učiněn. Zde tedy záleží na tom, na kterou hodnotu bylo kliknuto. V kontextovém menu je dále položka Textové filtry, nabízející formulaci podmínky pro libovolné textové hodnoty v daném sloupci - zde tedy nezáleží na hodnotě, na kterou bylo kliknuto.

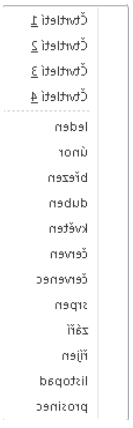


14.3.3 Filtrování dle datumových hodnot

Pravý klik na datumovou hodnotu (v ukázce na hodnotu *6.12.1975*) zobrazí kontextové menu pro datumová pole. Ve spodní části nabízí podmínky pro tu konkrétní hodnotu, na kterou byl pravý klik učiněn. Zde tedy záleží na tom, na kterou hodnotu bylo kliknuto. V kontextovém menu je dále položka Datumové filtry, nabízející formulaci podmínky pro libovolné datumové hodnoty v daném sloupci - zde tedy nezáleží na hodnotě, na kterou bylo kliknuto.



Protože však časové údaje mají oproti číselným a textovým svá další specifika, je k disposici další pod-menu pro výběr časových okamžiků v obdobích, do kterých současná - alespoň evropská - civilizace tok času seskupuje.

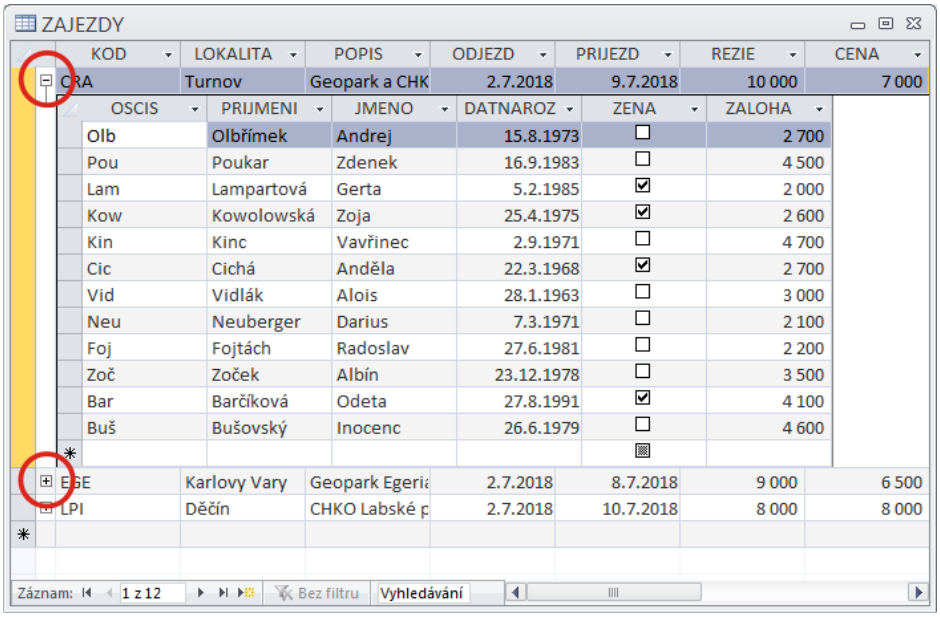


14.4 Vnořený datový list

Datový list programu Access umí využít vhodně vytvořenou relaci (např. s vlastnostmi uvedenými shora) v tzv. *vnořeném datovém listu*. Otevřením datového listu *odkazované* tabulky A relace (zde tedy ZÁJEZDY) je uživateli nabídnuta možnost otevřít pro každý její záznam pomocí tlačítek se znaménkem + vnořené datové listy obsahující záznamy *odkazující* tabulky B (zde tedy ÚČASTNÍCI), mající shodné hodnoty cizích klíčů s hodnotou primárního klíče záznamu tabulky A. Zde tedy konkrétně: ve vnořeném datovém listu se zobrazí všichni účastníci daného zájezdu.

Ve vnořeném datovém listu lze využívat všech nástrojů, které poskytuje prostředí "běžného" datového listu, zvláště řazení a filtrování (viz výše). Způsob řazení a filtrování je pak použit ve všech vnořených datových listech.

Vnořené datové listy lze tlačítkem na začátku každého řádku odkazované tabulky střídavě zobrazovat a skrývat:



15 Dotazy

Dotazy (angl. Queries) jsou de facto příkazy tzv. dotazovacího jazyka. V současné době je dominantním dotazovacím jazykem ten, který je znám pod zkratkou SQL (Structured Query Language - strukturovaný dotazovací jazyk) a který je na seznamovací úrovni popsán shora v *Dílu II: Dotazovací jazyk*. Jde o nástroj pro práci jak se strukturou, s daty relačních databází, ale i s databázemi jako celkem. Byl vyvíjen od roku cca 1970, primárně jako jazyk co nejvíce podobný běžnému anglickému vyjadřování. Postupem času se uplatnily snahy ho jakýmsi způsobem kodifikovat (např. do ANSI normy), nicméně díky obrovské akceleraci IT technologií se ukázaly mnohé slabiny původní definice SQL vzhledem k aktuálním potřebám jak uživatelů, tak programátorů (= autorům databázových aplikací). Důsledkem je sice jakási rádoby všezahrnující definice SQL, kterou však autoři různých databázových systémů (byť v detailech) ve svých aplikacích s gustem porušují, doplňují, mění. Z toho plyne: možnost přenositelnosti příkazů SQL mezi jednotlivými databázovými platformami je sice možná, ale klade přeci jen jisté nároky na jejich autory.

Autoři těchto učebních textů se - ve shora uvedených kapitolách - snažili popsat jen ty konstrukce SQL, které lze použít ve většině mutací tohoto jazyka. Příkazy ve tvarech tam uvedených plně postačují např. pro řešení úloh uvedených v Tématech I až III.

V této části učebních textů proto již nebudou vysvětlovány tvary a funkce příkazů, ale bude podán popis prostředí pro jejich vytvoření a úpravu tak, jak je nabízí databázové programy. Protože jsou ukázky předváděny v prostředí programu Access, byly voleny takové postupy, které jsou více či méně obdobně realizovány i v jiných databázových programech.

Základní ukázkou bude v obou následujících kapitolách dotazy, směřující do ÚČASTNÍKŮ v Tématu I:

a. Kteří účastníci zaplatili zálohu větší než 4.000,- Kč?

b. Kolik celkem zaplatili na zálohách účastnící jednotlivých zájezdů?

Výsledkem takových dotazů je v případě ad a. množina řádků (zde účastníků zájezdů), kteří vyhovují formulaci dotazu; v případě ad b. je to množina řádků (pro každý jednotlivý zájezd jeden) s informacemi o zájezdu a celkových zaplacených zálohách. Protože je výsledkem vždy *množina* řádků, může je databázový program zobrazit svými prostředky jako běžnou tabulku - program Access to činí ve svém datovém listě (viz výše).

15.1 Dotazy pro laiky

Pod pojmem "laik" (rozhodně ne hanlivým) zde rozumíme takového uživatele, který se z jakýchkoliv důvodů nehodlá zabývat syntaxí a sémantikou dotazovacích jazyka. Takový uživatel preferuje pro sestavení svého dotazu např. nějaké intuitivní grafické prostředí. Pro tuto třídu uživatelů poskytují databázové programy skutečně grafická návrhová prostředí. Jejich úkolem (protože dotazem *musí* být *text příkazu SQL*) je na základě uživatelem - laikem sestaveného grafického návrhu dotazu tento text vygenerovat, aniž by uživatel musel pravidla pro jeho zápis znát.

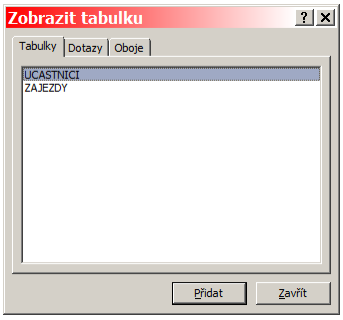
Nevýhodou tohoto řešení je velmi omezená množina tvarů příkazu SQL. Jinak řečeno, zdaleka se nevyužije mohutnost uplatnění SQL daná právě rozmanitostí jeho jednotlivých částí. Pokud by totiž návrhové prostředí mělo umožňovat konstrukci i výkonnějších dotazů, bylo by samo tak složité, že pro uživatele by bylo jednodušší se naučit přímo SQL.

15.1.1 Určení datových zdrojů a datových polí

Návrhové prostředí se uživateli předloží postupem

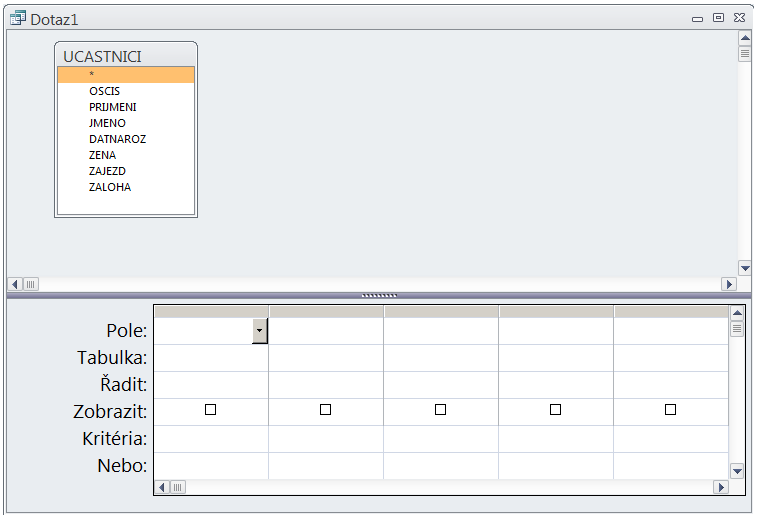
Vytvoření / Dotazy / Návrh dotazu

Následně je v pozadí otevřeno návrhové prostředí a v popředí aktivován formulář pro výběr jednoho nebo více datových zdrojů, do kterých směřuje dotaz:

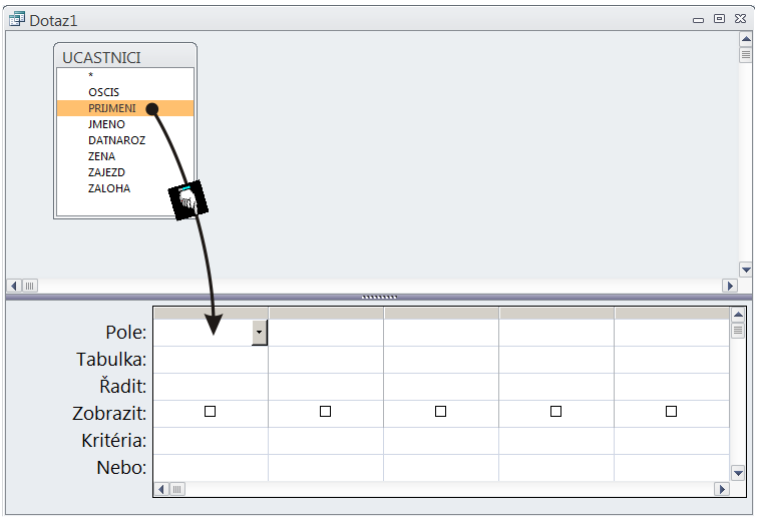


Ukázkový dotaz směřuje do tabulky ÚČASTNÍCI; ta se ve formuláři označí a tlačítkem *Přidat* se přidá na plochu návrhového prostředí. V případě čerpání z více datových zdrojů se tato činnost opakuje, na závěr se formulář zavře.

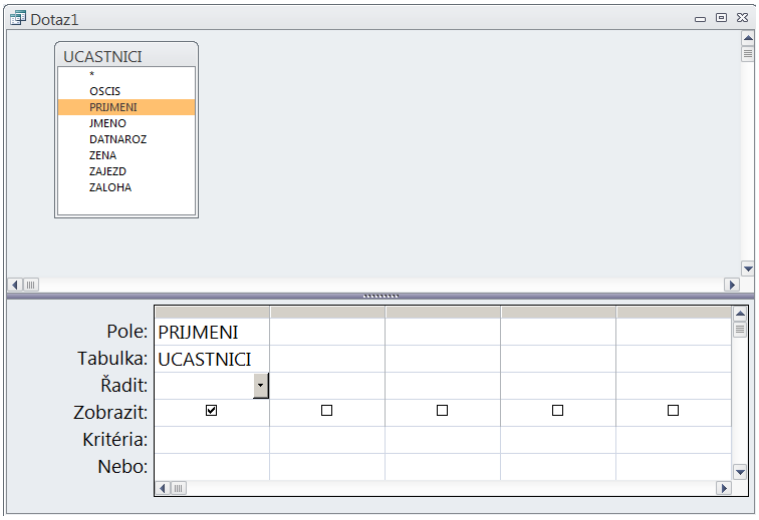
Nyní je návrhové prostředí připraveno pro formulaci dotazu:



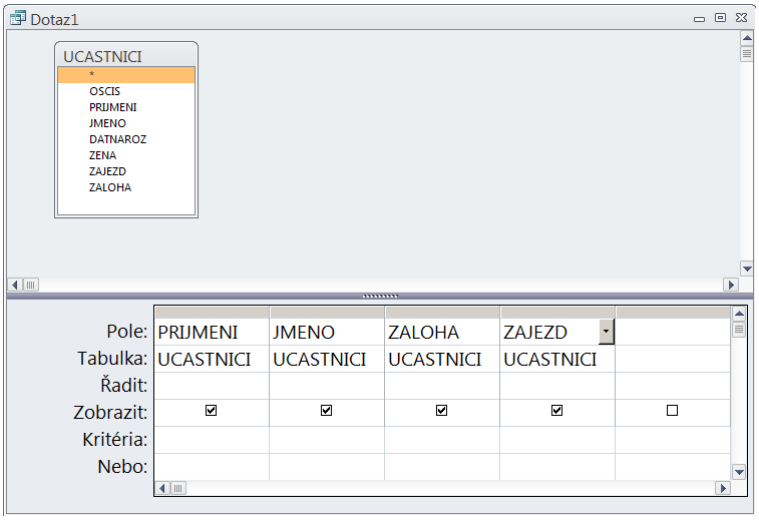
Prvním z úkolů je dát návrhovému prostředí na vědomí, která datová pole mají být zahrnuta do výsledku dotazu. Nejjednodušší je pro uživatele intuitivní přetažení požadovaných datových polí postupně do jednotlivých sloupců ve spodní části návrhového prostředí:



Tedy po přetažení prvního požadovaného pole PŘÍJMENÍ je stav návrhového prostředí následující:



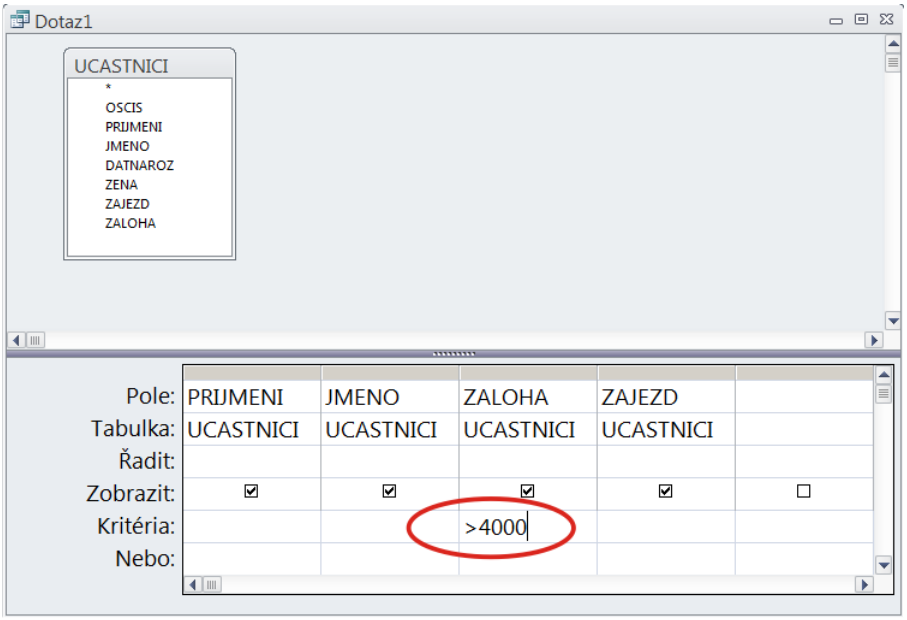
Tato činnost se následně opakuje pro všechna *požadovaná* datová pole (tedy ne nutně pro všechna pole datového zdroje):



V případě potřeby je možno za úzké tlačítko v záhlaví spodních sloupců jednotlivá pole přemístit nebo je z dotazu vypustit.

15.1.2 Určení kritérií a řazení

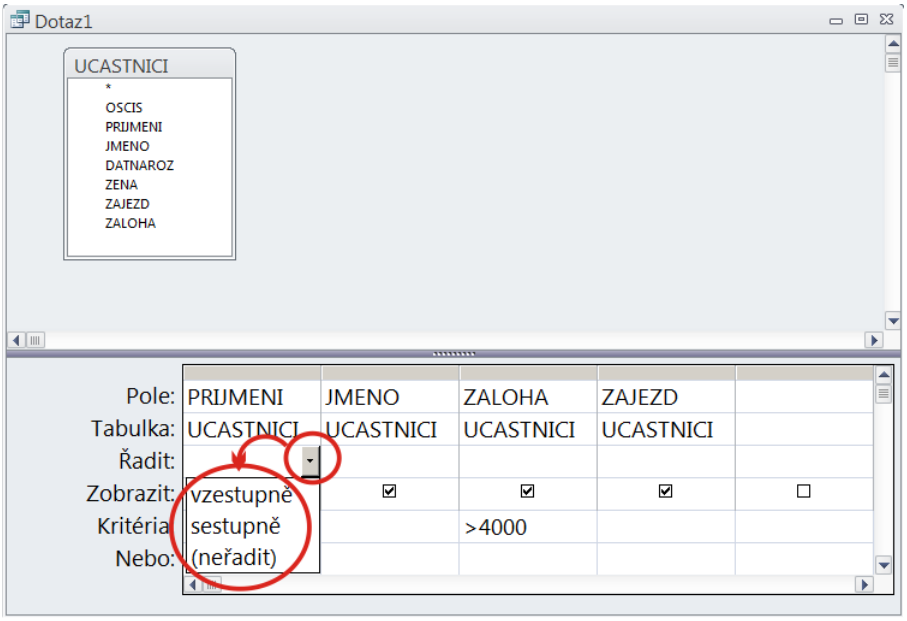
Následuje formulace podmínek (kritérií), za které budou řádky pojaty do výsledku dotazu. Podmínky se zapisují do řádku *Kritéria* zřejmým způsobem:



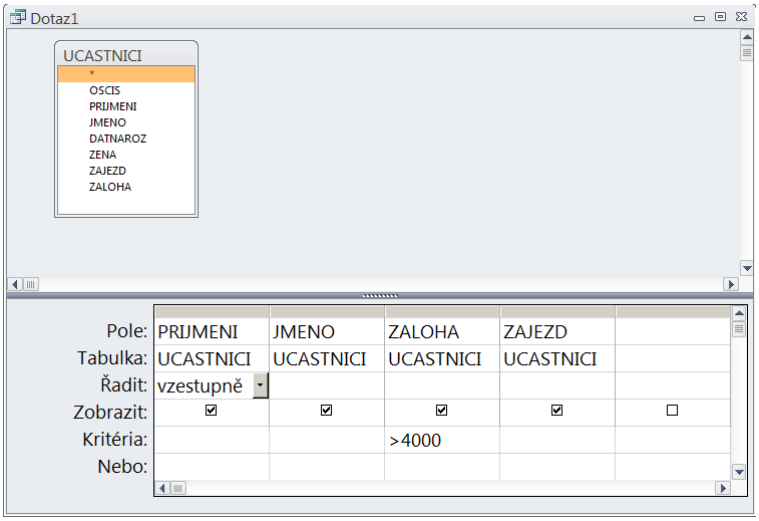
Pokud se do jednoho řádku *Kritéria* (tedy "vedle sebe") zapíší podmínky pro více datových polí, jsou v logickém vztahu *a současně*; pokud se zapíší do řádku *Nebo* (tedy "pod sebe"), jsou v logickém vztahu *nebo*.

Pozor, jde o matematický logický operátor **nebo**, *nikoliv o (často zaměňovaný) za vylučující* **buď anebo**!

Pro definici požadavku uživatele na řazení slouží řádek *Řadit*. Rozvíjecím tlačítkem v tomto řádku v příslušném datovém poli je zobrazen seznam nabídek, z něhož autor dotazu vybírá:

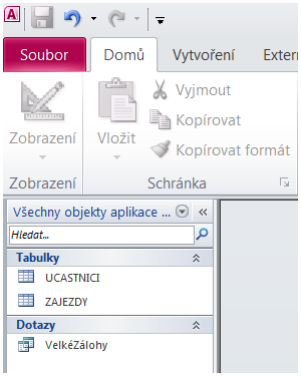


Po kompletní definici dotazu (viz následující obrázek) se formulář zavře.



15.1.3 Pojmenování dotazu a jeho provedení

Teprve při uzavření a po obvyklé otázce na uložení změn následuje žádost návrhového prostředí na zadání jména dotazu. V ukázce byl dotaz pojmenován *VelkéZálohy*; pod tímto označením se objeví v levém panelu databáze v části *Dotazy*:



Dvojitý klik na ikonu dotazu je pokynem pro databázový program, aby vyslal dotaz k provedení. Není-li v něm chyba, zobrazí databázový program po přijetí výsledku datový list tento výsledek obsahující:

VelkéZálohy

PRIJMENI	JMENO	ZALOHA	ZAJEZD
Barčíková	Odeta	4 100	CRA
Bušovský	Inocenc	4 600	CRA
Hokr	Kamil	4 500	LPI
Kinc	Vavřinec	4 700	CRA
Kliber	Štěpán	4 100	LPI
Machová	Emílie	4 800	LPI
Poukar	Zdenek	4 500	CRA
Sankot	Oskar	4 400	EGE
Sečkař	Mikuláš	4 100	LPI

Záznam: 1 z 9 Bez filtru Vyhledávání

15.1.4 Seskupování záznamů

Jednou z možností dotazu (přesněji v návrhovém prostředí konstruovaného příkazu) je možnost záznamy pocházející z určeného datového zdroje seskupovat, nad takto vytvořenými skupinami provádět souhrnně tzv. agregační operace, a z každé skupiny vytvořit jeden řádek s požadovanými daty. Taková možnost se využije při řešení druhé shora uvedené úlohy:

b. Kolik celkem zaplatili na zálohách účastnící jednotlivých zájezdů?

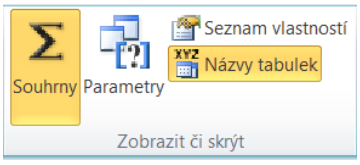
Je zřejmé, že návrhové prostředí potřebuje vědět

1. podle jakého pravidla (pravidel) vytvářet skupiny záznamů (zde skupiny účastníků se stejným zájezdem),
2. jakou souhrnnou početní operaci na jednotlivých skupinách provádět (zde součtovat zálohy).

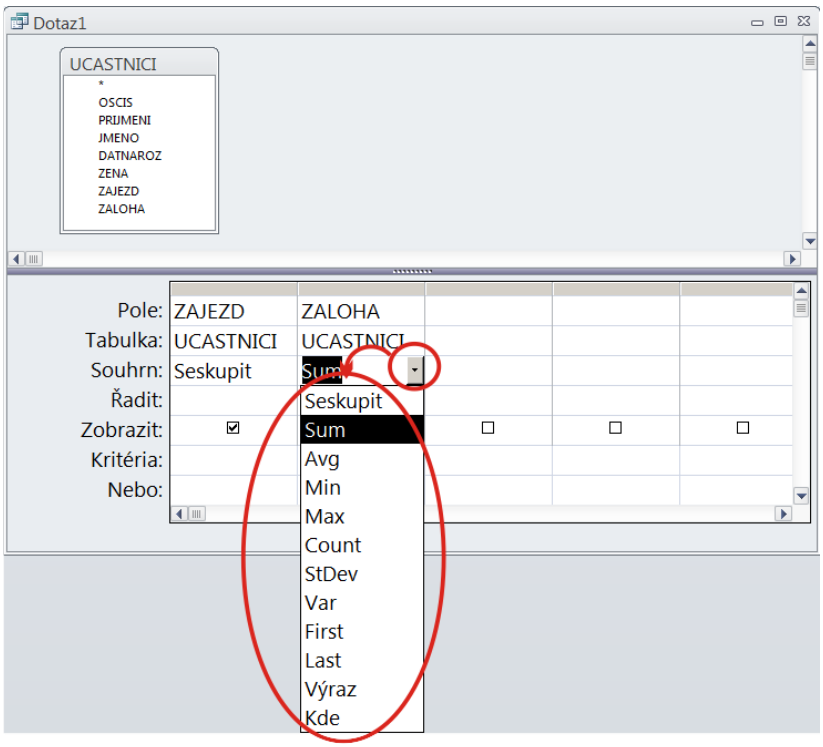
Tyto informace lze v návrhovém prostředí zadat do řádku *Souhrn*, který však není standardně přístupný. Jeho zobrazení se docílí postupem

Dotazy - nástroje / Návrh / Zobrazit či skrýt / Souhrny

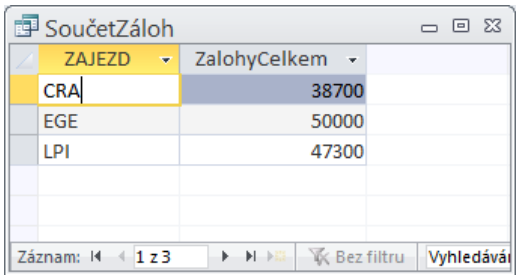
tedy stisknutím tlačítka *Souhrny* na kartě *Zobrazit či skrýt*:



Nyní následuje učení datových polí (přetažením popsaným výše). Minimálně to musí být alespoň jedno datové pole, podle kterého se bude seskupovat (zde tedy pole ZÁJEZD), a jedno datové pole, na které bude uplatňována souhrnná početní operace (zde součet hodnot v poli ZÁLOHA). Po přetažení se právě v nově zobrazeném řádku *Souhrn* sdělí postavení jednotlivých datových polí: pro datové pole ZÁJEZD to bude označení pole pro seskupování, pro datové pole ZÁLOHA jako pole sčítané (SUM - SUMACE - součtování):



Po nastavení těchto vlastností je návrh dotazu hotov, návrhové prostředí se tedy zavře, změny nechají uložit a přidělí se nově definovanému dotazu jméno - pro ukázku zvoleno jméno *SoučetZáloh*. Vytvořený dotaz lze předat k vykonání (dvojklik na jeho ikonu) a studovat výsledek:



15.1.5 Agregační funkce

V předchozím odstavci byla použita souhrnná (tzv. agregační) funkce SUM = součet. Je však možno použít řadu dalších funkcí, jak je patrno z nabídky. Níže je uveden jejich stručný popis, předtím však důležitá poznámka: Ne všechny databázové systémy akceptují všechny tyto agregační funkce, naopak některé mohou vykonávat další funkce v seznamu neuvedené. Proto v následujícím výčtu je ve sloupci *Standardně* uvedeno, zda tuto funkci standardně vykonávají všechny databázové systémy.

Ident	Funkce	Stdandardně
Count	Počet řádků ve skupině	Ano
Sum	Součet hodnot pole ve skupině	Ano
Avg	Průměrná hodnota pole ve skupině	Ano
Min	Nejmenší hodnota pole ve skupině	Ano
Max	Největší hodnota pole ve skupině	Ano
Var	Variance - Rozptyl hodnot daného pole	Ne
StDev	Standard Deviation - Směrodatná odchylka hodnot daného pole	Ne
First	První hodnota pole ve skupině	Ne
Last	Poslední hodnota pole ve skupině	Ne

***Poznámka**: Pokud v dotazu není požadováno řazení podle daného datového pole, pak výsledkem funkcí First a Last je náhodná hodnota ze skupiny.*

Dotaz může obsahovat požadavek na seskupování podle více datových polí současně - např. podle zájezdu a pohlaví: kolik zaplatili zvlášť ženy a zvlášť muži celkem záloh v každém zájezdu.

Dotaz může také obsahovat požadavek na více agregačních funkcí - např. kolik zaplatili účastníci celkem na zálohách, kolik byla průměrná záloha a kolik byla nejmenší záloha.

15.2 Výrazy v dotazech

Tato kapitola rozšiřuje předchozí odstavec *Databáze pro laiky*, jehož znalost se předpokládá. Bude použito stejné demonstrační databáze a stejné značení.

Připomenutí: Datovým zdrojem se pro účely těchto textů rozumí *tabulka* nebo *výsledek dotazu*. Pokud nebude hrozit nedorozuměni, bude se ve druhém případě používat termín bez slova "výsledek" (tedy jen *dotaz*). Datovým zdrojem se tedy bude rozumět *tabulka* nebo *dotaz*.

V citovaném odstavci byly uváděny pouze dotazy s funkcí filtrování a řazení. Tyto dotazy směřovaly do jediného datového zdroje, a to tabulky ÚČASTNÍCI. Byly uvedeny dva základní typy dotazů. Jeden sekvenčně procházel datový zdroj, event. vybíral, event. řadil; výstupem byly přímé hodnoty některých datových polí zdroje. Druhý typ dotazu seskupoval záznamy datového zdroje podle jediného kritéria, a na určených hodnotách každé skupiny vyhodnocoval zadanou tzv. agregační funkci.

Databázové systémy toho dovedou samozřejmě daleko více. Často mají zájemci o databáze před očima jiný, obecně známý program pro uchování a zpracovaní dat, tabulkový procesor Excel a jemu podobné. Doposud uvedené funkce databázových systémů (řazení, filtrování a seskupování) nejsou tedy pro uživatele tabulkových procesorů nic nového. Naopak: začne pociťovat absenci toho, co je pro tabulkové procesory charakteristické, vzorce. Tedy takového obsahu buňky, který má tvar např. =25*B13+9 a jehož hodnota je při zpracování listu na místě této buňky zobrazena.

V databázových systémech není *obecně* možno stanovit jako obsah datového pole výraz (vzorec). Je však možno požadovat, aby výsledek vyhodnocení nějakého výrazu byl obsahem polí nějakého sloupce *dotazu*. Ten se pak jeví ve výsledku celého dotazu jako další "vypočítávané" datové pole. Právě konstrukci takových dotazů je věnován tento odstavec.

> V tomto odstavci je běžně používán termín *Výraz* bez bližší specifikace. Autoři předpokládají, že čtenář intuitivně vytuší jeho význam z kontextu; více v tomto odstavci není třeba. Pro čtenáře zvídavé až hloubavé je detailní definice pojmu *Výraz* uvedena v závěrečné kapitole. Pro budoucí profesionály je její znalost skutečně nutná.

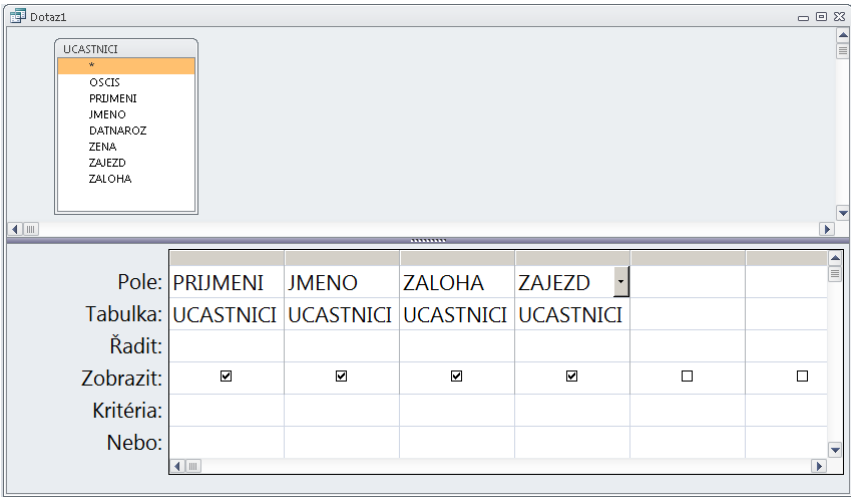
15.2.1 Motivační příklad

Tabulka ÚČASTNÍCI obsahuje datové pole ZÁLOHA s údajem o účastníkem zaplacené záloze. Z této platby musí ovšem naše cestovní kancelář odvést DPH ve výši 15% (stejně jako z později zaplaceného doplatku). Zajímalo by nás, jakou částku každý z účastníků zaplatil jako zálohu a kolik my z každé takové platby musíme odvést DPH. Pro tuto naši informovanost by postačoval dotaz, zobrazující kromě základních údajů o každém účastníku navíc sloupec s vypočítávanou daní z přidané hodnoty. Tento sloupec by mohl nést nějaký smysluplný název, např. DPH (nesmí však kolidovat s názvy existujících sloupců).

Jde o složitou matematickou operaci, proto nápověda: Je-li ZÁLOHA chápána jako 100%, pak 1% je rovno ZÁLOHA/100 a 15% je tedy 15x tolik. Proto je zapotřebí počítat hodnotu výrazu 15*ZÁLOHA/100.

15.2.2 Vytvoření dotazu

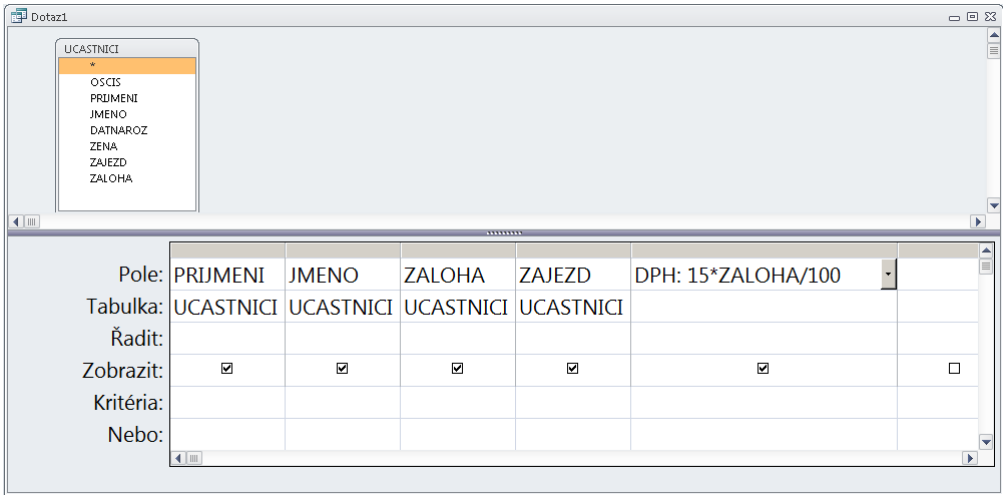
Dle odstavce shora se aktivuje návrhové prostředí dotazu s datovým zdrojem ÚČASTNÍCI. Požadujme ve výsledku dotazu nejprve např. datová pole stejná jako shora, tedy PŘÍJMENÍ, JMÉNO, ZÁLOHA a ZÁJEZD:



Nyní k vypočítávanému poli: je zapotřebí "ručně" zapsat do prvního volného sloupce do jeho řádku "Pole":

1. Zvolený název vypočítávaného pole.
2. Dvojtečku.
3. Výraz, který se má vyhodnocovat.

Pro čitelnější zápis a pro kontrolu je vhodné (tažením doprava za mezeru mezi sloupci) rozšířit sloupec pro vypočítávaný výraz:



Návrh dotazu se pak uloží s vhodným pojmenováním - např. *DPHzeZaloh*.

15.2.3 Výsledek dotazu

Podle odstavce *Pojmenování dotazu a jeho provedení* je možno nechat dotaz provést:

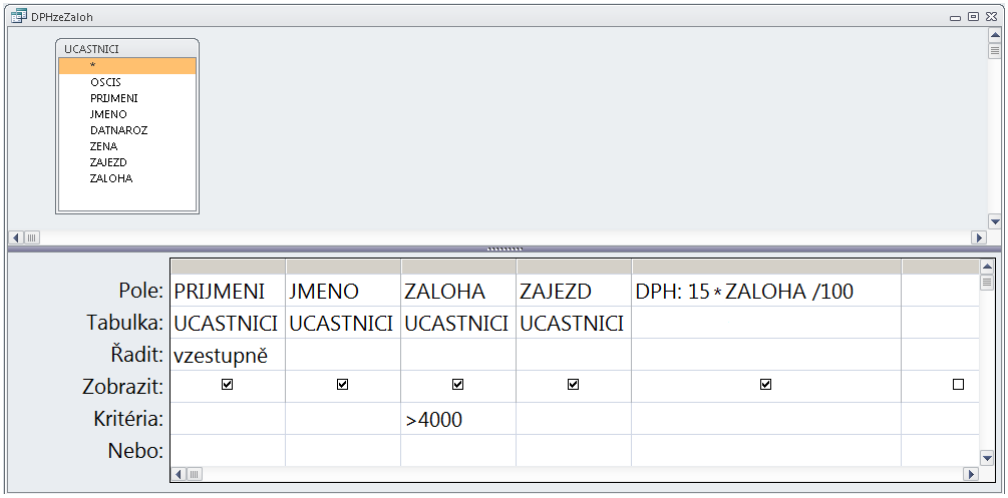
DPHzeZaloh

PRIJMENI	JMENO	ZALOHA	ZAJEZD	DPH
Kocour	Otakar	3 600	EGE	540
Bohoněk	Zdeněk	3 100	EGE	465
Koulová	Pavla	2 600	LPI	390
Olbřímek	Andrej	2 700	CRA	405
Albrechtová	Yveta	3 500	LPI	525
Oršulík	Stanomír	3 800	EGE	570
Rinková	Beáta	2 900	LPI	435
Stiovová	Daria	2 700	LPI	405
Pechancová	Táňa	3 700	LPI	555
Sečkař	Mikuláš	4 100	LPI	615
Látková	Blažena	3 200	EGE	480
Poukar	Zdenek	4 500	CRA	675
Vaičnerová	Milada	3 700	LPI	555

Záznam: 1 z 40 Bez filtru Vyhledávání

Dotaz může obsahovat libovolný počet vypočítávaných polí. Na jejich pořadí nezáleží (tedy nemusí to být poslední), je však nutno dodržet jedinečnost jejich pojmenování. V návrhovém prostředí dotazu v programu Access je nutno při *vytváření* začít zapisovat do prvního volného sloupce vpravo, poté však je možno pořadí sloupců návrhového prostředí (tedy i pořadí sloupců ve výsledku dotazu) libovolně měnit tažením za záhlaví přesouvaného sloupce.

V návrhovém prostředí dotazu - bez ohledu na to zda obsahuje nebo neobsahuje vypočítávaná pole - je možno využít podle potřeby všech možností dle odstavce *Určení kritérií a řazení*. Např. stejný dotaz s výsledkem řazeným podle abecedy jen s účastníky se zálohou větší než 4000 Kč:



s výsledkem

DPHzeZaloh

PRIJMENI	JMENO	ZALOHA	ZAJEZD	DPH
Barčíková	Odeta	4 100	CRA	615
Bušovský	Inocenc	4 600	CRA	690
Hokr	Kamil	4 500	LPI	675
Kinc	Vavřinec	4 700	CRA	705
Kliber	Štěpán	4 100	LPI	615
Machová	Emílie	4 800	LPI	720
Poukar	Zdenek	4 500	CRA	675
Sankot	Oskar	4 400	EGE	660
Sečkař	Mikuláš	4 100	LPI	615

Záznam: 1 z 9 Bez filtru Vyhledávání

15.3 Dotazy do dvou zdrojů

Výukové téma *Geoparky I* (viz shora) uvádí náměty pěti dotazů. První dva jsou směrovány do jediného datového zdroje. Avšak třetí dotaz:

- Kolik ještě doplatí účastníci v jednotlivých zájezdech?

evidentně směřuje do dvou datových zdrojů, zde do dvou tabulek. Požaduje se pro každého účastníka zjistit rozdíl ceny jeho zájezdu (ta je na jediném místě, a to v tabulce ZÁJEZDY v řádku zájezdu příslušejícího konkrétnímu účastníku) a účastníkem zaplacené zálohy (ta je také na jediném místě, a to v tabulce ÚČASTNÍCI v řádku konkrétního účastníka).

Současné zpracování dvou (a více) datových zdrojů se může principiálně odvíjet několikerým způsobem. Postupně popíšeme dva nejtypičtější.

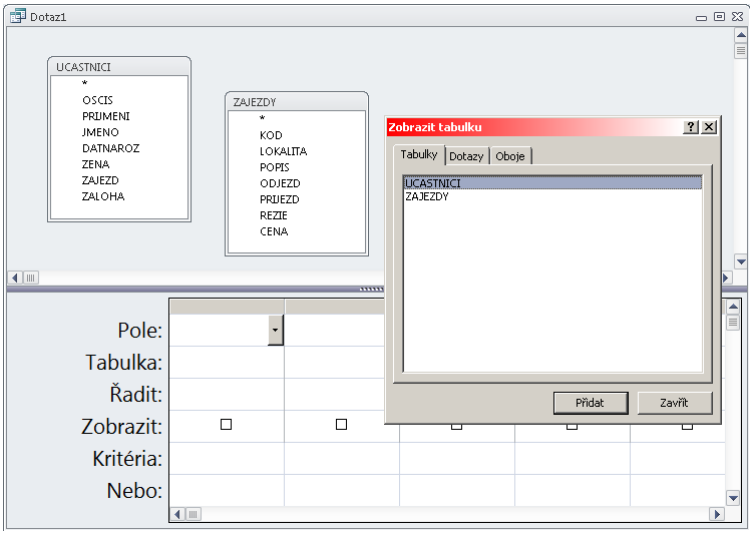
15.3.1 Kartézský součin

Připomenutí: Cílíme k naznačenému dotazu: Kolik doplatí účastníci ...

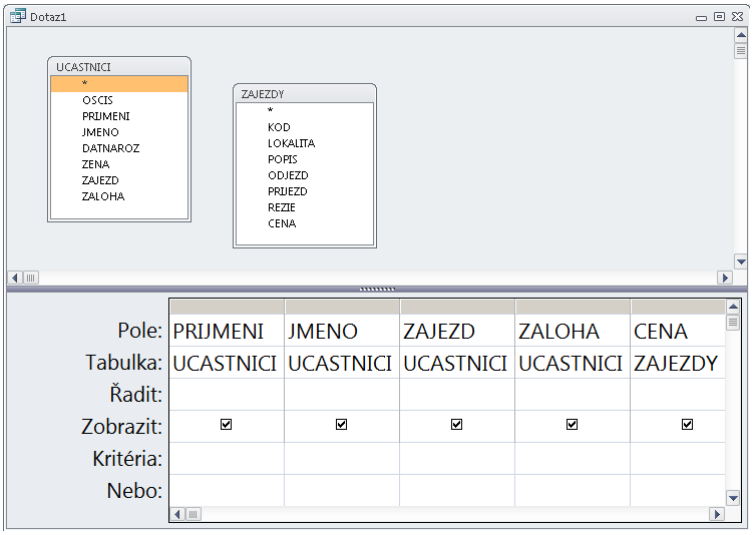
V intencích předchozího odstavce je tedy zapotřebí sestavit dotaz obsahující výraz

```
ZÁJEZDY.CENA - ÚČASTNÍCI.ZÁLOHA
```

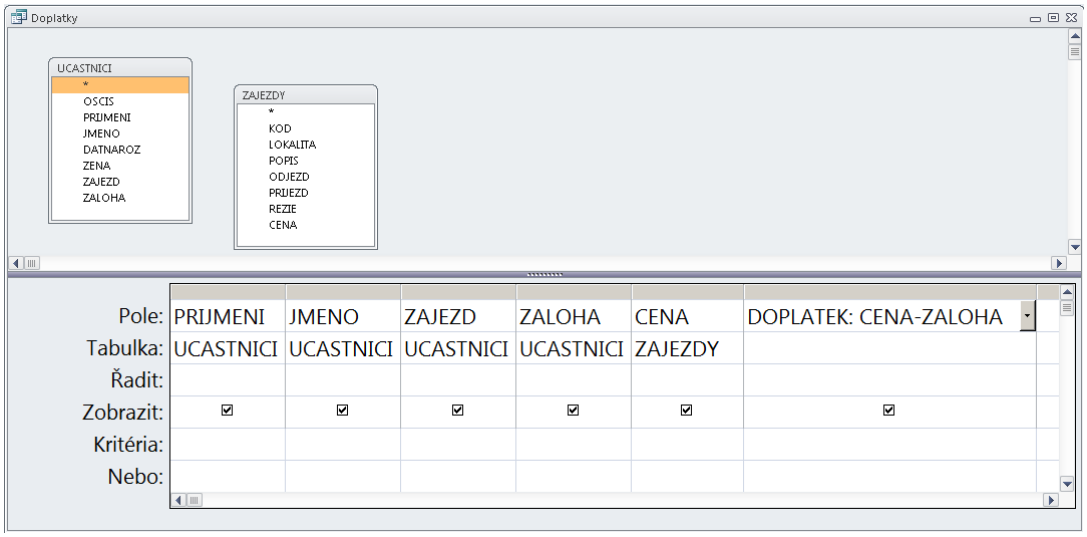
V návrhovém prostředí dotazu se proto přidají dvě tabulky postupem dle 15.1.1 skript:



a zvolí se datová pole, která je žádoucí mít ve výsledku dotazu (opět dle 15.1.1 skript) :



Nyní se "ručně" do prvního volného sloupce zprava zapíše zvolený název vypočítávaného pole (zde DOPLATEK), dvojtečka a vyhodnocovaný výraz:



Po uložení návrhu dotazu pod vhodným jménem (zde Doplatky) lze požádat o provedení dotazu:

Za prvé, zkontrolujeme správnost výpočtu - a radostně konstatujeme, že se skutečně doplatek rovná rozdílu ceny a zálohy. Databáze jsou prostě skvělé, umí odečítat. I my jsme ale dobří, protože jim to už umíme říct. Ovšem - když se tak kocháme výsledkem - všimneme si toho "za druhé". Je to nějaké divné: Kocour jede do Egerie třikrát pokaždé za jinou cenu? Stejně tak Bohoněk - a vůbec každý účastník je tam třikrát!

Toto není chyba Microsoftu, jak by se obvykle jako první usoudilo. Přesně takto má totiž dotaz tohoto typu pracovat. Jsou-li dva zdroje dat, utvoří databázový systém nejprve Kartézský součin (viz kapitola *Kartézský součin množin*) obou datových zdrojů, tedy "každý s každým": virtuálně pak za každý řádek prvního datového zdroje připojí postupně všechny řádky druhého datového zdroje. Měl-li by mít každý datový zdroj 1000 záznamů, vznikne něco jako široká tabulka mající 1000 x 1000 = 1 000 000 řádků. Teprve odtud se posílají jako výsledek dotazu žádaná datová pole a výrazy. V příkladu s cestovkou je 40 účastníků a 3 zájezdy, všech kombinací je tedy 40 x 3 = 120.

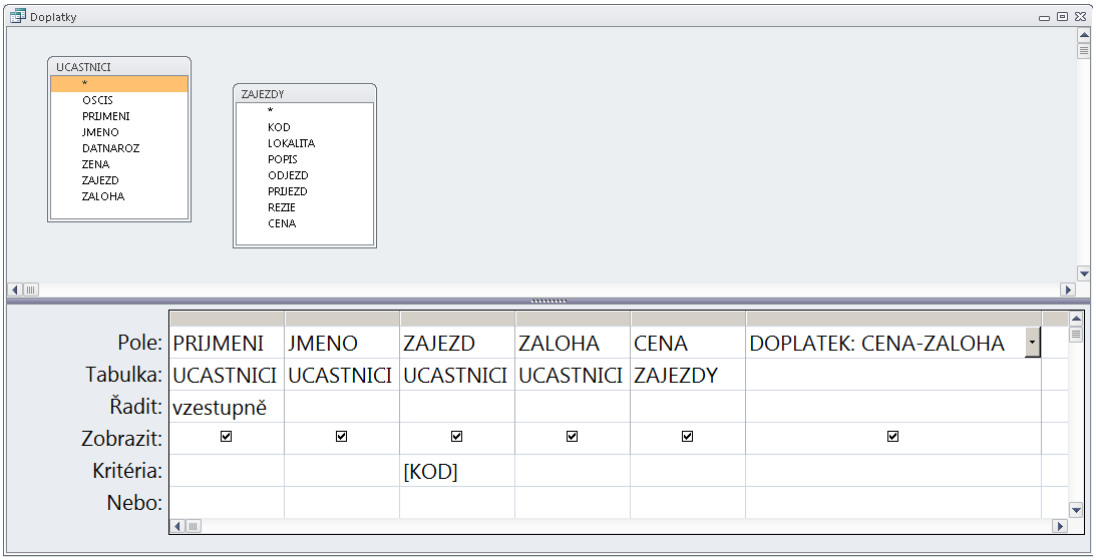
To je však něco, co zhola nepotřebujeme. Nepotřebujeme *všechny* kombinace, ale jen *některé*. To ovšem zajišťuje mechanismus filtrování uvedený shora. Zbývá stanovit - nejprve sami sobě - kritérium rozpoznávající ty "správné" kombinace. Zde je to poměrně jednoduché: kód zájezdu uvedený v poli ZÁJEZD daného účastníka v tabulce ÚČASTNÍCI se musí rovnat kódu zájezdu uvedenému v poli KÓD tabulky ZÁJEZDY, tedy

```
ÚČASTNÍCI.ZÁJEZD = ZÁJEZDY.KÓD
```

Protože jména datových polí nejsou stejná, stačí je uvést (viz výše) bez specifikace zdroje

```
ZÁJEZD = KÓD
```

V návrhu dotazu se to provede analogicky jako shora filtr pro hodnoty > 4000. Podmínka se zapíše do sloupce ZÁJEZD (levý operand), do řádku Kritéria se zapíše pravý operand (v případě testu na rovnost se nemusí psát znaménko =). Za povšimnutí stojí uzavření jména sloupce KÓD do hranatých závorek [a]. Tím se totiž dá na vědomí, že jde právě o jméno sloupce a nikoliv o tříznakový text "KÓD". Když už je návrhové prostředí otevřené, je možno přidat požadavek na řazení pro pozdější rychlé ověření, že už tam doopravdy nikdo není třikrát:



Po požadavku na provedení dotazu lze zkontrolovat výsledek:

15.3.2 Join (Propojení, Připojení, Spojení ...)

Nejprve obecná úvaha. Mějme jeden datový zdroj (zde např. tabulka ÚČASTNÍCI). Cena jejich konkrétního zájezdu je uvedena někde ve druhém datovém zdroji (zde v tabulce ZÁJEZDY - a to v řádku toho zájezdu, na který pojede daný účastník). Je-li třeba zjistit pro každého účastníka jeho požadovaný doplatek, je zapotřebí

A. pro každého účastníka (přesněji pro jeho řádek v tabulce ÚČASTNÍCI = první datový zdroj) získat data jeho zájezdu (přesněji řádek jeho zájezdu v tabulce ZÁJEZDY = druhý datový zdroj).

Tedy při procházení prvního datového zdroje zajistit přístup k tomu řádku druhého datového zdroje, který splňuje daná kritéria. Může však nastat situace, kdy z nějakého důvodu ve druhé tabulce nebude příslušný řádek nalezen (např. data tohoto zájezdu se zjišťují na poslední chvíli, chybou příslušného zaměstnance apod.). Určitě požadujeme zahrnout i ty účastníky (řádky první tabulky), pro které neexistují zájezdy (řádky druhé tabulky) a pro které tedy nebude doplatek vyčíslen (bude NULL).

Samozřejmě naopak může existovat mnoho zájezdů, na které se (třebas zatím) nikdo nepřihlásil. Jejich data nebudou přístupná žádnému účastníkovi.

Mějme opačný dotaz: Kolik zaplatili na zálohách účastníci každého zájezdu? Je proto zapotřebí

B. pro každý zájezd (přesněji pro jeho řádek v tabulce ZÁJEZDY = druhý datový zdroj) získat data jeho účastníků (přesněji řádky jeho účastníků v tabulce ÚČASTNÍCI = první datový zdroj)

tedy při procházení druhého datového zdroje zajistit přístup k tomu řádku prvního datového zdroje, který splňuje daná kritéria. Samozřejmě může nastat případ, že se na některý zájezd nikdo nepřihlásil; i tato situace nás zajímá: k řádku takového zájezdu nebude připojen žádný účastník, celkově zaplacené zálohy budou NULL.

Konečně třetí případ: jak je na tom naše cestovka? Kolik účastníků se přihlásilo na zatím nabízené zájezdy, kolik vyděláme nebo proděláme atd? Je tedy třeba

C. procházet jen ty řádky obou datových zdrojů, ve kterých existují "protějšky".

Nezajímají nás tedy klienti = účastníci, který nejsou přihlášeni na žádný zájezd, ani zájezdy, na které není přihlášen žádný klient = účastník.

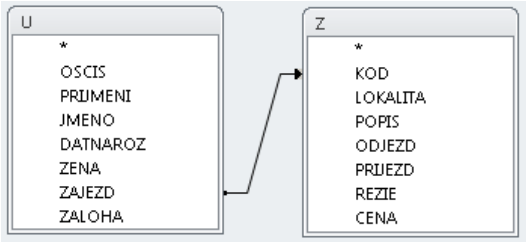
Ve všech třech případech je třeba zadat podmínku, podle které databázový systém pozná, který řádek prvního zdroje "patří" ke kterému řádku druhého zdroje. V diskutovaném příkladu je to stejná podmínka jako shora v případě kartézského součinu:

```
ÚČASTNÍCI.ZÁJEZD = ZÁJEZDY.KÓD
```

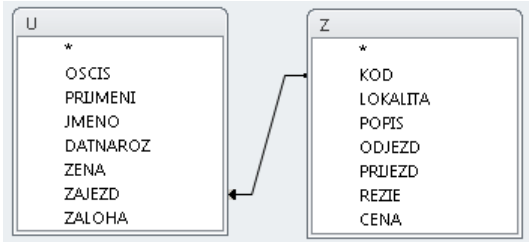
Pro mechanismus vazby mezi dvěma datovými zdroji pro účely dotazů (pozor, nejde o relace!) se vžil termín Join, důvěrně známý autorům dotazů používajícím SQL. Je však obtížné pro něj najít jednoslovný český ekvivalent. Zhruba mu odpovídá některý ze shora uvedených pojmů *Propojení*, *Připojení*, *Spojení*. Literární expert na Český jazyk by ovšem cítil mezi nimi velmi jemné rozdíly. Dokonce by mohl tvrdit, že Připojení by se mohlo nejlépe týkat případu A, Propojení případu B a Spojení případu C. Zde budeme v dalším používat jediný termín **připojení**.

Anglicky mluvící autoři dotazovacích jazyků, zvláště SQL, se tedy také omezili na jediný termín, a to zmíněný **Join**. Aby však rozlišili tři shora diskutované případy, navrhli na základě grafické představy toto:

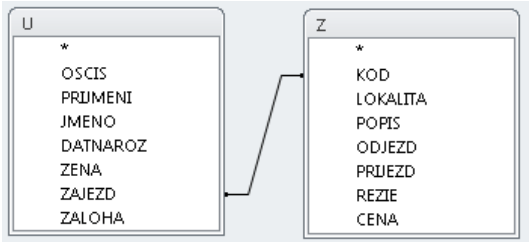
Představme si dva datové zdroje U a Z nakreslené vodorovně vedle sebe. Jeden (třeba U) je tedy vlevo, jeden (tedy Z) je vpravo. Potom ony tři výše uvedené situace lze znázornit spojnicí mezi nimi s případnou šipkou. V jednom případě zdroj U zleva připojuje zdroj Z, ve druhém případě zdroj Z zprava připojuje zdroj U, ve třetím případě jsou připojené "tam i zpět", tedy jaksi "vnitřně". Jde tedy o tyto tři případy:



Případ A: U zleva připojuje Z



Případ B: U zprava připojuje Z



Případ C: U vnitřně připojuje Z

Zároveň je tímto schématem určeno, na základě shody obsahu kterých datových polí je připojení realizováno.

Funkce dotazu, ve kterém je připojení definováno, je popsána takto:

A. **LEFT JOIN:** Dodá všechny řádky levého zdroje bez ohledu na to, zda v pravém existuje "protějšek". Existuje-li v pravém zdroji více "protějšků" (např. N), dodá takový řádek levého zdroje N-krát, pokaždé s jiným "protějším" řádkem pravého zdroje. Pokud naopak v pravém zdroji neexistuje "protějšek", dodá se levý řádek jedinkrát s daty pravé části rovnými hodnotě NULL.

B. **RIGHT JOIN:** Dodá všechny řádky pravého zdroje bez ohledu na to, zda v levém existuje "protějšek". Existuje-li v levém zdroji více "protějšků" (např. N), dodá takový řádek pravého zdroje N-krát, pokaždé s jiným "protějším" řádkem levého zdroje. Pokud naopak v levém zdroji neexistuje "protějšek", dodá se pravý řádek jedinkrát s daty levé části rovnými hodnotě NULL.

C. **INNER JOIN:** Dodá všechny řádky levého zdroje, pro které existuje v pravém "protějšek"- což budou všechny řádky pravého zdroje, pro které existuje v levém "protějšek". Existuje-li v pravém zdroji více "protějšků" (např. N), dodá takový řádek levého zdroje N-krát, pokaždé s jiným "protějším" řádkem pravého zdroje a naopak: Existuje-li v levém zdroji více "protějšků" (např. M), dodá takový řádek pravého zdroje M-krát, pokaždé s jiným "protějším" řádkem levého zdroje.

Je zřejmé, že (U left join Z) je totožné s (Z right join U), (U inner join Z) je totožné s (Z inner join U).

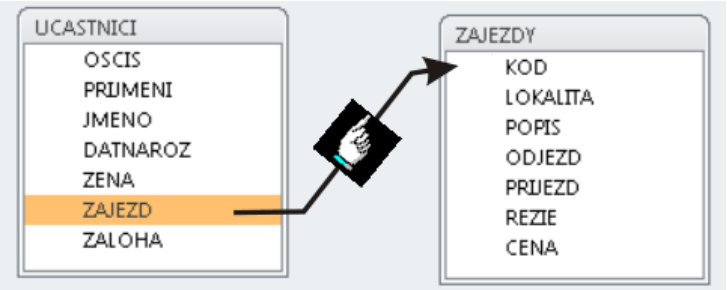
15.3.3 Dotaz pomocí Připojení

Připomenutí: Opět cílíme k dotazu uvedenému shora: Kolik doplatí účastníci ...

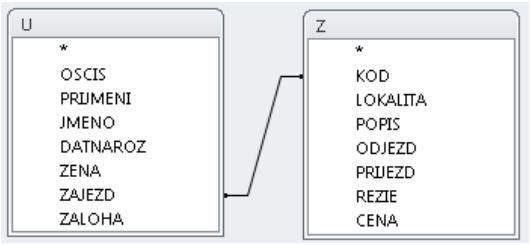
Ve smyslu předchozího odstavce: Dotaz bude směrován do tabulky ÚČASTNÍCI (zvolme ji jako *levý* datový zdroj) a do tabulky ZÁJEZDY (tedy bude stát na místě *pravého* datového zdroje).

K dosažení cíle je zapotřebí procházet všechny účastníky (= všechny řádky levého datového zdroje). Ke každému řádku účastníka pak připojit ten zájezd (= řádek pravého datového zdroje), který je v řádku účastníka uveden. Onen zájezd se pozná podle toho, že má v datovém poli KÓD stejnou hodnotu jako účastník v poli ZÁJEZD. Z trojice možností uvedených v předchozím odstavci jde tedy o možnost ad A, tedy *left join*.

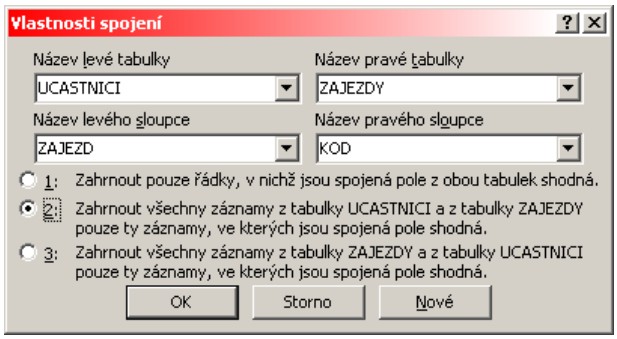
Postup - až na určení propojení - je velmi podobný postupu pro kartézský součin. V návrhovém prostředí dotazu se přidají dvě tabulky. Nyní je třeba stanovit způsob propojení. Podobně jako u relací se uchopí datové pole ZÁJEZD ve vzoru tabulky ÚČASTNÍCI a přetáhne se do datového pole KÓD ve vzoru tabulky ZÁJEZDY:



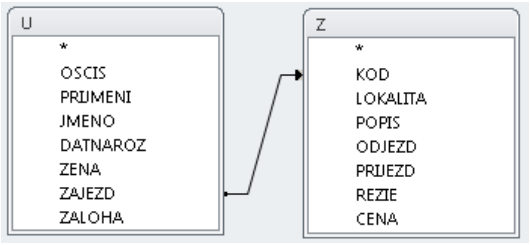
Po uvolnění tlačítka myši se zobrazí vytvořené propojení ve tvaru:



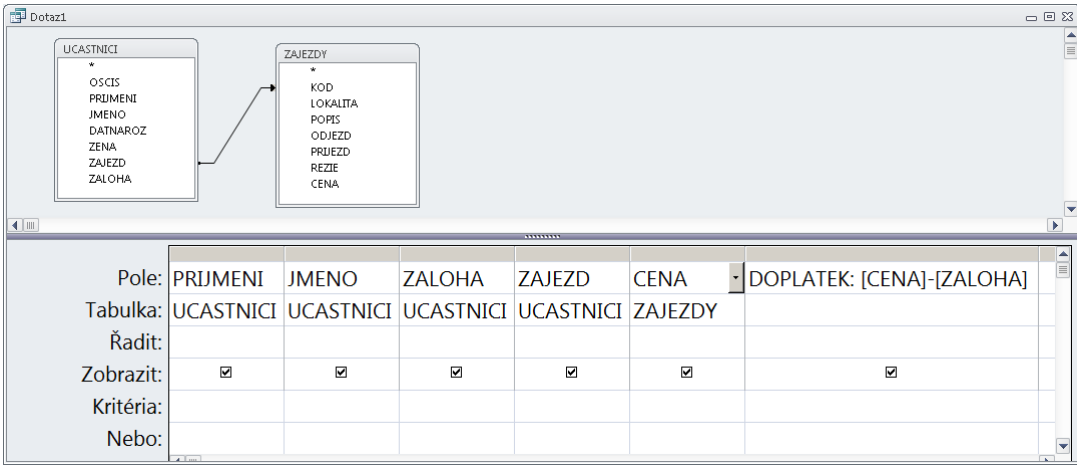
Není to námi požadovaný způsob propojení, proto otevřeme formulář pro výběr jedné ze dvou možností buď double-clickem na čáru spojující oba vzory tabulek, nebo pravým klikem na ni a volbou Vlastnosti spojení:



Velmi pečlivě pročteme a hlavně promyslíme všechny tři nabízené možnosti. Hledáme ... všechny ÚČASTNÍKY a ze ZÁJEZDŮ jen ten příslušný ... což je druhá z nabízených možností; tu zvolíme:



Teprve poté se zvolí datová pole, která je žádoucí mít ve výsledku dotazu, a stejně jako v případě kartézského součinu se zadá vypočítávané pole:



Po uložení návrhu dotazu pod vhodným jménem (zde Doplatky) lze požádat o provedení dotazu:

Stejně jako u kartézského součinu lze dotaz doplnit o řazení, filtrování aj.

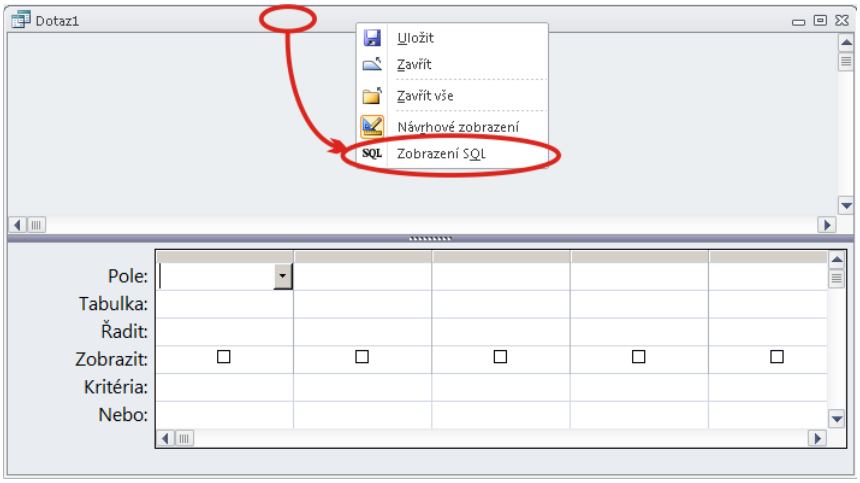
15.4 Dotazy jako příkazy SQL

V této kapitole nejsou popisovány příkazy jako takové (viz kapitolu *Jazyk SQL*), ale prostředí, ve kterém se příkazy zapisují a ukládají jako dotazy.

Prostředí databázových programů pro zápis textu příkazu SQL je analogické prostředí běžných textových editorů (např. Notepad = Poznámkový blok u Microsoftu). Kvalitnější prostředí poskytují navíc známé "našeptávání" a průběžnou kontrolu syntaxe, méně kvalitní databázové programy kontrolují syntaxi alespoň při uzavření tohoto editoru. Databázový program Access má při tvorbě návrhu dotazu k disposici dvě různá, avšak navzájem svázaná prostředí: pro "laiky" (plně popsáno výše, které je historicky nazýváno *Návrhové zobrazení*), a prostředí textového editoru, nazývaného *Zobrazení SQL*. Mezi těmito prostředími se nejjednodušeji přepíná pravým kliknutím kamkoliv do řádku nadpisu jednoho prostředí, a z následného kontextového menu výběrem druhého prostředí.

Z uvedeného plyne postup např. při tvorbě nového dotazu přímo jako text příkazu SQL:

- Návrh dotazu začíná stejně jako v předchozí kapitole.
- Formulář pro výběr datových zdrojů se (bez výběru) zavře.
- Pravý klik do řádku nadpisu a následný výběr *Zobrazení SQL* přepne prostředí do textového editoru SQL.



V textovém editačním okně mohou zůstat zbytky z přípravy Návrhovým prostředím. Cokoliv je v textovém editačním okně zobrazeno, smaže se (v následující ukázce je to slovo SELECT a středník) a lze začít zapisovat *jakýkoliv* příkaz SQL (tedy nejen příkaz SELECT):

Po ukončení zápisu textu příkazu je pak další postup opět stejný jako v případě shora popsaného Návrhového zobrazení: Editační okno se zavře, změny se nechají uložit, a v případě nově vytvářeného dotazu se mu na závěr přidělí jméno.

Je však nutno počítat s tím, že v případě chyby v zápise textu příkazu je tato chyba hlášena nelogicky až po zadání jména dotazu. V tom případě se dotaz neuloží, řízení se vrací textovému editoru a text příkazu je nutno dále upravit. Tento koloběh se opakuje, dokud není text příkazu bez syntaktické chyby.

16 Možná řešení dotazů jako SQL - Téma I

V úvodu tohoto dílu těchto učebních textů jsou uvedeny odkazy na možná řešení všech tří témat. Na závěr uveďme možné řešení dotazů s drobným komentářem alespoň Tématu I. Řešení se opírá o strukturu tabulek uvedenou výše.

Níže jsou uvedeny už jen tvary příkazu SQL, který danou činnost realizuje.

16.1 Popis řešení

Stejně jako v reálném životě je možno vymyslet řadu způsobů, jak úlohy řešit. Uváděná možnost využívá vlastnosti dotazu, který sjednocuje všechna datová pole jednoho (odkazujícího, zde ÚČASTNÍCI) datového zdroje s datovými poli příslušného záznamu druhého (odkazovaného, zde ZÁJEZDY) datového zdroje. Srozumitelněji řečeno: ke

každému řádku ÚČASTNÍKA připojí data toho řádku ZÁJEZDŮ, na který tento účastník jede. Výsledek tohoto dotazu pak bude sloužit jako datový zdroj při řešení požadovaných úloh.

Takový dotaz má - jako příkaz SQL - možný tvar:

```
select U.*, Z.*
  from UCASTNICI U left join ZAJEZDY Z on U.ZAJEZD=Z.KOD
  order by Z.KOD, U.PRIJMENI
```

Uložme ho a v dalších úlohách se na něj odkazujme jako na VSECHNA_DATA. Část výsledku tohoto dotazu:

VSECHNA_DATA

OSCIS	PRIJMENI	JMENO	DATNAROZ	ZENA	ZAJEZD	ZALOHA	KOD	LOKALITA	POPIS	ODJEZD	PRIJEZD	REZIE	CENA
Bar	Barčíková	Odeta	27.8.1991	☑	CRA	4 100	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Buš	Bušovský	Inocenc	26.6.1979	☐	CRA	4 600	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Cic	Cichá	Anděla	22.3.1968	☑	CRA	2 700	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Foj	Fojtách	Radoslav	27.6.1981	☐	CRA	2 200	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Kin	Kinc	Vavřinec	2.9.1971	☐	CRA	4 700	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Kow	Kowolowská	Zoja	25.4.1975	☑	CRA	2 600	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Lam	Lampartová	Gerta	5.2.1985	☑	CRA	2 000	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Neu	Neuberger	Darius	7.3.1971	☐	CRA	2 100	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Olb	Olbřímek	Andrej	15.8.1973	☐	CRA	2 700	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Pou	Poukar	Zdenek	16.9.1983	☐	CRA	4 500	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Vid	Vidlák	Alois	28.1.1963	☐	CRA	3 000	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Zoč	Zoček	Albín	23.12.1978	☐	CRA	3 500	CRA	Turnov	Geopark a CHKO Český	2.7.2018	9.7.2018	10 000	7 000
Boh	Bohoněk	Zdeněk	6.12.1975	☐	EGE	3 100	EGE	Karlovy Vary	Geopark Egeria - včetr	2.7.2018	8.7.2018	9 000	6 500

Záznam: 1 z 40 | Bez filtru | Vyhledávání

16.2 Dotaz 1 tématu I

Dotaz zní: Kolik účastníků se přihlásilo na jednotlivé zájezdy?

Možné řešení:

```
select
    KOD, LOKALITA,
    Count(PRIJMENI) as POCET_PRIHLASENYCH
  from VSECHNA_DATA
  group by KOD, LOKALITA
```

Dotaz je v ukázkové databázi uložen jako POCET_UCASTNIKU. Výsledek dotazu:

1_POCET_UCASTNIKU

KOD	LOKALITA	POCET_PRIHLASENYCH
CRA	Turnov	12
EGE	Karlovy Vary	14
LPI	Děčín	14

Záznam: 1 z 3 | Bez filtru | Vyhledávání

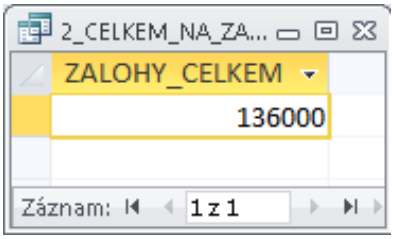
16.3 Dotaz 2 tématu I

Dotaz zní: Kolik je vybráno celkem na zálohách?

Možné řešení:

```
select
    Sum(ZALOHA) as ZALOHY_CELKEM
  from UCASTNICI
```

Dotaz je v ukázkové databázi uložen jako CELKEM_NA_ZALOHACH. Výsledek dotazu:



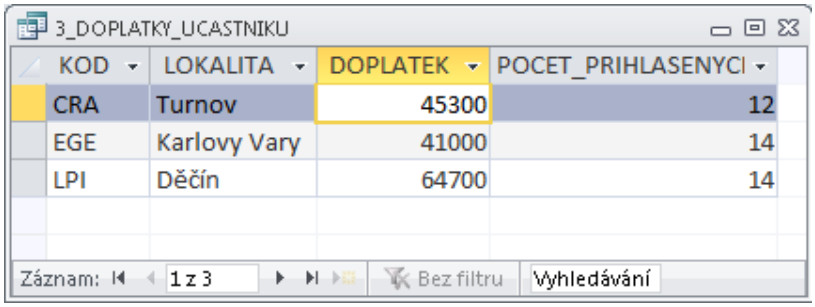
16.4 Dotaz 3 tématu I

Dotaz zní: Kolik ještě doplatí účastníci v jednotlivých zájezdech?

Možné řešení:

```
select
  KOD, LOKALITA,
  Sum(CENA)-Sum(ZALOHA) as DOPLATEK,
  Count(PRIJMENI) as POCET_PRIHLASENYCH
from VSECHNA_DATA
group by KOD, LOKALITA
```

Dotaz je v ukázkové databázi uložen jako DOPLATKY_UCASTNIKU. Výsledek dotazu:



16.5 Dotaz 4 tématu I

Dotaz zní: Kolik roků je nejstaršímu přihlášenému muži?

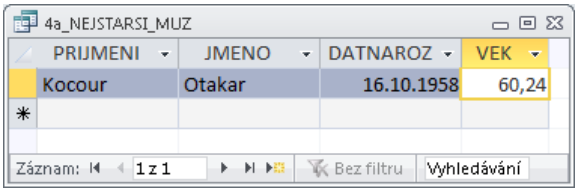
Možné řešení:

```
select
  PRIJMENI, JMENO, DATNAROZ,
  Round((date()-DATNAROZ)/365.25,2) as VEK
from UCASTNICI U
where
  (Date()-DATNAROZ) =
  (select max (Date()-V.DATNAROZ) from UCASTNICI V where not ZENA)
  and not ZENA
```

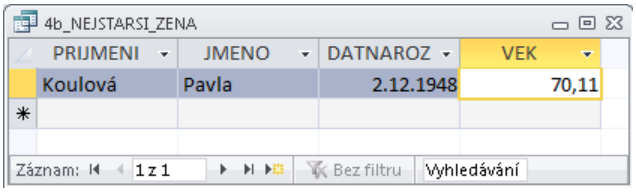
Dotaz je v ukázkové databázi uložen jako NEJSTARSI_MUZ.

*Poznámka: Dvě podmínky - zdánlivě zbytečně dvě stejné - **not ZENA** zbytečné nejsou. První z nich v poddotazu zajišťuje věk nejstaršího muže. Druhá řeší situaci, kdyby věk jak nejstaršího muže, tak nejstarší ženy byl stejný.*

Výsledek dotazu na nejstaršího muže:



Pro zajímavost a kontrolu dotaz na nejstarší ženu. Text dotazu se mění pouze v podmínce pohlaví (místo *not ZENA* je jen *ZENA*. Data jsou vytvořena tak, že nejstarší žena je starší než nejstarší muž:



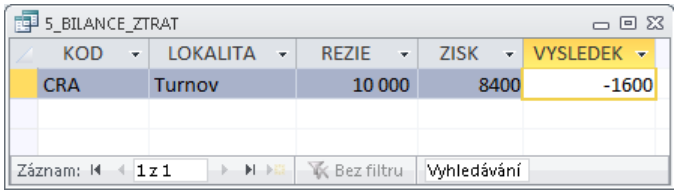
16.6 Dotaz 5 tématu I

Dotaz zní: Který zájezd je ztrátový?

Možné řešení:

```
select
   KOD, LOKALITA, REZIE,
   Sum(CENA)/10 as ZISK,
   ZISK-REZIE as VYSLEDEK
 from VSECHNA_DATA
 group by KOD, LOKALITA, REZIE
 having Sum(CENA)/10-REZIE<0
```

Dotaz je v ukázkové databázi uložen jako BILANCE_ZTRAT. Výsledek dotazu:



17 Výrazy

Formální jazyky (programovací, dotazovací a jiné) slouží jako nástroj pro vytvoření programu, který je realizací algoritmu zpracovávajícího data (třeba řeší kvadratickou rovnici). Algoritmus pracuje s nějakými daty, které mají **hodnotu** a **typ** (např. hodnotu 27 a typ Celé číslo). Právě **výraz** je ve formálních jazycích jediným nositelem hodnoty. Jde o konstrukci, která se při provádění algoritmu vyhodnotí a odevzdá do dalšího zpracování nějakou hodnotu nějakého typu. Příklad:

> Výraz **Alfa+4** (má-li Alfa celočíselnou hodnotu 27) odevzdá při zpracování hodnotu **31** typu **Celé číslo.**

Je zřejmé, že pro zápis výrazu ve formálních jazycích musí existovat přesná pravidla s přesným stanovením významu jeho jednotlivých částí. Tomu je věnována tato kapitola.

17.1 Definice pojmu Výraz

Pravidla pro zápis výrazu budou zvlášť sympatická uživatelům Excelu resp. obdobných tabulkových procesorů. Tzv. "vzorec" v Excelu má tvar =VÝRAZ, kdežto v návrhovém prostředí dotazu má tvar NÁZEV : VÝRAZ. Přitom VÝRAZ má v Excelu, v databázích a koneckonců i ve většině formálních jazyků stejnou syntaxi (stejná pravidla) i stejnou sémantiku (stejný význam). Drobná odchylka je v tom, že Excel může použít kromě jména buňky také adresu buňky. Pokud by částka nějaké zálohy byla v buňce v C14, pak analogický vzorec pro výpočet 15% DPH ze zálohy by v Excelu byl =15*C14/100. Pokud by však uživatel buňku C14 pojmenoval identifikátorem ZÁLOHA, pak i v Excelu by byl vzorec totožný: =15*ZÁLOHA/100.

Ve vzorcích je tedy základním pojmem výše zmíněný **Výraz**. Výraz **V** je tvořen jedním (a právě jedním) z následujících pojmů:

	Tvar	Příklad	Poznámka
(1)	zápis konstanty	234.12	
(2)	identifikátor	Alfa nebo Beta(12)	(proměnné, datového pole ...) nebo indexovaná proměnná
(3)	operace	3 * Alfa	Obecně **V** ⊗ **V** kde ⊗ je operační znaménko.
(4)	volání funkce	sin (Alfa)	Obecně **Fce (V, V, ... , V)**, kde **Fce** je identifikátor funkce.
(5)	(V)	(3 * sin (Alfa - 1)) / 5	Obecně jakýkoliv výraz uzavřený v kulatých závorkách.

Tabulka definuje pojem Výraz sice exaktně, ale pro mnohé čtenáře nezvykle. Jde totiž o tzv. *rekurzivní* definici: v definici jako takové už se používá přímo definovaný pojem. Protože naprosto vyčerpávající definice jednotlivých pojmů (1) až (5) je mimo zaměření těchto textů, jsou níže v jednotlivých odstavcích uvedeny jen nejnutnější poznámky vztahující se k výrazům v dotazech.

17.2 Příklad na rozmyšlenou

Na počáteční procvičení tento příklad: následující zápis:

```
3 + Alfa * 4 - sin(Beta / 2)
```

Tento zápis výrazem JE nebo NENÍ ?

Většina po krátkém přemýšlení odpoví, že to výraz JE. Ale pozor: pokud to skutečně je výraz, pak který: (1), (2), (3), (4) nebo (5)? Rozhodně to nemůže být i to i to i to!

Není to tak složité. Vylučovací metodou: Není to (1) = zápis konstanty. Ono to sice obsahuje zápis konstanty, ale jako celek to konstanta není. Analogicky (2) a (4). Pokud by to tedy měl být výraz, musí to být (3) = operace. Ale která?

I když mnozí protestují a chtějí začít operací sčítání +, začněme naschvál operací násobení *. Vyznačme to barevně:

```
3 + Alfa * 4 - sin(Beta / 2)
```

Ovšem měla-li by to být tato operace - viz (3) - pak musí být

```
3 + Alfa
```

výraz a

```
4 - sin(Beta / 2)
```

musí být rovněž výraz. Měla-li být konstrukce (opět vyznačme barevně)

```
3 + Alfa
```

výrazem (opět vylučovací metodou = operací), musí být

```
3
```

výrazem a rovněž

```
Alfa
```

musí být výrazem. Obě tyto konstrukce výrazy jsou: **3** je výraz ad (1) a **Alfa** je výraz ad (2).

Analogicky se dojde ke kladné odpovědi pro druhou část výchozího zápisu - doporučujeme dokončit samostatně.

Dále doporučujeme promyslet: začali jsme operací násobení. Ke stejnému kladnému výsledku (že zápis JE výrazem) bychom došli, kdybychom začali sčítáním nebo odečítáním. V definici výrazu se totiž nevyskytuje ani slovo o *prioritě* operací; priorita operací je totiž definována až při definici jednotlivých operací, které jsou dány **operačním znaménkem** a **počtem a typem operandů**. Stejné operační znaménko tedy může mít na různých typech operandů různou prioritu (protože jde o různé operace).

17.3 Zápis konstanty

17.3.1 Zápis číselné konstanty

Celočíselná konstanta se zapisuje běžným způsobem, avšak bez případných tisícových oddělovačů. Mohou se zapisovat i případné počáteční bezvýznamné nuly.

Pro zápis konstanty s desetinnou částí platí stejná pravidla, navíc jedno důležité: zásadně se používá desetinná tečka, nikoliv desetinná čárka!

Správně	Chybně	Poznámka
27		
-09627		Znaménko se zapisuje těsně před první cifrou
	21 700	Obsahuje mezeru
15.28		
15.28e-3		= 15.28 x 10^{-3} = 0.0153
	15,28	Obsahuje čárku
	1 545.283 759	Obsahuje mezery

17.3.2 Zápis textové konstanty

Znaky textu jsou uvozeny znakem " (uvozovky) a stejný znak stojí za posledním znakem textu. V databázových výrazech (přesněji v dotazovacím jazyku SQL) lze místo uvozovek použít i znak ' (apostrof). V obou případech ale platí: jakým znakem je text uvozen, takovým musí být zakončen.

Je-li text textové konstanty uvozen (a tedy i zakončen) uvozovkami a je-li zapotřebí mít jako jeden z vnitřních znaků textu také uvozovky, pak se tyto "vnitřní" uvozovky zdvojí - zapisují se dvě hned vedle sebe. Stejné platí o znaku apostrof.

Správně	Chybně	Poznámka
"Jan Novák"		
"Jan ""Ringo"" Novák"		Jan "Ringo" Novák
	"Jan Novák'	Nekončí tím čím začíná
	#Jan Novák#	Nezačíná ani " ani '

17.3.3 Zápis konstanty typu datum a čas

Mezi dva znaky s kódem 35 (znak #, angl. **Hash**) se zapíše nejprve datum bez mezer ve tvaru MM/DD/YYYY - pozor na americké pořadí, pak mezera, a nakonec čas bez mezer ve tvaru HH:MM:SS.

Část s časem může chybět (ale ne datum), pak se nezapisuje ani mezera za datem. Část s datem může chybět (ale ne čas), pak se nezapisuje ani mezera před časem.

Tato pravidla akceptuje většina dostupných databázových systémů. V různých systémech mohou být dílčí (ne však uvedená pravidla omezující) variace. Může např. chybět část s vteřinami a dvojtečka před ní; není to omezení, protože 13:25 je to samé jako 13:25:00. Některé systémy povolují zadat čas s větší přesností, tj. časovou část např. ve tvaru HH:MM:SS.SSS (vždy ovšem s desetinnou tečkou, nikoliv čárkou - Microsoft Jet to však neumožňuje).

U různých systémů je však vhodné se seznámit s interpretací chybějící časové resp. datumové části. Co je např. výsledkem dotazu na čas, chybí-li čas v zadávané konstantě? *Většinou* to bývá půlnoc ...

Správně	Chybně	Poznámka
#01/02/2020#		2. ledna 2020
#1/2/2020#		2. ledna 2020
#02/17/2020 12:15:00#		Konec třetí dvouhodinovky v PO 17. února
	#20/1/2020#	20. měsíc není
	#1.1.2020#	Obecně nepřijmou všechny systémy

17.3.4 Zápis logické konstanty

Tento datový typ je v různých systémech nazývaný různě (např. Boolean, Logical, Yes/No). Vždy však tuto množinu tvoří dvě "protikladné" hodnoty.

Zápisem konstanty tohoto typu je jedno ze dvou slov: **True** a **False** ve smyslu, který skutečně je významem těchto slov: Pravda a Nepravda. Negací pravdy je nepravda, negací nepravdy je pravda.

Při zápisu těchto slov se v databázových (obecně Case-nonsensitive) systémech nebere ohled na velikost písmen (True = true = TRUE). Samotná slova se zapisují bez mezer.

Správně	Chybně	Poznámka
TRUE		Pravda, Ano, logická 1
FaLse		Nepravda, Ne, logická 0
	Nepravda	Není to jedno ze dvou vyhrazených slov

17.4 Identifikátor

Poznámka: *Viz také odstavec "Tvar a zápis jmen" shora v těchto textech.*

Identifikátor je autorem aplikace, databáze ap. vymyšlené označení pro datovou hodnotu, která je předmětem zpracování a která je uložena na nějakém paměťovém mediu (operační paměť, disk ...). Obecně jde o symbolické označení adresy paměti, kde se daná hodnota nachází.

Definice identifikátoru je následující:

```
Identifikátor je posloupnost písmen a číslic začínající písmenem.
```

V databázových (obecně case-nonsensitive) systémech nezáleží na *velikosti* písmen. To si lze představit např. tak, jako by je systém všechna považovat za velká.

Znakem identifikátoru však nesmí být zvláště mezera. Identifikátorem proto nemůže být např. **Cena zájezdu** (obsahuje mezeru). Protože mnohdy je vhodné nějak opticky oddělit některé znaky identifikátoru, rozšiřuje většina zvláště databázových systémů množinu číslic ještě o znak "_" (podtržítko).

Správně	Chybně	Poznámka
Alfa		= alfa = ALFA

aLfA		= alfa = ALFA
Cena_Zajezdu		Obsahuje povolené podtržítko
Beta_12		= beta_12 = BETA_12
	12_Beta	Nezačíná písmenem

Poznámka 1: Některé systémy, povolující jako znak identifikátoru podtržítko, jej považují nikoliv za číslici, ale za písmeno. Znamená to, že v takovém případě může identifikátor podtržítkem začínat.

Poznámka 2 zvláště pro databázové systémy: Identifikátory se používají při tvorbě databáze na každém kroku - jako jména datových polí, tabulek, dotazů atd. Na druhé straně jsou databáze svými dnes už mezinárodními formáty vhodným prostředkem pro vskutku mezinárodní výměnu dat. Tam však zhusta nastává problém. Projeví-li o naše data zájem v zahraničí, dopředu nevíme, jaký databázový systém u nich používají a jak se vyrovnává se znaky národních abeced v identifikátorech. Proto autoři našich databází v případě, že by mohla být zasílána do zahraničí, nepoužívají v identifikátorech diakritiku.

Poznámka 3: Závěr předchozí poznámky o diakritice netýká obsahu dat, pouze identifikátorů použitých jako jména objektů v databázi.

17.5 Operace

Konkrétní operace jakožto část výrazu zajišťuje provedení konkrétního algoritmu na jistých operandech a vrácení výsledku provedeného algoritmu. Např. zápis 2+3 zajistí provedení známého algoritmu (viz 1. ročník ZŠ) a vrátí hodnotu 5. Ovšem zápis "Jan_"+"Novák" zajistí provedení už méně známého algoritmu a vrátí hodnotu "Jan_Novák". Přitom obě operace mají stejné operační znaménko, liší se *typem* operandů.

Dále: Je-li hodnota B rovna 5 (slovy pětka), pak operace B-2 vrátí hodnotu 3, kdežto operace -B vrátí hodnotu "záporná pětka". Přitom obě operace mají stejné operační znaménko, liší se *počtem* operandů.

Operace tedy mohou mít zejména různý počet operandů. Operace s jedním operandem se nazývají *unární*, se dvěmi operandy *binární*, se třemi operandy *ternární* atd. Operace s nulovým počtem operandů je *nulární* - není nesmyslná, jak by se zdálo; může vracet při každém provedení svého algoritmu např. aktuální čas.

Poznámka: Protože textový zápis operací je lineární, nastává problém s operačními znaménky. Problém celkem není s binárními operacemi (až na omezený počet typografických znaků pro znaménka). Unární operace používají v databázových systémech prefixovou notaci (znaménko stojí *před* operandem) a také s nimi není problém. Už je však problém s operacemi ternárními, obecně s větším počtem operandů. V editoru lineárního textu nejde zapsat znaménko a pravidelně plošně kolem něj jednotlivé operandy. Proto jsou požadavky na provedení algoritmu operací s větším počtem operandů realizovány nikoliv výrazy typu (3) = operace, ale výrazy typu (4) = volání funkce.

Z uvedeného vyplývá existence celé řady operací se stejným operačním znaménkem, ale různým typem operandů. Pro účely databází se omezme na unární a binární operace a uveďme jen některé z nich.

Upozornění: *Obsah tabulky odpovídá pravidlům databázového systému Microsoft Jet, kterého využívá např. program Access. V jiných databázových systémech a zvláště v různých programovacích jazycích je nutno nahlédnout do jejich dokumentace.*

Typ levého operandu L	Operační znaménko	Typ pravého operandu P	Výsledek	Typ výsledku	Poznámka
číslo	+	číslo	Součet L+P	číslo	Typ čísla je vyšší z typů L a P
číslo	-	číslo	Rozdíl L-P	číslo	Typ čísla je vyšší z typů L a P
číslo	*	číslo	Součin L*P	číslo	Typ čísla je vyšší z typů L a P
číslo	/	číslo	Podíl L/P	racionální číslo	
číslo	\	číslo	Celočíselné dělení L a P	celé číslo	L a P se nejdříve případně zaokrouhlí

číslo	mod	číslo	Zbytek po dělení L a P	celé číslo	L a P se nejdříve případně zaokrouhlí: 7 mod 2
číslo	^	číslo	Umocnění L^P	číslo	Realizováno vztahem $L^P = e^{P . ln(L)}$
datum/čas	+	Číslo	Pozdější	datum/čas	Okamžik o P "dní" dále za L
Číslo	+	datum/čas	Pozdější	datum/čas	Okamžik o L "dní" dále za P
datum/čas	-	Číslo	Dřívější	datum/čas	Okamžik o P "dní" dříve před L
datum/čas	-	datum/čas	Interval	Číslo	Kolik "dní"uplynulo od P k L
text	+	text	spojení P za L	text	"Jan_" + "Novák" = "Jan_Novák"
Ano/Ne	and	Ano/Ne	logický součin	Ano/Ne	viz binární aritmetika: L and P
Ano/Ne	or	Ano/Ne	logický součet	Ano/Ne	viz binární aritmetika: L or P

Operace + a - definované pro typ datum/čas se týkají jazyků, které pracují s tímto datovým typem. Předpokládá, že interní uložení je ve tvaru racionálního čísla, přičemž celá část tohoto čísla je pořadové číslo dne počínaje nějakým počátkem (u Microsoftu je den 0 stanoven u většiny jejich produktů na 31. prosince 1899), a necelá část tohoto čísla de facto určuje čas v tomto dni. Jinak řečeno, jednotkou určující datum (a čas) je jeden den. Racionální hodnota 1.75 (tedy 1 ¾) chápána jako datum a čas je 1. ledna 1900 18:00 hod. Den číslo 1 je 1.1.1900 a 0.75 = 3/4 dne, tedy 6 hodin odpoledne.

17.6 Volání funkce

Volání funkce jakožto část výrazu zajišťuje provedení konkrétního algoritmu na jistých operandech a vrácení výsledku provedeného algoritmu. Např. zápis sqr(4) zajistí provedení algoritmu výpočtu druhé odmocniny (angl. Square Root) z hodnoty 4 a vrátí hodnotu 2. Obdobně jako operace je funkce definována svým jménem a počtem a typem operandů.

V databázových systémech, na které je zaměřena tato publikace, existují kromě tzv. agregačních funkcí popsaných v kapitolách o dotazech seskupujících dat, také tzv. skalární funkce. S nimi je drobný problém: je jich značné množství, ale v různých systémech mohou být jednak různé, jednak různě pojmenované. To opomíjíme ty systémy s vynikající možností (např. Microsoft) umožňujících autorům databází definovat v podstatě libovolný počet svých vlastních funkcí.

Vytvořit seznam skalárních funkcí společných všem databázovým systémům jak jménem, tak funkcí je práce nadlidská, autor této publikace to může potvrdit. Po hodinách pátrání ve všech možných zdrojích se začíná obávat, že taková funkce je snad jediná, a to **abs** (absolutní hodnota čísla).

Proto následující tabulka obsahuje alespoň minimální výčet funkcí použitelných v prostředí Microsoft Jet, což je základní databázové prostředí, se kterým se studenti setkají. Tabulka je maximálně zestručněná, zájemce o podrobnější informace je získá např. v msdn.microsoft.com.

Pro přehlednost je v tabulce datumová hodnota #01/02/2020 13:28:51# označena symbolem ⁝⁝ a textová hodnota "Nováková" symbolem x͡x.

Identifikátor funkce	Typ operand(ů)	Příklad volání	Výsledek	Typ výsledku	Poznámka
abs	číslo	abs (-27)	27	číslo	Absolutní hodnota operandu
sin	číslo v [rad]	sin (3.14/6)	0.5	číslo	sinus; 3.14/6 [rad] = 30 [º]
cos	číslo v [rad]	cos (3.14/3)	0.5	číslo	kosinus; 3.14/3 [rad] = 60 [º]
atan	číslo v [rad]	atan (1)	0.7854	číslo v [rad]	arkus tangens; 0.7854 [rad] = 3.14/4 [rad] = 45 [º]
exp	číslo	exp (2)	7.3891	číslo	= e^2
log	číslo	log (7.3891)	2	číslo	**přirozený** logaritmus
sqr	nezáporné číslo	sqr (4)	2	číslo	square root = druhá odmocnina √2

format	číslo C, text F	format (1234.567891, "# ##0.000 000")	1 234.567 891	text	hodnota C formátovaná podle vzoru F
trim	text T	trim(" Čas ")	"Čas"	text	T bez počátečních a koncových mezer
length	text T	length ("Čas")	3	celé číslo	počet znaků T
lcase	text T	lcase ("Čas")	"čas"	text	T v němž jsou všechna **písmena** převedena na malá
ucase	text T	ucase ("Čas")	"ČAS"	text	T v němž jsou všechna **písmena** převedena na velká
left	text T, číslo C	left (xx, 3)	"Nov"	text	C znaků zleva z textu T
right	text T, číslo C	right (xx, 2)	"vá"	text	C znaků zprava z textu T
mid	text T, číslo C, číslo N	mid (xx, 3, 4)	"vákо"	text	N znaků z T počínaje C-tým znakem
date		date ()	#01/02/2020#	datum/čas	datumová část data/času v okamžiku volání
time		time ()	#13:28:51#	datum/čas	časová část data/času v okamžiku volání
now		now ()	;	datum/čas	úplná hodnota data/času v okamžiku volání
day	datum/čas D	day (;)	2	číslo	číslo dne v hodnotě D
month	datum/čas D	month (;)	1	číslo	číslo měsíce v hodnotě D
year	datum/čas D	year (;)	2020	číslo	číslo roku v hodnotě D
hour	datum/čas D	hour (;)	13	číslo	hodiny v hodnotě D
minute	datum/čas D	minute (;)	28	číslo	minuty v hodnotě D
second	datum/čas D	second (;)	51	číslo	vteřiny v hodnotě D
weekday	datumčas D, číslo C	weekday (;, 2)	1	číslo	číslo dne v týdnu - viz poznámky níže
weekdayname	číslo D, ano/ne Z, číslo C	weekdayname (1, true, 2)	"po"	text	jméno dne v týdnu - viz poznámky níže
monthname	číslo M, ano/ne Z	monthname (4, false)	"duben"	text	jméno měsíce v roce - viz poznámky níže

WeekDay: Výsledkem je číslo dne v týdnu datumu D. Druhý parametr C udává, který den je první v týdnu (kterým dnem týden začíná): C=1 znamená, že den č. 1 je neděle, C=2 znamená, že den číslo 1 je pondělí atd.

WeekDayName: Výsledkem je jméno dne číslo D v týdnu. Je-li druhý parametr Z roven True (ano), pak je výsledkem zkratka dne, je-li druhý parametr Z roven False (ne), je výsledkem plný název dne. Třetí parametr C udává, který den je první v týdnu (kterým dnem týden začíná): C=1 znamená, že den č. 1 je neděle, C=2 znamená, že den číslo 1 je pondělí atd. Jméno dne respektuje národní prostředí, v českém prostředí je zkratka dvoupísmenová.

MonthName: Výsledkem je jméno měsíce číslo M v roce. Je-li druhý parametr Z roven True (ano), pak je výsledkem zkratka měsíce, je-li druhý parametr Z roven False (ne), je výsledkem plný název měsíce. Jméno měsíce respektuje národní prostředí, v českém prostředí je zkratka třípísmenová.

18 Závěr

Předložené výukové texty byly původně deklarovány pro obor *Geovědní a montánní turismus*. Databázová problematika je však natolik obecná, že rozdíl mezi zaměřením na jiné obory spočívá více méně ve vhodně volených příkladech, na kterých jsou pojmy a postupy demonstrovány. Snahou autorů bylo proto volit taková témata, která by byla přijatelná pro mnohé další (nejen) geo-vědní obory, a tím se rozšířila využitelnost těchto výukových textů.

Z velmi široké teoretické databázové problematiky byly do těchto textů vybrány jen ty oblasti, které jsou na jedné straně nutné pro pochopení fungování (zvláště relačních) databází jako takových, na druhé straně usnadní samostatnou práci v prostředí databázových programů při řešení vlastních odborných úloh.

Autoři si jsou vědomi jednoho velkého nedostatku předkládaných textů. Pouze v minimálním rozsahu a jen náznakově pro jednu jedinou úlohu (byť ve třech stupních složitosti) je naznačena problematika snad nejdůležitějšího jádra databázové problematiky, a tím je analýza a vůbec předprojektová příprava úlohy. Důvod je nasnadě: databázové aplikace pokrývají dnes snad všechny oblasti lidské činnosti, a tak jako je pestrý sám život, jsou nepřeberné možnosti realizace databázově podpory jednotlivých oblastí. I když byly činěny pokusy o formalizaci analytických postupů, o automatizované návrhy jak datových struktur, tak vlastního zpracování dat - v každé úloze se najde byť jediná výjimka z pravidel těchto algoritmizovaných činností, že v důsledku toho se v konečné fázi stejně vytváří řešení "na míru". A tam pak nastupují osobní, především zkušenostní vlastnosti řešitelů. Uvádí se, že jako kodér je pracovník použitelný po roce praxe, jako realizátor hotové analýzy po třech letech, ale jako kvalitní analytik nejméně po pěti až osmi letech nepřetržité praxe. Ovšem ta praxe v sobě zahrnuje i sledování vývoje jak v teorii, tak v jejím uplatnění při používání databázových nástrojů. To je zřejmý důvod, proč je databázová problematika nejen mezi studenty, ale i mezi odbornými pracovníky jiných vědních oborů nepopulární a mnohdy přímo odmítaná: jednoduše díky absenci praxe tomu (alespoň v počátcích) nerozumí. To rozhodně není nedostatek jejich duševního potenciálu, to je běžná ukázka psychologie lidské osobnosti. Nejsem-li schopen dohlédnout na důsledek své činnosti (a to při nedostatku praxe prostě nejsem), raději volím takové postupy, které znám a které se osvědčily.

Nicméně přes uvedený nedostatek se autoři domnívají, že vytvořili materiál poskytující jak základní teoretická východiska, tak zejména popisující praktické kroky při tvorbě jednoduchých databázových úloh. Tyto kroky byly zvoleny tak, aby byly po jisté, celkem nenáročné míře zobecnění, použitelné v podobných problematikách.

Literatura a další výukové zdroje

[1] SCHEJBAL, C., HOMOLA, V., STANĚK, F.: *Geoinformatika.* PONT, s.r.o., 2004. ISBN 80-967611-8-8.

[2] CODD, E.F.,: *The Relational Model for Database Management.* (Version 2 ed.) Addison Wesley Publishing Company, 1990. ISBN 0-201-14192-2.

[3] DATE, C. J., DARWEN, H.: *Databases, types and the relational model: the third manifesto.* (Version 2 ed.) Reading, MA: Addison-Wesley, 2006. ISBN 0-321-39942-0.

[4] CHLAPEK, D., STANOVSKÁ, I., ŘEPA, V.: *Analýza a návrh informačních systémů.* Praha: Oeconomica, 2011. ISBN: 978-80-245-1782-7.

[5] MICROSOFT: *Description of the database normalization basics.* [online]. Microsoft, 2017. [cit. 11. 1. 2019]. Dostupné z: https://support.microsoft.com/en-us/help/283878

[6] HOMOLA, V.: *Hardwarové uložení číselných dat.* [online]. VŠB Ostrava, 2016. [cit. 11. 11. 2020]: Dostupné z: http://homel.vsb.cz/~hom50/INFORM/HWDDATA/HWDDATA.HTM