VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ
Ústav automatizace a informatiky

DATABÁZOVÉ SYSTÉMY

(doplňující text ke konzultacím v 3. ročníku kombinovaného bakalářského studia oboru *Aplikovaná informatika a řízení*)

Doc. RNDr. Ing. Miloš Šeda, Ph.D.

BRNO 2002

Obsah

1. Teoretické aspekty zpracování dat

1.1. Úvod

Tradiční a relativně nejčastější použití počítačů je v oblasti zpracování dat, dříve také nazývané *hromadným zpracováním dat* (HZD).

Rozsáhlejší systémy pro zpracování dat se nazývají *informační systémy*. Rozumíme jimi systémy pro sběr, uchování, vyhledání a zpracování dat za účelem poskytnutí informací. *Data* jsou údaje získané pozorováním nebo měřením a *informace* interpretací těchto dat a vztahů mezi nimi. Informační systémy zajišťují následující činnosti:

- výběr informace,
- prognózy vývoje,
- plánování,
- rozhodování,
- použití pro automatizaci inženýrských prací (AIP),
- použití pro zpracování ekonomických agend (mzdy, faktury, skladové hospodářství, účetnictví apod}.

Činnost informačního systému může probíhat jako

- zpracování v dávkách,
- konverzace se systémem.

První způsob je vhodný zvláště pro soubory, kde se zpracování účastní všechny nebo většina záznamů jako např. mzdová agenda. V přípravě podkladů je ovšem potřeba rozsáhlá administrativa a interval mezi počátkem zpracování a časem, kdy jsou připraveny výstupní sestavy může být dosti dlouhý.

Konverzační systémy byly původně vytvořeny pro aplikace vyžadující rychlou zpětnou vazbu jako je např. rezervace letenek, zpracování dat v bankovnictví apod. Výhodou je to, že vstup dat se provádí přímo tam, kde data vznikají, a odpadá tak mnoho administrativy a kódování, které je typické pro dávkové zpracování.

V HZD se používají systémy řízení souborů, které podporují některé techniky jako např. *sekvenční*, *index-sekvenční*, *indexové soubory* a *B-stromy*.

Charakteristickým rysem klasického hromadného zpracování dat je, že soubory jsou strukturovány podle potřeb konkrétních programů, které je používají. Ze zadání problému usuzujeme na to, jaká data budeme potřebovat pro zpracování a podle toho navrhneme soubory, které jsou "ušity na míru" pro daný program. V každém programu je pak přesně udáno, které soubory potřebuje, jak jsou strukturovány a organizovány, jaké výsledné informace se mají získat a do kterých výstupních souborů se mají uložit.

Ze skutečnosti, že popisy souborů jsou součástí uživatelských programů vyplývá řada těžkostí:

- *Programy a data jsou navzájem závislé*. Pokud je nutné změnit organizaci dat, je třeba tyto změny promítnout do všech programů, které s daty pracují.
- *Redundance* (*nadbytečnost*) *dat*. Agendy jsou pojímány jako relativně izolované části informačního systému bez širšího sdílení dat. Tak se může stát, že stejné údaje o pracovnících podniku jsou uvedeny váněkolika souborech - v evidenci osobní, účetní, kvalifikační, zdravotní atd.
- *Nekonzistence dat*. Je-li některý údaj redundantní, pak musí mít ve všech souborech stejnou hodnotu. Tato vlastnost se nazývá *konzistence*. Může se však stát, že při změně hodnoty se nepromítne tato změna do všech souborů a data se stanou nekonzistentní, např. pracovnice se vdá a změnu příjmení nahlásí pouze na mzdovou účtárnu.

- *Nekompatibilita dat.* Data stejného významu se v různých agendách mohou získávat odlišnými metodami a v různém čase, což může mít za následek odlišnost výstupních informací jednotlivých agend.
- *Obtížná dosažitelnost dat.* Chceme-li získat odpověď na dotaz ohledně stavu dat, znamená to vytvořit zvláštní aplikační program, který odpovídá právě na tento dotaz, případně se spokojit s již existujícími výstupy aplikačních programů a požadovanou informaci vánich vyhledat "ručně".
- *Izolovanost dat.* Data mohou být roztroušena v různých souborech. Soubory mohou být různě organizovány a data mohou mít různý formát. Tím se značně komplikuje tvorba nových aplikačních programů.
- *Problém současného přístupu více uživatelů.* Např. v systému rezervace místenek se jedná o koordinaci paralelních procesů, kdy jeden může modifikovat data a druhý číst. V podstatě to není možné v souborech s vysokou redundancí, kdy je nutné aktualizovat údaj na mnoha místech.
- *Problém ochrany dat před zneužitím.* Např. při zpracování dat v bankovnictví není žádoucí, aby mohl kdokoliv provádět s daty jakékoliv operace nebo aby měl přístup ke všem informacím. Při zpracování dat klasickým způsobem však aplikační program musí mít k dispozici tolik podrobností, že to ochranu dat proti zneužití velmi ztěžuje.
- *Problém integrity dat.* Hodnoty dat podléhají omezením, které odrážejí vlastnosti skutečných objektů ve světě (např. měnový kurs, limity váhových kategorií závodníků apod.). Také všechna data v datových souborech musí odpovídat stavu reálného světa. Této vlastnosti říkáme *integrita* (*celistvost*). Součástí aplikačních programů musí být kontrola vstupujících dat, což v případě, že se kontroly týkají více souborů může aplikační program zbytečně zkomplikovat.

1.2. Databázové systémy

Výše uvedené problémy klasických metod hromadného zpracování dat vedly ke vzniku a rozvoji *databázových systémů*. Mají následující vlastnosti:

1. Struktury datových souborů jsou odděleny od aplikačních (uživatelských) programů.
2. Přístup k datům je možný jen prostřednictvím programů databázového systému.
3. Data je možné vyhodnotit jakýmkoliv způsobem.
4. Je umožněn přístup více uživatelů současně a vyřešena ochrana dat před zneužitím.

Data již nejsou organizována v izolovaných souborech, ale v komplikovanější centrálně zpracovávané struktuře dat, zvané *databáze* DB (*database*) nebo také *báze dat*, pro kterou je vytvořena jediná interní organizace dat, společná pro všechny oblasti a způsoby využití těchto dat. Centrální správa databáze, tzn. všechny implementační programy, jsou realizovány prostřednictvím speciálního programového vybavení, které se nazývá *systém řízení báze dat* SŘBD (*database management system*, DBMS). Ten spolu s databází tvoří *databázový systém* DBS (*database system*). Zjednodušeně lze tedy říci, že

DB + SŘBD = DBS

Systém řízení báze dat zahrnuje:

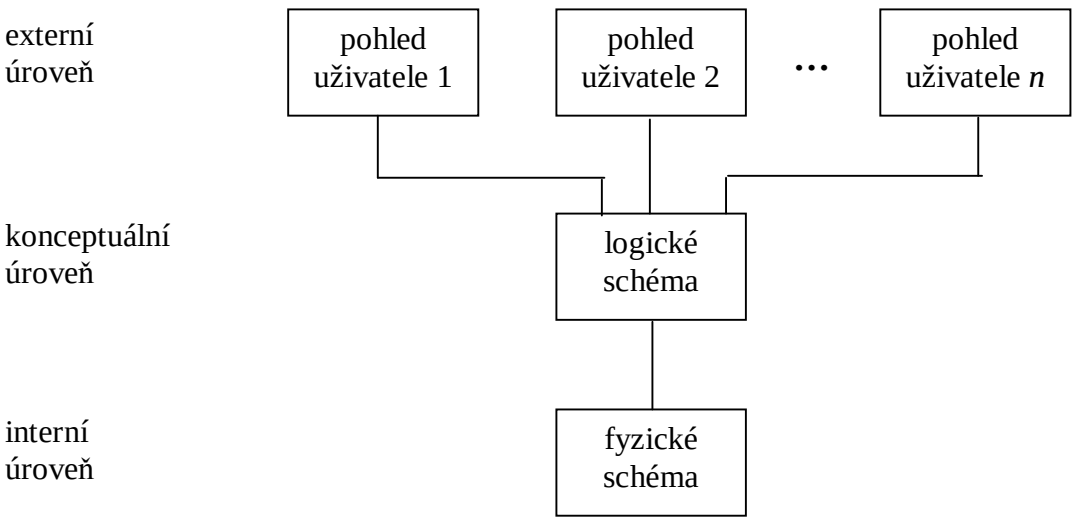
- prostředky pro popis dat, které se někdy označují jako *jazyk typu DDL* (*data definition language - jazyk pro definici dat*)
- prostředky pro popis algoritmu, označované jako *jazyk typu DML* (*data manipulation language - jazyk pro manipulaci s daty*).

Jazyk typu DDL slouží k vytvoření všech definic uživatelských dat potřebných v aplikaci.

Jazyk typu DML se používá jednak k aktualizaci dat (tj. k změnám dat v databázi, přidávání a rušení), jednak k výběru dat z databáze podle daných požadavků. Část DML určená pro výběr dat se nazývá *dotazovací jazyk* (*query language*).

Na obrázku je znázorněna architektura databázového systému. Lze rozlišit 3 úrovně:

1. *Externí úroveň* (*external level*) je reprezentována daty z pohledu uživatele. Pohled uživatele na data se též nazývá *externí schéma*. Mohou je představovat např. výstupní tiskové sestavy, formuláře pro vstup dat, popř. jiná data, která obsahují informaci užitečnou pro uživatele systému. Různí uživatelé mohou "vidět" (z důvodu odborného zaměření, přístupových práv apod.) různě vymezené části informačního obsahu databáze.
2. *Konceptuální úroveň* (*conceptual level*) je integrovaným pohledem nebo schématem celé databáze. Popis databáze na konceptuální úrovni se nazývá *logické schéma databáze* (*logical database scheme*).
3. *Interní úroveň* (*internal level*) koresponduje s vlastním fyzickým uložením dat na vnějších paměťových médiích a metodami přístupu k datům. Popis databáze na této úrovni se nazývá *fyzické schéma databáze* (*physical database scheme*).



Obr. 1. Architektura databázového systému

1.2.1 Nezávislost dat

Nezávislostí dat se v databázových systémech rozumí možnost změnit definice dat na nižší úrovni abstrakce, aniž by se tím ovlivnila definice na vyšší úrovni abstrakce. Mluvíme o dvou úrovních nezávislosti dat.

1. *Fyzická nezávislost dat* umožňuje změnit fyzické schéma a přitom nedojde ke změně logického schématu ani uživatelských aplikačních programů.
2. *Logická nezávislost dat* umožňuje změnit konceptuální úroveň popisu dat, aniž by bylo třeba přepisovat aplikační programy. Na externí úrovni se přitom nemění pohled těch uživatelů, jichž se změna logického schématu přímo netýká.

1.2.2 Sdílení dat

Na rozdíl od klasického agendového zpracování je při databázovém zpracování konceptuální úroveň prostřednictvím externí úrovně přístupná všem aplikačním programům a je zaručena její stabilita v čase i při změnách fyzického schématu na interní úrovni. Proto není nutné získávat redundantní údaje a celkový objem dat se sníží.

Skutečnost, že se data ***sdílejí*** všemi aplikačními programy má kromě snížení redundance příznivý vliv i na celkovou integrovanost informačního systému. Nedochází k náhodným rozdílům v hodnotách sdílených dat, které se používají na různé účely.

Centralizací dat do *sdílené databáze* se však odpovědnost za správu dat přenesla z jednotlivých agend na databázový systém. V rámci uživatelské organizace se předpokládá, že jeho provoz má na starosti pověřená osoba, tzv. *administrátor* (*správce*) *databáze*. Do jeho kompetence patří vytváření schémat na všech úrovních a definice a popis vazeb mezi jednotlivými úrovněmi. Externí schémata vytváří přitom podle požadavků uživatelů.

1.2.3 Integrita

O tomto pojmu jsme se již částečně zmínili v závěru 1. kapitoly. *Integrita* (*celistvost*) *databáze* je stav, váněmž jsou data v plném rozsahu přípustná a využitelná v aplikačních programech a mezi hodnotami položek souborů databáze platí vztahy, které jsou stanoveny k zaručení sémantické korektnosti databáze. Jinak se dá také říci, že je to stav, kdy hodnoty dat jsou správné, konzistentní a aktuální.

K narušení integrity může dojít chybami technického i základního programového vybavení, chybami v aplikačních programech a v datech apod.

Jak jsme již uvedli, součástí aplikačních programů musí být kontrola správnosti vstupujících dat. Některé systémy řízení báze dat (SŘBD) mají možnost uchovat *integritní omezení*, např. formule v predikátovém kalkulu 1. řádu, které po interpretaci hodnot položek databáze nabývají hodnotu `True` anebo `False` a informují o tom, zda stav databáze je správný či nikoliv. Tímto způsobem se dá zamezit provedení aktualizačních příkazů, které by vedly k nesprávnému stavu databáze. Příkladem integritního omezení je požadavek, aby věk oprávněného voliče bylo celé kladné číslo větší nebo rovné 18.

Procedury ukládání dat musí být takové, aby systém řízení báze dat v případě vzniku chyby mohl obnovit data beze ztrát. Z tohoto důvodu je SŘBD vybaven funkcemi *kopírování* celé databáze anebo jejich částí na záložní paměťové médium. V případě, že došlo ke ztrátě integrity, použije se kopie databáze k její obnově. Vzhledem k většinou velkým rozsahům databáze a z toho vyplývající časové náročnosti kopírování nelze jej provádět příliš často, což způsobuje ztrátu dat, která byla získána v době od posledního kopírování po poruchu. Pokud je riziko ztráty velké anebo má značné důsledky, používají se jemnější kopírovací procedury. Jednou z nich je použití tzv. *žurnálové záložní paměti*, kam se při každé změně dat v databázi zaznamená stav menších oblastí (záznamu, bloku) před změnou a po změně.

1.2.4 Náhodný přístup

Agendové zpracování je zaměřené jednoúčelově a předpokládá, že všechny požadavky na výstupní informace jsou specifikovány předem. Při používání programu se však velmi často objeví dodatečné požadavky na jeho funkci, jejichž realizace si může vyžádat značné programátorské úsilí.

V databázovém systému vzhledem k nezávislosti dat na aplikačních programech je snadné napsat nový aplikační program, který zabezpečí provedení požadované akce. V mnoha případech jde o požadavek na výběr dat podle zadaného kritéria. Většina SŘBD je vybavena speciálními uživatelskými jazyky neprocedurálního charakteru, které jsou orientovány na využití neprogramátory.

Možnost realizace náhodného přístupu k databázi výrazně zlepšuje využití dat v informačním systému a je často jedním z hlavních důvodů přechodu na databázovou technologii zpracování dat.

1.3. Modely dat

Posláním automatizovaného informačního systému je poskytovat informace o určité části reálného světa (*realitě*). Informační systém a jeho konstrukce je současně určitým druhem *modelu reality*. Před vybudováním databáze je tedy nutné analyzovat realitu a vytypovat *objekty*, o nichž chceme v databázi udržovat informaci. Objekty lze přitom rozdělit do dvou tříd:

- *Entity* - rozumíme jimi abstrakce libovolných existujících věcí.
- *Hodnoty* - charakterizují, popisují entity.

Vlastnosti entit se nazývají *atributy*. Atribut přiřazuje každé entitě z množiny entit hodnotu z nějaké neprázdné množiny nazývané *doména atributu*. Atribut je tedy funkce z množiny entit do domény atributu. Neformálně řečeno atributy jsou vztahy mezi entitami a hodnotami. (Později se budeme ještě bavit o vztazích mezi entitami, které nazýváme *relace*.)

Příklad 1.3.1
Entity množiny studentů VUT mají například atributy jméno, pohlaví, věk. Jsou to funkce z množiny studentů VUT po řadě do množiny alfabetických řetězců možných jmen studentů, množiny {muž,žena} a intervalu [18,60].

Atribut či množina atributů, jehož hodnoty jednoznačně určují každou entitu v množině entit se nazývají *klíčovým atributem* (*klíčem*) dané množiny entit.

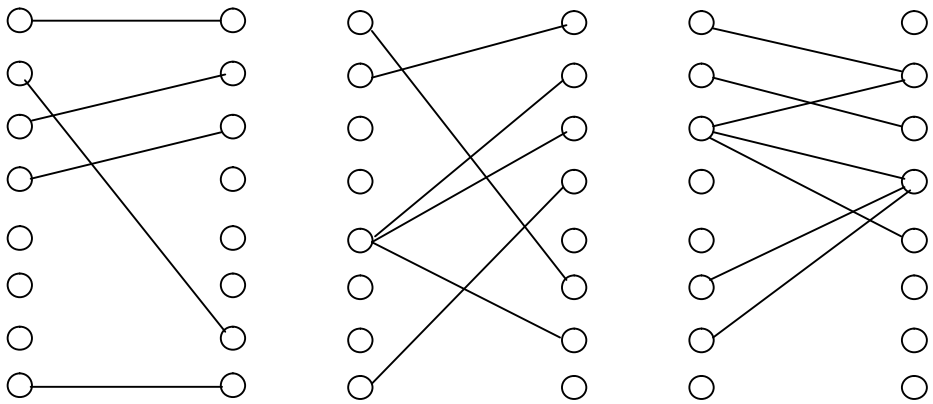
Příklad 1.3.2
Pro množinu zaměstnanců nějaké organizace může být klíčem např. rodné číslo nebo osobní číslo zaměstnance přidělené osobním oddělením.

Uvažujme uspořádaný seznam množin entit $E_1, E_2, \ldots, E_k$ a nechť k-tice entit $(e_1, e_2, \ldots, e_k)$, $e_i \in E_i$ je navzájem mezi sebou v nějakém vztahu v. Existence tohoto vztahu nás opravňuje považovat k-tici $(e_1, e_2, \ldots, e_k)$ za entitu a množinu V všech takových k-tic, které jsou navzájem mezi sebou ve stejném vztahu v, nazýváme *vztahem* V mezi množinami $E_1, E_2, \ldots, E_k$.

Nejčastější případ vztahů mezi množinami entit je pro $k = 2$. Tyto vztahy lze dále klasifikovat.

- *Vztah* 1:1 - každá entita jedné množiny je spojena vztahem s nejvýše jednou entitou druhé množiny

- *Vztah* 1: N mezi množinami E_1 a E_2 reprezentuje situace, kdy každá entita z E_1 je spojena vztahem s žádnou či více entitami z E_2, ale každá entita z E_2 je spojena vztahem s nejvýše jednou entitou z E_1.
- *Vztah* M : N je nejobecnější, nejsou kladena žádná omezení na množinu dvojic entit spojených příslušným vztahem.



Obr. 2. Schématické vyjádření vztahů mezi 2 množinami entit

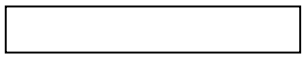
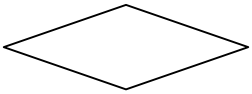
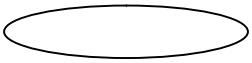
Příklad 1.3.3

1. Příkladem vztahu 1:1 je vztah manželství mezi muži a ženami v evropské části světa. Jeden muž může mít současně nejvýše jednu manželku a jedna žena nejvýše jednoho manžela.
2. V arabském světě jde v uvedeném příkladě o vztah 1: N. Jeden muž může současně mít více než jednu ženu, žena může mít nejvýše jednoho manžela.
3. Pokud bychom upustili v daném příkladě od požadavku současnosti, pak i v Evropě může jít o vztah M : N. Každý muž mohl být několikrát ženatý, stejně jako žena vícekrát vdaná.

Od počátku 70. let se objevují pokusy o zachycení *sémantiky* (*významu*) dat ukládaných v informačních systémech. Vznik prakticky použitelného sémantického modelu umožňujícího popsat konceptuální úroveň databáze je svázán s tzv. *Entity-Relationship modelem*, který se také zkráceně označuje jako *E-R model*. Jde o schématické znázornění množin entit, jejich atributů a vztahů mezi nimi grafickou formou. Používané grafické symboly ukazuje následující obrázek.

Postup při vytváření E-R modelu je následující:

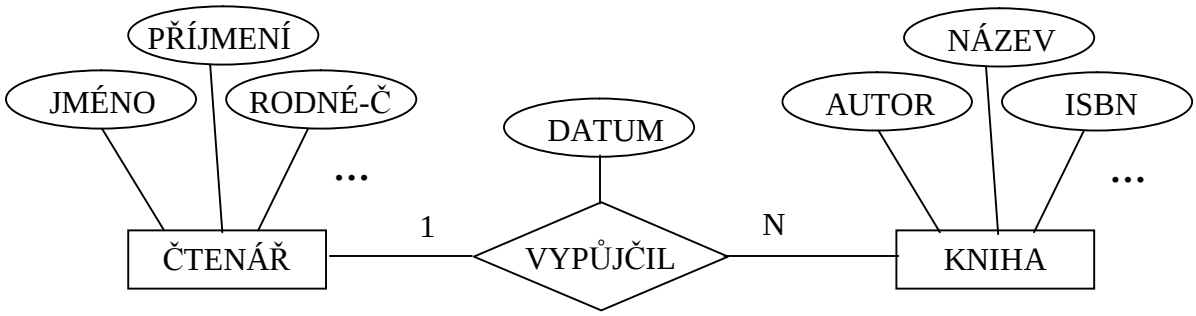
1. Určí se a pojmenují zobrazované objekty z reality a vztahy mezi nimi, které nás budou zajímat.
2. Rozhodne se o rozdělení objektů na entity a hodnoty.
3. Definičním oborům vztahů mezi entitami a atributům se podle potřeby přiřadí nová jména.
4. Stanoví se identifikátory entit.
5. U vztahů mezi entitami se určí jejich typ (1:1, 1:N, M:N).
6. Sestrojí se grafické zobrazení modelu.

Grafický symbol	Význam symbolu
	obdélník reprezentuje entity nebo množiny entit
	kosočtverec znázorňuje vztahy mezi entitami nebo množinami entit
	oválné uzly reprezentují atributy
	čáry jsou použity pro spojení symbolů
1	znak 1 je použit pro označení jednoduchého výskytu ve vztahu
M, N, …	M, N, atd. označují vícenásobný výskyt ve vztahu

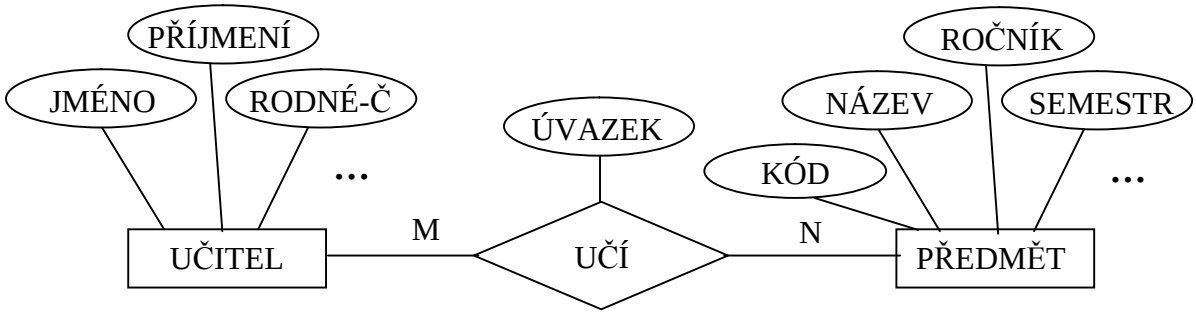
Tab. 1. Grafické symboly E-R modelů

Příklad 1.3.4

Na obr. … je vyjádřen vztah mezi čtenáři a knihami v knihovně. Tento vztah je typu 1 : N, protože každý čtenář mít vypůjčeno vice knih a naopak každá kniha může být v každém okamžiku vypůjčena nejvýše jedním čtenářem. Na obr. E-R diagramu jde o vztah M : N mezi učiteli a předměty, protože jeden učitel může učit více předmětů a přitom jeden předmět může být vyučován více učiteli. Vidíme, že vztah může mít svůj vlastní atribut, v našem případě úvazek učitele pro daný předmět.



Obr. 3. E-R model se vztahem 1 : N



Obr. 4. E-R model se vztahem M : N

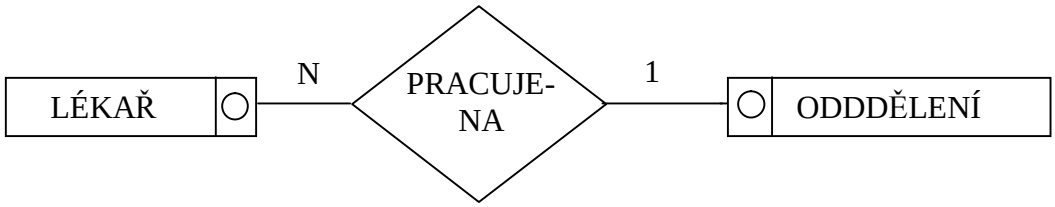
1.3.1 Integritní omezení pro vztahy

1. Kardinalita vztahu – 1:1, 1:N, M:N
2. Členství ve vztahu
3. Slabé entitní typy, cizí klíč
4. Min-max integritní omezení

Členství ve vztahu
- povinné (úplné)
- nepovinné (částečné)

Příklad 1.3.5

Lékaři v nemocnici

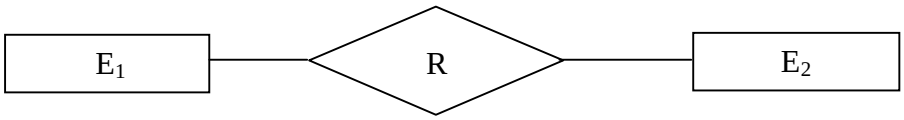


(Každý učitel učí alespoň 1 předmět, ne každý předmět musí být vyučován, např.volitelný předmět nemusí být otevřen)

(Sportovec nemusí vyhrát žádnou soutěž, každá soutěž má vítěze)

(Čtenář nemusí mít nic vypůjčeno, knihu si nikdo nevypůjčil)

Nechť zápis **R(E_1, E_2)** vyjadřuje diagram



Pak **E(min, max)** označuje minimální, resp. maximální počet výskytů entity E ve vztahu R.

Pro binární typy vztahů lze např. psát

R (E_1: (1,1), E_2 : (1,1))	odpovídá	1:1
R (E_1: (1,1), E_2 : (0,n))		1: N
R (E_1: (0,1), E_2 : (0,n))		(0 nebo 1) : N
R (E_1: (0,m), E_2 : (0,n))		M :N

Příklad: Popis režimu v knihovně

VÝPŮJČKA (ČTENÁŘ : (0,5), EXEMPLÁŘ : (0,1))
REZERVACE-KNIH (ČTENÁŘ : (0,4), KNIHA : (0,n))
EVIDENCE (KNIHA : (1,n), EXEMPLÁŘ : (1,1))
n – kniha může být v knihovně ve více kusech,
EVIDENCE (KNIHA : (0,n), EXEMPLÁŘ : (1,1))
0 – kniha nemusí být ve fondu knihovny, např. jde o ztracenou knihu nebo knihu získanou z jiné knihovny meziknihovní výpůjčkou

Závěrem se dá říci, že E-R modely mají pro návrh databáze podobný význam jako *vývojové diagramy* pro návrh algoritmu.

Po vytvoření E-R modelu se následuje další etapa, v níž se určí způsob reprezentace tohoto modelu v databázi. V první řadě jde o návrh logického schématu, tj. organizace dat na konceptuální úrovni. Pak se navrhne organizace dat na externí a interní úrovni.

Organizace dat se navrhují prostřednictvím *modelů dat* (*datových modelů*). V 60. letech byly vyvinuty tři nejznámější datové modely, které tvoří základ dnešních komerčních systémů řízení báze dat. Jsou to modely: *Relační, hierarchický* a *síťový*.

V relačním modelu jsou jednotným prostředkem pro zobrazení množin entit i vztahů mezi nimi tzv. *relační schémata* a *relace*.

V hierarchickém a síťovém modelu množinám entit odpovídají množiny záznamů, pro znázornění vztahů jsou speciální prostředky. V hierarchickém modelu se vztahy mezi množinami entit reprezentují *stromovými strukturami*, v nichž uzly reprezentují entity, hrany existenci vztahu mezi entitami.

V síťovém modelu lze vztahy budovat jako obecné *grafové struktury*. Hierarchický model je speciálním případem síťového modelu.

Nejvýznamější a současně nejrozšířenější datový model je model relační, a proto se jím nyní budeme podrobněji zabývat.

1.3.2 Relační model

Úvodem zavedeme ještě několik nových pojmů.

Definice 1.
Nechť je dán systém D_i, $1 \leq i \leq n$ neprázdných množin, tzv *domén*. Pak podmnožina kartézského součinu $R \subseteq D_1 \times D_2 \times \ldots \times D_n$ se nazývá *relace stupně n* (*n-ární relace*) nad doménami $D_1, D_2, \ldots, D_n$. Prvky relace R jsou uspořádané n-tice $(d_1, d_2, \ldots, d_n)$, kde $d_i \in D_i$, $1 \leq i \leq n$.

Z hlediska reálné existence databáze nemá smysl uvažovat nekonečné relace R, omezíme se proto pouze na konečné podmnožiny kartézského součinu.

Při tomto omezení si můžeme relaci jednoduše představit jako matici s m řádky a n sloupci. Pro lepší názornost sloupcům přiřadíme jména, která označují příslušné domény z oboru definice relace. Těmto jménům říkáme *atributy*. Pokud budeme chápat atributy jako funkce z n-tic relace do jednotlivých domén, dostaneme se k atributům ve smyslu úvodu kapitoly 3.

Je výhodné zobrazovat relace také v alternativní formě pomocí *tabulky* takto:
1. Každému prvku relace odpovídá jeden řádek tabulky, žádné dva řádky nejsou shodné (protože relace je množina a prvek množiny uvádíme jen jednou).
2. i-tý sloupec obsahuje pouze hodnoty z domény D_i.
3. Záhlaví sloupců reprezentují atributy.

Poznámka:
Prvky relace jsou uspořádané n-tice, vyměníme-li tedy pořadí domén, dostaneme relaci, která se od původní relace liší.

V tabulkové interpretaci však nezáleží na pořadí sloupců, pokud se permutují i se jmény v záhlaví. Z matematického hlediska tedy nejsou reprezentace pomocí relace a tabulky ekvivalentní. Tento nedostatek lze odstranit alternativní definicí relace, viz např. [4]. Protože však z pohledu reprezentace reálného světa na pořadí atributů nezáleží, nebudeme mezi oběmi interpretacemi rozlišovat.

Příklad 1.3.6

UČITELÉ

číslo_učitele	jméno	honorář/hod	funkce
U1	Starý	150	docent
U2	Novák	100	asistent
U3	Kříž	200	profesor
…	…	…	…

Tab. 2. Relace UČITELÉ

Domény relace UČITELÉ lze popsat tímto způsobem:

D_1 množina řetězců tvaru U_n, n přirozené číslo
D_2 množina možných jmen učitelů
D_3 množina nezáporných čísel menších než 100, představujících honorář v Kč za hodinu přednášky
D_4 množina alfabetických řetězců označujících funkční zařazení učitelů zaměstnání.

Strukturu relace budeme zapisovat takto: Uvedeme název relace a v závorce seznam jejích atributů.

Struktura relace z příkladu 1.3.6 je: UČITELÉ(číslo,jméno,honorář,funkce)

Zápis tohoto tvaru nazýváme *relační schéma*. Atributy, které pojmenovávají jednu doménu (sloupec) budeme nazývat *jednoduché atributy* jejich kombinace *složené atributy*.

Definice 2.
Klíč K relace R je podmnožina atributů relace *R* s těmito vlastnostmi, které jsou nezávislé na čase:
V1: *Jednoznačná identifikace.* Každá *n*-tice relace je hodnotami atributů tvořících *K* jednoznačně určena.
V2: *Neredundance.* Žádný atribut z *K* nelze vynechat, aniž by přestala platit vlastnost V1.

Schéma relace může obsahovat více klíčů. Ze všech možných klíčů se vybere jeden a označí se jako *primární klíč*. V zápisu relačního schématu budeme atributy tvořící primární klíč podtrhovat.

Je zřejmé, že v relaci existuje alespoň jeden klíč, protože kombinace všech klíčů má vlastnost V1. Vyplývá to z toho, že relace je množina a nemůže tedy obsahovat 2 stejné prvky.

Je nasnadě analogie mezi *relačním schématem* a *deklarací záznamu* (stanovení a pojmenování položek záznamů a jejich typu). Stejně tak existuje analogie mezi *relací* a *souborem* a mezi *prvkem relace* a *záznamem souboru*.

Tyto analogie ukazují jednu z možných implementací relace v paměti počítače, totiž implementovat relaci jako soubor záznamů s formátem daným relačním schématem.

Všechny 3 přístupy k popisu dat shrnuje tabulka

relace	tabulka	soubor
relační schéma	jméno a záhlaví tabulky	definice typu záznam a soubor
prvek relace	řádek tabulky	záznam (věta)

Tab. 3. Ekvivalentní termíny

Definice 3.
Databází (*bází dat*) nazýváme konečnou množinu v čase proměnných konečných relací, které jsou definovány nad doménami ze systému množin D_i, $1 \leq i \leq n$.

Aktualizace databáze, tj. její změna v čase, která umožňuje zachytit v databázi změny probíhající v reálném světě, spočívá ve změně aktuálních relací databáze - přidáváním, vynecháváním prvků relace nebo změnou hodnot některých komponent některých prvků relací.

Výsek reality popsaný v E-R modelu lze pomocí relačního modelu popsat tímto způsobem:

- Množina entit se reprezentuje pomocí relace, jejíž schéma obsahuje atributy těchto entit. Každý prvek relace reprezentuje jednu entitu dané množiny.
- Vztah mezi množinami entit $E_1, E_2, \ldots, E_k$ se reprezentuje relací R, jejíž schéma obsahuje klíčové atributy entit každé množiny $E_1, E_2, \ldots, E_k$.

V případě binárního vztahu typu M : N bude výsledný klíč této relace složen z dvojice (K_1,K_2), kde K_1, K_2 jsou klíčové atributy množin entit E_1, E_2, v případě vztahu 1: N bude výsledným klíčem přímo klíč K_2.

Případným přejmenováním atributů splníme podmínku jednoznačnosti jmen atributů relace R. Má-li vztah svoje vlastní atributy, přidají se do relačního schématu R.

Příklad 1.3.7
Uvažujme E-R model z příkladu 3.4. Pomocí relací jej vyjádříme takto:

UČITELÉ(ČU, jméno, honorář, funkce)
PŘEDMĚTY(ČP, název, ročník, hodin)
CO_UČÍ(ČU,ČP, úvazek)

V příkladu jsme vzhledem k relaci CO_UČÍ popisující vztah mezi učiteli a předměty a sdružující mimo jiné klíčové atributy relací UČITELÉ a PŘEDMĚTY museli nejednoznačné označení atributu číslo přejmenovat. Nová označení atributů jsou ČU (číslo učitele) a ČP (číslo předmětu). Tyto dva atributy tvoří složený klíč relace CO_UČÍ.

1.3.3 Relační algebra

Zavedením relací jako prostředku na modelování databáze se řeší jeho první - *definiční* - stránka. Relační model databáze dále tvoří *operace* s databázemi.

Nejznámějším nástrojem pro práci s relacemi je *relační algebra*.

(1) *Sjednocení* (*union*) relací R a S téhož stupně se označuje $R \cup S$ a je definováno

$$R \cup S = \{ t \mid t \in R \ \vee \ t \in S \}$$

(2) *Rozdíl* (*difference*) relací R a S se označuje $R - S$ a je definován

$$R - S = \{ t \mid t \in R \ \wedge \ t \notin S \}$$

Příklad 1.3.8

R

A	B
a	1
d	2
c	1

S

A	B
b	3
c	1

$\Rightarrow$

$R \cup S$

A	B
a	1
d	2
c	1
b	3

R - S

A	B
a	1
d	2

Tab. 4. Sjednocení a rozdíl relací

(3) *Kartézský součin* (*Cartesian product*) relace R stupně n a relace S stupně m se označuje $R \times S$ a je definován

$$R \times S = \{ rs \mid r \in R \ \wedge \ s \in S \}, \quad \text{kde } rs = (r_1, r_2, \ldots, r_n, s_1, s_2, \ldots, s_m)$$

Příklad 1.3.9

R

A	B	C
a	1	x
b	1	y

S

D	E
2	a
2	b

$\Rightarrow$

$R \times S$

A	B	C	D	E
a	1	x	2	a
a	1	x	2	b
b	1	y	2	a
b	1	y	2	b

Tab. 5. Kartézský součin

(4) *Projekce* (*projection*) relace R stupně n na atributy $A = (i_1, i_2, \ldots i_m)$, kde $1 \le i_j \le n$, $i_j \ne i_k$ pro $j \ne k$, je seznam indexů různých komponent relace R (sloupce tabulky nebo alternativně jména atributů), se označuje $R[A]$ a je definována

$$R[A] = \{ r[A] \mid r \in R \}, \text{ kde } r[A] = (r_{i1}, r_{i2}, \ldots, r_{im} \text{ pro } r \in R$$

(projekce slouží k výběru sloupců tabulky)

Příklad 1.3.10

R

A	B	C
a	1	x
b	2	x

$\Rightarrow$

$R[A]$

A
a
b

R[C]

C
x

R[A,C]

A	C
a	x
b	x

Tab. 6. Projekce

(5) *Selekce* (*selection*) nebo také *restrikce* (*restriction*).
Nechť R je relace, φ logická formule sestavená obvyklým způsobem s případným užitím logických operátorů $\wedge \vee \neg$. Pak selekce (výběr) z relace R podle φ se označuje $R[\varphi]$ a je definována

$$R[\varphi] = \{r \mid r \in R \ \wedge \ \varphi (r)\}$$

(selekce slouží k výběru řádků tabulky)

Příklad 1.3.11

R

A	B	C
1	a	2
3	b	2
2	a	2
1	c	1

$\Rightarrow$

R[A>2]

A	B	C
3	b	2

R[A≤C]

A	B	C
1	a	2
2	a	2
1	c	1

R[(A>C)∨ (B=”c”)]

A	B	C
3	b	2
1	c	1

Tab. 7. Selekce

(1) až (5) tvoří pět základních operací relační algebry, další uvedené operace se dají vyjádřit pomocí těchto pěti základních.

Jsou to tyto operace:

(6) *Průnik* (*intersection*) relací R a S se označuje $R \cap S$ a je definován

$$R \cap S = \{ t \mid t \in R \ \wedge \ t \in S \}$$

Příklad 1.3.12

R

A	B
a	1
d	2
c	1

S

A	B
b	3
c	1

$\Rightarrow$

$R \cap S$

A	B
c	1

Tab. 8. Průnik relací

(7) Θ-*spojení* (Θ-*join*)

Nechť R je relace stupně m, $i \in \{1,2, \ldots ,m\}$, S relace stupně n, $j \in \{1,2, \ldots ,n\}$,

$\Theta \in \{<, \leq, =, \geq, >, \neq\}$

Θ-*spojení* relací R a S podle Θ na i-té komponentě relace R a j-té komponentě relace S se označuje $R[i \ \Theta \ j]S$ a je definováno

$$R[i \ \Theta \ j]S = \{rs \mid r \in R \ \wedge \ s \in S \ \wedge \ r[i] \ \Theta \ s[j]\}$$

(Θ-spojení je tedy kartézský součin omezený podmínkou Θ, tj. do výsledného spojení jsou vybrány jen ty řádky, které splňují tuto podmínku)

Příklad 1.3.13

R

A	B	C
a	1	1
b	1	2
b	2	2

S

D	E
1	x
2	y
3	z

$\Rightarrow$

$R\ [B = D]\ S$

A	B	C	D	E
a	1	1	1	x
a	1	2	1	x
b	2	2	2	y

R [$C > D$] S

A	B	C	D	E
a	1	2	1	x
b	2	2	1	x

R [$C < D$] S

A	B	C	D	E
a	1	1	2	y
a	1	1	3	z
a	1	2	3	z
b	2	2	3	z

Tab. 9. Θ-spojení

Z příkladu je vidět, že pokud bude Θ podmínkou rovnosti, pak výsledné spojení bude obsahovat 2 shodné sloupce. Zavádí se proto operace:

$$R[i*j]S = (R[i = j]S)\ [1,2, \ldots ,m+j-1, m+j+1, \ldots,m+n]$$

která automaticky jeden ze shodných sloupců vypouští.

V našem příkladě tedy dostaneme $R[B*D]S$

(8) *Přirozené spojení* (*natural join*) relací R a S se označuje $R[*]S$.
Vzhledem k poměrně složitému formálnímu popisu uvedeme jeho definici pouze slovně. Přirozené spojení ze součinu $R \times S$ vybere ty záznamy, které mají na stejně pojmenovaných sloupcích stejné hodnoty, přitom stejně pojmenované sloupce se objeví ve výsledném spojení pouze jednou.

(9) *Dělení* (*division*).

Nechť R je relace stupně m a S relace stupně n.

Nechť $A = (i_1, i_2, \ldots ,i_k)$, $i_j \in \{1,2, \ldots ,m\}$, $j = 1, \ldots ,k$, $i_j \neq i_k$ pro $j \neq k$,
$B = (g_1, g_2, \ldots ,g_t)$, $g_j \in \{1,2, \ldots ,n\}$, $j = 1, \ldots ,t$, $g_j \neq g_k$ pro $j \neq k$,
jsou seznamy vybraných atributů relací R, resp. S (atributy zde pro jednoduchost označujeme přirozenými čísly).

Označme $\bar{A} = (j_1, j_2, \ldots , j_{m-k})$, kde $j_p \in \{1,2, \ldots , m\}-A$ pro $p = 1,2, \ldots , m-k$,
$j_p < j_k$ pro $p < k$ (tj. $\bar{A}$ jsou atributy z R, které nebyly vybrány do seznamu A).

Definujme pro $r \in R$ množinu $\mathrm{im}_R(r[\bar{A}])$, (*image set*), která obsahuje všechny doplňky k $r[\bar{A}]$, které v součinu s $r[\bar{A}]$ tvoří prvek relace R, tedy

$$\mathrm{im}_R(r[\bar{A}]) = \{y \mid \mathrm{r}[\bar{A}]\ y \in R[\bar{A}, A]\ \}.$$

Dělení relace R na A relací S na B se označuje $R[A : B]S$ a je definováno:

$$R[A : B]S = \{\mathrm{r}[\bar{A}] \mid r \in R \wedge (S[B] \subseteq \mathrm{im}_R(r[\bar{A}])\}.$$

Lze dokázat, že platí

$$R[A : B]S = R[\bar{A}] - ((R[\bar{A}] \times S[B]) - R[\bar{A}, A]\ )\ [1,2, \ldots ,m–k]$$

Příklad 1.3.14 Image set

R

D_1	D_2	D_3	D_4	D_5
1	a	x	f	2
2	a	y	g	3
1	b	x	f	2
2	c	y	h	3
3	a	x	f	1
1	b	y	f	2
2	a	x	h	3

Nechť $A = \{D_3, D_2, D_4\}$, pak $\bar{A} = \{D_1, D_5\}$. Je-li $r = (1, a, x, f, 2)$, pak $r[A] = (x, a, f)$, $r[\bar{A}] = (1,2)$, a tedy $\text{im}_R(r[\bar{A}]) = \text{im}_R(1,2) = \{(x, a, f), (x, b, f), (y, b, f)\}$.

Příklad 1.3.15

Úkolem je pro údaje zadané v tabulce 10 zjistit příjmení a jména všech kupujících, kteří kupují všechny typy výrobků.

Jestliže položíme $A = \{$příjmení, jméno$\}$, $B = \{$kód_zboží$\}$, pak řešení získáme operací dělení

KUPUJÍCÍ [$\bar{A}$: B] VÝROBEK, tzn. operací dělení
KUPUJÍCÍ [{kód_zboží}:{kód_zboží}] VÝROBEK.

KUPUJÍCÍ

příjmení	jméno	kód-zboží
Hora	Jan	X
Kos	Petr	Y
Novák	Ivo	X
Hora	Jan	Y
Kos	Petr	X
Hora	Jan	Z

VÝROBEK

kód-zboží	výrobní-cena	prodejní-cena
X	5	8
Y	4	4
Z	6	9

KUPUJÍCÍ [kód-zboží : kód-zboží]VÝROBEK

příjmení
Hora

Tab. 10. Příklad operace dělení

Ověření:

R= KUPUJÍCÍ, A={příjmení,jméno},
$\bar{A}$={kód-zboží}, B={kód-zboží}

R[$\bar{A}$] ={ KUPUJÍCÍ}[příjmení] ={Hora,Kos,Novák}

im_R (r[$\bar{A}$]):
im_R (Hora,Jan) ={X,Y,Z}
im_R (Kos,Petr) ={X,Y}
im_R (Novák,Ivo) ={X}

S[B]={VÝROBEK}[kód-zboží]={X,Y,Z},
podmínce $S[B] \subseteq im_R(r[\bar{A}])$ vyhovuje pouze Hora.

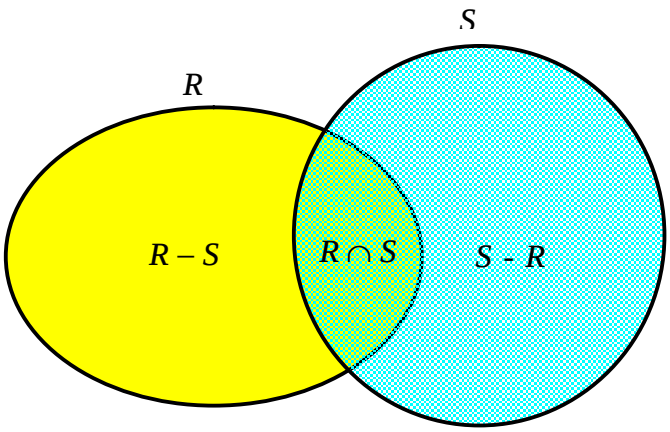
Poznámka:

- *Relační dělení* je spíše dělení ve smyslu *rozdělování* (vyber ty, kteří se ti hodí), než dělení v aritmetickém smyslu. Relační dělení vyjadřuje *univerzální kvantifikátor* $\forall$.
- *Spojení* obsahuje podmínku *pro každý* nebo podmínku typu *existuje*, a tedy může vyjadřovat *existenční kvantifikátor* $\exists$.

Poznámka:
Relace v relačním modelu uvažujeme výhradně konečné, to samozřejmě tedy platí i o operandech operací relační algebry. Rovněž relace, které jsou výsledkem uvedených operací jsou konečné. Proto do relační algebry nezahrnujeme např. operaci *doplňku* $\sim R = \{r \mid r \notin R\}$.

Z definic operací je vidět, že některé operace lze vyjádřit pomocí jiných operací.

i) Θ-spojení lze definovat pomocí kartézského součinu a selekce,
ii) přirozené spojení pomocí kartézského součinu, selekce a projekce,
iii) průnik lze vyjádřit několika způsoby pomocí rozdílu a případně sjednocení:
   * $R \cap S = R \cup S - (R - S) - (S - R)$,
   * $R \cap S = R - (R - S)$,
   * $R \cap S = S - (S - R)$.



Odtud vyplývá, že *minimální množina operací relační algebry* je určena např. takto: {sjednocení, kartézský součin, rozdíl, selekce, projekce}.

1.3.4 Relační algebra jako dotazovací jazyk

Příklad 1.3.16

Uvažujme relace

UCHAZEČI(rodné-č, jméno, příjmení, místo, ulice, čp, PSČ)
JAZYKY(kód, název)
ZNALOSTI(rodné-č, kód, stupeň-znalosti).

Stupeň znalosti budeme klasifikovat jako "začátečník", "pokročilý" a "velmi pokročilý".

Máme určit odpovědi na tyto dotazy:

(D1) Zjistit základní údaje o všech uchazečích.
Q : = UCHAZEČI

(D2) Rodná čísla, jména a příjmení všech uchazečů z Brna.
Q : = (UCHAZEČI[místo="Brno"]) [rodné-č, jméno, příjmení]

(D3) Jména a příjmení uchazečů, kteří jsou velmi pokročilí v angličtině.
Q : = {UCHAZEČI [∗] (ZNALOSTI [kód="Eng" AND stupeň-znalosti="velmi pokročilý"])}[jméno,příjmení]

(D4) Jazyky, v nichž je uchazeč zadaného rodného čísla přinejmenším pokročilý.
Q : = { (ZNALOSTI [rodné-č=zadané-rodné-číslo AND
(stupeň-znalosti="pokročilý" OR stupeň-znalosti="velmi pokročilý")])
[∗] JAZYKY}[název]

(D5) Uchazeči, kteří jsou velmi pokročilí v ***alespoň jednom*** jazyku a v kterém.
Q : = {UCHAZEČI [∗] (ZNALOSTI [stupeň-znalosti="velmipokročilý"]) }
[jméno,příjmení,kód]

(D6) Uchazeči, kteří neovládají ***žádný*** (cizí) jazyk.

neznají žádný = všichni – ti, co znají alespoň 1 jazyk

Q : = (UCHAZEČI [rodné-č] – ZNALOSTI [rodné-č] [∗]
(UCHAZEČI [rodné-č, jméno, příjmení])

(D7) Uchazeči, kteří ovládají ***pouze*** angličtinu.

znají pouze angličtinu = ti, co znají angličtinu – ti, co znají jiný jazyk

Q : = { (ZNALOSTI[kód="Eng"]) [rodné-č] – (ZNALOSTI [kód<>"Eng"])
[rodné-č] }
[∗] (UCHAZEČI[rodné-č, jméno, příjmení])

(D8) Uchazeči, kteří mají alespoň základní vlastnosti ***všech*** jazyků.

*Q*3 ti, co znají všechno = ti, co něco znají – ti, co něco neznají
Q2: ti, co něco neznají = všichni znají vše – ti, co něco znají
*Q*1: všichni znají vše = kartézský součin UCHAZEČI [rodné-č] × JAZYKY[kód]

*Q*1 : = UCHAZEČI [rodné-č] × JAZYKY[kód]
*Q*2 : = *Q*1 – ZNALOSTI [rodné-č, kód]
*Q*3 : = *Q*1 [rodné-č] – *Q*2 [rodné-č]
Q : = *Q*3 [∗] (UCHAZEČI [rodné-č, jméno, příjmení])

Totéž pomocí operace dělení

$Q1$: = ZNALOSTI [{A_1, A_2, … }−{ rodné-č } : kód] JAZYKY
Q : = $Q1$ [∗] (UCHAZEČI [rodné-č, jméno, příjmení])

1.3.5 Návrh struktury relační databáze, normalizace

V dalším omezíme třídu relací, které budeme používat k zobrazení výseku reálného světa v databázi.

Příklad 1.3.17
Předpokládejme, že chceme sledovat údaje organizace, a to z jakých oddělení se skládá, kdo je na odděleních vedoucím a kteří pracovníci jsou na jednotlivá oddělení zařazeni. Pro jednoduchost budeme pracovníky označovat pouze příjmením a předpokládáme, že tato příjmení jsou jednoznačná v rámci celé organizace.

Tyto údaje bychom tedy mohli vyjádřit relací ZAMĚSTNANCI(oddělení,vedoucí,pracovníci)

ZAMĚSTNANCI

oddělení	vedoucí	pracovníci
Konstrukce	Novák	Novotný, Navrátil, Sedláček, Janíček, ...
Projekce	Růžička	Karas, Kos, Sova, ...
Účtárna	Konečná	Nová, Stará, Hrubá, ...
…	…	…

Tab. 11. Relace ZAMĚSTNANCI v 0. normální formě

Z tabulky 11 je vidět, že organizace i správa dat je velmi neefektivní:
- Šířka pole pro pracovníky se musí nastavit podle oddělení s nejvyšším počtem pracovníků (resp. s nejvyšším počtem znaků pro jejich jména), a na odděleních s malým počtem pracovníků bude pole jen málo využito.
- Je obtížné zjistit, kolik zaměstnanců mají jednotlivé oddělení.
- Modifikace údajů o pracovnících je složitá.
- Řešení přestává být zvládnutelné, jestliže bychom chtěli o pracovnících sledovat více údajů, např. také křestní jméno, rodné číslo, místo, ulici, číslo popisné a PSČ jejich bydliště apod.

Všechny zmíněné problémy byly způsobeny tím, že relace `ZAMĚSTNANCI` ve sloupci `pracovníci` neobsahuje "jednoduché hodnoty", ale seznamy údajů. Proto se v relačním modelu databáze uvažují pouze relace v 1. *normální formě* (1NF) (*first normal form*), tj. relace, jejichž domény jsou množinami v jistém smyslu jednoduchých hodnot, např. množiny čísel, znaků, slov v nějaké abecedě ap. Prvkem tabulky nesmí být pole ani záznam, hodnoty atributů musí být atomické. Relaci, která není v 1NF (nenormalizovanou relaci), lze za předpokladu, že klíč relace neobsahuje atributy, jejichž domény jsou množinami relací, zpravidla nahradit jednou nebo několika relacemi v 1NF se stejným informačním obsahem.

Nenormalizovanou relaci z příkladu můžeme nahradit jednou relací v 1NF, jak je vidět z tabulky zaměstnanců
.

ZAMĚSTNANCI

oddělení	vedoucí	pracovník
Projekce	Růžička	Karas
Projekce	Růžička	Kos
Projekce	Růžička	Sova
…	…	…
Konstrukce	Novák	Novotný
Konstrukce	Novák	Navrátil
Konstrukce	Novák	Sedláček
Konstrukce	Novák	Janíček
…	…	…
Účtárna	Konečná	Nová
Účtárna	Konečná	Stará
Účtárna	Konečná	Hrubá
…	…	…
…	…	…

Tab. 12. Relace ZAMĚSTNANCI v 1. normální formě

Je vidět, že i toto řešení je neefektivní. Jeho nedostatky jsou tyto:

1. *Redundance* (*redundancy*), (*nadbytečnost*).
   Informace o tom, jak se oddělení jmenuje a kdo je jeho vedoucím, se v tabulce vyskytuje tolikrát, kolik má oddělení pracovníků.
2. *Nebezpečí nekonzistence* (*risk of inconsistency*) *při aktualizaci*.
   Z existence redundance vyplývá i složitější oprava údajů. Je-li např. jmenován nový vedoucí projekce nebo vedoucí účtárny změní jméno, případně některé oddělení změní název, musí se tento údaj opravit u všech pracovníků příslušného oddělení. Pokud se to provede nedůsledně, pak se data stanou nekonzistentní (totéž oddělení má dva vedoucí, resp. dva různé názvy).
3. *Anomálie vložení* (*insertion anomaly*).
   Do tabulky nelze vložit informaci o nově zřízeném oddělení, pokud ještě nemá žádné pracovníky.
4. *Anomálie zrušení* (*deletion anomaly*).
   Pokud všichni pracovníci některého oddělení odejdou, pak při jejich zrušení v tabulce zrušíme i informaci o vlastním oddělení, což není žádoucí.

Z uvedeného lze již intuitivně odvodit, že obvykle nevystačíme s jedinou tabulkou a je nutné původní tabulku rozložit na několik tabulek v 1NF, které ovšem musí obsahovat spojující údaje, aby nedošlo ke ztrátě informačního obsahu.

V našem příkladě můžeme rozdělit výchozí relaci `ZAMĚSTNANCI` na dvě relace `ZAM1(oddělení,vedoucí)` a `ZAM2(oddělení,pracovník)`.

Funkční závislosti

Definice 4
Nechť A a B jsou atributy relace, $D(A)$ je doména atributu A a $D(B)$ je doména atributu B a nechť f je funkce měnící se v čase taková, že

$$f: D(A) \rightarrow D(B).$$

[V matematickém smyslu není funkce, protože je dovoleno, aby se v čase relace v databázi měnily].

Jestliže tedy f znamená množinu uspořádaných dvojic $\{ (a, b) \mid a \in D(A), b \in D(B) \}$, pak v každém bodě časové osy existuje nejvýše jedna hodnota b z domény $D(b)$.

Abychom rozlišili f od funkce v matematickém smyslu, budeme ji nazývat *funkční závislost* (*functional dependency*).

Kvůli stručnosti budeme místo $f: D(A) \rightarrow D(B)$ psát $f: A \rightarrow B$.

Existuje-li funkční závislost $f: A \rightarrow B$, pak řekneme, že doména B je *funkčně závislá* nebo stručně *závislá* na doméně A a řekneme, že A *funkčně determinuje* nebo stručně *determinuje* doménu B.

Existuje-li pouze jediná funkční závislost z domény A do domény B, budeme stručně psát $A \rightarrow B$, tj. A funkčně determinuje B.

Označení $A \nrightarrow B$ znamená, že neexistuje žádná funkční závislost z A do B.

Platí-li $A \rightarrow B$ a současně $B \rightarrow A$, pak v každém okamžiku existuje mezi A a B jednoznačná korespondence (tj. korespondence 1:1), což budeme zapisovat $A \Leftrightarrow B$.

Definice 5
Jsou-li X, Y podmnožiny množiny atributů A, pak řekneme, že Y *funkčně závisí* na X (nebo X *funkčně determinuje* Y), píšeme $X \rightarrow Y$, když pro každou možnou aktuální relaci R platí, že mají-li libovolné dva prvky relace R stejné hodnoty atributů (množiny atributů) X, pak mají i stejné hodnoty atributů (množiny atributů) Y. Formálně zapsáno tedy

$$X \rightarrow Y \text{ na } R(A) \Leftrightarrow \forall r1, r2 \in R, (r1.X = r2.X) \Rightarrow (r1.Y = r2.Y) \qquad (1)$$

Poznámka:
Zde je důležité si uvědomit, že funkční závislost je definována na základě vlastností všech možných aktuálních relací (jinými slovy pro všechny populace relace), a není tedy možné soudit na funkční závislost z vlastností jediné (třeba právě nyní) aktuální relace. Naopak je z jediné aktuální relace možné ukázat neplatnost funkční závislosti mezi některými atributy (resp. množinami atributů).

Definice 6
Jestliže je dána množina funkčních závislostí F, pak řekneme, že F *logicky implikuje* závislost $X \rightarrow Y$ (resp. $X \rightarrow Y$ je funkční závislost (*logicky*) *odvoditelná* z F), jestliže je splněna ve všech relacích, v nichž jsou splněny závislosti z F. Množina všech funkčních závislostí odvoditelných z F se nazývá *uzávěr* (*closure*) množiny F, značíme jej F^+.

Vzhledem k nekonečnému počtu takových relací prakticky nelze F^+ určovat podle definice. Snahou tedy je k pojmu *logické implikace* (*logical implication*) najít ekvivalentní nástroj pro

výpočet F^+, resp. pro určení, zda daná závislost $X \to Y$ patří do F^+ či nikoliv. V klasické teorii relačních databází k tomuto účelu existují tři *odvozovací pravidla* (*inference rules*) známá jako *Armstrongovy axiomy* (*Armstrong's axioms*). Tato pravidla splňují požadované vlastnosti axiomatického systému. Tj. jsou:

(i) *bezesporná* (nebo také *korektní*) (*sound*), to znamená, že pomocí nich lze z F odvodit pouze závislosti patřící do F^+;
(ii) *úplná* (*complete*), dovolují odvodit z F všechny závislosti z F^+ a
(iii) *nezávislá* (*independent*), tj. odstraněním kteréhokoliv axiomu porušíme platnost úplnosti.

Nechť X, Y, Z jsou podmnožiny množiny atributů A. Pak platí:

A1: *Pravidlo inkluze* (*Inclusion rule*).
Jestliže $Y \subseteq X$, pak $X \to Y$.

A2: *Pravidlo zvětšení* (*Augmentation rule*).
Jestliže $X \to Y$, then $XZ \to YZ$ (XZ zastupuje $X \cup Z$).

A3: *Pravidlo tranzitivity* (*Transitivity rule*).
Jestliže $X \to Y$ a $Y \to Z$, pak $X \to Z$.

Aplikací Armstrongových axiomů lze určit další užitečná pravidla pro odvozování funkčních závislostí (W v posledním z nich označuje podmnožinu množiny atributů A):

P1: *Pravidlo kompozice* (*Union rule*).
Jestliže $X \to Y$ a $X \to Z$, pak $X \to YZ$.

P2: *Pravidlo dekompozice* (*Decomposition rule*).
Jestliže $X \to YZ$, pak $X \to Y$ a $X \to Z$.

P3: *Pravidlo pseudo-tranzitivity* (*Pseudo-transitivity rule*).
If $X \to Y$ and $YW \to Z$, then $XW \to Z$.

Důkaz [P1] Z prvního předpokladu $X \to Y$ pravidla P1 dostaneme podle A2, že $XZ \to YZ$. Analogicky z druhého předpokladu $X \to Z$ pravidla P1 dostaneme podle A2, že $XX \to XZ$, a tedy $X \to XZ$. Konečně z $X \to XZ$ a $XZ \to YZ$ dostaneme podle A3, že $X \to YZ$.

[P2] Z pravidla A1 plyne platnost $YZ \to Y$ a $YZ \to Z$. Odtud a z předpokladu $X \to YZ$ pravidla P2, aplikujeme-li pravidlo A3, přímo vyplývá existence funkčních závislostí $X \to Y$ a $X \to Z$.

[P3] Z prvního předpokladu $X \to Y$ pravidla P3 dostaneme aplikací pravidla A2 platnost $XW \to YW$, což společně s dalším předpokladem $YW \to Z$ pravidla P3 implikuje podle pravidla A3 existenci funkční závislosti $XW \to Z$, což jsme chtěli dokázat.

S pojmem funkční závislosti se úzce váže pojem klíče relačního schématu.

Definice 7
Je-li dáno relační schéma $R(A)$ a $K \subseteq A$, pak K je *klíčem* (*key*) relačního schématu $R(A)$, jestliže splňuje následující vlastnosti:
(V1) $K \to A$.
(V2) Neexistuje $K' \subseteq K$ tak, že $K' \to A$.

První vlastnost vyjadřuje jednoznačnost, druhá neredundanci. S využitím pojmu uzávěr lze klíč alternativně definovat takto:

Je-li dáno relační schéma $R(A)$ a $K \subseteq A$, pak K je *klíčem* relačního schématu $R(A)$, jestliže splňuje následující vlastnosti:

(V1') $K \rightarrow A \in F^+$.
(V2') Pro každé $K' \subseteq K$ platí $K' \rightarrow A \notin F^+$.

Speciální funkční závislosti, normální formy relací

V metodice návrhu datových struktur hrají důležitou roli pojmy *úplné* a *částečné funkční závislost* (*full, partial functional dependence*).

Definice 8

Nechť $R(A)$ je relační schéma, X, Y jsou podmnožiny množiny atributů A. Řekneme, že:

- *Y úplně funkčně závisí* na X, $X \rightarrow Y$, jestliže neexistuje X', $X' \subseteq X$, $X' \neq \emptyset$ tak, že $X' \rightarrow Y$.
- *Y částečně funkčně závisí* na X, $X \rightarrow Y$, jestliže existuje X', $X' \subset X$, $X' \neq \emptyset$ tak, že $X' \rightarrow Y$.

S využitím těchto pojmů lze pak vlastnost (V2') v druhé variantě definice klíče vyjádřit takto:

(V2") A úplně funkčně závisí na K.

Existence částečných funkčních závislostí v relačním schématu způsobuje výše zmíněné problémy: *nebezpečí nekonsistence* při modifikaci a *aktualizační anomálie* (*update anomalies*) - *anomálie vložení* a *anomálie zrušení*. Příčinou prvního z nich je *redundance* dat.

Příklad 1.3.18
Jestliže např. v tabulce odpovídající relaci z předchozího příkladu je více řádků se stejnou hodnotou atributu A_1, pak při její změně (reprezentující např. změnu jména, adresy apod.) je třeba ji promítnout do všech příslušných řádků, jinak se informace stanou nekonsistentními (tatáž osoba se pak může vyskytovat v tabulce se dvěma různými jmény, atd.).

Anomálie vložení nastává tehdy, když nemůžeme do tabulky vložit jinak užitečnou informaci, pokud k ní není určena hodnota klíče.

Příklad 1.3.19
Pokud například v jediné tabulce společně zpracováváme údaje doktorandů (rodné číslo, jméno, příjmení) s uvedením názvu předmětu, který si vybrali ze skupiny volitelných předmětů do svého studijního plánu, pak nelze vložit informaci o předmětu, jestliže si jej nikdo nevybral.

Naopak, jestliže jediný student si vybral určitý předmět a pak byl z tabulky vyřazen (např. ukončil studium), zruší se s údaji studenta i informace o předmětu. Tento nežádoucí jev se označuje *anomálie zrušení*. Vzhledem k tomu, že studenti volí větší počet předmětů není klíčem pouze rodné číslo ani pouze kód předmětu, protože ten si mohlo zapsat více studentů, ale oba atributy (rodné číslo, kód předmětu), přičemž informace o termínu složení zkoušky

bude záviset na celém klíči, údaje vážící se k studentovi závisí pouze na rodném čísle studenta (tj. na části klíče) a údaje o předmětech pouze na kódu předmětu.

Proti uvedeným problémům se br níme rozkladem výchozího schématu na více schémat, která neobsahují částečné funkční závislosti.
Tj. atributy, které úplně závisí na celém klíči umístíme se všemi složkami klíče do jedné schématu a do dalšího (dalších) vložíme atributy s částí klíče, na níž závisí.

V uvedeném příkladu tedy původní schéma rozložíme na 3 schémata:
STUDENTI(rodné-číslo}, jméno, příjmení),
PŘEDMĚTY(kód-předmětu, název),
ZÁPIS-PŘEDMĚTU(rodné-číslo, kód-předmětu, termín zkoušky).

Podtržením jsou označeny klíčové atributy. V konkrétním příkladu byly všechny uvedené problémy odstraněny. V obecném případě tomu však tak být nemusí. V některém z rozložených schémat se mohou ještě vyskytovat tranzitivní funkční závislosti, které způsobují tytéž problémy jako částečné funkční závislosti. Jejich definice je následující:

Definice 9
Nechť X, Y jsou podmnožiny množiny atributů A, C je jednoduchý atribut, který se nevyskytuje v X ani v Y. Nechť dále platí $X \rightarrow Y$, $Y \rightarrow C$, a neplatí $Y \rightarrow X$. Pak řekneme, že C *tranzitivně funkčně závisí* na X.

Tranzitivní závislost odstraníme další dekompozicí na dvě relační schémata. Závislost $X \rightarrow Y \rightarrow C$ odstraníme vytvořením dvou projekcí, z nichž jedna obsahuje atributy X, Y a druh atributy Y, C.

Příklad 1.3.20
NEMOCNICE(pacient-ID, jméno-pacienta, příjmení-pacienta,
zaměstnavatel, adresa-zaměstnavatele, datum-přijetí, čas-přijetí, přijímající-lékař,
oddělení, diagnóza)

klíč= (pacient-ID, datum-přijetí, čas-přijetí)

funkční závislosti:

pacient-ID → jméno-pacienta,
pacient-ID → příjmení-pacienta,
pacient-ID → zaměstnavatel → adresa-zaměstnavatele,
(pacient-ID, datum-přijetí, čas-přijetí) → přijímající-lékař,
(pacient-ID, datum-přijetí, čas-přijetí) → oddělení,
(pacient-ID, datum-přijetí, čas-přijetí) → diagnóza.

Dekompozice:

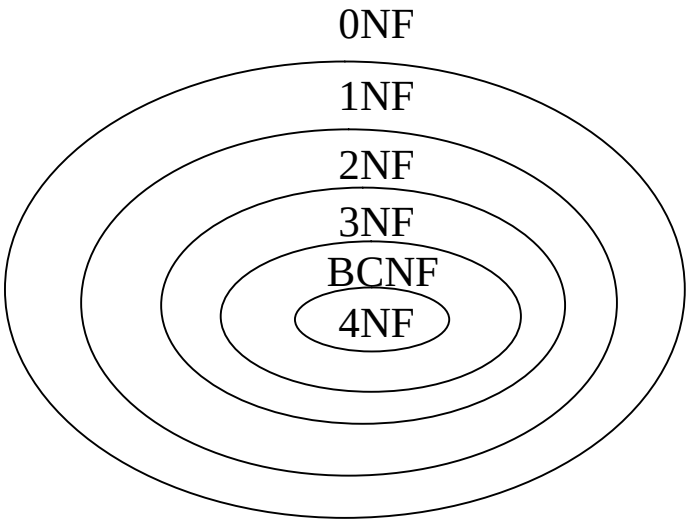
PŘIJETÍ(pacient-ID, datum-přijetí, čas-přijetí, přijímající-lékař, diagnóza)
PACIENT(pacient-ID, jméno-pacienta, příjmení-pacienta)
LÉKAŘ(přijímající-lékař, oddělení)
ZAMĚSTNAVATELÉ-PACIENTŮ(zaměstnavatel, adresa-zaměstnavatele)
ZAMĚSTNÁN(pacient-ID, zaměstnavatel)

Vlastnosti relačních schémat jsou klasifikovány jako *normální formy* (*normal forms*). 1. normální forma (1NF) byla již zmíněna v úvodu.

Definice 10
Řekneme, že relační schéma je v:

- 2. *normální formě* (2NF), jestliže je v 1NF a neobsahuje žádné částečné funkční závislosti neklíčových atributů na klíči;
- 3. *normální formě* (3NF), jestliže je v 2NF a žádný neklíčový atribut není tranzitivně závislý na žádném klíči;
- *Boyce-Coddově normální formě* (BCNF), když je v 3NF a neobsahuje ani tranzitivní závislost klíčových atributů (jednoho klíče) na (jiném) klíči.

Je definována i 4. normální forma (4NF), která se odkazuje na další typ funkční závislost, tzv. *multizávislost*. Vztah mezi relacemi z hlediska normální formy ukazuje následující obrázek.



Definice11
Atribut Y v $R(A)$ *multizávisí* na atributu X, jestliže platí, že každá hodnota atributu X určuje množinu hodnot atributu Y a tato množina přitom nezávisí na hodnotě jiných atributů v $R(A)$. Píšeme $X \rightarrow\rightarrow Y$. Formálně:

$X \rightarrow\rightarrow Y \quad \hat{U}_{\mathrm{df}}$

$\forall\, r_1, r_2 \in R\ (r_1.X = r_2.X \Rightarrow \exists\, v_1, v_2 \in R$, kde

(a) $v_1.X = v_2.X = r_1.X = r_2.X$

(b) $(v_1.Y = r_1.Y) \wedge (v_1.\ \overline{XY} = r_2.\ \overline{XY}\ )$

(c) $(v_2.Y = r_2.Y) \wedge (v_2.\ \overline{XY} = r_1.\ \overline{XY}\ ))$

To znamená, že zaměníme-li hodnoty atributu Y v n-ticích r_1, r_2, pak tyto modifikované n-tice opět musí být v instanci relace R.

Příklad 1.3.21
ROZVRH(učitel, předmět, učebna)

Učitel může učit více předmětů, každý předmět může být vyučován ve více učebnách. Dané relační schéma obsahuje dvě multizávislosti:

učitel $\rightarrow\!\!\rightarrow$ předmět a předmět $\rightarrow\!\!\rightarrow$ učebna. Tato dvojice multizávislostí způsobuje následující problémy:

Jestliže některý všeobecný předmět může být vyučován např. v 10 učebnách a přibude nový učitel vyučující tento předmět, pak je nutné přidat 10 nových záznamů.

Přibude-li nová učebna pro výuku určitého předmětu, pak je nutné přidat tolik řádků, kolik učitelů učí tento předmět.

Uvedené problémy odstraníme rozkladem výchozího relačního schématu na dvě schémata:

R1(učitel, učebna)
R2(předmět, učebna)

Definice 12
Relace R je v 4. *normální formě* (4NF), jestliže v případě, že obsahuje multizávislost $X \rightarrow\!\!\rightarrow Y$, kde not($Y \subseteq X$) a XY nezahrnuje všechny atributy A, pak X obsahuje i klíč relace A.

Shrneme-li tento odstavec, pak při návrhu datových struktur rozkladem relačního schématu na několik relačních schémat postupně odstraňujeme všechny nežádoucí funkční závislosti, které způsobují problémy s nekonsistencí, resp. aktualizační anomálie.
Tento postup se označuje jako *normalizace* a cílem je, aby všechna výsledná relační schémata byla v 4NF nebo alespoň minimálně v 3NF. Při rozkladu je třeba dbát na to, aby byl *bezeztrátový* (*lossless*) a zachovával vazby mezi souvisejícími údaji. To znamená, abychom operací *spojení* (*join*) přes společné atributy rozložených schémat byli schopni rekonstruovat původní relační schéma.

Definice 13
Nechť $R(A)$ je relační schéma a $\rho = \{R(A_1), R_2(A_2)\}$ je jeho rozklad a F je množina funkčních závislostí. Řekneme, že při rozkladu *nedojde ke ztrátě informace vzhledem k F*, jestliže pro každou relaci $R(A)$ splňující F platí:
$R = R_1(A_1)\ [*]\ R_2(A_2)$.

Věta 1 (*dekompoziční teorém*)
Jestliže v $R(A_1, A_2, A_3)$ platí funkční závislost $A_1 \rightarrow A_2$, pak R můžeme *bez ztráty informace* rozložit do projekcí $R_1(A_1, A_2)$ a $R_2(A_1, A_3)$.

Věta 2
Nechť $\rho = \{R(A_1), R_2(A_2)\}$ je rozklad relačního schematu $R(A)$ a F je množina funkčních závislostí. Pak při rozkladu ρ nedochází ke ztrátě informace vzhledem k F právě tehdy, když: $(A_1 \cap A_2) \rightarrow A_1 - A_2$ nebo $(A_1 \cap A_2) \rightarrow A_2 - A_1$.

Příklad 1.3.22
UČITELÉ(jméno, ústav, předmět, úvazek)
F = {jméno $\rightarrow$ ústav, (jméno, předmět) $\rightarrow$ úvazek}

ρ = { UČ(jméno, ústav), UČ-PŘ(jméno, předmět, úvazek) }

{jméno, ústav}$\cap${jméno, předmět, úvazek}= {jméno}

{jméno, ústav} – {jméno, předmět, úvazek}= {ústav}

Odtud tedy podle předchozí věty je ověřeno, že tento rozklad je bezeztrátový.

Rozpoznání funkčních závislostí tvoří základ porozumění významu (sémantiky) dat a jejich odstranění vede k efektivnímu návrhu databáze.

2. Dotazovací jazyk SQL

SQL (Structured Query Language)

2.1. Výběrový dotaz

SELECT [**ALL** | **DISTINCT** | **DISTINCTROW** | [**TOP** n [**PERCENT**]]]
{* | *tabulka*.* | [*tabulka*.] *položka*1 [**AS** *alias*1]
[,*tabulka*.] *položka*2 [**AS** *alias*2]
[,...] }
FROM {[1]*tabulka*1 [**AS** *alias*1*t*] |
[2]*výběrový-dotaz*1 [**AS** *alias*1*d*] |
[3]=1|2} [**LEFT** | **RIGHT** | **INNER**] **JOIN**
*tabulka*2 [**AS** *alias*2*t*] **ON** *spojovací-podmínka*
[**IN** *externí-databáze*] }
[,{...}]
[**WHERE** *vyhledávací-podmínka*}]
[**GROUP BY** *klíč-agregace*]
[**HAVING** *vyhledávací-podmínka-skupin*]
[**ORDER BY** {*název-sloupce* | *číslo-sloupce* [**ASC** | **DESC**] }
[,{...}]
[**WITH OWNERACCESS OPTION**]

spojovací-podmínka

- *tabulka*1.*položka*1{= |<> | < | <= | > | >= } *tabulka*2.*položka*2

vyhledávací-podmínka

- *výraz*1 {= |<> | < | <= | > | >= } *výraz*2
- *výraz* [**NOT**] **BETWEEN** *dolní* **AND** *horní*
- *výraz* [**NOT**] **IN** *množina-hodnot*
- *výraz* [**NOT**] **LIKE** *vzorový-řetězec*
- *výraz* {= |<> | < | <= | > | >= } **ALL** (*poddotaz*)
- *výraz* {= |<> | < | <= | > | >= } **ANY** | **SOME**
- (*poddotaz*)
- *výraz* [**NOT**] **IN** (*poddotaz*)
- [**NOT**] **EXISTS** (*poddotaz*)

Příklady
jméno **LIKE** "K*"
jméno **LIKE** "J?rka"
jméno **LIKE** "P[A-F]###"
jméno **IN** ("Jan","Jiří","Pavel")
jméno **IN** ([Osoby].[Jméno])

Poznámka:

Jestliže se v klauzuli **FROM** uvede několik tabulek bez omezujících kritérií (v klauzulích **WHERE**, resp v **JOIN**), pak je výsledkem dotazu kartézský součin těchto tabulek. To je možné ve speciálních případech využít, jak ukazuje následující příklad.

Příklad:

ŠACH(hráč-ID, jméno, příjmení)
Úkolem je vygenerovat hrací listinu hráčů systémem každý s každým.

Při řešení příkladu otevřeme tabulku dvakrát, podruhé s náhradním jménem. Předpokládáme přitom jednokolový hrací systém. Jestliže bychom chtěli vygenerovat hrací listinu pro dvoukolový hrací systém, pak stačí operátor < nahradit operátorem nerovnosti <>.

SELECT šach.jméno, šach.příjmení, š2.jméno, š2.příjmení
FROM šach, šach **AS** š2
WHERE šach.hráč-ID < š2.hráč-ID

Poznámka:

SELECT ∗
FROM Tab1,Tab2
WHERE Tab1.ID = Tab2.ID
je zpracováno Accessem stejně jako
SELECT ∗
FROM Tab1 **INNER JOIN** Tab2 **ON** Tab1.ID=Tab2.ID

Příklad

ODDĚLENÍ(č-oddělení, název-oddělení)
PRACOVNÍCI(rodné-číslo, jméno, příjmení, č-oddělení)

1. **INNER JOIN** - kombinuje všechny záznamy, které splňují operaci spojení, tj. vyberou se všechna oddělení, na nichž jsou nějací pracovníci
2. **LEFT JOIN** - vybrat všechna oddělení, i když na některých nemusí být žádní pracovníci
3. **RIGHT JOIN** - vybrat všechny pracovníky včetně těch, kteří nejsou přiřazeni na žádná oddělení (např. externisté).

Příklad: V seznamu vystupujících informací nemusí být uvedeny pouze položky tabulky, ale také i složitější výrazy.

OSOBY(rodné-číslo, jméno, příjmení, ...)
ZAMĚSTNÁNÍ(rodné-číslo, organizace, datum-nástupu, datum-odchodu, ...)

Zjistit seznam osob, jejich zaměstnání a počet let odpracovaných u jednotlivých zaměstnavatelů a vypsat je seřazené abecedně.

SELECT DISTINCTROW osoby.příjmení, osoby.jméno, zaměstnání.organizace,
Val(**Format**(**Iif**(**IsNull**(zaměstnání.datum-odchodu,
Date()–zaměstnání.datum-nástupu,

```
                    zaměstnání.datum-odchodu–zaměstnání.datum–nástupu),
        "yy")) AS "počet odpracovaných let"
FROM osoby LEFT JOIN zaměstnání
                    ONosoby.rodné-číslo=zaměstnání.rodné-číslo
ORDER BY osoby.příjmení, osoby.jméno
```

2.1.1 Agregační funkce

název agregační funkce	význam
COUNT(*)	počet záznamů včetně těch, které obsahují **Null** hodnoty
COUNT(*položka*)	počet záznamů, neuvažují se položky s hodnotou **Null**
AVG(*výraz*)	aritmetický průměr
SUM(*výraz*)	součet
MIN(*výraz*)	minimální hodnota
MAX(*výraz*)	maximální hodnota
STDEV(*výraz*)	směrodatná odchylka základního souboru číselných výrazů
STDEVP(*výraz*)	odhad směrodatné odchylky základního souboru ze směrodatných odchylek skupin vytvořených pomocí **GROUP BY**
VAR(*výraz*)	rozptyl základního souboru číselných výrazů
VARP(*výraz*)	odhad rozptylu základního souboru z rozptylů skupin vytvořených pomocí **GROUP BY**

Tab. 13. Agregační funkce

Příklad:
VOLITELNÉ-PŘEDMĚTY(student, název-předmětu, ročník, semestr, ...)

Vypsat seznam předmětů, do nichž se přihlásilo alespoň 10 studentů a setřídit je sestupně podle počtu přihlášených.

```
SELECT název-předmětu, COUNT(název-předmětu)
    FROM volitelné-předměty
    GROUP BY název-předmětu
        HAVING {COUNT(název-předmětu) >=10
    ORDER BY 2 DESC
```

Agregační klíč může být i vícesložkový, jak ukazuje následující příklad.

Příklad:
Uvažujme relační schéma ŠKOLA(číslo-uč, jméno, příjmení, katedra, funkce, ...).

Chceme určit, seznam kateder a kolik profesorů, docentů a asistentů na jednotlivých katedrách pracuje.

```
SELECT katedra, funkce, COUNT() AS počet
    FROM škola
    GROUP BY katedra, funkce
```

Výsledek vyhodnocení tohoto dotazu má tvar podle tabulky.

katedra	funkce	počet
Matematika	profesor	2
	docent	6
	asistent	17
Fyzika	profesor	2
	docent	4
	asistent	11

Tab. 14. Vícevrstvá agregace

Ve vícevrstvé agregaci záleží na pořadí složek agregačního klíče. Jestliže toto pořadí v předchozím příkladu obrátíme, pak dostaneme odlišný výstup, jak ukazuje jak ukazuje následující tabulka.

```
SELECT funkce, katedra COUNT() AS počet
    FROM škola
    GROUP BY funkce, katedra
```

funkce	funkce,	počet
profesor	Matematika	2
	Fyzika	2
	…	…
docent	Matematika	6
	Fyzika	4
	…	…
asistent	Matematika	17
	Fyzika	11
	…	…

Tab. 15. Vícevrstvá agregace (podruhé)

2.1.2 SQL s poddotazy

Příklad

FIRMA1(rodné-č, jméno, příjmení, plat, funkce, ...)
FIRMA2(rodné-č, jméno, příjmení, plat, funkce, ...)

Všichni pracovníci z 1. firmy, jejichž plat převýšuje plat *všech* pracovníků z 2. firmy.

```
SELECT jméno, příjmení, plat
    FROM firma1
    WHERE plat > ALL (SELECT plat
                      FROM firma2)
```

Všichni pracovníci z 1. firmy, jejichž plat převyšuje plat *alespoň jednoho* pracovníka z 2. firmy.

```
SELECT jméno, příjmení, plat
    FROM firma1
    WHERE plat > ANY (SELECT plat
                      FROM firma2)
```

Příklad

ZÁKAZNÍCI(č-zákazníka, jméno-zák, místo, ulice, čp, PSČ)
FAKTURY(č-faktury, č-zákazníka, datum, ...)

Seznam zákazníků, kteří v tomto roce neplatili *žádnou* fakturu.

```
SELECT jméno-zák
    FROM zákazníci
    WHERE č-zákazníka NOT IN
                      (SELECT č-zákazníka
                          FROM faktury
                          WHERE datum >= #01.01.2001#)
```

Totéž zadání řešit s využitím klauzule **EXISTS**

```
SELECT jméno-zák
    FROM zákazníci
    WHERE NOT EXISTS
            (SELECT *
                FROM faktury
                WHERE zákazníci.č-zákazníka=faktury.č-zákazníka
                      AND datum >= #01.01.2001#)
```

Příklad

SOUTĚŽ(tým, body, dal, dostal, ...)

a) Sestavit tabulku sportovní soutěže podle získaných bodů, při rovnosti bodů se rozhoduje podle rozdílu vstřelených a obdržených gólů.
b) Z tabulky *Q* získané dotazem v předchozím kroku určit tým s největším počtem bodů. Pokud je více týmů s nejvyšším počtem bodů, pak je všechny vypsat.

```
SELECT tým, SUM(body) AS Sbody, SUM(dal) AS Sdal, SUM(dostal) AS Sdostal
    FROM soutěž
```

```
    GROUP BY tým
    ORDER BY 2 DESC, SUM(dal–dostal) DESC

SELECT Q.tým, Q.Sbody, (Q.Sdal–Q.Sdostal) AS skóre
    FROM Q                ‘ Q ≡ sportovní-tabulka získaná předchozím dotazem
    WHERE Q.Sbody = (SELECT MAX(Q.Sbody)
                          FROM Q)
    ORDER BY skóre DESC
```

V poddotazu je agregační funkce **MAX** vztažena k celé tabulce, protože zde chybí klauzule **GROUP BY**.

Příklad:

ČTENÁŘI(ID-čtenáře, jméno, příjmení, ...)
KNIHY(ISBN, název, autor, nakladatelství ...)
REZERVACE(ID-čtenáře, ISBN)

Jména čtenářů, kteří mají rezervovanou knihu "Zločin a trest".

```
SELECT čtenáři.jméno, čtenáři.příjmení
    FROM čtenáři
    WHERE čtenáři.ID-čtenáře IN
            (SELECT rezervace.ID-čtenáře
                FROM rezervace
                WHERE rezervace.ISBN =
                        (SELECT knihy.ISBN
                            FROM knihy
                            WHERE knihy.název = "Zločin a trest"))
```

Existenční a univerzální kvantifikátor v SQL

Výrok "pro každé x platí $p(x)$" je logicky ekvivalentní výroku "neexistuje x takové, že $p(x)$ neplatí". Formálně zapsáno:
$(\forall x)\ p(x) \equiv \neg\ (\exists\ x)(\ \neg\ p(x))$

Příklad:
$(\forall x)\ (\exists\ y)\colon y>x$
je ekvivalentní
$\neg\ (\exists\ x)(\ \neg\ (\exists\ y)\colon y>x)$

Poznámka:
Je jednodušší "myslet" v univerzálních kvantifikátorech, než v negacích existenčních kvantifikátorů. V SQL však není definován univerzální kvantifikátor.

Poznámka:
Výraz ... **EXISTS** (**SELECT** ∗ ...) se vyhodnotí `true`, je-li množina daná v závorkách neprázdná.

Příklad:

Jména čtenářů, kteří mají rezervovanou *nějakou* knihu.

```
SELECT čtenáři.jméno, čtenáři.příjmení
    FROM čtenáři
    WHERE čtenáři.ID-čtenáře IN
            (SELECT rezervace.ID-čtenáře
                FROM rezervace)
```

Zadání předchozího příkladu lze vyjádřit jinými slovy tak, že chceme určit jména čtenářů takových, že *existuje* kniha, kterou mají rezervovánu. Při tomto vyjádření se nabízí použití klauzule **EXISTS**.

```
SELECT čtenáři.jméno, čtenáři.příjmení
    FROM čtenáři
    WHERE EXISTS
        (SELECT *
            FROM rezervace
            WHERE čtenáři.ID-čtenáře = rezervace.ID-čtenáře)
```

Čísla čtenářů, kteří mají rezervovánu alespoň jednu knihu, přitom však *žádnou* z nakladatelství *UNIS*. Jinými slovy tedy čísla čtenářů takových, že *neexistuje* jimi rezervovaná kniha, která by byla z UNISu.

```
SELECT rezervace.ID-čtenáře
    FROM rezervace, rezervace AS R2
    WHERE NOT EXISTS
        (SELECT *
            FROM rezervace
            WHERE rezervace.ID-čtenáře = R2.ID-čtenáře
                AND ISBN IN
                    (SELECT ISBN
                        FROM knihy
                        WHERE knihy.nakladatelství = "UNIS"))
```

Čísla čtenářů, kteří mají rezervovány *všechny* knihy.

```
SELECT čtenáři.ID-čtenáře
    FROM čtenáři
    WHERE (SELECT DISTINCT ISBN
            FROM rezervace
            WHERE rezervace.ID-čtenáře = čtenáři.ID-čtenáře)
        =
        (SELECT ISBN
            FROM knihy)
```

Z povahy problému plyne, že obě množiny jsou identické, jestliže mají stejný počet prvků, a tedy jej lze řešit i takto:

```
SELECT čtenáři.ID-čtenáře
    FROM čtenáři
    WHERE (SELECT COUNT(DISTINCT ISBN)
                FROM rezervace
                WHERE rezervace.ID-čtenáře = čtenáři.ID-čtenáře)
        =
            (SELECT COUNT(ISBN)
                FROM knihy)
```

2.2. Křížový dotaz

TRANSFORM *agregační-funkce* (*výraz*1)
výběrový-dotaz
PIVOT *výraz*2

Příklad:

```
TRANSFORM SUM(qOsoba.doba) As celková-doba
    SELECT qOsoba.rodné-čislo, qOsoba.příjmení, qOsoba.jméno
        FROM Osoba
        GROUP BY qOsoba.rodné-čislo, qOsoba.příjmení, qOsoba.jméno
    PIVOT Osoba.místo
```

rodné číslo	příjmení	jméno	<>	ABB	KPS	…
	Hora	Jan	0			…
	Kos	Petr		10	4	…
	Novotný	Pavel			5	…
…	…		…	…	…	…

Tab. 16. Křížový dotaz

Příklad:

```
TRANSFORM COUNT(Student.ročník)
    SELECT Student.ročník
        FROM Student
        GROUP BY Student.ročník
    PIVOT Partition(průměrný-prospěch,1,3,0.50)
```

ročník	1 : 1.50	1.51 : 2	2.01 : 2.50	2.51 : 3
1	40	150	250	60
2	…	…	…	…
3	…	…	…	…
4	…	…	…	…
5	…	…	…	…

Tab. 17. Křížový dotaz

2.3. Akční (aktualizační) dotazy

Vytvoření nové tabulky.

SELECT [**ALL** | **DISTINCT** | ...]
seznam-polí
INTO *nová-tabulka* [**IN** *externí-databáze*]
FROM *zdrojová-tabulka*

Přidání jednoho záznamu do existující tabulky nebo dotazu.

INSERT INTO { *tabulka* | *dotaz* }
[(*seznam polí, do nichž se data přidávají*)]
VALUES (*seznam hodnot*)

Výběr záznamů z jedné tabulky a jejich přidání do nových záznamů jiné tabulky.

INSERT INTO { *tabulka* | *dotaz* }
[(*seznam polí, do nichž se data přidávají*)]
SELECT ...
FROM ...
WHERE ...

Změna obsahu položek v tabulce.
UPDATE { *tabulka* | *dotaz* }
[**IN** *externí-databáze*]
SET *název-sloupce* = { *výraz* | **NULL** }
[, ...]
WHERE *vyhledávací-podmínka*

Vymazání (zrušení) záznamů v tabulce.

DELETE [∗ | *tabulka*. ∗ | *seznam-sloupců*]
FROM ...
[**IN** *externí-databáze*]
WHERE *vyhledávací-podmínka*

2.4. Definiční dotazy

Vytvoření nové tabulky, definice položek (název, datový typ, velikost).a indexů

CREATE TABLE *tabulka*
(*položka*1 *datový-typ*[(*velikost*)]
[**NOT NULL**] [**CONSTRAINT** *index*1…]
[, *položka*2 *datový-typ*[(*velikost*)]
[**NOT NULL**] [**CONSTRAINT** *index*2…]
[, ...]]
[,**CONSTRAINT** *vícesložkový-index* …
[, ...]])

Klauzule CONSTRAINT
- *definice indexů* (*a vytvoření relace s jinou tabulkou*)
v příkazech **CREATE TABLE a ALTER TABLE**

- *Jednoduchý index*

CONSTRAINT *jméno-indexu*
{ **PRIMARY KEY** | **UNIQUE** | **NOT NULL** |
REFERENCES *cizí-tabulka* [(*cizí-pole*1, *cizí-pole*2)]}

- *Vícesložkový index* (lze jej definovat pouze pro primární klíč.)

CONSTRAINT *jméno-indexu*
{ **PRIMARY KEY** (*primární-segm*1[,*primární-segm*2 [, ...]]) |
UNIQUE} (*jedinečný-segm*1 *jedinečný-segm*2 [, ...]]) |
NOT NULL (*nenulový-segm*1[,*nenulový-segm*2 [, ...]]) |
FOREIGN KEY (*ref*1[,*ref*2 [, ...]]) **REFERENCES** *cizí-tabulka*
[(*cizí-pole*1 [,*cizí-pole*2
[, ...]])]}

- *Modifikace struktury existující tabulky (přidávání nových položek, resp. odstraňování existujících položek.*

ALTER TABLE *tabulka*
{ **ADD** { **COLUMN** *položka datový-typ* [(*velikost*)]
[**NOT NULL**}] [**CONSTRAINT** *index* …] |
CONSTRAINT *vícesložkový-index* …} |
DROP { **COLUMN** *položka* | **CONSTRAINT** *jméno-vícesložkového indexu* } }

- *Zrušení existující tabulky z databáze, resp. zrušení existujícího indexu z tabulky.*

DROP {**TABLE** *tabulka* | **INDEX** *index* **ON** *tabulka*}

- *Vytvoření nového indexu pro existující položku v tabulce.*

CREATE [**UNIQUE**] **INDEX** *index*
ON *tabulka* (*položka* [**ASC** | **DESC**]
[,*položka* [**ASC** | **DESC**], ...])
[**WITH** { **PRIMARY** | **DISALLOW NULL** | **IGNORE NULL**}]

3. Visual Basic pro aplikace MS Access

Definice konstant

[**Public** | **Private**] **Const** *název-konstanty* [**As** *datový_typ*] = *konstantní_výraz*

Private (implicitní) – konstanta přístupná pouze v modulu, kde byla definována
Public v deklarační sekci (modulu, formuláře nebo sestavy) – globální platnost
datový_typ: **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Date**, **String**, **Variant**

3.1. Řídící struktury Visual Basicu

3.1.1 Přiřazovací příkazy

proměnná = výraz
Set *proměnná* ={ **New** *objektový-výraz* | **Nothing**}

3.1.2 Příkazy větvení

If *podmínka* **Then** *příkaz*1 [**Else** *příkaz*2]

```
If podmínka Then
   příkazy1
[Else If podmínka2 Then
          příkazy2
       [Else
          příkazy3]]
End If
```

```
Select  Case testovaný-výraz
        Case seznam-hodnot-1
              příkazy1
        Case seznam-hodnot-2
              příkazy2
        ...
        [Case Else
              příkazy-n]
End Select
```

3.1.3 Příkazy cyklu

For *řp* = *poč* **To** *konc* [**Step** *krok*]
    *příkazy*1
[**Exit For**]
    *příkazy*2
Next *řp*

Jestliže se tělo cyklu provádí pro všechny prvky nějaké kolekce, je výhodné použít následující verzi cyklu **For**.

For Each *prvek* **In** *skupina*
    *příkazy*1
[**Exit For**]
    *příkazy*2
Next *prvek*

Příklad:
Určení názvů všech otevřených formulářů a prvků v nich obsažených. Pro srovnání ukážeme řešení příkladu pro obě verze cyklu **For**.

```
Sub VýpisNázvů1()
    Dim nf As Integer, nc As Integer, i As Integer, j As Integer
    Dim s As String
    Dim frm As Form
    s = ""
    Set nf = Forms.Count
    If nf > 0 Then
       For i = 0 To nf–1
            Set frm = Forms(i)
            s = s & frm.Name & Chr$(13) & Chr$(10)
            nc = frm.Count
            If nc > 0 Then
               For j = 0 To nc–1
                    s = s & "          " & frm(j).Name & Chr$(13) & Chr$(10)
               Next j
            Else
               s = s & "          ve formuláři nejsou žádné prvky" & Chr$(13) & Chr$(10)
            End If
       Next i
    Else
       s = s & "Nejsou otevřeny žádné formuláře" & Chr$(13) & Chr$(10) & Chr$(13) _
       & Chr$(10)
    End If
    MsgBox s
End Sub
```

```
Sub VýpisNázvů2()
    Dim s As String
    Dim frm As Form, ctl As Control
    s = ""
    Set nf = Forms.Count
    If Forms.Count > 0 Then
       For Each frm In Forms
            s = s & frm.Name & vbCrLf
            If frm.Count > 0 Then
               For Each ctl In frm.Controls
                    s = s & vbTab &  ctl.Name & vbCrLf
               Next ctl
            Else
               s = s & vbTab & "ve formuláři nejsou žádné prvky" & vbCrLf
            End If
       Next frm
    Else
       s = s & "Nejsou otevřeny žádné formuláře" & vbCrLf
    End If
    MsgBox s
End Sub
```

```
Do {While | Until} podmínka
    příkazy1
    [Exit Do]
    příkazy2
Loop
```

```
Do
    příkazy1
    [Exit Do]
    příkazy2
Loop {While | Until} podmínka
```

```
While podmínka
    příkazy
Wend
```

3.1.4 Příkazy skoku

- *Nepodmíněný skok* na jiný příkaz v proceduře nebo funkci
  GoTo {*návěští* | *číslo-řádku*}

Příklad
GoTo nav
...
nav:

- *Podmíněný skok při vyhodnocení chyby*
  On Error {**GoTo** *identifikátor-řádku* | **Resume** [**Next**] | **GoTo** 0}

- Je-li uvedeno pouze **Resume**, provede se skok na příkaz, který způsobil chybu a Access se pokusí jej znovu provést.
- Konstrukce **Resume Next** zachycuje chyby, přitom se však pokračuje v provádění následujícího příkazu.
- Jestliže je použito **GoTo** 0, pak je zachycování chyb v aktuální proceduře vypnuto a chyba je předána chybové rutině ve volající proceduře. Pokud žádná předcházející chybová rutina neexistuje, otevře se chybové dialogové okno.

Poznámka:
V chybové rutině (podprogramu ošetření chyby) lze testovat:

1. Hodnotu vestavěné proměnné **Err**(číslo chyby). Jestliže **Err**=0, pak žádná chyba nenastala.
2. Chybové hlášení pomocí funkce **Error**.
3. Lze využít i objekt **Err** a jeho vlastnosti **Err**.**Description**, v níž je uložena textová informace o vzniklé chybě, a **Err**.**Number** obsahující číslo chyby.

3.1.5 Příkaz With

With *objekt*
    příkazy
End With

Příklad:

With *můj-objekt*
    .**Height** = 2000
    .**Width** = 2000
    .**Caption** = "Toto je moje označení"
End With

Odkazy na prvky kolekce ve Visual Basicu

Forms … kolekce otevřených formulářů
Forms(*i*) … (*i*+1)-ní otevřený formulář
Forms[*objednávky*] … formulář `objednávky`

Forms("objednávky") … formulář `objednávky`

Jestliže je jméno prvku kolekce uloženo v proměnné, např. je parametrem procedury nebo funkce, pak musíme použít poslední zápis s kulatými závorkami, uvozovky se ovšem nepoužijí.

Příklad:
Jestliže při návrhu formuláře do něj začleníme řídící prvek *karta*, pak zjistíme, že tento prvek má pouze dvě stránky a v interaktivním režimu není možné počet stránek karty zvýšit. Jedinou možností je programové řešení. Například jej můžeme implementovat takto: Do pomocného formuláře zařadíme dvě editační pole
(jedno je určeno pro vstup jména formuláře a druhé pro jméno karty ve formuláři) a tlačítko, po jehož stisku se spustí procedura, která otevře příslušný formulář v návrhovém zobrazení, do určené karty přidá novou stránku a nakonec formulář zavře. Kostru procedury znázorňuje následující úsek zdrojového kódu.

Sub PřidatStránkuNaKartu(*názevform* **As String**, *názevkarty* **As String**)
    ...
    DoCmd.**OpenForm** *názevform*, `acDesign`
    ...
    Forms(*názevform*).**Controls**(*názevkarty*).**Pages**.**Add**
    DoCmd.**Close**
    ...
End Sub

3.1.6 Procedury a funkce

[**Public** | **Private**] [**Static**] **Sub** *název-procedury* ([*seznam-parametrů*])
    [*příkazy*1]
    [**Exit Sub**]
    [*příkazy*2]
End Sub

	Rozsah platnosti procedury/funkce
Public	Ve všech procedurách/funkcích *všech* modulů.
Private	V dalších procedurách/funkcích *stejného* modulu.
Static	Všechny proměnné deklarované (implicitně či explicitně) po celou dobu otevření modulu obsahující tuto proceduru budou uchovány. **Příklad:** *Čítač* uvnitř procedury, jehož hodnota se zvyšuje o 1 při každém volání procedury.

Tab. 18. Rozsah platnosti procedury/funkce

3.1.6.1 Parametry procedur a funkcí

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *název-parametru* [**As** *datový-typ*]

- **Optional** označuje nepovinný parametr. Umožňuje deklarovat parametr typu **Variant**. Za nepovinným parametrem musí být všechny další parametry rovněž nepovinné. Test nepřítomnosti nepovinných parametrů lze provést pomocí funkce **IsMissing**().
- **ByVal** - volání hodnotou. Jestliže je skutečným parametrem výraz, Visual Basic s ním zachází jako by byl deklarován pomocí **ByVal**.
- **ByRef** - volání odkazem. Pole se vždy předávají odkazem.
- **ParamArray** musí být v seznamu parametrů poslední.

3.1.6.2 Volání procedur a funkcí

Proceduru můžeme volat dvěma způsoby:

Call *název-procedury*(*seznam-skutečných-parametrů*)

nebo

název-procedury seznam-skutečných-parametrů

Výsledkem funkce je hodnota, a tedy je možné ji použít všude tam, kde je přípustný výraz, např. na pravé straně přiřazovacího příkazu:

proměnná = *název-funkce*(*seznam-skutečných-parametrů*)

výchozí hodnoty: (Pokorný/Kopp, str. 40-41)

číslo	0
řetězec	prázdný řetězec
typ **Variant**	**Empty**
objektový typ	**Nothing**

Příklad: (z modulu `modUtility`)

```
Function IsNothing(varToTest As Variant) As Integer
    'testuje "logical nothing" podle datového typu
    'Empty a Null = Nothing
    'číslo=0 = Nothing
    'řetězec nulové délky = Nothing
    'Date/Time nikdy není Nothing
    IsNothing = True
    Select Case VarType(varToTest)
        Case vbEmpty
            Exit Function
        Case vbNull
```

```
            Exit Function
        Case vbBoolean
            If varToTest Then IsNothing = False
        Case vbByte, vbInteger, vbLong, vbSingle, vbDouble, vbCurrency
            If varToTest < > 0 Then IsNothing = False
        Case vbDate
            IsNothing = False
        Case vbString
            If Len(varToTest) < > 0 And varToTest < > " " Then IsNothing = False
    End Select
End Function
```

3.2. Formuláře

3.2.1 Událostní procedury ve formulářích

Příklad: Hledání záznamu po stisku tlačítka podle pole, na němž stál kurzor před jeho stiskem.

```
Private Sub Command7_Click()
    On Error GoTo Err_Command7_Click
    Screen.PreviousControl.SetFocus
    DoCmd.DoMenuItem acFormBar, acEditMenu, 10, , acMenuVer70
Exit-Command7_Click:
    Exit Sub
Err-Command7_Click:
    MsgBox Err.Description
    Resume Exit_Command7_Click
End Sub
```

Příklad: Formulář kritérií pro výběr (bez podkladové tabulky) a událost po stisknutí tlačítka způsobující otevření nového formuláře s podkladovou tabulkou omezenou filtrem z prvního formuláře.

(A)

```
Option Compare Database
Option Explicit

Private Sub Command12_Click()
    On Error GoTo Err_Command12_Click
    Dim stDocName As String
```

```
    Dim stLinkCriteria As String
    stDocName = "zaměstnání"
    stLinkCriteria = "[kraj] = " & "'" & Me![Combo6] & "'" _
    & " AND [příjmení] LIKE" & "'" & Me![Text8] & "*" & "'"_
    & " AND [plat] > = " & Me![Text10]
    DoCmd.OpenForm stDocName, , , stLinkCriteria
Exit-Command12_Click:
    Exit Sub
Err-Command12_Click:
    MsgBox Err.Description
    Resume Exit_Command12_Click
End Sub
```

(B)

```
stDocName = "osoby"
stLinkCriteria ="[pohlaví] =" & "'" & IIf(Me![Frame0] = 1, "muž", "žena") & "'" _
    & " AND [vìk] <=" & Me![maxvek] _
    & " AND [vzdělání] =" & "'" & Me![Combo11] & "'" _
    & " AND [počet dětí] <=" & Me![maxdeti] _
    & " AND [místo] LIKE " & "'" & Me![místo] & "*'" _
    & " AND [plat] >=" & Me![minplat]
DoCmd.OpenForm stDocName, , , stLinkCriteria
```

```
stDocName = "osoby"
stLinkCriteria = "[pohlaví] ='" & IIf(Me![Frame0] = 1, "muž", "žena") & "'" _
    & " AND [věk] <=" & Me![maxvek] _
    & " AND [vzdělání] ='" & Me![Combo11] & "'" _
    & " AND [počet dětí] <=" & Me![maxdeti] _
    & " AND [místo] LIKE '" & Me![místo] & "*'" _
    & " AND [plat] >=" & Me![minplat]
DoCmd.OpenForm stDocName, , , stLinkCriteria
```

Příklad: Víceúrovňový výběr.

Máme k dispozici dvě tabulky:

OSOBY(rč, jméno, příjmení, pohlaví, stav, ...)
STAVY(pohlaví, stav)

V první tabulce jsou položky `pohlaví` i `stav` textového typu, v druhé tabulce je `pohlaví` číselného typu. Tato tabulka se označuje jako `číselník` a obsahuje všechny možné stavy, které osoby mohou mít.
Nemá definován žádný klíč.
Konkrétní údaje číselníku jsou uvedeny v tab. 19.

pohlaví	stav
1	ženatý
1	svobodný
1	rozvedený
1	vdovec
2	vdaná
2	svobodná
2	rozvedená
2	vdova

Tab. 19. Číselník stavů

Ve formuláři s podkladovou tabulkou `OSOBY` se stav vybírá tak, že nejdříve v přepínači **Option Group** se dvěma polohami `muž`, `žena` vybereme pohlaví a to způsobí vykreslení příslušné čtveřice stavů v seznamu **ListBox** a z něj pak vybereme skutečný stav. Vybraný stav ze seznamu se přímo ukládá do položky `stav` v tabulce `OSOBY`, to znamená, že tato položka je uvedena ve vlastnosti `Control Source` seznamu.
Jestliže přepínač nastavíme do 1. polohy, pak se do položky `OSOBY.pohlaví` uloží hodnota "muž", přepnutí do 2. polohy generuje zápis hodnoty "žena" do uvedené položky.

Předpokládejme, že přepínač je definován tak, že první poloha vrací hodnotu 1 a druhá poloha hodnotu 2. Nechť dále jméno seznamu je *List*9 a jméno skupiny voleb je *Frame*2. Za těchto předpokladů je seznam ve vlastnosti `Row Source` definován SQL dotazem:

SELECT DISTINCTROW stav **FROM** stavy **WHERE** pohlaví=*Frame*2.**Value**;

Kdykoliv přepneme polohu ve skupině voleb, musí se znovu překreslit seznam příslušné čtveřice stavů. Musíme tedy definovat událost *After_Update* přepínače. Současně však musíme zrušit případně již definovaný stav osoby, protože ten se týká pohlaví, které bylo nastaveno před přepnutím, a ve formuláři je třeba jej znovu zadat.

```
Private Sub Frame2_AfterUpdate()
        pohlaví = IIf(Frame2.Value=1,"muž","žena")
        List9.Visible = True
        List9. Requery
        stav = " "
End Sub
```

Poslední věc, kterou musíme ošetřit, je to, aby při procházení tabulkou pomocí selektorů záznamů ve formuláři se ve formuláři rekonstruovalo nastavení přepínače, vykreslil se seznam příslušné čtveřice stavů a v tomto seznamu se zvýraznil vybraný stav. Všechny tyto akce zařadíme do události formuláře *Form_Current*(). Větev **Else** odpovídá případu, kdy v tabulce ještě není uložena informace o pohlaví osoby, např. jsme se dostali na konec tabulky a vkládáme údaje do nově přidávaného řádku. V tom případě musíme zajistit, aby v přepínači nebyla vybrána žádná poloha a dokud nějakou polohu v přepínači nezvolíme, se ani nezobrazovala žádná čtveřice stavů.

```
Private Sub Form_Current()
        If Not IsNull(pohlaví) Then
```

```
            Frame2.Value = IIf (pohlaví ="muž",1,2)
            List9.Visible = True
            List9.Requery
            List9.Value = stav
        Else
            Frame2.Value = 0
            List9.Visible = False
        End If
End Sub
```

3.3. Objekt RecordSet

Příklad:
Cyklus modifikace ve všech řádcích tabulky (např. zvýšení poplatku za elektřinu o 10 procent).

```
    ...
    Dim ws As Workspace
    Dim db As Database
    Dim rst As RecordSet
'1.
    Set ws = DBEngine.Workspaces(0)
    Set db = ws.Databases(0)

'2  Set db = DBEngine(0)(0)

'3. Set ws = DBEngine.Workspaces(0)
    'Set db = ws.OpenDatabase("název souboru .mdb")

'4. Set db  = CurrentDb()

    Set rst = db.OpenRecordset("název tabulky")
    rst.MoveFirst
    While Not(rst.EOF)                  '  Do Until rst.EOF
        rst.Edit                        '      rst.Edit
        rst![sazba] = rst![sazba] *1.1  '      rst![sazba] = rst![sazba] *1.1
        rst.Update                      '      rst.Update
        rst.MoveNext                    '      rst.MoveNext
    Wend                                '  Loop
    rst.Close
    ...
```

Příklad: Aktualizace záznamů s nevyplněným údajem o telefonním čísle.

```
...
Dim db As Database, rst As RecordSet
Dim strHledej, strMsg, strTab As String
Set db = CurrentDb()
strTab = "Zákazníci"
Set rst = db.OpenRecordset(strTab,dbOpenDynaset)
strHledej = "IsNull([telefon])"

rst.FindFirst strHledej
Do Until rst.NoMatch
    rst.Edit
    strMsg = "Zadejte telefonní číslo zákazníka: " _
    & rst![příjmení] & " " & rst![jméno]
    rst![telefon] = InputBox(strMsg)
    rst.Update
    rst.FindNext strHledej
Loop
rst.Close
...
```

Příklad:
Tentýž příklad v obecné verzi - doplňování údajů do libovolné tabulky a libovolného jejího sloupce.

```
Sub DoplňHodnotuÚdaje(tabulka As String, sloupec As String)
    Dim db As Database, rst As Recordset, f As Field
    Dim strHledej As String, záznam As String, s As String, i As Integer
    Set db = CurrentDb()
    Set rst = db.OpenRecordset(tabulka, dbOpenDynaset)
    strHledej = "IsNull(" & sloupec & ")"
    rst.FindFirst strHledej
    Do Until rst.NoMatch
        rst.Edit
        záznam = ""
        For Each f In rst.Fields
            Select Case VarType(rst.Fields(f.Name))
                Case vbString
                    s = rst.Fields(f.Name)
                Case vbByte, vbInteger, vbLong, vbSingle, _
                    vbDouble, vbCurrency
                    s = Str(rst.Fields(f.Name))
                Case vbBoolean
                    s = IIf(rst.Fields(f.Name), "True", "False")
                Case vbDate
                    s = Format(rst.Fields(f.Name), "dd.mm.yy")
```

```
                    Case Else
                        s = ""
                End Select
                záznam = záznam & f.Name & ": " & s & vbCrLf
            Next f

'alternativní zápis cyklu For
'           For  i = 0 To rst.Fields.Count – 1

'                Select  Case VarType(rst.Fields(rst.Fields(i).Name))
'                    Case vbString
'                        s = rst.Fields(rst.Fields(i).Name)
'                    Case vbByte, vbInteger, vbLong, vbSingle, _
'                        vbDouble, vbCurrency
'                        s = Str(rst.Fields(rst.Fields(i).Name))
'                    Case vbBoolean
'                        s = IIf(rst.Fields(rst.Fields(i).Name), "True", "False")
'                    Case vbDate
'                        s = Format(rst.Fields(rst.Fields(i).Name), "dd.mm.yy")
'                    Case Else
'                        s = ""
'                End Select
'                záznam = záznam & rst.Fields(i).Name & ": " & s & vbCrLf
'           Next f

            rst.Fields(sloupec) = InputBox(záznam & vbCrLf & _
                                  "Zadej hodnotu údaje " & sloupec & ": ")
            rst.Update
            rst.FindNext strHledej
        Loop
        rst.Close
End Sub
```

3.4. Volání SQL dotazu z VBA

1. Uložený dotaz (výběrový, křížový, …)
DoCmd.**OpenQuery** *název-dotazu* [, *pohled*][, *datový-mód*]

pohled: `acViewDesign`, `acViewNormal`, `acViewPreview`
datový-mód: `acAdd`, `acEdit`, `acReadOnly`

2. Výběrový dotaz
Set *rst* = *db*.**OpenRecordset**(*výběrový_dotaz*,`dbOpenDynaset`)

3. Akční a definiční dotazy

DoCmd.**RunSQL** *dotaz* [, *použít-transakci*]
nebo
db.Execute *dotaz*

V příkazu **DoCmd**.**RunSQL** lze jako *dotaz* použít pouze *akční* nebo *definiční* dotaz, to znamená dotazy **INSERT INTO**, **DELETE**, **UPDATE**, **SELECT INTO**, resp. dotazy **CREATE TABLE**, **DROP TABLE**, **ALTER TABLE**,**CREATE INDEX**, **DROP INDEX**.

Příklad: Zjištění hodnoty pro položku typu `automatické číslo`.

Jestliže zapisujeme z programu např. pomocí akčního dotazu **INSERT INTO** do tabulky, pak je nutné pro definici údaje typu automatické číslo zjistit nejvyšší přidělené číslo v dosud zapsaných záznamech. Samozřejmě, pokud je tabulka prázdná, musí toto číslo být rovno nule.

```
Function MaxHodnota(tabulka As String, sloupec As String) As Byte
    Dim db As Database, rst As RecordSet
    Set db = CurrentDb()
    Set rst = db.OpenRecordset("SELECT COUNT(*) As mxf0 FROM " & tabulka)
    If rst!mxf0 = 0 Then
       MaxHodnota = 0
    Else
       Set rst = db.OpenRecordset("SELECT  MAX(" & sloupec & ") As mxf0 " & _
                                  "FROM "  & tabulka)
       rst.MoveFirst
       MaxHodnota = rst!mxf0
    End If
    rst.Close
End Function
```

Příklad:

"Teploměr", přesýpací hodiny, cyklus, transakce.

```
    Set ws = DBEngine.Workspaces(0)
    Set db = ws.Databases(0)
    Set rst = db.OpenRecordset("název tabulky")
    teploměr = SysCmd(acSysCmdInitMeter,"text co se děje", počet-kroků)
    DoCmd.Hourglass True
    ws.BeginTrans
    ...
    For i = 1 To počet-kroků
        ...
        rst.AddNew
        ...
```

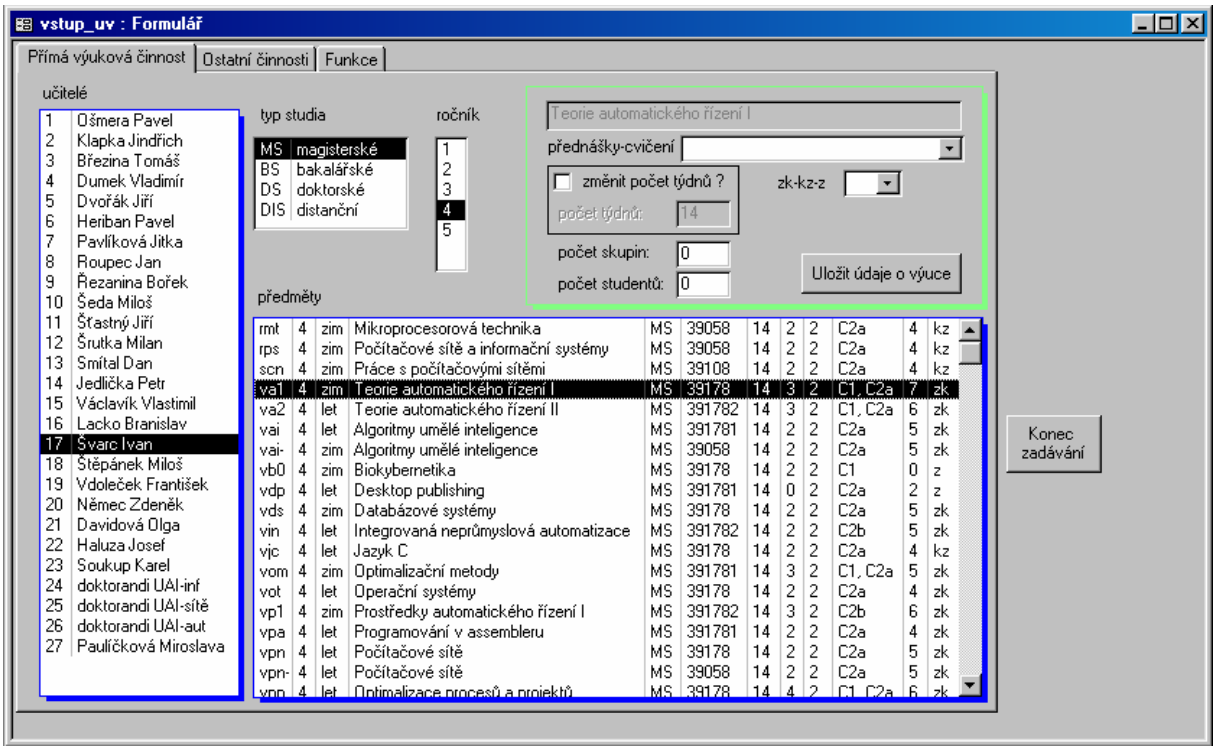
```
        rst.Update
        ...
        teploměr = SysCmd(acSysCmdUpdateMeter, i)
    Next i
    ws.CommitTrans
    rst.Close
    DoCmd.Hourglass False
    teploměr = SysCmd(acSysCmdClearStatus)
```

4. Složitější příklad

V tomto odstavci ukážeme dva typické formuláře z rozsáhlejší aplikace řešené autorem textu za účelem sledování aktivit učitelů Ústavu automatizace a informatiky FSI VUT v Brně. Aplikace je řízena z centrálního formuláře, plnícího roli jakéhosi menu.

Obr. 4.1. Menu aplikace

Na ukázku vybereme formulář, který se aktivuje po stisku volby Úvazky učitelů. V tomto formuláři se zadávají informace o učiteli a jeho výuce tak, že se postupným zpřesňováním (typ studia, ročník) vytřídí seznam předmětů, z nichž se vybere vyučovaný předmět, pro který se pak v pravé horní části formuláře vyplní podrobné informace o typu výuky, počtu hodin, počtu skupin, počtu týdnů apod.



Obr. 4.2. Zadávání výuky předmětu učitelem ústavu

Některé položky specifikující předmět jsou nedosažitelné, protože jsou již předvyplněny z tabulky předmětů, jiné jsou aktivní. Tvar formuláře je vidět na obr. 4.2.

Zdrojový kód událostních procedur prvků formuláře je uveden dále, nebudeme jej rozebírat, protože obsahuje buď již známé příkazy Visual Basicu nebo je lze s pomocí helpu či vzhledu formuláře identifikovat a místo toho jej ponecháme čtenáři k samostatnému studiu. Formulář je řešen jako vícestránkový, přičemž obsah jednotlivých stánek se liší.

```
Option Compare Database
Option Explicit
Dim ppp As Integer, ccc As Integer, kkk As Single
Dim sss As String

Private Sub Form_Load()
    TabCtl33.Value = 0          ' první stránka na kartě
    ListTypStudia.Value = "MS"  ' typ studia
    ListRocnik.Value = 1        ' ročník
    ListPredmety.Requery        ' vybraný seznam předmětů
    ListUcitele.Value = 1       ' učitel
    ComboFunkce.Value = 201     ' funkce
    Label_pom0.Visible = False
    Label_pom1.Visible = False
    Label_pom2.Visible = False
    Label_pom3.Visible = False
    Label_pom4.Visible = False
    Label_pom5.Visible = False
    Label_pom6.Visible = False
    Label_pom7.Visible = False
    Label_pom8.Visible = False
    Label_pom9.Visible = False
    LabelCv.Caption = " "
    ComboZk.Value = " "
    ListUcFun.Value = Null
'    ListUcFun.Requery
    ListUcOst.Value = Null
    ListUcOst.Requery
    TextPocetOstatni.Value = 0
'    TextPocetOstatni.Enabled = False
    CommandSaveChange.Enabled = False
    CheckTydny.Value = False
    TextTydny.Enabled = False
    TextSkupiny.Enabled = False
    TextStudenti.Enabled = False
    CheckTydny.Enabled = False
    ComboTyp_prcv.Enabled = False
    TextPredmet.Enabled = False
End Sub

Private Sub InicializaceDetailů(enab As Integer)
    TextTydny.Enabled = False
    CheckTydny.Value = False
    TextSkupiny.Value = 0
    TextStudenti.Value = 0
    ComboTyp_prcv.Value = 0
    LabelCv.Caption = " "
    ComboZk.Value = " "
    If enab = 1 Then
```

```
      TextSkupiny.Enabled = False
      TextStudenti.Enabled = False
      CheckTydny.Enabled = False
      ComboTyp_prcv.Enabled = False
      TextPredmet.Enabled = False
   End If
End Sub

Private Sub ListUcitele_AfterUpdate()
   TextPocetOstatni.Value = 0
   ListUcOst.Requery
End Sub

Private Sub ComboOstatni_AfterUpdate()
'    TextPocetOstatni.Enabled = True
   TextPocetOstatni.Value = 0
End Sub


Private Sub ListTypStudia_AfterUpdate()    ' typ studia
   ListRocnik.Value = 1
   Select Case ListTypStudia.Value
      Case "DS"
         ListRocnik.RowSource = "1"
      Case "BS"
         ListRocnik.RowSource = "1;2;3"
      Case "MS"
         ListRocnik.RowSource = "1;2;3;4;5"
      Case "DIS"
         ListRocnik.RowSource = "1;2;3;4;5;6"
   End Select
   ListRocnik.Requery
   ListPredmety.Requery
   TextPredmet.Value = " "
   TextTydny.Value = 0
   InicializaceDetailů (1)
   If ListTypStudia.Value = "DIS" Then
     ComboTyp_prcv.Value = 2
   Else
     ComboTyp_prcv.Value = 0
   End If
End Sub

Private Sub ListRocnik_AfterUpdate()    ' ročník
   ListPredmety.Requery
   ListPredmety.Value = 0
   TextPredmet.Value = " "
   TextTydny.Value = 0
   InicializaceDetailů (1)
End Sub

Private Sub ListPredmety_AfterUpdate()    ' předměty
   Dim db As Database, rst As Recordset
   Set db = CurrentDb()
   Set rst = db.OpenRecordset("predmety")
   rst.Index = "PrimaryKey"
   rst.Seek "=", ListPredmety.Value
   TextSkupiny.Enabled = True
   TextStudenti.Enabled = True
   CheckTydny.Enabled = True
```

```
    ComboTyp_prcv.Enabled = True
'    TextPredmet.Enabled = True
    TextPredmet.Value = rst!nazev
    TextTydny.Value = rst!tydnu
    ppp = rst!predn
    ccc = rst!cvic
    sss = rst!semestr
    rst.Close
    If sss = "zim" Then
        Label_pom5.Caption = 0    ' t_LS
        Label_pom7.Caption = 0    ' h_LS
    Else
        Label_pom4.Caption = 0    ' t_ZS
        Label_pom6.Caption = 0    ' h_ZS
    End If
' další složky z recordsetu
    InicializaceDetailů (0)
End Sub

Private Sub CheckTydny_AfterUpdate()
    TextTydny.Enabled = CheckTydny.Value
End Sub

Private Sub CommandPridatFunkci_Click()
    On Error GoTo Err_CommandPridatFunkci_Click
    Dim db As Database, rst As Recordset
    Dim strQ As String, mx As Integer

    Set db = CurrentDb()

'(1) Set rst = db.OpenRecordset("uc_fun")
'    mx = 0
'    rst.MoveFirst
'    While Not rst.EOF
'        If rst!IDautom > mx Then mx = rst!IDautom
'        rst.MoveNext
'    Wend
'    mx = mx + 1
'    MsgBox "mx1=" + Str(mx)
'    rst.Close

'(2) Set rst = db.OpenRecordset("MXuc_fun", dbOpenDynaset)
'    rst.MoveFirst
'    mx = rst!mxf + 1
'    MsgBox "mx2=" + Str(mx)
'    rst.Close

'(3) Set rst = db.OpenRecordset("SELECT MAX(IDautom) As mxf0 FROM uc_fun")
'    rst.MoveFirst
'    mx = rst!mxf0 + 1
'    MsgBox "mx3=" + Str(mx)
'    rst.Close

'(4)
    mx = MaxHodnota("uc_fun", "IDautom") + 1
'    MsgBox "MaxHodnota=" + Str(mx)
    Label_pom0.Caption = mx
    If Not IsNull(ListUcitele.Value) And Not IsNull(ComboFunkce.Value) Then
        strQ = "INSERT INTO uc_fun" _
            & "(IDuc,IDfu,IDautom)" _
```

```
            & "VALUES (ListUcitele.Value,ComboFunkce.Value,Label_pom0.Caption)"
'           & "VALUES (ListUcitele.Value,ComboFunkce.Value,MaxHodnota('uc_fun', 'IDautom') + 1)"
        DoCmd.RunSQL strQ
        ListUcFun.Requery
    Else
        MsgBox "Nový záznam o funkci nelze zapsat," & vbCrLf & _
             "protože údaje jsou neúplné"
    End If

'(5)   If Not IsNull(ListUcitele.Value) And Not IsNull(ComboFunkce.Value) Then
'       strQ = "INSERT INTO uc_fun" _
'            & "(IDuc,IDfu" _
'            & "VALUES (ListUcitele.Value,ComboFunkce.Value)"
'       DoCmd.RunSQL strQ
'       ListUcFun.Requery
'    Else
'       MsgBox "Nový záznam o funkci nelze zapsat," & vbCrLf & _
'            "protože údaje jsou neúplné"
'    End If

Exit_CommandPridatFunkci_Click:
    Exit Sub
Err_CommandPridatFunkci_Click:
    MsgBox Err.Description
    Resume Exit_CommandPridatFunkci_Click
End Sub

Private Sub CommandZrusitFunkci_Click()
    On Error GoTo Err_CommandZrusitFunkci_Click
    Dim strQ As String
    If Not IsNull(ListUcFun.Value) Then
        strQ = "DELETE * FROM uc_fun WHERE IDautom=ListUcFun.Value"
        DoCmd.RunSQL strQ
        ListUcFun.Value = Null
        ListUcFun.Requery
    Else
        MsgBox "Nelze nic zrušit, protože žádný záznam" & vbCrLf & _
             "o zastávané funkci nebyl vybrán"
    End If
Exit_CommandZrusitFunkci_Click:
    Exit Sub
Err_CommandZrusitFunkci_Click:
    MsgBox Err.Description
    Resume Exit_CommandZrusitFunkci_Click
End Sub

Private Sub CommandPridatOstatni_Click()
On Error GoTo Err_CommandPridatOstatni_Click
    Dim db As Database, rst As Recordset
    Dim strQ As String, mx As Integer
    Set db = CurrentDb()
    mx = MaxHodnota("uc_ost", "IDautom") + 1
    Label_pom1.Caption = mx
    If Not IsNull(ListUcitele.Value) And Not IsNull(ComboOstatni.Value) _
       And TextPocetOstatni.Value > 0 Then
        strQ = "INSERT INTO uc_ost" _
             & "(IDuc,IDost,pocet,IDautom)" _
& "VALUES (ListUcitele.Value,ComboOstatni.Value,TextPocetOstatni.Value,Label_pom1.Caption)"
        DoCmd.RunSQL strQ
        ListUcOst.Requery
```

```
    Else
        MsgBox "Nový záznam o činnosti nelze zapsat," & vbCrLf & _
            "protože údaje jsou neúplné"
    End If
Exit_CommandPridatOstatni_Click:
    Exit Sub
Err_CommandPridatOstatni_Click:
    MsgBox Err.Description
    Resume Exit_CommandPridatOstatni_Click
End Sub

Private Sub CommandZrusitOst_Click()
On Error GoTo Err_CommandZrusitOst_Click
    Dim strQ As String
    If Not IsNull(ListUcOst.Value) Then
        strQ = "DELETE * FROM uc_ost WHERE IDautom=ListUcOst.Value"
        DoCmd.RunSQL strQ
        ListUcOst.Value = Null
        ListUcOst.Requery
    Else
        MsgBox "Nelze nic zrušit, protože žádný záznam" & vbCrLf & _
            "o ostatní činnosti učitele nebyl vybrán"
    End If
Exit_CommandZrusitOst_Click:
    Exit Sub
Err_CommandZrusitOst_Click:
    MsgBox Err.Description
    Resume Exit_CommandZrusitOst_Click
End Sub

Private Sub CommandOpravitOst_Click()
On Error GoTo Err_CommandOpravitOst_Click
    Dim db As Database, rst As Recordset
    Dim strQ As String
    If Not IsNull(ListUcOst.Value) Then
      Set db = CurrentDb()
      Set rst = db.OpenRecordset("uc_ost")
      rst.Index = "PrimaryKey"
      rst.Seek "=", ListUcOst.Value
'    TextPocetOstatni.Enabled = True
      Label_pom1.Caption = rst!IDautom
      ComboOstatni.Value = rst!IDost
      TextPocetOstatni.Value = rst!pocet
      rst.Close
      ListUcitele.Enabled = False
      TextPocetOstatni.SetFocus   ' změna fokusu, aby se tlačítko opět mohlo znepřístupnit
      CommandSaveChange.Enabled = True
    Else
       MsgBox "Nelze nic opravovat, protože žádný záznam" & vbCrLf & _
            "o ostatní činnosti učitele nebyl vybrán"
    End If
Exit_CommandOpravitOst_Click:
    Exit Sub
Err_CommandOpravitOst_Click:
    MsgBox Err.Description
    Resume Exit_CommandOpravitOst_Click
End Sub

Private Sub CommandSaveChange_Click()
On Error GoTo Err_CommandSaveChange_Click
```

```
    Dim strQ As String
    strQ = "UPDATE uc_ost SET IDost=ComboOstatni.Value, pocet=TextPocetOstatni.Value " & _
        "WHERE IDautom=Label_pom1.Caption"
    DoCmd.RunSQL strQ
    ListUcOst.Requery
    ListUcitele.Enabled = True
    ComboOstatni.SetFocus
    CommandSaveChange.Enabled = False
Exit_CommandSaveChange_Click:
    Exit Sub
Err_CommandSaveChange_Click:
    MsgBox Err.Description
    Resume Exit_CommandSaveChange_Click
End Sub

Private Sub ComboTyp_prcv_AfterUpdate()
    Dim db As Database, rst As Recordset
    Set db = CurrentDb()
    Set rst = db.OpenRecordset("pred_cv")
    rst.Index = "PrimaryKey"
    rst.Seek "=", ComboTyp_prcv.Value
    kkk = rst!koef
    rst.Close
    Select Case ComboTyp_prcv.Value
        Case 1, 7, 11, 15
            LabelCv.Caption = "P"
            Select Case ListPredmety.Value
                Case "1in", "1in-", "ai", "bzi", "0in", "dtx", "fza", "rdb", "rmt", "rps", "scn", _
                    "vci", "vir", "vjc", "vm2", "v2a", "vzp", "vb0", "vd", "vdp", "vr0", "vu0"
                    ComboZk.Value = "  "
                Case Else
                    ComboZk.Value = "zk"
            End Select
        Case 2
            LabelCv.Caption = "Konz"
            ComboZk.Value = "zk"
        Case 3, 8, 12, 16
            LabelCv.Caption = "C1"
            Select Case ListPredmety.Value
                Case "dtx", "fza", "rdb", "rmt", "rps", "scn", _
                    "vci", "vir", "vjc", "vm2", "v2a", "vzp", "vb0"
                    ComboZk.Value = "kz"
                Case "bzi", "0in", "vd", "vdp", "vr0", "vu0"
                    ComboZk.Value = "z"
                Case Else
                    ComboZk.Value = "  "
            End Select
        Case 4, 9, 13, 17
            LabelCv.Caption = "C2a"
            Select Case ListPredmety.Value
                Case "1in", "1in-", "ai", "dtx", "fza", "rdb", "rmt", "rps", "scn", _
                    "vci", "vir", "vjc", "vm2", "v2a", "vzp", "vb0"
                    ComboZk.Value = "kz"
                Case "bzi", "0in", "vd", "vdp", "vr0", "vu0"
                    ComboZk.Value = "z"
                Case Else
                    ComboZk.Value = "  "
            End Select
        Case 5, 10, 14, 18
            LabelCv.Caption = "C2b"
```

```
        Select Case ListPredmety.Value
          Case "1in", "1in-", "ai", "dtx", "fza", "rdb", "rmt", "rps", "scn", _
               "vci", "vir", "vjc", "vm2", "v2a", "vzp", "vb0"
            ComboZk.Value = "kz"
          Case "bzi", "0in", "vd", "vdp", "vr0", "vu0"
            ComboZk.Value = "z"
          Case Else
            ComboZk.Value = "  "
        End Select
      Case 6
        LabelCv.Caption = "sem."
        ComboZk.Value = "  "
  End Select
End Sub

Private Sub CommandSaveUV_Click()
On Error GoTo Err_CommandSaveUV_Click
  Dim strQ As String, mx As Integer, koef_zk As Single
  mx = MaxHodnota("uc_uv", "IDuv") + 1
  Label_pom2.Caption = mx
  Select Case ListTypStudia.Value
    Case "DS"
      Label_pom3.Caption = "51"
    Case "BS"
      Label_pom3.Caption = "81"
    Case "MS"
      Label_pom3.Caption = "11"
    Case "DIS"
      Label_pom3.Caption = "??"
  End Select
  If sss = "zim" Then
    Label_pom4.Caption = TextTydny.Value    ' t_ZS
    Label_pom6.Caption = IIf(Left(LabelCv.Caption, 1) = "C", ccc, ppp) ' h_ZS
  Else
    Label_pom5.Caption = TextTydny.Value     ' t_LS
    Label_pom7.Caption = IIf(Left(LabelCv.Caption, 1) = "C", ccc, ppp) ' h_LS
  End If
  Label_pom8.Caption = kkk
  Select Case ComboZk.Value
    Case "zk"
      koef_zk = 0.7
    Case "kz"
      koef_zk = 0.5
    Case Else
      koef_zk = 0
  End Select
  If IsNull(ComboTyp_prcv.Value) Or (TextSkupiny.Value = 0) Or _
    (TextStudenti.Value = 0) Then
    MsgBox "Nový záznam o výuce nelze zapsat," & vbCrLf & _
           "protože údaje jsou neúplné"
  Else
    Label_pom9.Caption = (Label_pom4.Caption * Label_pom6.Caption + _
         Label_pom5.Caption * Label_pom7.Caption) * TextSkupiny.Value * kkk _
      + TextStudenti.Value * koef_zk
    strQ = "INSERT INTO uc_uv" _
    & "(IDuv,IDuc,IDpred,rocnik,forma_st,t_ZS,t_LS,h_ZS,h_LS," & _
      "IDprcv,typ_prcv,forma_zk,n_skupin,n_stud,koef,sum_radek)" & _
 "VALUES (Label_pom2.Caption,ListUcitele.Value,ListPredmety.Value," & _
      "ListRocnik.Value,Label_pom3.Caption," & _
      "Label_pom4.Caption,Label_pom5.Caption,Label_pom6.Caption,Label_pom7.Caption," & _
```

```
        "ComboTyp_prcv.Value,LabelCv.Caption,ComboZk.Value," & _
        "TextSkupiny.Value,TextStudenti.Value,Label_pom8.Caption,Label_pom9.Caption)"
      DoCmd.RunSQL strQ
      ListUcOst.Requery
   End If
Exit_CommandSaveUV_Click:
   Exit Sub
Err_CommandSaveUV_Click:
   MsgBox Err.Description
   Resume Exit_CommandSaveUV_Click
End Sub

Private Sub CommandKonec_Click()
On Error GoTo Err_CommandKonec_Click
   DoCmd.Close
Exit_CommandKonec_Click:
   Exit Sub
Err_CommandKonec_Click:
   MsgBox Err.Description
   Resume Exit_CommandKonec_Click
End Sub
```

Zdroj řádků seznamu předmětů je reprezentován následujícím SQL dotazem.

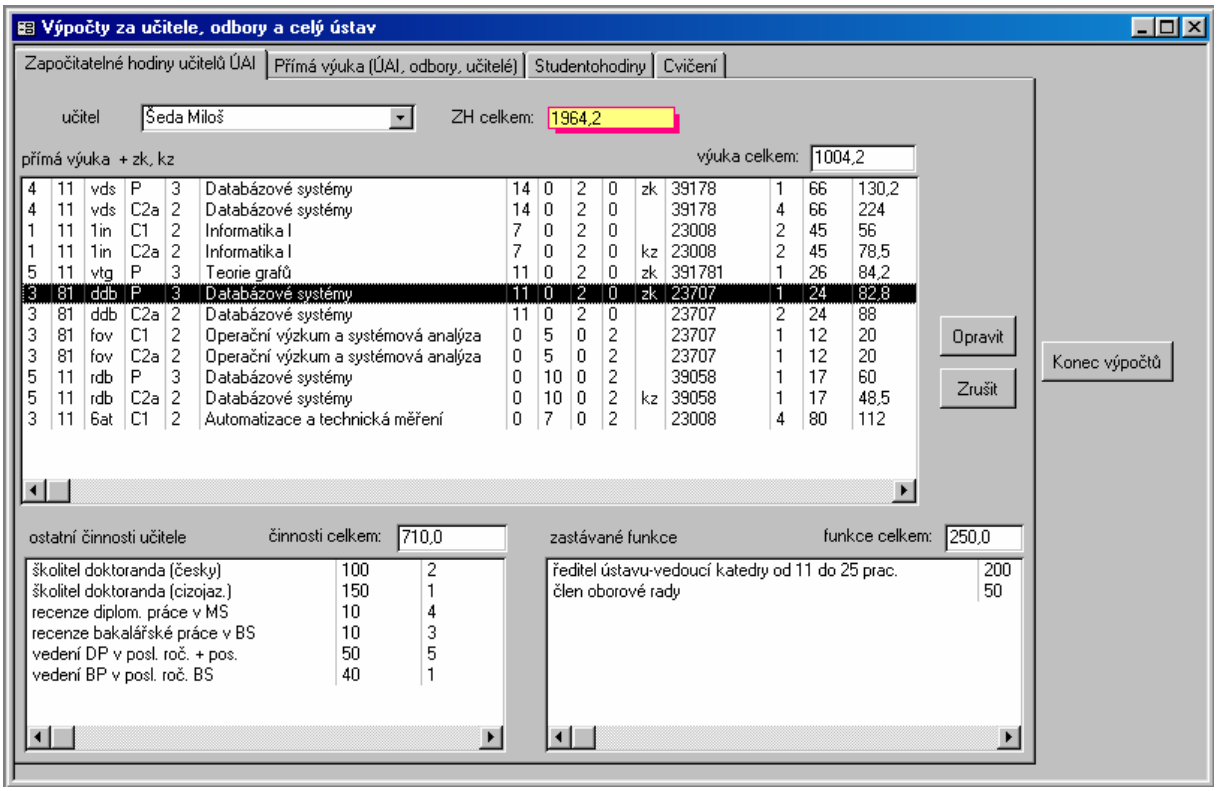
```
SELECT DISTINCTROW [predmety].[kod], [predmety].[rocnik], [predmety].[semestr],
[predmety].[nazev], [predmety].[studium], [predmety].[obor], [predmety].[tydnu],
[predmety].[predn], [predmety].[cvic], [predmety].[typcvic], [predmety].[kredit], [predmety].[zk]
FROM [predmety] WHERE studium=ListTypStudia.Value AND rocnik=ListRocnik.Value;
```

Druhým formulářem, který zde budeme demonstrovat, je formulář, který se otevře po stisku tlačítka Výpočty ZH a je určen k výpočtu započitatelných hodin učitele, odborů ústavu atd. Zatímco v prvním formuláři převládaly výběrové a akční dotazy SQL, ve formuláři z obr. 4.3 vzhledem k jeho funkci jsou hlavní agregační funkce dotazů.

Seznam přímá výuka + zk, kz je plněn dotazem

```
SELECT DISTINCTROW [Uc_UV].[IDuv], [Uc_UV].[rocnik], [Uc_UV].[forma_st],
[Uc_UV].[IDpred], [Uc_UV].[typ_prcv], [Uc_UV].[koef], [predmety].[nazev], [Uc_UV].[t_ZS],
[Uc_UV].[t_LS], [Uc_UV].[h_ZS], [Uc_UV].[h_LS], [Uc_UV].[forma_zk], [predmety].[obor],
[Uc_UV].[n_skupin], [Uc_UV].[n_stud], [Uc_UV].[sum_radek] FROM Uc_UV,predmety
WHERE Uc_UV.IDuc=ComboV_uc.Value AND Uc_UV.IDpred=predmety.kod;
```

Vlastní zdrojový kód událostních procedur prvků tohoto formuláře je uveden za obrázkem 4.3.



Obr. 4.3. Výpočty započitatelných hodin

```
Option Compare Database
Option Explicit

Private Sub Form_Load()
   Me!ComboV_uc = 0
   ListV_ostatni.Requery
   ListV_funkce.Requery
   ListV_vyuka.Requery
End Sub

Private Sub ComboV_uc_AfterUpdate()
   Dim db As Database, rst As Recordset
   Dim a As Single, celkem As Single
   Set db = CurrentDb()
   celkem = 0
   Me!Textsum_cin = 0
   Me!Textsum_fun = 0
   Me!Textsum_vyuka = 0

   Set rst = db.OpenRecordset("SELECT SUM(extra_cin.koef * uc_ost.pocet) " _
            & "AS sum_cin " _
            & "FROM uc_ost,extra_cin " _
            & "WHERE uc_ost.IDuc=" & Me!ComboV_uc & " AND " _
            & "uc_ost.IDost=extra_cin.IDextra", dbOpenSnapshot)
   If IsNull(rst!sum_cin) Then
      a = 0
   Else
      rst.MoveFirst
      a = rst!sum_cin
```

```
      Me!Textsum_cin = Format(a, "###0.0")
   End If
   ListV_ostatni.Requery
   rst.Close
   celkem = celkem + a

   Set rst = db.OpenRecordset("SELECT SUM(funkce.ZH) AS sum_fun " _
            & "FROM uc_fun,funkce " _
            & "WHERE uc_fun.IDuc=" & Me!ComboV_uc & " AND " _
            & "uc_fun.IDfu=funkce.IDfun", dbOpenSnapshot)
   If IsNull(rst!sum_fun) Then
      a = 0
   Else
      rst.MoveFirst
      a = rst!sum_fun
      Me!Textsum_fun = Format(a, "###0.0")
   End If
   ListV_funkce.Requery
   rst.Close
   celkem = celkem + a

   Set rst = db.OpenRecordset("SELECT SUM(uc_UV.sum_radek) AS sum_vyuka " _
            & "FROM uc_UV " _
            & "WHERE uc_UV.IDuc=" & Me!ComboV_uc, dbOpenSnapshot)
   If IsNull(rst!sum_vyuka) Then
      a = 0
   Else
      rst.MoveFirst
      a = rst!sum_vyuka
      Me!Textsum_vyuka = Format(a, "###0.0")
   End If
   ListV_vyuka.Requery
   rst.Close
   celkem = celkem + a

   Me!TextSum_ZH = Format(celkem, "###0.0")

End Sub

Private Sub CommandCelkemOdbory_Click()
On Error GoTo Err_CommandCelkemOdbory_Click
   Dim db As Database
   Dim rstOdb As Recordset, rstUc As Recordset, rst As Recordset
   Dim ZH As Single, ost As Single, fun As Single, vyuka As Single
   Dim celkemUc As Single, celkem As Single
   Dim strQ As String

   strQ = "DELETE * FROM VyslOdbory"
   DoCmd.RunSQL strQ

   Set db = CurrentDb()
   Set rstOdb = db.OpenRecordset("VyslOdbory")
   Set rstUc = db.OpenRecordset("Ucitele")

   rstUc.MoveFirst
   celkem = 0
   Do While Not rstUc.EOF
      celkemUc = 0

      Set rst = db.OpenRecordset("SELECT SUM(uc_UV.sum_radek) AS sum_vyuka " _
```

```
            & "FROM uc_UV " _
            & "WHERE uc_UV.IDuc=" & rstUc!IDped, dbOpenSnapshot)
    If IsNull(rst!sum_vyuka) Then
       vyuka = 0
    Else
       rst.MoveFirst
       vyuka = rst!sum_vyuka
    End If
    rst.Close

    celkemUc = celkemUc + vyuka

    Set rst = db.OpenRecordset("SELECT SUM(extra_cin.koef * uc_ost.pocet) " _
            & "AS sum_cin " _
            & "FROM uc_ost,extra_cin " _
            & "WHERE uc_ost.IDuc=" & rstUc!IDped & " AND " _
            & "uc_ost.IDost=extra_cin.IDextra", dbOpenSnapshot)
    If IsNull(rst!sum_cin) Then
       ost = 0
    Else
       rst.MoveFirst
       ost = rst!sum_cin
    End If
    rst.Close

    celkemUc = celkemUc + ost

    Set rst = db.OpenRecordset("SELECT SUM(funkce.ZH) AS sum_fun " _
            & "FROM uc_fun,funkce " _
            & "WHERE uc_fun.IDuc=" & rstUc!IDped & " AND " _
            & "uc_fun.IDfu=funkce.IDfun", dbOpenSnapshot)
    If IsNull(rst!sum_fun) Then
       fun = 0
    Else
       rst.MoveFirst
       fun = rst!sum_fun
    End If
    rst.Close

    celkemUc = celkemUc + fun

    rstOdb.AddNew
    rstOdb!Iduc = rstUc!IDped
    rstOdb!odbor = rstUc!odbor
    rstOdb!jmeno = rstUc!jmeno
    rstOdb!ZH = vyuka
    rstOdb!ost = ost
    rstOdb!fun = fun
    rstOdb!Uc_celkem = celkemUc
    rstOdb.Update

    rstUc.MoveNext
    celkem = celkem + celkemUc
Loop

rstOdb.Close
rstUc.Close
ListUAI.Requery
ListOdbory.Requery
ListSumyUc.Requery
```

```
Exit_CommandCelkemOdbory_Click:
   Exit Sub
Err_CommandCelkemOdbory_Click:
   MsgBox Err.Description
   Resume Exit_CommandCelkemOdbory_Click
End Sub

Private Sub CommandZrusitVyuku_Click()
On Error GoTo Err_CommandZrusitVyuku_Click
   Dim strQ As String
   If Not IsNull(ListV_vyuka) Then
      strQ = "DELETE * FROM uc_uv WHERE IDuv=ListV_vyuka.Value"
      DoCmd.RunSQL strQ
      ListV_vyuka.Value = Null
      ListV_vyuka.Requery
      Call ComboV_uc_AfterUpdate
   Else
      MsgBox "Nelze nic zrušit, protože žádný záznam" & vbCrLf & _
           "o výuce předmětu nebyl vybrán"
   End If
Exit_CommandZrusitVyuku_Click:
   Exit Sub
Err_CommandZrusitVyuku_Click:
   MsgBox Err.Description
   Resume Exit_CommandZrusitVyuku_Click
End Sub

Private Sub CommandOpravitVyuku_Click()
On Error GoTo Err_CommandOpravitVyuku_Click
   Dim stDocName As String
   Dim stLinkCriteria As String
   If Not IsNull(ListV_vyuka) Then
      stDocName = "OpravaVyuky"
      DoCmd.OpenForm stDocName, , , stLinkCriteria, , , ListV_vyuka
   Else
      MsgBox "Nelze nic opravovat, protože žádný záznam" & vbCrLf & _
           "o výuce předmětu nebyl vybrán"
   End If
Exit_CommandOpravitVyuku_Click:
   Exit Sub
Err_CommandOpravitVyuku_Click:
   MsgBox Err.Description
   Resume Exit_CommandOpravitVyuku_Click
End Sub

Private Sub CommandKonecVypoctu_Click()
On Error GoTo Err_CommandKonecVypoctu_Click
   DoCmd.Close
Exit_CommandKonecVypoctu_Click:
   Exit Sub
Err_CommandKonecVypoctu_Click:
   MsgBox Err.Description
   Resume Exit_CommandKonecVypoctu_Click
End Sub
```

5. Témata projektů

1. Sestavte program, který bude poskytovat informace o odjezdech autobusů z výchozí stanice (např. Brno). Uživatel zadá cílovou stanici a čas, kdy do ní chce přijet. Program mu poskytne informaci o odjezdu autobusu a zda je možné zakoupit jízdenku. V případě, že je autobus obsazen, může uživatel požadovat informaci o následujícím nebo předcházejícím spoji.
2. Sestavte program, který povede evidenci skladu. O každém skladovaném materiálu si program pamatuje číslo, stav zásob a požadovanou minimální zásobu. Uživatel se může informovat o stavu zásob, požadovat seznam materiálu, jehož stav poklesl pod minimální množství. Dále uživatel může požadovat zrušení evidence, případně zaevidování nového materiálu, aktualizaci při dodání, resp. vyexpedování materiálu.
3. Sestavte program na pomoc zdravotním sestrám v nemocnici. Program si pamatuje u každého pacienta jméno, číslo pokoje, druh léku, který užívá, dobu, kdy lék užívá (ráno, v poledne, večer) a množství. Sestra zadá dobu a počítač ji informuje, do kterých pokojů, komu a jaké, léky má přinést. Dále má sestra možnost evidovat nového pacienta, případně zrušit údaje o pacientovi propuštěném z nemocnice.
4. Sestavte program na pomoc čtenáři v knihovně. O každé knize je evidován autor, název knihy, tématická skupina a počet výtisků, které je možno ještě zapůjčit. Čtenář zadá některé z těchto údajů (např. autora, tématický okruh, název, případně kombinace těchto voleb) a počítač mu poskytne seznam knih, které přicházejí v úvahu, a informaci, zda je možné knihu zapůjčit. Dále program umožní zaevidování vypůjčení výtisku a jeho vracení (snížení nebo zvýšení počtu volných výtisků).
5. Program na pomoc diabetikům. Oběd diabetika nesmí překročit určitou energetickou hodnotu. Předpokládejme, že oběd se skládá z polévky, z hlavního jídla, zákusku (resp. salát, kompot) a nápoje. U každého chodu je k dispozici několik variant s různou energetickou hodnotou. Sestavte jídelníček diabetika na týden dopředu. Energetická hodnota jídla přitom musí dosáhnout alespoň 90% určené hodnoty.
6. Sestavte program pro evidenci motorových vozidel. U každého evidovaného vozidla je známa SPZ, druh vozidla, barva vozidla a jeho majitel. Službu konající policista zadá některý z těchto údajů (např. část SPZ a barvu auta, část SPZ a druh vozidla, pouze část SPZ, pouze barvu a všechny další kombinace) a počítač mu zpřístupní seznam vozidel, která odpovídají zadanému popisu, a pro majitele vytiskne předvolání na policii.
7. Sestavte evidenci dárců krve (jméno, krevní skupina, RH-faktor, datum posledního odběru). Obsluha zadá datum, krevní skupinu a požadované množství krve a čeká na seznam vhodných dárců a náhradníků, kteří budou pozváni k odběru. Pozn.: Dárce může darovat 200 ml krve jednou za tři měsíce. Nezapomeňte, že některé skupiny mohou darovat krev i jiným skupinám. Uvažujte i RH-faktor.
8. Sestavte program, který určí výši prospěchového stipendia studentů. Pro každou studijní skupinu uložte abecední seznam studentů spolu s jejich výsledky u jednotlivých zkoušek. Vytiskněte jmenný seznam studentů všech skupin, ve kterém bude u každého jména uvedena skupina, průměrný prospěch a výše stipendia. Umožněte tak, určit nejlepší a nejhorší studenty v jednotlivých skupinách a situaci v každém ročníku.

9. Agenda časopisů. Evidujte časopisecké články tak, že u každého článku si pamatujete jméno časopisu, který ho uveřejnil, datum, obor, klíčová slova. Dávejte uživateli informace o článcích podle oboru, podle klíčových slov, podle časopisu, podle data a poskytujte různé další služby.
10. Sestavte program, který vytvoří katalog cestovní kanceláře. Katalog bude obsahovat tyto údaje: číslo zájezdu, země, místo, počet dní, datum odjezdu, cena, druh dopravního prostředku. Poskytujte uživateli informace o zájezdech podle zadaných požadavků, např. země, přibližného termínu odjezdu, cenových relací, výběru dopravního prostředku ap.
11. Rozhlasová stanice vysílá hitparádu písniček populární hudby. Z dvaceti skladeb posluchači určí 5, které se jim nejvíce líbí. Žebříček popularity bude určen z datového souboru, jehož záznamy obsahují tyto informace: Jméno a příjmení posluchače a pět písniček seřazených podle priority, tomu by odpovídaly body, např. za 1. místo 9 bodů, za 2. místo 6 bodů ap. Program určí nový žebříček popularity a seznam posluchačů v pořadí, nakolik se jejich tipy s ním shodují.
12. Vytvořte program pro tisk výsledků soutěže v krasobruslení. V databázi jsou uloženy tyto údaje: Jméno soutěžícího, stát, který reprezentuje, 9 známek rozhodčích za provedení a 9 známek za umělecký dojem. U každého závodníka se v každé z obou kategorií hodnocení ruší nejvyšší a nejnižší známka a ze zbytku se vytváří průměr. Tiskněte průběžně pořadí na prvních třech místech po každém výsledku a nakonec celkové pořadí.
13. Vytvořte program pro výpočet výsledků v cyklistické časovce jednotlivců. V databázi jsou uloženy tyto údaje: Jméno cyklisty, stát, čas startu a dojezdu do cíle. Vytvořte přehlednou tabulku výsledků, seřazenou podle dosaženého času.
14. V registru svazků bývalé státní bezpečnosti (StB) byly vedeny údaje o spolupracovnících s udáním kategorie jejich spolupráce (agent, důvěrník, kandidát tajné spolupráce apod.). Napište program, který by umožnil takový svazek vytvořit na počítači a poskytoval informace oprávněným uživatelům o spolupracovnících vybrané kategorie. Oprávnění přístupu k datům zabezpečte heslem, které se při vkládání z klávesnice nebude zobrazovat.
15. Sestavte program, který vytvoří trestní rejstřík. V rejstříku budou následující údaje: Jméno a příjmení odsouzen,ho, přečin, za nějž byl odsouzen, doba odnětí svobody, nápravná skupina, místo vykonávání trestu. Umožněte uživateli vypsat informace o osobách odsouzených za určitý druh trestné činnosti, jména recidivistů, kteří byli odsouzeni vícekrát za některý druh zločinu i s jeho uvedením, jména osob, které strávili ve vězení více než zadaný počet let ap.
16. Sestavte program pro registraci kupˇnových knížek. V databázi budou uloženy následující údaje: Jméno a příjmení, bydliště (místo, ulice), rodné číslo, datum zaregistrování, číslo registračního místa. Program musí vyhodnotit pokus o zaregistrování více než jedné knížky jednou osobou. Podle zadaných voleb poskytuje informace např. o celkovém počtu zaregistrovaných, o počtu registrací za určité období, v jednom registračním místě a možné kombinace těchto voleb.
17. Sestavte program pro seznamovací kancelář. V databázi budou údaje o osobách: Jméno a příjmení, pohlaví, věk, vzdělání, výška, váha a požadavky na zvolené charakteristiky partnera. Program by měl každému klientovi vypsat informace o třech nejvhodnějších partnerech opačného pohlaví, přitom se předpokládá, že muž by neměl

být menší než žena, Úroveň vzdělání by se neměla lišit o více než jeden stupeň. Hodnocení charakteristik navrhne programátor.

18. Sestavte program testující úroveň znalostí. Předpokládejme, že lze vybírat ze 3 možností a), b), c). Otázky testu jsou uloženy v záznamech databáze včetně správného řešení a bodového ohodnocení správné odpovědi. Program se dotáže na jméno a zadá otázky testu. Výstupem je zpráva o dosaženém výsledku, např. v procentech z maximálního počtu bodů.

19. Předpokládejme, že máme k dispozici dva datové soubory. V jednom jsou uloženy údaje o objednávkách zahrnující číslo objednávky, název zboží, požadované množství, název dodavatele,datum dodávky. V druhém souboru jsou údaje z dodacích listů obsahující číslo objednávky, název zboží, název dodavatele, dodané množství, datum dodání. Sestavte program, který příslušnému dodavateli napíše urgenci, pokud kontrahované množství určitého zboží není dodáno např. do 14 dnů po dohodnutém termínu dodávky.

20. Sestavte program, který určí výši úroku na spořitelních vkladních knížkách za kalendářní rok. Máme k dispozici dva soubory. První obsahuje tyto informace: Číslo spořitelní knížky, výše vkladu, resp. výběru a datum. Vklad je reprezentován kladným číslem, výběr záporným číslem. Předpokládejme, že 1.1. byl uložen zůstatek z předchozího roku a všechny knížky v souboru mají stejný úrok. Ten se však může průběhu roku měnit podle vývoje inflace a tyto informace jsou uloženy v druhém souboru. Předpokládejme, že první záznam obsahuje hodnotu výchozího úroku k 1.1. a další záznamy hodnotu nového úroku a datum, odkdy byl zaveden.

21. Napište program pro amortizaci materiálu. O každém evidovaném materiálu se udržují tyto informace: Evidenční číslo, druh materiálu, pořizovací cena, hodnota odpisu za jeden rok v procentech z pořizovací ceny, datum posledního odpisu, zůstatková cena. Program v závislosti na momentálním datu a datu posledního odpisu sníží zůstatkovou cenu materiálu. Pokud tato hodnota poklesne pod určitý zlomek z pořizovací ceny, vyřadí materiál z evidence a informaci o tom uloží do druh,ho souboru, který může být např. podkladem pro nabídku partiového prodeje.

6. Kontrolní otázky

I. Návrh datových struktur

Pro následující relační schámata je úkolem provést dekompozici na několik relačních schémat v 3. normální formě tak, aby nedošlo ke ztrátě informace (v případě potřeby je možné výchozí schéma před dekompozicí upravit (rozšířit)):

1. PACIENTI(rodné-č, jméno, příjmení, diagnóza, den-nástupu, č-oddělení, název-oddělení)
2. BYDLIŠTĚ(rodné-č, jméno, příjmení, místo, ulice, ČP, PSČ, městská-čtvrt, okres, kraj)
3. TRESTNÍ-REJSTŘÍK(rodné-č, jméno, příjmení, trestný-čin, paragraf, sazba-od, sazba-do, výše-trestu, nápravná-skupina)
4. AUTA(SPZ, typ, barva, obsah-válců, rok-výroby, prodejní-cena, dovezené{typ boolean}, rodné-č-majitele, jméno, příjmení)
5. VÝPŮJČKA(rodné-č, jméno, příjmení, název-knihy, nakladatelství, autor, ISBN, tématické-zaměření)
6. STUDENTI(rodné-číslo, jméno, příjmení, škola, fakulta, typ-studia, délka-studia, obor, místo)

II. Dotazy v relační algebře

V následujících příkladech jsou dánay2, 3 nebo 4 tabulky a je třeba zformulovat dotaz relační algebry, který z nich určí požadované informace.

1. HEREC(č-herce, jméno, příjmení)
   REPERTOÁR(č-hry, název-hry)
   OBSAZENÍ(č-herce,č-hry)

   Dotazem relační algebry určit všechny herce (jejich jména a příjmení), kteří hrají ve hře ‘Hamlet’.

2. SKLAD(č-skladu, adresa, ...)
   MNOŽSTVÍ(č-skladu, č-součástky, počet-kusů)

   Dotazem relační algebry zjistit adresy všech skladů, kde mají součástku ‘10’ alespoň ve 20 kusech.

3. MAJITEL(rodné-č, jméno, příjmení)
   BARVA(kód-barvy, barva)
   TYP(kód-typu, název-typu)
   MÁ-AUTO(rodné-č, SPZ, kód-typu, kód-barvy)

   Dotazem relační algebry zjistit jména a příjmení všech majitelů modré Felicie z okresu Kroměříž.
   Pozn. Okres Kroměříž identifikujeme úvodními znaky "KM" v číslu auta,
   (v Accessu bychom v této selekci použili zápis LEFT(MÁ-AUTO.číslo, 2) = "KM"

4. LÉKAŘ(č-licence, specializace)
   PACIENT(rodné-č, jméno, příjmení)

NÁVŠTĚVA(rodné-č, č-licence, datum)

Dotazy relační algebry zjistit
a) specializace lékařů, jejichž pomoc vyhledal pacient s rodným číslem 600101/111
b) jméno a příjmení tohoto pacienta

5. UČITEL(os-číslo, jméno, příjmení)
   PŘEDMĚT(kód-př, název-předmětu)
   CO-UČÍ(os-číslo, kód-př)

   Dotazem relační algebry zjistit názvy všech předmětů, které učí učitel ‘Karásek’.
   (Předpokládáme, že neexistují 2 učitelé se stejným jménem.)

III. SQL

V následujících příkladech je úkolem zjistit z daných tabulek požadované informace pomocí jazyka SQL.

1. UČITELÉ(č-uč, jméno, příjmení, č-odboru)
   ODBORY(č-odboru, název-odboru)

   Dotazem SQL zjistit názvy všech odborů a počty učitelů na nich zařazených.

2. ZAMĚSTNANCI(rodné-č, jméno, příjmení)
   DĚTI(rodné-č-rodiče, jméno-dítěte)

   Dotazem SQL zjistit všechny bezdětné zaměstnance.

3. PRODEJCI(č-prodejce, jméno, příjmení)
   FAKTURY(č-faktury, č-zboží, počet-kusů, datum)
   ZBOŽÍ(č-zboží, cena-za-kus)

   Dotazem SQL zjistit všechny prodejce a celkové částky, které jimi byly vyfakturovány.

4. HEREC(č-herce, jméno, příjmení)
   REPERTOÁR(č-hry, název-hry)
   OBSAZENÍ(č-herce,č-hry)

   Dotazem SQL určit, kolik herců je obsazeno ve hře ‘Hamlet’.

5. HEREC(č-herce, jméno, příjmení)
   REPERTOÁR(č-hry, název-hry)
   OBSAZENÍ(č-herce,č-hry)

   Dotazem SQL určit všechny herce (jejich jména a příjmení), kteří hrají ve hře ‘Hamlet’ a uspořádat je v abecedním pořadí.

6. SKLAD(č-skladu, adresa, ...)
   MNOŽSTVÍ(č-skladu, č-součástky, počet-kusů)

   Dotazem SQL zjistit adresy všech skladů, kde mají součástku ‘10’ alespoň ve 20 kusech

7. MAJITEL(rodné-č, jméno, příjmení)

BARVA(kód-barvy, barva)
TYP(kód-typu, název-typu)
MÁ-AUTO(rodné-č, SPZ, kód-typu, kód-barvy)

Dotazem SQL zjistit jména a příjmení všech majitelů modré Felicie.

8. LÉKAŘ(č-licence, specializace)
   PACIENT(rodné-č, jméno, příjmení)
   NÁVŠTĚVA(rodné-č, č-licence, datum)

   Dotazem SQL zjistit specializace lékařů, jejichž pomoc vyhledal pacient s rodným číslem 600101/111

9. LÉKAŘ(č-licence, specializace)
   PACIENT(rodné-č, jméno, příjmení)
   NÁVŠTĚVA(rodné-č, č-licence, datum)

   Dotazem SQL zjistit datum všech návštěv pacienta s rodným číslem 600101/111

10. UČITEL(os-číslo, jméno, příjmení)
    PŘEDMĚT(kód-př, název-předmětu)
    CO-UČÍ(os-číslo, kód-př)

    Dotazem relační algebry zjistit názvy všech předmětů, které učí učitel ‘Karásek’. (Předpokládáme, že neexistují 2 učitelé se stejným jménem.)

IV. Visual Basic pro aplikace Accessu

1. Napsat proceduru ve Visual Basicu, která z tabulky OSOBY1(rodné-č, jméno, příjmení, místo, ulice, čp, PSČ) překopíruje všechny záznamy osob bydlících v Brně do tabulky OSOBY2 se stejnou strukturou. Úlohu vyřešte bez použití SQL s pomocí objektů typu recordset.
2. Napsat proceduru ve Visual Basicu, která z tabulky skladových položek SKLAD(ID, název, počet-kusů) překopíruje do tabulky OBJEDNAT(ID, název) všechny položky, jejichž počet kusů na skladě je nulový. Úlohu vyřešte bez příkazu SQL s využitím objektů typu recordset.
3. Napsat proceduru ve Visual Basicu, která do tabulky OSOBY1(rodné-č, jméno, příjmení) přidá všechny řádky tabulky OSOBY2(rodné-č, jméno, příjmení). Úlohu vyřešte bez použití SQL s pomocí objektů typu recordset.
4. Napsat funkci ve Visual Basicu, jejíž návratovou hodnotou je název uměleckého díla, za které byla v dražbě nabídnuta nejvyšší cena. Jsou k dispozici 2 tabulky: DÍLA(ID, název), NABÍDKY(ID, cena, zájemce). Úlohu vyřešte bez použití SQL s pomocí objektů typu recordset.
5. Napište funkci ve Visual Basicu, jejíž návratovou hodnotou je rodné číslo první osoby z tabulky ZÁKAZNÍCI(rodné-č, jméno, příjmení), která v tabulce OBJEDNÁVKY(rodné-č, zboží-č, rok, …) nemá žádný záznam v roce 2000. Úlohu vyřešte bez použití SQL s pomocí objektů typu recordset.
6. Napište funkci ve Visual Basicu, jejíž návratovou hodnotou je, kolik procent zákazníků z tabulky ZÁKAZNÍCI(rodné-č, jméno, příjmení) nemá žádný letošní záznam v tabulce

OBJEDNÁVKY(rodné-č, zboží-č, rok, …). Úlohu vyřešte bez použití SQL s pomocí objektů typu recordset.

7. Napište funkci ve Visual Basicu, jejíž návratovou hodnotou je rodné číslo čtenáře z tabulky ČTENÁŘI(rodné-č, jméno, příjmení), který v tabulce VÝPŮJČKY(rodné-č, ISBN) má největší počet záznamů. Úlohu vyřešte bez použití SQL s pomocí objektů typu recordset.
8. Napište funkci ve Visual Basicu, jejíž návratovou hodnotou je pořadí prvního řádku, na kterém se dvě tabulky stejné struktury a stejného počtu řádků liší v hodnotě klíčového pole ID. Jestliže všechny odpovídající si řádky mají stejné hodnoty klíče, pak funkce vrátí hodnotu 0. Úlohu vyřešte bez použití SQL s pomocí objektů typu recordset.

Literatura

[1] Benyon-Davies, P.: *Database Systems*. Macmillan Press, London, 1996. ISBN 0-333-63667-8.

[2] Bíla, J., Král, F.: *Databázové a znalostní systémy*. Skriptum ČVUT FS, Praha, 1999.

[3] Date, C.J.: *An Introduction to Database Systems*. Addison-Wesley, New York, (6th edition), 1995.

[4] Duží, M.: *Konceptuální modelování – datový model HIT*. Slezská univerzita, Ostrava, 2000.

[5] Elmasri, R. and Navathe, S.B.: *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, Amsterdam, (2nd edition), 1997.

[6] Farana, R.: *Tvorba relačních databázových systémů*. VŠB TU, Ostrava, 1999.

[7] Farana, R.: *Aplikace počítačů v řízení. Relační databáze.* VŠB TU, Ostrava, 1995.

[8] Fortier, P.J.: *Database Systems Handbook*. McGraw-Hill, 1997, ISBN 0-07-021626-6.

[9] Havlát, T., Benešovský, M.: *Úvod do databázových systémů*. Skriptum UJEP PřF, Brno, 1984.

[10] McCullough-Dieter, C.: *Mistrovství v Oracle 8*. Computer Press, Praha, 1999.

[11] Pokorný, J.: *Visual Basic pro aplikace Accessu 2000*. Kopp, České Budějovice, 2000.

[12] Pokorný, J.: *Office 97 a Internet*. Kopp, České Budějovice, 1997.

[13] Pokorný J.: *Učíme se SQL*. PLUS, Praha, 1993.

[14] Pokorný, J.: *Konstrukce databázových systémů*. Skriptum ČVUT FEL, Praha, 1999.

[15] Pokorný, J., Halaška, I.: *Databázové systémy. Vybrané kapitoly a cvičení*. Karolinum, Praha, 1993.

[16] Pokorný, J., Halaška, I.: *Databázové systémy. Vybrané kapitoly a cvičení*. Karolinum – nakladatelství Univerzity Karlovy, Praha, 1998.

[17] Straka, M.: *Vývoj databázových aplikací*. Grada, Praha, 1992.

[18] Šimůnek, M.: *SQL - kompletní kapesní průvodce*. Grada, Praha, 1999.

[19] Viescas, J.: *Mistrovství v Microsoft Access 2000*. Computer Press, Praha, 2000.

Poděkování:

Autor děkuje za podporu z Projektu rozvoje bakalářských programů na Fakultě strojního inženýrství VUT v Brně a za možnost začlenit do textu odborné poznatky získané při řešení výzkumného záměru CEZ J22/98: 261100009 „Netradiční metody studia komplexních a neurčitých systémů“.