Masarykova univerzita v Brně
Fakulta informatiky

Relační datové modely v informačních systémech

Brno, duben 1996

Pavel Rychlý

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a že jsem uvedl všechny literární prameny, kterých jsem v práci využil.

V Brně 11. dubna 1996

Za odborné vedení, poskytnutou literaturu, cenné rady a připomínky děkuji vedoucímu diplomové práce Ing. Jiřímu Sochorovi, CSc.

Obsah

Úvod

Význam informačních systémů pro obor informačních technologií je značný. Na souhrné produkci softwaru se podle statistik tvorba informačních systémů podílí drtivou většinou. *Informační systém* (IS) je systém hromadného zpracování dat. Většinou se jedná o obchodní aplikace, ve kterých jsou z jednoduchých datových elementů (řetězec znaků, číslo, datum) tvořeny složité struktury. S těmito strukturami se provádí opět jednoduché operace.

Vývoj libovolného složitějšího systému vyžaduje použití modelu. *Model* je schematické popsání reality, které má usnadňovat komunikaci mezi zákazníkem a tvůrcem a mezi členy programátorského týmu. Pro vývoj větších IS se vytvářejí dva základní modely: datový a funkční. Datový model je úzce spojen s cílovým databázovým systémem. Dříve se používaly modely hierarchické a síťové, nyní to jsou modely relační a objektové. Pro klasické obchodní IS se jako nejvýhodnější stále jeví *relační datový model*.

Kapitola 1 popisuje relační datový model ve své podstatě. Nejdříve uvádí základní principy relačního modelu, potom formální popis jeho normálních forem a relační algebry. Také jsou zde uvedeny některé modifikace relačního modelu.

O reprezentaci relačního modelu pojednává kapitola 2. Protože *jeden obrázek je lepší než tisíce slov*, reprezentace to je grafická — entitně-relační diagram. Jsou zde uvedeny součásti klasických entitně-relačních diagramů a jejich nejpoužívanější modifikace. Ze závěrů této kapitoly vycházejí následující dvě kapitoly, ve kterých byl učiněn pokus zavést do entitně-relačních diagramů nové prvky a postupy. Tyto doplňky mohou ještě více usnadnit vývoj datových modelů pro informační systémy.

Kapitola 3 popisuje metody zakrytí informace v entitně-relačních diagramech. Jedná se zejména o postupnou specifikaci jednotlivých prvků diagramu, která je doplněna hierarchickým uspořádáním diagramů a specifickým zobrazením prvků v diagramu.

Kapitola 4 se zabývá funkcemi, které lze provádět nad diagramy. Nejdříve jsou teoreticky rozebrány možnosti generování z diagramů a transformace diagramů, potom jsou uvedeny příklady pro obě možnosti.

V praktické části byl vytvořen nástroj, program KREDI, který umožňuje demonstrovat to, co je popsáno v teoretické části. Jedná se o univerzální otevřený systém umožňující práci s libovolným typem diagramu. V kapitole 5 jsou popsány jeho funkce a v kapitole 6 je nastíněna jeho implementace. Kapitola 7 uvádí další možná rozšíření programu.

1 Relační datový model

První návrh relačního datového modelu (RDM) vytvořili E. F. Codd a C. J. Date ve firmě IBM na počátku 70. let. Základní principy uvedeme v následujícím odstavci.

Přestože programová implementace relační databáze není příliš složitá, v praxi se RDBS rozšířily až počátkem 80. let. Bylo to způsobeno výpočetní náročností operací nad relacemi. Dřívější počítače nebyly dostatečně výkonné, aby tyto nároky pro rozsáhlé systémy zvládly. Dalším faktorem, který umožnil přechod k RDBS, bylo prozkoumání efektivních třídících a vyhledávacích algoritmů a struktur, zejména B-stromů. Rozšíření relačních databázových systémů potom znamenalo asi největší průlom v používání databází v celé jejich historii.

1.1 Princip relačního modelu

Relační datový model se opírá o tři základní myšlenky:

1. Oddělení fyzického a logického pohledu na data.

   Při vytváření relační databáze se tedy samostatně popíše logická struktura databáze (tj. datový model) a až při vlastní implementaci vznikne fyzická struktura databáze. Tento fyzický popis lze provést buď ručním přepisem, nebo generováním za použití počítače. Datový model je tedy implementačně nezávislý.

2. Stejné a jednoduché operace nad daty.

   Každý objekt v relačním modelu je reprezentován relací, nad kterou lze provádět vždy stejné operace (viz 1.3). Jiným přístupem je objektově orientované programování, kdy pro každý objekt je potřeba explicitně uvést přípustné operace (metody).

3. Neprocedurální jazyk pro manipulaci s daty.

   Pro operace s daty a se schématem databáze se používá jednoduchý neprocedurální jazyk (viz 1.3.1).

1.2 Normální formy

Za normální formu (NF) se vždy považuje nějaký „pěkný“ tvar, se kterým se lépe pracuje. Dále je tento tvar buď jednoznačný, nebo alespoň „jednoznačnější“ než jeho jiný ekvivalent.

V první práci Codda [2] o relačních modelech jsou popsány první tři normální formy. Další vznikaly postupně později a objevují se stále vyšší. Až do třetí NF existuje algoritmický přechod od jedné normální formy k vyšší.

Hlavní důvody transformace datového modelu do vyšších normálních forem jsou:

- odstranění nežádoucích závislostí při vkládání, modifikaci a rušení relací,
- vytvoření relačního modelu, který je pro uživatele i programátora jednodušší na pochopení,
- snížení potřeby měnit strukturu databáze při zavádění nových typů dat (atributů) a tím prodloužení životnosti aplikačních programů.

Normální formy vycházejí z formálních matematických definic. Jak je uvedeno například v [1, str. 178], lze neformálně jednotlivé normální formy popsat takto:

První normální forma
: Hodnoty atributů jsou atomické.

Druhá normální forma
: Požadavky první NF a navíc každý neklíčový atribut popisuje to, co je identifikováno pouze daným klíčem.

Třetí normální forma
: Požadavky druhé NF a navíc každý neklíčový atribut závisí pouze na klíči a není pouze dalším popisem jiného neklíčového atributu.

Čtvrtá normální forma
: Požadavky třetí NF a navíc hodnoty dvou nebo více neklíčových atributů se nikdy nemapují na jiný neklíčový atribut.

Pátá normální forma
: Požadavky třetí NF a navíc hodnoty dvou nebo více neklíčových atributů se nikdy nemapují na jiný neklíčový atribut ani při realizaci operace spojení.

Na tomto místě je třeba podotknout, že čtvrtá a pátá normální forma již nemají svoji formální definici.

1.2.1 Základní definice

Definice: *Relační schéma* je systém $R = (\{A_1, A_2, \ldots, A_n\}, f)$, kde $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ je množina jmen atributů, $f : \mathcal{A} \to \mathcal{D}$, $f(A_i) = D_i$, $D_i \neq \emptyset$ jsou domény atributů.

Definice: Nechť $\mathcal{R} = (\mathcal{A}, f), \mathcal{X}, \mathcal{Y} \subseteq \mathcal{A}$. Pak $\mathcal{Y}$ je *funkčně závislá na* $\mathcal{X}$ (píšeme $\mathcal{X} \to \mathcal{Y}$), jestliže pro každou instanci R relačního schématu $\mathcal{R}$ platí: mají-li dva prvky relace R stejné hodnoty na atributech $\mathcal{X}$, pak mají stejné hodnoty i na $\mathcal{Y}$.

Definice: Nechť $\mathcal{F}$ je množina funkčních závislostí pro relační schéma $\mathcal{R} = (\mathcal{A}, f)$. Nechť $\mathcal{X} \rightarrow \mathcal{Y}$ je funkční závislost. Řekneme, že $\mathcal{F}$ *logicky implikuje* $\mathcal{X} \rightarrow \mathcal{Y}$, jestliže v každé instanci R schématu $\mathcal{R}$, v nichž jsou splněny závislosti z $\mathcal{F}$, je také splněna $\mathcal{X} \rightarrow \mathcal{Y}$.

Množina všech funkčních závislostí, které jsou logicky implikovány $\mathcal{F}$, říkáme *uzávěr* $\mathcal{F}$ a značíme $\mathcal{F}^+$

.

Definice: Nechť $\mathcal{R} = (\mathcal{A}, f)$, $\mathcal{F}$ množina funkčních závislostí, $\mathcal{X} \subseteq \mathcal{A}$.

Řekneme, že $\mathcal{X}$ je *klíč* schématu $\mathcal{R}$ jestliže:

1. $\mathcal{X} \rightarrow \{A_1, A_2, \ldots, A_n\} \in \mathcal{F}^+$,
2. $\forall \mathcal{Y} \subset \mathcal{X} : \mathcal{Y} \rightarrow \{A_1, A_2, \ldots, A_n\} \notin \mathcal{F}^+$.

1.2.2 První normální forma

První normální forma není ještě zcela formálně definována. V podstatě nám říká, že se jedná o relační model, tedy domény všech atributů relací jsou dále nedělitelné. Žádný atribut nemá svoji vnitřní strukturu, není tvořen opakováním nebo sjednocením dalších hodnot.

1.2.3 Druhá normální forma

Definice: Nechť jsou $\mathcal{X}$, $\mathcal{Y}$ podmnožiny množiny atributů relačního schématu $\mathcal{R}$ a $\mathcal{F}$ je množina funkčních závislostí.

Řekneme, že $\mathcal{Y}$ *úplně závisí* na $\mathcal{X}$, jestliže $\mathcal{X} \rightarrow \mathcal{Y}$ pro všechny $\mathcal{X}'$ taková, že $\mathcal{X}' \subset \mathcal{X} \Rightarrow \mathcal{X}' \not\rightarrow \mathcal{Y}$.

Definice: Atribut relačního schématu se nazývá *primární*, je-li součástí alespoň jednoho klíče.

Definice: Relační schéma je ve *druhé normální formě*, jestliže je v 1. NF a každý atribut, který není primární, je úplně závislý na každém klíči.

1.2.4 Třetí normální forma

Definice: Nechť $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ jsou podmnožiny množiny atributů relačního schématu $\mathcal{R}$ s funkčními závislostmi $\mathcal{F}$ a platí:

$$\mathcal{X} \cup \mathcal{Z} = \emptyset,\ \mathcal{Y} \cup \mathcal{Z} = \emptyset,\ \mathcal{Y} \not\subseteq \mathcal{X}.$$

Je-li

$$\mathcal{X} \rightarrow \mathcal{Y} \in \mathcal{F}^+,\ \mathcal{Y} \rightarrow \mathcal{Y} \in \mathcal{F}^+ \text{ a } \mathcal{Y} \rightarrow \mathcal{X} \notin \mathcal{F}^+,$$

pak

$$\mathcal{X} \rightarrow \mathcal{Z} \in \mathcal{F}^+ \text{ a } \mathcal{Z} \rightarrow \mathcal{X} \notin \mathcal{F}^+$$

a říkáme, že $\mathcal{Z}$ je *tranzitivně závislé* na $\mathcal{X}$.

Definice: Relační schéma je ve *třetí normální formě*, jestliže je v 2. NF a žádný atribut, který není primární, není tranzitivně závislý na žádném klíči.

1.2.5 Boyce-Coddova normální forma

Definice: Relační schéma je v *Boyce-Coddově* NF (BCNF), jestliže pro každé $\mathcal{X}$ a $\mathcal{A}$ taková, že $\mathcal{X} \rightarrow \mathcal{A}$ a $\mathcal{A} \not\subseteq \mathcal{X}$, obsahuje $\mathcal{X}$ klíč $\mathcal{R}$.

Sporem přímo z jednotlivých definic lze dokázat, že relační schéma, které je v BCNF, je v 3. NF.

V praxi se nicméně ukazuje, že pro dosažení efektivity dotazů je nutné vyšší normální formy porušit. V reálných systémech proto velice zřídka narazíme na datový model, který by byl dokonce jen v třetí normální formě.

1.3 Relační algebra

Relační algebra slouží k provádění operací s matematickým modelem relací. V tomto případě tedy chápeme *relaci* jako množinu uspořádaných n-tic, mluvíme také o relaci stupně n.

Operace realční algebry můžeme logicky rozdělit do dvou skupin:

- množinové operace,
- operace specifické pro relace.

Množinové operace lze provádět s libovolnými množinami, tím spíše s relacemi. K těmto operacím řadíme:

sjednocení relací R a S stejného stupně
$R \cup S := \{t | \, t \in R \vee t \in S\}$

průnik relací R a S stejného stupně
$R \cap S := \{t | \, t \in R \wedge t \in S\}$, tedy $R \cap S = R - (R - S)$

rozdíl relací R a S stejného stupně
$R - S := \{t | \, t \in R \wedge t \notin S\}$

součin relací R a S
$R \times S := \{rs | \, r \in R \wedge s \in S\}$, $rs = (r_1, r_2, \ldots, r_n, s_1, s_2, \ldots, s_m)$

Druhá skupina operací již předpokládá, že prvky relací jsou výše zmíněné uspořádané n-tice. Jedná se o následující operace:

projekce relace R na množinu atributů $\mathcal{A}$
$\mathcal{A} = (i_1, \ldots, i_m),\ 1 \le i_j \le n, \quad i_j \ne i_k$ pro $j \ne k$

$$R[\mathcal{A}] := \{r[\mathcal{A}] | r \in R\},$$

kde $r[\mathcal{A}] = (r_{i_1}, r_{i_2}, \ldots, r_{i_m})$

výběr (restrikce, selekce) relace R stupně n

$$R[\phi] := \{r|\ r \in R\ \wedge\ \phi(r)\},$$

kde ϕ je totální predikát na $r \in R$

dělení relace R stupně m a relace S stupně n,

$$A = (i_1, i_2, \ldots, i_k), \quad i_j \in \{1, 2, \ldots, m\}, \quad i_j \ne i_l \text{ pro } l \ne j,$$

$$B = (g_1, g_2, \ldots, g_k), \quad g_j \in \{1, 2, \ldots, n\}, \quad g_j \ne g_l \text{ pro } l \ne j,$$

$$\bar{A} = (j_1, j_2, \ldots, j_{m-k}), \quad j_p \in \{1, 2, \ldots, m\} - A, \quad j_p < j_l \text{ pro } p < l.$$

Dále definujeme

$$im_R(r[\bar{A}]) := \{y|\ r[\bar{A}]y \in R[\bar{A}, A]\}.$$

Potom $R[A\!:\!B]S$ dělení relace R na A relací S na B definujeme:

$$R[A\!:\!B]S := \{r[\bar{A}]|\ r \in R\ \wedge\ S[B] \subseteq im_R(r[\bar{A}])\}.$$

podíl relace R stupně m a relace S stupně n, $m > n$, $S \ne \emptyset$

$$R\!:\!S := R[A : B]S,$$

kde $A = (m - n + 1, m - n + 2, \ldots, m)$, $B = (1, 2, \ldots, n)$

spojení relací R a S pomocí operace θ na i-té komponentě R a na j-té komponentě S, relace R je stupně m, relace S je stupně n, $i \in \{1, 2, \ldots, m\}$, $j \in \{1, 2, \ldots, n\}$, $\theta \in \{<, =, >, \le, \ne, \ge\}$

$$R[i\,\theta\,j]S := \{rs|\ r \in R\ \wedge\ s \in S\ \wedge\ r[i]\,\theta\,s[j]\},$$

tedy $R[i\,\theta\,j]S = (R \times S)[i\,\theta\,(m + j)]$.
Častěji se používá speciální případ zapisovaný $R[i * j]S$, který je definován jako:

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

přirozené spojení $R : (A_1, A_2, \ldots, A_m), S : (B_1, B_2, \ldots, B_n)$. Nechť dále $(C_1, C_2, \ldots, C_k)$ jsou navzájem různé a vyskytují se jak v R tak v S. $\forall h :\ 1 \leq h \leq k,\ \exists i, j\ (C_h = A_i\ \wedge\ C_h = B_j)\ \wedge\ \forall g \neq h :\ C_h \neq C_g$.

$(A_1, A_2, \ldots, A_m, D_1, D_2, \ldots, D_n)$ jsou atributy $R \times S$, kde

$$D_i = \begin{cases} S.B_i, & \text{pokud } \exists h : B_i = C_h \\ B_i, & \text{jinak} \end{cases}$$

Označme $(E_1, E_2, \ldots, E_{n-k})$ atributy z $(D_1, D_2, \ldots, D_n)$ s vypuštěním přejmenovaných $(S.X)$.

$$R[*]S := ((R \times S)[\phi])[A_1, A_2, \ldots, A_m, E_1, E_2, \ldots, E_{n-k}],$$

kde $\phi :\ C_1 = S.C_1\ \wedge\ C_2 = S.C_2\ \wedge\ \ldots\ \wedge\ C_k = S.C_k$

1.3.1 SQL

Na základě relační algebry byl ve firmě IBM vyvinut jazyk *SQL* (*Structured Query Language*), pro manipulaci s relacemi v databázových systémech. Tento jazyk byl navržen tak, aby i dotazy zadané nezkušeným programátorem mohly být optimalizovány a efektivně vyhodnoceny. Jazyk SQL byl později podporován (s různými modifikacemi) většinou producentů velkých databází jako *de facto standard*. V současné době je již i formálně standardizován normou ISO (SQL-92) a pracuje se na standardu jeho objektově orientovaného rozšíření.

V terminologii databází se relacím říká *tabulky* a atributům *sloupce*. Operace relační algebry jsou zakomponovány do jediného příkazu SQL, příkazu `select`. Ten v sobě zahrnuje pouze operace *projekce*, *výběru* a *spojení*.

Jazyk SQL dále zahrnuje příkazy pro manipulaci s daty (`insert`, `update`, `delete`), příkazy pro definici dat (`create`, `alter`, `drop`) a příkazy pro správu přístupových práv (`grant`, `revoke`).

1.4 Modifikace RDM

Modifikace relačního datového modelu se mohou týkat jedné ze dvou oblastí:

- matematické – formální, teoretické;
- praktické – databázové.

Nejvíce propracovanou teoretickou prací rozšiřující původní myšlenky relací jsou tzv. *ohebné relace* (*Flexible Relations*). Jak je ukázáno v [4], podařilo se autorům modelovat nestejnorodou strukturu atributů v rámci jedné relace, a to při zachování všech výhod původního relačního modelu. Navíc je dokázáno, že operace nad jistou podmnožinou ohebných relací lze implementovat stejně efektivně jako nad běžnými relacemi.

Modifikace na poli praktickém provádí v podstatě každý výrobce relačního databázového systému. Zvláště v poslední době, s nástupem objektově orientované technologie, to je porušení atomičnosti atributů. Hodnotou atributu potom může být nejenom dlouhý text či obrázek, ale například i celá tabulka (relace).

Dále se v databázích používají různé „technické detaily", které nejsou součástí matematického relačního modelu. Jedna skupina nástrojů se snaží řešit problémy redundance a konzistence v databázích. K tomuto účelu se používají zejména transakční zpracování a přidávání trigrů do databáze. Další skupinou jsou nástroje pro zefektivnění práce databáze. Jedná se např. o indexy, clustry atd.

Výše zmíněné prostředky se různí s každým dodavatelem relační databáze a přesto, že jde jen o „technické detaily", nebylo by možno bez nich vytvořit větší spolehlivou databázi.

1.5 Zhodnocení RDM

Hlavní výhody relačního modelu lze shrnout do několika následujících bodů:

- jednoduchost, formální teorie
  Teorie datových struktur a vlastních dat přináší jejich nezávislost a přehlednost. Oddělení fyzického a logického pohledu na data umožňuje poměrně vysokou implementační nezávislost.

- vyspělost
  Po teoretické stránce byli publikovány snad všechny potřebné práce, týkající se jak formálního modelu (normální formy apod.), tak implementačních algoritmů (zejména vyhledávací algoritmy). Po praktické stránce jsou současné relační databázové systémy roky prověřované a úspěšně používané v mnoha rozsáhlých aplikacích.

- standard
  Jazyk pro práci s relačními databázemi (SQL) je standardizován, tento standard je široce známý a používaný.

Z tohoto hodnocení vyplývá, že relační databáze se velice dobře hodí pro informační systémy. Zejména to platí v obchodní oblasti, kde nevadí hlavní nedostatek relačních databází, tedy atomičnost atributů.

Díky rozvoji hardware, software a poklesu cen počítačů (hlavně osobních) se stále častěji objevují snahy používat databáze nejen pro zpracování jednoduše strukturovaných obchodních dat, ale i složitě strukturovaných dat. Jde např. o aplikace v oborech CAD, GIS, CASE. Na tyto úkoly již klasický relační model nestačí a jako výhodnější řešení se jeví spíše objektově orientované databáze.

2 Entitně relační diagramy

Jak jsme si ukázali v předchozí kapitole, použití RDM může být velice výhodné pro návrh nejrůznějších systémů. Otázkou ovšem zůstává, jak daný relační model znázornit. Právě k tomuto účelu slouží entitně relační diagramy (ERD).

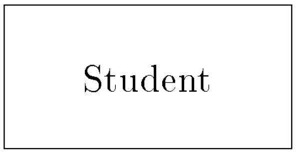
2.1 Součásti ERD

Všechny typy entitně relačních diagramů se skládají, jak samotný název ukazuje, z *entit* a *relací*. Různé modifikace E-R digramů navíc obsahují symboly pro: *atribut*, *asociační entitu*, *subtyp/supertyp* a jiné.

2.1.1 Entita

Entita je v E-R diagramu reprezentována obdélníkem, ve kterém je uvedeno jméno entity (obrázek 1). Reprezentuje skupinu nebo množinu objektů (věcí) v reálném světě, jejíž jednotlivé prvky mají následující charakteristiku:

- každý je jednoznačným způsobem identifikován,
- každý hraje určitou roli v modelovaném systému,
- každý může být popsán jedním nebo několika datovými elementy (atributy).



Obrázek 1: Entita

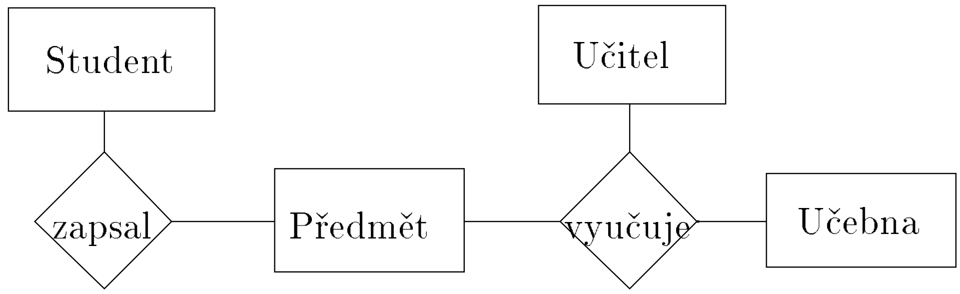
2.1.2 Relace

Entity jsou mezi sebou navzájem propojeny relacemi. Relace reprezentuje množinu vazeb mezi entitami a je znázorněna kosočtvercem. Unvitř kosočtverce je uveden název relace. Kosočtverec je čárami spojen s reprezentací entit, které propojuje. Na těchto čarách může být navíc explicitně vyjádřena maximální kardinalita relace a to buď přímo číselnou, nebo znakovou hodnotou (1, n), nebo za tímto účelem zvolenými symboly.

Je důležité, aby relace reprezentovala *množinu* vazeb. Každá instance relace potom reprezentuje spojení jednoho, nebo více výskytů jednoho objektu (entity) a žádného, nebo více výskytů druhého objektu (entity). Relace také může spojovat dva, nebo více výskytů jediného objektu.

Také je nutné, aby relace reprezentovala vztah, který je potřeba uchovávat v modelovaném systému. Nemůže to být něco, co lze spočítat nebo mechanicky odvodit z jiných dat.

V některých modelech může být mezi dvěma entitami více různých relací. Jinde může jedna relace vyjadřovat vazbu mezi více než dvěmi relacemi. Obě situace jsou vidět na obrázku 2.

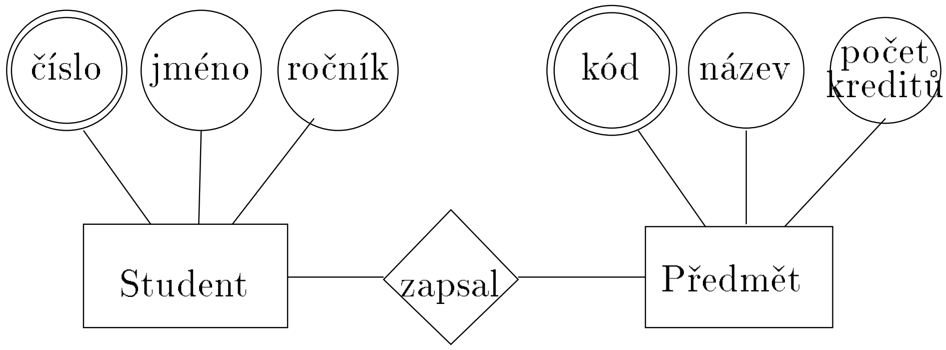


Obrázek 2: Relace

2.1.3 Atribut

Atribut se znázorňuje kruhem se jménem uvnitř a reprezentuje jeden datový element zvolené entity. V některých diagramech jsou navíc určitým způsobem zvýrazněny klíčové atributy (obrázek 3). Diagramy, které obsahují zobrazení atributů, se někdy označují jako E-R-A diagramy.

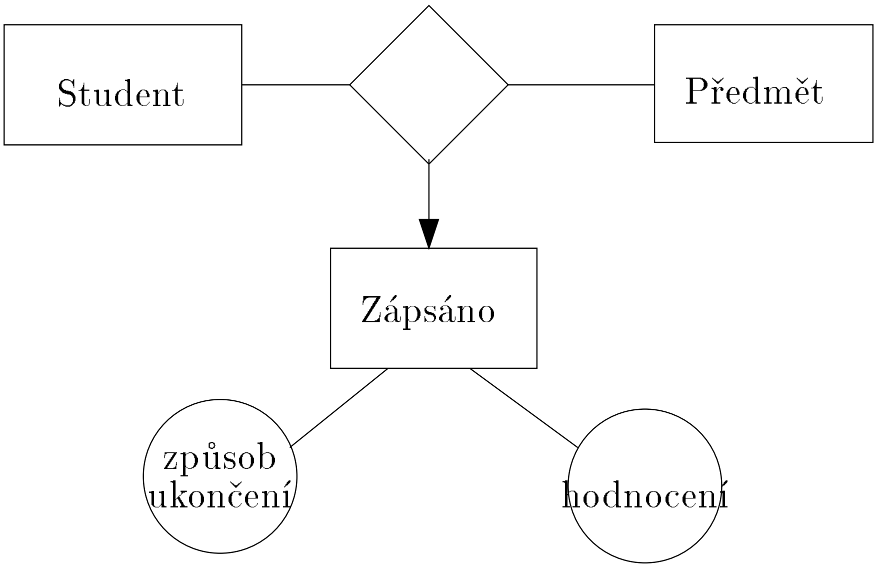
Některé metodiky nezaznamenávají atributy explicitně do diagramů, ale pouze je uvádějí v datovém slovníku projektu.



Obrázek 3: Atribut

2.1.4 Asociační entita

Asociační entita je určitou kombinací entity a relace. Můžeme ji považovat za relaci, u které chceme udržovat nějaké informace. Z toho také vyplývá její označení. Je zobrazena jako nepojmenovaná relace, ke které je „připojena“ entita nesoucí dodatečnou informaci (obrázek 4).



Obrázek 4: Asociační entita

Ve starších diagramech se používaly atributy navázané přímo na relaci. Při použití asociační entity tato nutnost odpadá. Atributy se tedy mohou vázat pouze na entity, což vede k zpřehlednění diagramů a možnosti dalších formálních kontrol diagramů.

2.1.5 Subtyp/supertyp

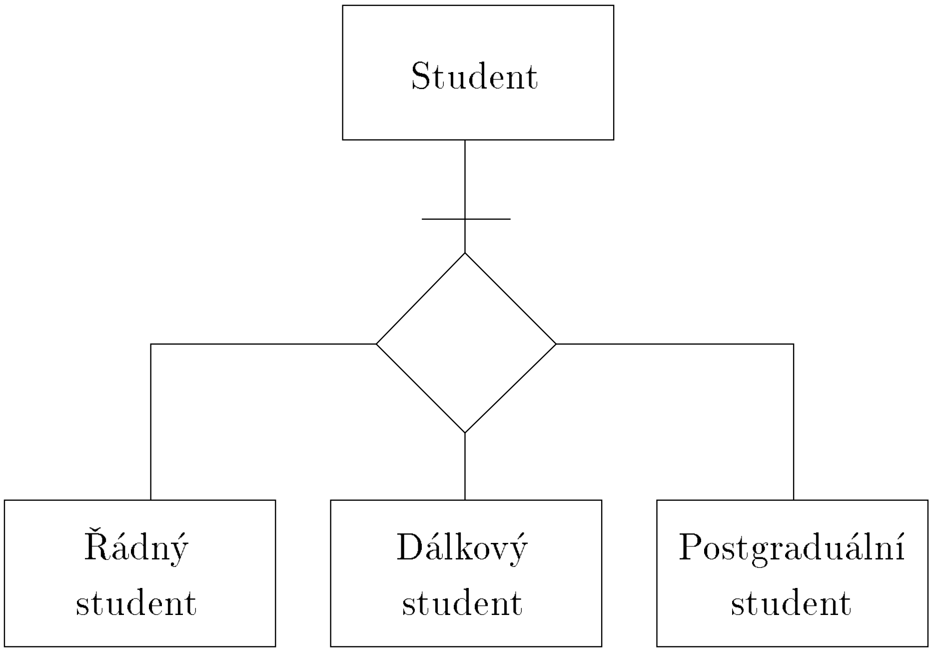
Subtyp/supertyp se skládá z nadřazené entity (supertyp) a jedné nebo více entit (subtyp), které jsou specializací nadřazené entity. Všechny subtypy jsou navzájem spojeny nepojmenovanou relací a supertyp je s touto relací spojen přeškrtnutou čárou (obrázek 5).

V takovém případě supertyp obsahuje atributy použitelné i pro všechny jeho subtypy. Naopak každý subtyp obsahuje atributy specifické pouze pro tuto entitu.

2.2 Modifikace a rozšíření ERD

Výše popsané entitně-relační digramy jsou nejčastěji používaný nástroj pro popis datového modelu při vývoji informačních systémů. Přesto existují další populární notace popisující strukturu dat.

Snad prvním široce používaným popisem struktury dat byly Bachmanovy diagramy, vyvinuté již v 60. letech. Používaly se tedy mnohem dříve, než vznikly relační databáze. Typický Bachmanův diagram je na obrázku 6.



Obrázek 5: Subtyp/supertyp

Jsou ve své podstatě stejné jako jednoduché E-R diagamy, pouze explicitně nepopisují relace mezi entitami a symbolicky znázorňují kardinalitu relací.

Další rozšířenou alternativou je DeMarcův diagram struktury dat (obrázek 7). Hlavní odlišností oproti E-R diagramům je zobrazení klíčových atributů entit přímo v jejich symbolech a také absence symbolů pro relace.

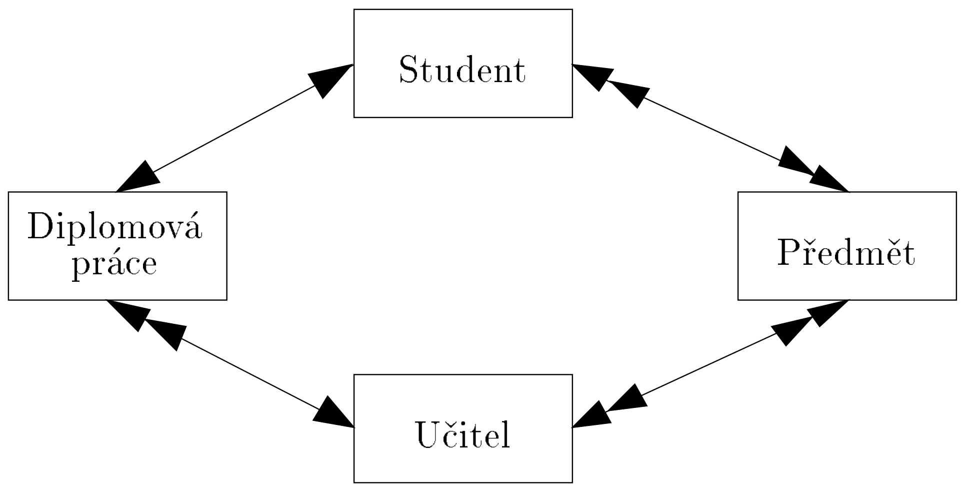
Užitečným rozšířením E-R diagramů, které navrhl James Martin v [6], je symbolické označování kardinality relací. V takovém diagramu potom nejsou, stejně jako v DeMarcově diagramu, použity symboly pro relace. Na obou koncích čáry, která spojuje dvě entity, jsou zobrazeny dva symboly reprezentující minimum a maximum dané vazby (obrázek 8).

S rozvojem objektově orientovaného programování a následně rozvinutím objektově orientované analýzy a návrhu se v E-R diagramech začaly používat pro některé relace speciální symboly. Nejčastěji to jsou symboly pro reprezentaci části entity (*is-part-of*) a podtypu entity (*is-a*).

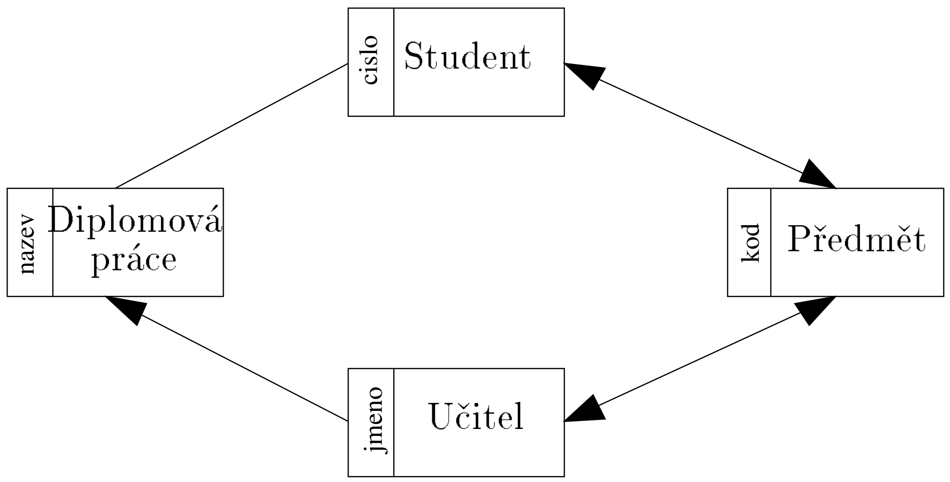
2.3 Zhodnocení ERD

Hlavní výhodou použití ERD je bezesporu jejich jednoduchost a názornost. Právě proto jsou poměrně rozšířeny a alespoň v základní verzi, kdy se používají pouze entity, relace a atributy, je důvěrně zná snad každý návrhář databázových systémů.

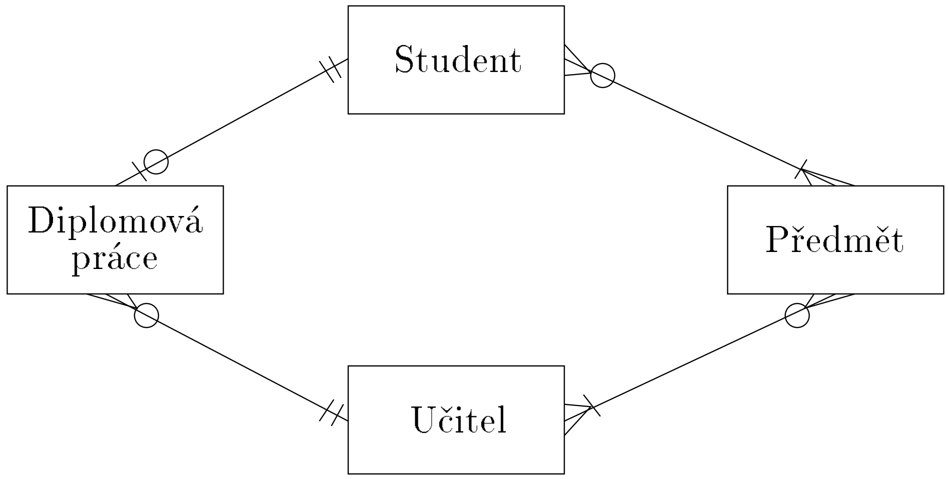
Hůře to vypadá s výše zmíněnými rozšířeními. Ty se většinou objevují



Obrázek 6: Příklad Bachmanova diagramu



Obrázek 7: Příklad DeMarcova diagramu struktury dat



Obrázek 8: Označení kardinality relací symboly

pouze v jednom či několika málo nástrojích pro tvorbu ERD (pokud se nevyskytují pouze v publikacích), a proto nejsou používané ani všeobecně známé. Pro některé vlastnosti reálných databázových strojů neexistuje žádný ekvivalent v diagramech, a tak je nutno tyto vlastnosti zaznamenávat v dokumentaci mimo vlastní diagram.

Zkušenost ukazuje, že se většinou nepodaří udělat kompletní analýzu a potom kompletní design datového modelu, proces není lineární. Dokonce to nezáleží ani tak na zkušenostech tvůrců informačního systému, ale spíše na schopnosti zákazníka spolupracovat. Jedinou výjimkou snad je reiplementace fungujícího systému. Potom lze mnoho informací zjistit z existujícího programu (systému) a není potřeba je dolovat z uživatelů.

Zásadním nedostatkem ERD proto stále zůstává nutnost vytvářet přímo konečné verze diagramů. Nelze tak postihnout postupný vývoj datového modelu, jako je tomu například u diagramu datových toků při analýze procesů. Některé nástroje se snaží tento problém řešit, zabývají se ale spíše dodatečnou optimalizací datového modelu. Například umožňují poloautomatický přechod od abstraktních datových typů k fyzickým. Jedná se tedy o přechod z modelu logického do modelu fyzického. Zejména pro vývoj větších systémů by vytváření logického datového modelu postupně, ve více krocích, bylo velkým přínosem.

3 Metody zakrytí informace v ERD

Zakrytí informace je snad nejvíce používaná metoda pro zpřehlednění programů a efektivnější vývoj softwaru vůbec. Na tomto principu je založeno několik desítek let staré modulární programování a je jedním z principů v současné době stále více se rozšiřujícího objektově orientovaného programování.

Použití principu zakrytí informace vede k rychlejšímu vývoji. Programátor se zabývá jen tím co právě potřebuje, nemusí neustále promýšlet vše do nejjemnějších detailů.

Přesto se tento princip nepodařilo uspokojivě zahrnout do vývoje datových modelů pomocí ERD. V dalších odstavcích si ukážeme, jaké metody zakrytí informace lze při tvorbě ERD použít. Můžeme je logicky rozdělit na následující typy:

- postupná specifikace — informaci vůbec nemáme nebo ji nemáme dostatečně přesnou,
- faktické zakrytí — máme informaci, ale nezobrazujeme ji.

Dále pak navrhneme způsob ovládání a zobrazení navržených prostředků.

3.1 Postupná specifikace

Při použití postupná specifikace jde o návrh datového modelu po částech, podobně jako tomu je u procesního modelu. Do diagramů tedy zaznamenávám jen ty informace, které jsou jisté, a případné upřesnění je provedeno až následně. Přitom máme zdokumentován celý postup a ne jenom konečnou verzi. Postupnou specifikaci můžeme aplikoval na entity nebo relace.

Pro entitu tento postup použijeme, pokud máme v datovém modelu reálný objekt, který se pravděpodobně bude skládat z více entit (jeho reprezentace bude v databázi uložena do více tabulek), ale ještě nejsme schopni přesně určit z jakých.

V takovém případě do diagramu zakreslíme *virtuální entitu*. V podstatě se jedná o přidání další vrstvy abstrakce do ERD, lze na to také pohlížet jako na *pohled*. Virtuální entita je entita, kterou můžeme běžně zakreslovat do E-R diagramů s vazbami na relace a atributy, ale před dokončením návrhu (před generováním kódu) je nutno tuto entitu „upřesnit“. Musíme nakreslit E-R diagram, který popisuje danou virtuální entitu. V tomto ERD musí být také označena „klíčová“ entita, tedy entita, na kterou se váží relace a atributy navázané na virtuální entitu.

Na obrázku 9 je znázorněn příklad virtuální entity a na obrázku 10 je tato virtuální entita rozkreslena.

Druhou možností je použití postupné specifikace pro relace. Tento případ použijeme, pokud nevíme přesně kardinalitu dané relace. Pak použijeme

Obrázek 9: Virtuální entita

Obrázek 10: Rozkreslení virtuální entity

symbol pro zatím neznámou kardinalitu, popřípadě upřesnění ve stylu „skoro vždy jeden".

Příkladem může být účtenka, na které je v 90 % jenom jedna položka, ale jinak tam může být položek libovolně mnoho. Virtuální entita reprezentující účtenku má přímo atribut s první položkou a některé programy pracující jen s touto jednou položkou můžeme napsat přímo bez specifikace vazeb.

Ve fázi analýzy tedy nemusíme některá rozhodnutí dělat, ty se potom přenesou do další fáze a tam se pomocí dalších informací rozhodne. Může nastat situace, že zbývajících 10 % je úplně jiný druh účtenky, který se zpracovává úplně jinak a zvolí se proto vazba 1:1, resp. se položka stane atributem entity účtenka; nebo se musí těchto 10 % postihnout a zvolí se vazba 1:n.

Tato metoda umožňuje, aby bylo možno některé struktury programů psát ještě před přesnou specifikací virtuální entity. V zápisu programu se používají pouze atributy a relace vázané na virtuální entitu. Výsledný program se potom generuje podle rozkreslení virtuální entity.

V uvedeném příkladě se podle konečného závěru (vazba 1:1, resp. 1:n) daný program vygeneruje s jednou tabulkou, resp. se spojením dvou tabulek.

3.2 Hierarchicky vnořené ERD

Výše popsaný princip virtuálních entit přímo vybízí k hierarchickému způsobu práce s diagramy.

Pokud má programátor zobrazen E-R diagram obsahující nějakou virtuální entitu, má možnost přímo (kliknutím myši, horkou klávesou) přejít na zobrazení ERD s rozkreslením této virtuální entity. Stejně jednoduše musí mít možnost přejít z ERD virtuální entity na její použití. Zde ovšem mohou nastat problémy, protože daná entita může být použita ve více diagramech.

Jinou možností je „síťový model", kde má programátor možnost vybrat si pro zvolenou entitu (ne nutně virtuální) ze všech diagramů, kde je daná entita použita.

3.3 Specifické zobrazení ERD

Jinou možností faktického zakrytí informace je zobrazení pouze specifických částí diagramů. Zobrazeny mohou být jen ty části, které potřebuji k právě prováděné činnosti.

Zde je uvedeno několik praktických příkladů:

- nezobrazovat atributy, ale pouze entity a relace;
- jiné forma zobrazení relací (při programování entita odpovídá tabulce a nezajímá nás jestli tam je subtyp/supertyp, is-part-of apod., ale stačí vědět, že tam je vazba);
- požadavky zákazníka.
  Pokud vývojový tým používá ustálenou notaci diagramů, ale zákazník vyžaduje jinou, lze jiným typem zobrazení vytvořit dokumentaci dle zákazníkových požadavků.

3.4 Závěr

Jak jsme si ukázali v této kapitole, lze pro návrh datového modelu používat různé metody zakrytí informace.

Použití všech výše popsaných vylepšení může při návrhu datového modelu pomoci, samo o sobě ovšem nevede k jeho lepšímu návrhu. Proto je vhodné je používat pouze tam, kde to opravdu vystihuje situaci. Vývoj datového modelu pravděpodobně nepůjde nikdy automatizovat, vždy se bude uplatňovat především zdravý rozum.

4 Generování a transformace

V současných vývojových prostředcích je nutno dělat spoustu mechanických činností „ručně“. Většinou to je způsobeno nedostupností vhodného nástroje, který by umožňoval definovat požadované činnosti. Úspěšný vývojový prostředek by měl umožnit veškerou mechanickou práci přenechat počítači.

4.1 Princip generování a transformací

Každá transformace je určitá funkce, která jednomu datovému modelu M přiřadí jiný model M'. Analogicky je každé generování funkce, přiřazující datovému modelu textovou zprávu Z obsahující program, hlášení chyb apod.

Když se na tuto věc podíváme podrobněji, zjistíme, že je dobré, když transformace modelu M, kromě modelu M' podává také zprávu Z o své činnosti. Generování potom můžeme chápat jako transformaci s žádnými změnami, pouze se zprávou Z. Transformace může navíc kromě zprávy generovat i *program*, takže výstupů z jedné funkce může být hned několik. Příkladem může být změna datového modelu. Automaticky se provede změna schématu, vygeneruje se hlášení o změnách a skript pro změnu databáze.

Tansformace lze realizovat několika způsoby:

1. jako jednorázovou záležitost, která vytvoří nový model M'.
   V takovém případě je nutné, aby vývojový nástroj podporoval práci s nějakými vrstvami či fázemi (fyzická / logická; analýza / design / implementace), popř. uživatelem definovanými verzemi.

2. jako jednorázovou záležitost, která modifikuje model M a původní informace není zachována.

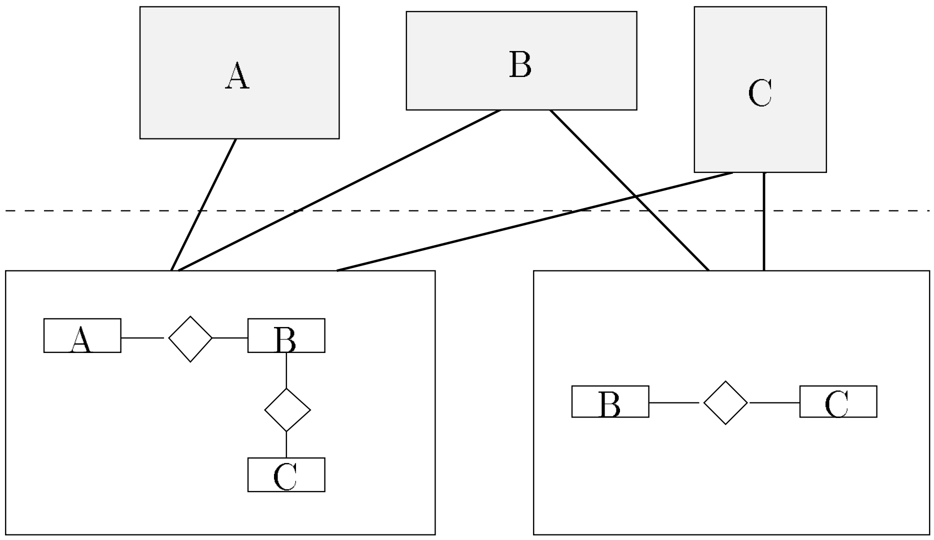
3. jako funkce, která v každém okamžiku ukáže aktuální stav nového M'. Odpovídá to jisté formě pohledu či změně zobrazení.

V praxi si lze představit situace, kdy bude každá možnost užitečná (viz např. kapitola 4.3). Problémy mohou nastat snad pouze u poslední možnosti, která může být implementačně i výpočetně náročná.

Při transformacích může být jako model M použit celý datový model, jeden nebo několik diagramů, popř. jen jejich část. Z tohoto pohledu můžeme datový model rozdělit do dvou rovin (obrázek 11):

- logická rovina — reprezentuje vlastní datový model, tj. entity, relace, atributy apod.

- rovina diagramů — grafická reprezentace datového modelu.



Obrázek 11: Dvě úrovně datového modelu

4.2 Příklady generování

4.2.1 Kontroly

Kontrola generuje, pro daný diagram či model, textovou zprávu obsahující hlášení chyb, upozornění na nepřesnosti apod. Některé chyby úplně zamítne a neumožní další práci s diagramem resp. modelem. Na jiné chyby pouze upozorní. Kontrolovat lze jak vrstvu diagramů (např. „visící“ atributy), tak vrstvu logickou (např. nutnost vyplnění typu v každém atributu). V úvahu také přichází uživatelem definované kontroly, jako je například kontrola konvece tvorby názvů objektů.

Dobrý vývojový prostředek by měl umožnit některé výstupy kontrol zanechat součástí diagramů, resp. celého modelu jako dokumentace. Potom se nabízí možnost „okomentovat“ výstup programátorem a následně může nadřízený (popř. člověk zodpovědný za dodržení kvality) zkonrolovat dodržení všech pravidel.

4.2.2 Generování kódu

Při generování kódu z datového modelu se jedná především o skripty pro vytváření databázových tabulek. Může se ovšem jednat i o programy pro transformace z jedné databáze (počítačové platformy) do druhé. Pokud bychom měly k dispozici správu verzí, lze generovat i programy pro transformaci dat při změně datového modelu.

4.2.3 Generování sestav

Uživatelsky definovatelné sestavy jsou do jisté míry základní výstup, který by měl vývojový systém poskytovat.

Takové sestavy potom mohou být součástí programové dokumentace, mohou sloužit jako podklady pro zákazníka. Dále je možné tyto sestavy statisticky zpracovávat a získat tak potřebné metriky přibližující náročnost projektu. V neposlední řadě lze pomocí sestav zajistit export do jiného vývojového prostředí.

4.3 Příklady transformací

Velkou skupinou transformací jsou pomocné technické operace, které je potřeba občas provést. Jedná se například o změnu konvence pojmenovávání objektů. Jiným příkadem může být úprava zobrazení diagramu, jako je rušení „visících" prvků (samostatné relace, atributy) diagramu.

Další možností je překlad modelu do jiného jazyka. Tuto fukci lze realizovat jako pohled (3. typ realizace, viz výše), i když by se jednalo spíše o efektní detail než nutnost.

Velkou skupinu transformací tvoří také přechody mezi jednotlivými vrstvami či verzemi modelu. Například při přechodu z logické vrstvy do fyzické je možno transformovat typy atributů na fyzické typy podporované cílovým databázovým systémem. Dále může jít o různé optimalizace modelu, dané opět cílovou databází.

Jinou tranformací může být optimalizace modelu na základě volumetrických dat. V takovém případě programátor doplní do datového modelu další dodatečné informace o předpokládané velikosti tabulek, množství dotazů a aktualizací, požadované doby odezvy apod. Podle těchto údajů lze potom provést např. sloučení entit spojených navzájem vazbou 1:1 nebo supertyp/subtyp. Tím sice dochází k porušení vyšších normálních forem, dosahuje se ovšem vyšší efektivity výsledné aplikace.

5 KREDI a jeho funkce

Pro demonstraci prvků a postupů z předchozích kapitol byl vytvořen program KREDI. Název programu vznikl zkrázením slov *kreslič diagramů*.

Principy popsané v kapitolách 3 a 4 lze aplikovat i na jiné typy diagramů, než jsou E-R diagramy. Proto byl program řešen jako univerzální a otevřený a lze ho použít pro libovolný typ diagramu. Program KREDI poskytuje tyto funkce:

- definice struktury diagramů a datového modelu,
- kreslení diagramů,
- správa repozitáře,
- generování z diagramů a datového modelu pomocí šablon, popř. transformace diagramů a modelu,
- grafický tisk diagramů,

Již z této hrubé specifikace je vidět, že se nejedná o jednoduchý nástroj pro kreslení jednoho typu diagramu (nějaké „kreslítko“), ale (při správné konfiguraci programu a definici diagramů) o provázaný systém, potenciálně umožňující kompletní vývoj rozsáhlých informačních systémů.

V této kapitole budou popsány jednotlivé funkce programu a budou demonstrovány na příkladu E-R diagramů. Bude ukázáno jak definovat vzhled a datový model ERD, jak potom vypadá práce při tvorbě diagramů, jak vytvořit jednoduchou šablonu pro generování SQL skriptu na vytvoření tabulek databáze.

V grafickém uživatelském rozhraní je ve všech textech (položky menu, popisy vstupních prvků, jména oken) použita angličtina. Stejně tomu je v konfiguračních a definičních skriptech, kde jsou všechna klíčová slova v angličtině. Tento výběr byl proveden na základě následujících úvah.

Většina programátorů je zvyklá na anglické názvy a české ekvivalenty jsou pro ně neznámé a matoucí. V některých případech nejsou české ekvivalenty ustáleny, různí autoři používají jiná slova či slovní spojení. Pro některé výrazi dokonce české ekvivalenty vůbec neexistují. Byl tedy zvolen jazyk anglický, aby mohli programátoři začít používat program okamžitě, bez podrobného čtení dokumentace.

5.1 Definice datového modelu a diagramu

Jak již bylo uvedeno výše, program KREDI je koncipován jako univerzální nástroj, a proto v sobě nemá zakomponovánu žádnou definici konkrétního diagramu. Aby mohl poskytovat jednotlivé funkce, potřebuje definovat struktury, nad kterými má pracovat.

Definice se provádějí vytvořením zdrojového kódu v jednom či několika textových souborech, které se při inicializaci programu načtou a interpretují. Zdrojový kód je napsán v jazyce k tomuto účelu vytvořeném, který je založen na objektově orientovaném principu a svou syntaxí vychází z jazyka Tcl.

Definice se skládá ze dvou částí. První popisuje jednotlivé *objekty* diagramů, druhá popisuje styly zobrazení diagramu.

Objektem je každý prvek diagramu nebo datového modelu, jako např. entita, relace nebo čára spojující entitu s relací. Každý objekt je určité třídy. Rozeznáváme tři základní třídy[1]:

box — objekt, který zaujímá v diagramu určitou polochu. Většinou nese další upřesňující informace.

line — čára spojující navzájem určité boxy. Může nést další informace.

connection — spojení čáry a boxu. Z jiného pohledu jde o zakončení čáry. Většinou nenese další informaci nebo dokonce není ani zobrazeno v diagramu.

Spojení a čára nemohou v diagramu existovat samostatně, vždy musí být spolu s nějakým boxem.

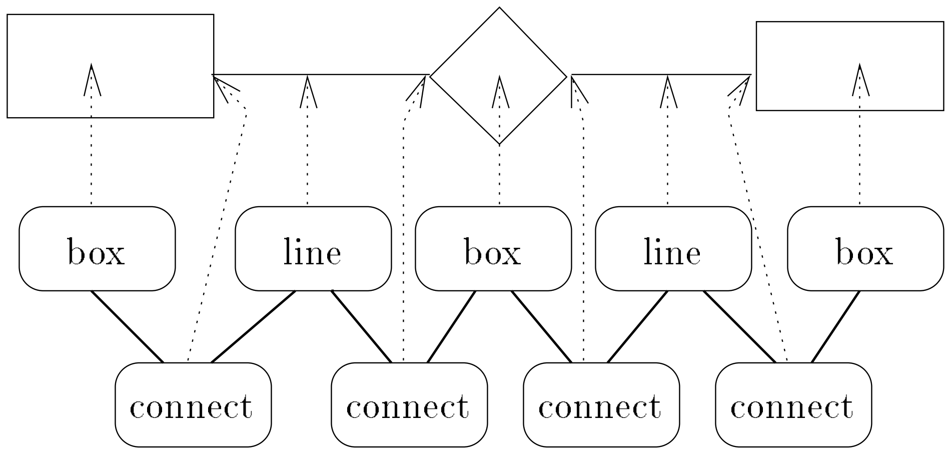
V diagramu i v datovém modelu jsou zaznamenány u každého objektu odkazy na jiné objekty, se kterými „sousedí“. Jedná se tedy o jistý neorientovaný graf, kde uzly tvoří instance jednotlivých objektů a hrany tvoří odkazy mezi nimi. Hrany ovšem nemohou být mezi libovolnými uzly. Povoleny jsou pouze kombinace: *box — spojení*, *spojení — čára*. Navíc z každého *spojení* vede hrana do právě jednoho *boxu* a právě jedné *čáry*. Z každé *čáry* vedou právě dvě hrany do různých *spojení*. Příklad pro jednoduchý E-R diagram je znázorněn na obrázku 12.

Každý prvek platného (již dokresleného) diagramu musí obsahovat své jméno, které je potom jednoznačnou identifikací prvku v repozitáři. U boxů je toto jméno většinou přímo zobrazeno v diagramu — v příkladu ERD jsou to název relace resp. entity. U čáry a spojení toto jméno nemusí být zobrazeno, může být dokonce u všech prvků stejného objektu totožné. V takovém případě se prvky se stejným jménem odkazují na stejný prvek z repozitáře.

Popis každého objektu lze rozdělit na dvě části, podle toho, co popisují:

- datový model,
- vzhled objektu v diagramu.

[1]Poznámka k terminologii: Mohlo by se zdát přesnější nazývat „objekty“ „třídami“, protože jednotlivé prvky diagramů jsou až instancemi právě definovaných objektů. Na druhé straně ovšem jsou tyto objekty určeny hodnotami atributů tříd, tedy to jsou instance tříd – objekty. Navíc je možno definovat nové třídy, obvozené od základních tří tříd. Proto budu dále nazývat typy jednolivých prvků „*objekty*“ a jednotlivé prvky „*instance*“ objektů.



Obrázek 12: Vztah diagram – model

Jednotlivé části popisu si ukážeme na příkladu definice entity z ERD. Zde je uvedena příslušná část zdrojového kódu definice:

```
object entita {
   class kdBox
   var identifikator varchar(20)
   var poznamka varchar(60)
   construct {}
   style default {
      create {
         set item [$w create rec [expr $x -30] [expr $y -15] \
                [expr $x +30] [expr $y +15] -fill gray
      }
      actions  move edit select modify
      findtags entita
   }
}
```

Klíčovým slovem `object` tedy definujeme objekt se jménem *entita*. Dále následuje blok s vlastní definicí objektu. Blok obsahuje na každém řádku příkaz začínající klíčovým slovem následovaný několika parametry. Parametrem může být i blok, který obsahuje několik řádků a je uzavřen do složených závorek.

Definice objektu může obsahovat následující příkazy:

- `class`
  Povinný příkaz definující třídu objeku.

- `var`
  Definuje atribut datového modelu, může být uveden opakovaně v rámci

jednoho objektu. První parametr udává jméno atributu, druhý jeho databázový typ, tak jak bude uložen v repozitáři.

- `construct`
  Nepovinný příkaz definující blok příkazů jazyka Tcl, který se má provést při vytvoření instance objektu.

- `intersec`
  Nepovinný příkaz definující blok příkazů jazyka Tcl, který se má provést pro zjištění průsečíku okraje boxu a čáry. Používá se při kreslení diagramu. Má smysl pouze pro objekty třídy *box*.

- `style`
  Definuje styl zobrazení prvku v diagramu. První parametr je jméno stylu, druhý je blok definující daný styl. Blok může obsahovat následující příkazy:

  - `create`
    Definuje kód, který se provede při zobrazení prvku v diagramu. Je nutno v něm nastavit proměnnou `item` na hodnotu, reprezentující grafický obrazec na obrazovce. K tomu lze použít proměnných `x` a `y`, obsahující pozici prvku v diagramu.
  - `findtags`
    Za tímto klíčovým slovem jsou uvedeny značky, které slouží k vyhledávání jednotlivých prvků diagramu při generování a transformacích diagramů. Značku tvoří libovolný řetězec znaků. Jednotlivé značky jsou od sebe odděleny mezerou. Příkaz lze použít opakovaně v rámci jednoho stylu.
  - `actions`
    Tento příkaz následují akce, které lze s daným prvkem provádět při editaci diagramu. Lze uvést výběr z těchto akcí:

    select — označení,

    move — přesununí,

    size — změna velikosti,

    name — změna jména,

    modify — změna odpovídajícího prvku v datovém modelu.

  Pro každý objekt můžeme definovat několik stylů, nejméně však jeden se jménem *default*.

- `connect`
  Je použitelný pouze pro objekty třídy *line* a definuje objekt třídy spojení, kterým má začínat, resp. končit čára (první parametr je `begin`, resp. `end`).

- `check`
  Je také použitelný pouze pro objekty třídy *line* a definuje stejným způsobem jako `connect` seznam objektů třídy *box*, na které se čára může vázat.

Druhou částí definice diagramu je popis stylů sobrazení. Tato část je nepovinná. Pokud není uveden žádný styl diagramu, používá se pouze implicitní styl *normal*. Tento styl zobrazuje všechny prvky diagramu ve stylu *default*.

Definice stylu se vytváří klíčovým slovem `diagramstyle`. První parametr udává jméno stylu ve tvaru, ve kterém bude uvedeno v menu editoru. Druhý parametr udává seznam objektů, které jsou v tomto stylu zobrazeny. Pokud je prvkem seznamu opět seznam, první člen je jméno objektu a druhý člen je styl tohoto objektu, ve kterém se objekt zobrazí.

Následující příklad definuje styl se jménem *Bez atributů*. Diagram zobrazený v tomto stylu může obsahovat tři druhy objektů – *entita*, *relace*, *cara*. Objekty *entita* jsou zobrazeny ve stylu *velka*, ostatní objekty jsou zobrazeny ve stylu *default*.

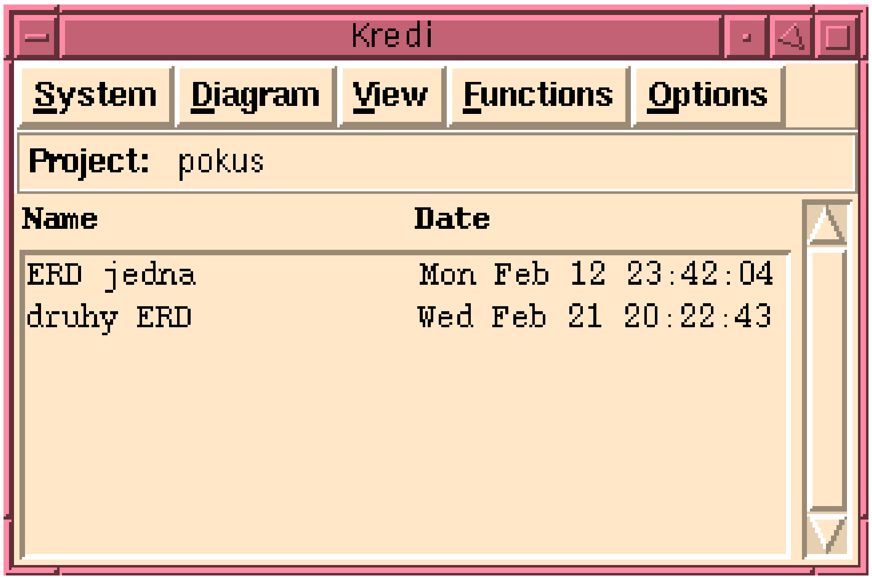
```
diagramstyle "Bez atributů" {
    {entita velka}
    relace
    cara
}
```

5.2 Projekt

Po spuštění programu Kredi se zobrazí na obrazovce okno projektu (obrázek 13). Projekt, se kterým se právě pracuje, je dán již při spuštění programu, uvedením jména projektu, a nelze ho měnit. Každý projekt pracuje s jedinou databází v systému Postgres95, která má stejný název jako projekt. Pro vytvoření nového projektu slouží externí dávka, která vytvoří příslušnou databázi, vytvoří v ní tabulky požadované příslušným datovým modelem a některé nainicializuje.

Z okna projektu lze především otvírat jednotlivé diagramy. Spouštět lze ale i jiné funkce a uživatel může do menu projektu zařazovat svoje vlaství položky. Všechny funkce jsou dostupné z menu, některé lze navíc spouštět kombinací kláves či myší. Některé funkce pracují s aktuálním diagramem, jiné s vybranými diagramy. Vybrané diagramy jsou v souhrnném přehledu zvýrazněny, aktuální je diagram, který je vybrán jako jediný. Standardní menu okna projektu obsahuje následující položky:

- System
  - Info — souhrnné informace o projektu
  - Exit — ukončení programu



Obrázek 13: Okno projektu

- Diagram
  - Open — vytvoření okna editoru s aktuálním diagramem
  - New — submenu, ve kterém každá položka odpovídá jednomu typu diagramu. Po výběru položky a zadání názvu nového diagramu se vytvoří okno editoru s prázdným diagramem daného názvu.
  - Print — vytiskne vybrané diagramy na tiskárnu nebo do souboru (podle konfigurace)
  - Rename — přejmenování aktuálního diagramu
- View — submenu, které obsahuje stejně jako Diagram/New položky pro každý typ diagramu. Každou z položek lze zatrhnout a v souhrném přehledu jsou potom zobrazeny diagramy pouze zatržených typů.
- Functions — submenu, které je implicitně prázdné, pro uživatelsky definované funkce (generování, kontroly modelu apod.)
- Options
  - Diagram — konfigurace parametrů diagramů
  - Printer — konfigurace tisku diagramů

Toto standardní menu lze doplnit o vlastní položky nastavením volby `projectMenu` na blok s definicí menu. Definice je napsána v jazyce pro popis menu, jehož struktura je obdobná jazyku pro definici modelu. Každý příkaz (kromě `separator`) má jako první parametr uvedeno jmého položky menu. Ve jméně se může vyskytovat tilda (~), která způsobí ve výsledném menu podtržení následujícího znaku. Tento znak potom tvoří horkou klávesu dané položky. Pokud se ve jméně tilda nevyskytuje, je podrhnut první znak jména. V definici menu lze použít následující příkazy:

- `main`
  Jedinný příkaz, který lze použít na nejvyšší úrovni. Definuje prvek hlavního menu. Druhý parametr udává blok definující submenu. V definici submenu mohou být použity všechny následující příkazy.

- `command`
  Vytvoření položky, jejíž aktivace způsobí provedí daného příkazu či bloku příkazů (druhý parametr). V příkazech lze použít proměnnou `this`, která obsahuje objekt okna s právě vytvářeným menu.

- `radio`
  Vytvoření přepínací položky. Druhý parametr udává globální proměnnou, kterou tato položka kontroluje, třetí pak hodnotu, kterou má proměnná nabývat.

- `check`
  Vytvoření zatrhávací položky. Parametry jsou stejné jako u příkazu `radio`.

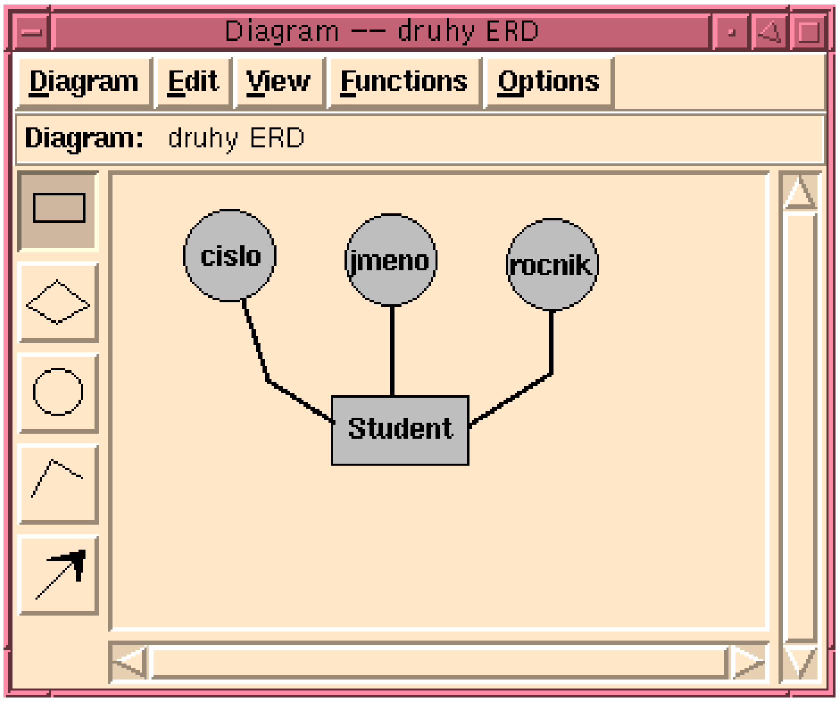
- `separator`
  Vytvoří vodorovnou linku v menu.

- `cascade`
  Vytvoří submenu, definované blokem v druhém parametru. Pokud již submenu se stejným jménem existuje, položky definované v bloku se přidávají na konec k existujícímu submenu.

5.3 Editace diagramu

Hlavní částí vývoje modelu je tvorba vlastních diagramů. Ta se provádí v okně editoru. Vzhled tohoto okna závisí na typu diagramu a celkové konfiguraci systému. Na obrázku 14 je zobrazeno okno pro uvedený příklad E-R diagramů. V editoru se diagram kreslí pomocí myši a tlačítek v levé části okna. Celkové úpravy, změna zobrazení a další funkce se aktivují z menu.

Vzhled a funkce tlačítek v okně je nutno nastavit v konfiguraci. Provádí se to nastavením volby `diagramTools:`*diagram* na blok s definicí tlačítek, kde *diagram* je označení typu diagramu. Blok je popsán v jazyce pro popis tlačítek. Zápis definice je podobný definici objektů. Jednotlivé příkazy jazyka si ukážeme na uvedeném příkadě E-R diagramů.



Obrázek 14: Okno editoru

```
option diagramMenu:erd {
   newbox entita entita
   newbox relace relace
   newbox atribut atribut
   choose cara {
      cara_r {entita relace}
      cara_a {entita atribut}
   }
   command vyber select
}
```

Každý příkaz vytvoří jedno tlačítko. Tlačítka se v okné umísťují postupně shora dolů. Prvním parametrem každého příkazu je jméno ikony, které se má na tlačítku zobrazit. Ikony musí být ve formátu *X Window BitMap* s koncovkou `.xbm` a musí být uloženy v adresáři s ikonami. Adresář je dán volbou `iconDir`, implicitně nastavenou na `./icons`.

Při popisu tlačítek lze použít následující příkazy:

- `newbox` — vytvoření nového prvku objektu třídy box,
- `command` — provedení příkazů uvedených v bloku,

- `choose` — vyvtoření nového prvku čáry podle počátečního a koncového bodu.

Okno editoru obsahuje v menu standardně následující položky:

- Diagram
  - Save — uložení diagramu do databáze
  - Print — tisk diagramu na tiskárnu nebo do souboru podle nastavení
  - Close — ukončení práce editoru
- Edit
  Většina akcí tohoto submenu pracuje s označenými (vybranými) prvky diagramu. Některé z nich pracují, pouze pokud je vybrán pouze jeden (aktuální) prvek. V závislosti na definici objektů diagramu (viz kapitola 5.1) mohou být některé akce pro některé prvky zakázány. Položka akce, kterou v dané situaci nelze použít, je potom v menu neaktivní.
  - Open — otevření jiného diagramu ve kterém se vyskytuje aktuální prvek. Pokud je takových diagramů více, zobrazí se okno s názvy diagramů, ze kterého je možno si žádaný diagram vybrat.
  - Rename — změna jména prvku
  - Modify — modifikace prvku v repozitáři se jménem aktuálního prvku diagramu
  - Cetre — vycentrování diagramu a upravení celkové plochy diagramu podle nastavení voleb
  - Delete — smazání vybraných prvků diagramu. Při mazání boxů jsou smazány i všechny z něho vycházející čáry.
- View
  - Zoom — změna velikosti zobrazení, podle zadaných procent skutečné velikosti
  - Style — submenu obsahující styly, ve kterých je možno daný diagram zobrazit. Vždy se zde vyskytuje položka Normal, která zobrazuje všechny prvky ve stylech `Default`. Dále se zde vyskytují položky pro každý styl popsaný v definici diagramu.
- Functions — submenu, které je implicitně prázdné, pro uživatelsky definované funkce (generování, kontroly diagramu apod.)
- Options
  - Diagram — konfigurace parametrů diagramu

– Printer — konfigurace tisku diagramu

Jsou to stejné položky jako v menu projektu. Pokud se u diagramu nespecifikují bere se nastavení z projektu, jinak se bere nastavení pro daný diagram. Změna konfigurace se tedy týká pouze aktuálního diagramu a je zachována i po uzavření diagramu.

Pro každý typ diagramu lze toto standardní menu stejným způsobem jako menu projektu doplnit o vlastní položky. Blok s definicí menu se v tomto případě přiřadí volbě `diagramMenu:`*diagram*, kde *diagram* je označení typu diagramu. V uvedeném příkladě to tedy je volba `diagramMenu:erd`.

5.4 Generování a transformace

Použití generování a transformací již vyžaduje podrobnou znalost jazyka Tcl. Funkce nad diagramy lze vytvářet dvěma způsoby:

- tradiční procedura Tcl,
- šablona systému KREDI.

Při použití normální procedury Tcl je tato procedura volána s parametry identifikující diagram a vrácená hodnota je vypsána jako výstup generování, resp. transformace. Tuto metodu je vhodné používat zejména pro transformace a kontroly, kde výstup není příliš strukturován.

Druhou možnost, použití šablony, je naopak vhodné použít tam, kde má výstup pevný formát, pouze na některých místech je potřeba doplnit nějaké hodnoty. Šablona je textový soubor s koncovkou `.tpl`, který je uložený v adresáři pro šablony. Adresář je daný volbou `templateDir`, který má implicitně hodnotu `./templates`.

Při expanzi šablony se nejdříve načte soubor daný jménem šablony, v tomto souboru se provedou potřebné substituce a výsledek je vrácen na výstup. Prováděné substituce jsou stejné jako v jazyce Tcl, tedy `$prom` se nahradí hodnotou proměnné `prom` a `[proc param]` se nahradí hodnotou, kterou vrátí volání procedury `proc` s parametrem `param`. V hranatých závorkách může být více příkazů (oddělených koncem řádku), jako výsledek se potom bere hodnota vrácená posledním provedeným příkazem.

Obě metody lze vzájemně kombinovat. Můžeme tedy v libovolné funkci volat proceduru `kdExpandTemplate` s parametrem udávajícím jméno šablony. Procedura vrátí vyplněnou šablonu, se kterou můžeme pracovat jako s běžným řetězcem znaků. Naopak můžeme v šabloně uvést do hranatých závorek libovolnou proceduru, tedy i proceduru generující nějaký výstup. Hodnota tohoto výstupu potom bude vložena na místo volání. Samozřejmě také můžeme v šabloně volat expanzi jiné šablony.

V procedurách a procedurálních substitucích šablon můžeme používat libovolné příkazy jazyka Tcl. Navíc byl pro šablony tento jazyk rozšířen o další užitečné procedury a konstrukty:

- `expandeach` — konstrukt analogický `foreach`. Také se prochází seznam hodnot, zadaný blok se ovšem neprovádí, ale po vyhodnocení se přidává do seznamu. Seznam se nakonec naformátuje a vrátí se výsledek.
- `seteach` — analogie příkazu `set` pro více proměnných. Každé proměnné v seznamu prvního parametru se přiřadí hodnota ze seznamu druhého parametru.
- `swapcoords` — funkce vracející přehozené souřadnice ze vstupu.
- `getObjects` — metoda třídy diagramu. Vrací seznam všech prvků diagramu, které vyhovují daným podmínkám.
- `find` — metoda třídy diagramu. Vrací prvky, které leží na specifikované cestě od jednoho prvku k jinému.

Jako příklad, demonstrující použití popsaných příkazů, zde uvádíme šablonu, která vygeneruje SQL skript na vytvoření tabulek z E-Rdiagramu.

```
-- create table
-- generated by Kredi
[expandeach e [$diagram getObjects entita] "" "\n\n" {
    --  [$e getItem poznamka]
    CREATE TABLE [$e getItem identifikator] \(
[    expandeach a [$diagram find $e -b atribut] "\t" ",\n" \
    {[$a getItem identifikator] [$a getItem typ]}]
    \);
}]
-- end of file
```

Výsledek z této šablony pro diagram na obrázku 14 vypadá takto:

```
-- create table
-- generated by Kredi

    --  Student ve skole
    CREATE TABLE student (
        cis INT,
        jmeno VARCHR(20),
        roc INT
    );

-- end of file
```

6 Programová implementace KREDI

V této kapitole je popsána implementace programu KREDI. Kapitoly 6.1 a 6.2 se dotýkají použitých principů spíše obecně. Naopak kapitola 6.3 podrobně popisuje implementaci jedné části programu. Pro její správné pochopení je asi nutné znát principy jazyka Tcl a knihovny Tk (viz např. [9], [10]).

6.1 Použité prostředky

Pro implementaci programu byl zvolen operační systém UNIX s grafickým uživatelským prostředím X Window System. Program byl napsán v jazyce Tcl s použitím rozšiřujících knihoven Tk a [incr Tcl]. Pro repozitář byla použita databáze Postgres95.

Tcl (Tool Command Language) je jednoduchý jazyk pro psaní skriptů na správu a rozšiřování aplikací. Poskytuje obecné prostředky pro programování jako jsou proměnné, cykly, procedury apod. Navíc je snadno rozšiřitelný. Je to vlastně knihovna funkcí v jazyce C, která může být přidána do jiné aplikace a která může být pomocí dalších funkcí rozšířena dle potřeby dané aplikace. Byla použita verze 7.4.

Knihovna Tk je jedno z nejpoužívanějších rozšíření Tcl. Přidává prostředky pro vytváření uživatelského rozhraní pro X Window System. Je také realizována v jazyce C a lze ji stejně snadno doplňovat o nové prvky rozhraní, potřebné v aplikaci. Byla použita verze 4.0.

[incr Tcl] tvoří objektově orientované rozšíření jazyka Tcl, stejně jako je C++ objektově orientované rozšíření jazyka C[2]. Přidává do jazyka možnost definice tříd, čímž umožňuje tvorbu rozsáhlých knihoven, při zachování dostatečné přehlednosti a srozumitelnosti kódu. Byla použita verze 1.5.

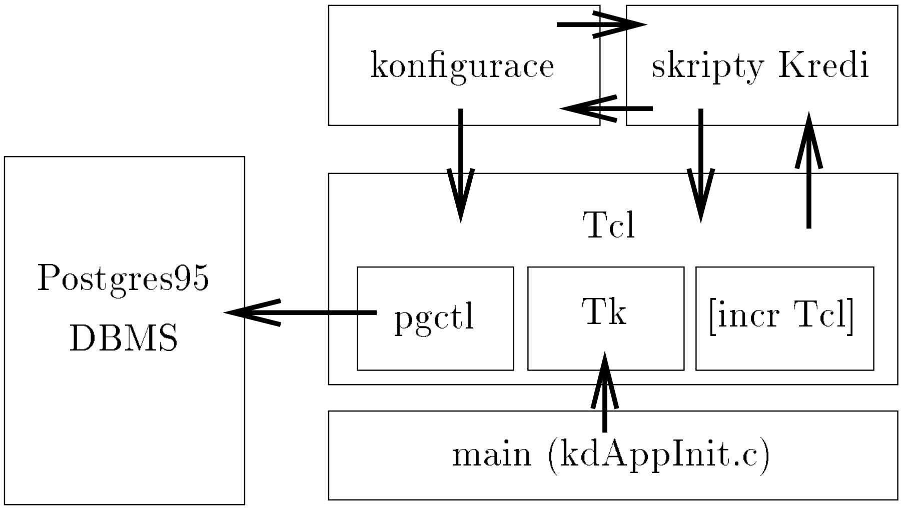
Postgres95 je relační databázový systém, který je rozšířen o třídy, dědičnost, typy, funkce a pravidla, při zachování jednoduchosti relační databáze. Pro práci v Tcl je k dispozici knihovna pgtcl, která zajišťuje komunikaci s databázovým strojem a umožňuje přímo v Tcl zadávat SQL příkazy a zpracovávat jejich výsledky. Byla použita verze 1.0, jako následník verze 4.2 systému Postgres.

Propojení jednotlivých prostředků ukazuje obrázek 15. Šipka z A do B znamená, že modul A volá funkci z modulu B.

Vlastní program se tedy skládá ze skriptů v jazyce Tcl, které využívají všechna výše zmíněná rozšíření. Jednotlivé konfigurační skripty a části kódu v definici diagramů musí být také psány v tomto jazyce.

Program byl sice vyvinut v prostředí UNIX & X Window, nabízí se ovšem možnost převést program do prostředí MS Windows přechodem na vyšší verzi Tcl 7.5 / Tk 4.1. Jedinou podmínkou by bylo použití protokolu

[2] Dokonce i název rozšíření tím byl inspirován. V jazyce C znamená `C++` zvětšení proměnné `C` o 1, analogicky v Tcl znamená `[incr Tcl]` zvětšení proměnné `Tcl` o 1.



Obrázek 15: Struktura použitých prostředků.

TCP/IP na komunikaci s databázovým strojem Postgres95, běžícím na jiném počítači.

6.2 Struktura programu

Celý program je uložen v několika souborech, přímo ve zdrojovém kódu jazyka Tcl. K jeho spuštění je tedy potřeba interpret Tcl, který navíc obsahuje všechny potřebná rozšíření (Tk, [incr Tcl], pgtcl). Příkladem takového interpretru může být program `kredish`, který lze vytvořit z přiložených souborů `kdAppInit.c`, `Makefile` příkazem operačního systému `make`. Při nestandardní konfiguraci systému je potřeba upravit v souboru `Makefile` cesty ke knihovnám jednotlivých rozšíření.

Vlastní běh programu lze spustit příkazem `kredi`. Tím se začne interpretovat soubor `kredi`, který obsahuje počáteční inicializaci spojení s databázovým strojem, inicializaci proměnných, spuštění konfiguračních striptů a následné načtení struktury datového modelu. Na závěr vytvoří okno projektu, kterému předá řízení.

Další soubory programu můžeme podle obsahu rozdělit do čtyř skupin:

- pomocné funkce,
- třídy diagramů a prvků modelu,
- třídy uživatelského rozhraní,
- konfigurační soubory.

6.2.1 Pomocné funkce

Do této skupiny patří pět souborů, které poskytují pomocné a do jisté míry obecné funkce, použitelné v celém programu.

V souboru `control.tcl` jsou procedury pro vytvoření jednoduchých oken na obrazovce, které zajišťují vstup textového řetězce, popř. výstup zprávy.

Soubor `database.tcl` zajišťuje komunikaci s databázovým strojem. Je určitou nadstavbou knihovny *pgtcl*, kterou využívá, a poskytuje služby vyšší úrovně specifické pro program KREDI.

Soubor `menu.tcl` obsahuje procedury pro zpracování definice menu. Je zde nadefinován jazyk, ve kterém se definice menu píše, a způsob zpracování jednotlivých příkazů tohoto jazyka. Také je zde procedura, která načte definici menu a vytvoří odpovídající vnitřní struktury.

Podobně je soubor `model.tcl` definicí jazyka pro popis datového modelu a vzhledu diagramů. Obsahuje definice jednotlivých příkazů a proceduru pro načtení a zpracování celého definičního souboru.

Soubor `utils.tcl` obsahuje opravdu pomocné funkce k uložení a obnovení globálního stavu aplikace, nalezení nejbližšího prvku v seznamu. Dále obsahuje procedury pro výpočet průsečíku úsečky s elipsou, obdélníkem a polygonem.

6.2.2 Třídy diagramů a prvků modelu

Soubor `diagram.tcl` obsahuje definici jediné třídy *kdDiagram*. Tato třída obsahuje metody pro uložení diagramu do databáze, načtení z databáze, vyhledávání jednotlivých prvků v diagramu, generování ze šablony a dále několik pomocných metod.

V souboru `diobjs.tcl` jsou definovány základní třídy prvků diagramu. Třída *kdBase* je předkem všech dalších tříd. Jsou v ní definovány různé pomocné a konfigurační metody, které jsou společné všem ostatním třídám. Je to třída abstraktní, tedy nemá smysl vytvářet objekty této třídy, pouze lze z *kdBase* odvodit novou třídu a tvořit objekty této nové třídy. Třídy *kdBox*, *kdLine* a *kdConn* jsou odvozeny z *kdBase* a umožňují vytváření objektů boxu, čáry a spojení (viz 5.1). V každé z těchto tříd jsou definovány specifické metody pro vytvoření, přesunutí, označení prvku a další metody umožňující vyhledávání sousedních uzlů v grafu diagramu.

V souboru `edobjs.tcl` jsou definovány třídy potřebné při práci editoru. Jde zejména o třídy označení obdélníka a čáry. Dále jsou zde definovány některé užitečné funkce pro práci editoru s jednotlivými prvky diagramu.

6.2.3 Třídy uživatelského rozhraní

Soubor `editor.tcl` definuje jedinou třídu *kdEditor*, která reprezentuje okno editoru. Tato třída obsahuje metody, které převádějí jednotlivé akce v uživatelském rozhraní na odpovídající akce v diagramu. Dále obsahuje metody zajišťující správné zobrazení diagramu na obrazovce a také metodu pro tisk diagramu na tiskárně (resp. vytvoření souboru popisujího diagram v jazyce PostScript). V neposlední řadě jsou zde metody pro ovládání editoru a nastavení parametrů zobrazení a tisku diagramů.

V souboru `project.tcl` je definována také pouze jedna třída – *kdProject*. Tato třída reprezentuje hlavní okno programu KREDI se zvoleným projektem. Obsahuje metody pro zobrazení seznamu diagramů v daném projektu, otevření diagramu (vyvtoření okna editoru s tímto diagramem) a vytvoření nového diagramu zvoleného typu. Stejně jako třída *kdEditor* obsahuje také metody pro nastavení konfiguračních parametrů.

6.2.4 Konfigurační soubory

Tato skupina soubrů nepatří přímo do programu KREDI. Jsou to soubory vyvořené uživatelem a závisí na typu tvořených diagramů.

V našem případě pracujeme s E-R diagramy a vytvořili jsme tedy soubor `erd.dia`. Tento soubor obsahuje definici ERD tak, jak je popsána v kapitole 5.1. Pro jiné typy diagramů bychom vytvořily soubory s jinými jmény.

Soubor `kredirc` je kondigurační soubor, který se načítá při startu programu. Musí v něm být nastaveny všechny hodnoty, kde nám nestačí implicitní nastavení, popř. se liší systmémová konfigurace. Jednotlivé parametry jsou srozumitelné z následující ukázky:

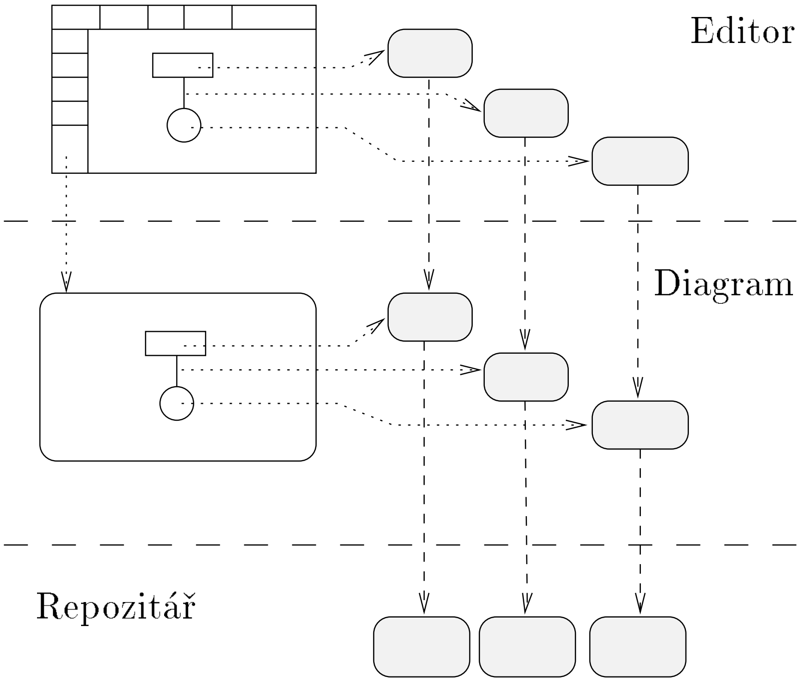
```
#kredirc
option templateDir "./templates"

option projectMenu {
   main Functions {
      command "Check diagram" "$this check_diagram"
      command "Check model" "$this check_model"
   }
   main Options {
      check "Generate names" gennames
   }
}

kdLoadModel erd
```

6.3 Editor diagramu

Nejdůležitější částí celého programu KREDI je editor diagramu. Implementace editoru je tvořena jedinou třídou *kdEditor*. Objekty této třídy v sobě ovšem zahrnují odkazy na další objekty. Na jedné straně to jsou objekt diagramu (instance třídy *kdDiagram*), na druhé pak jednotlivé prvky diagramu. Přehledně jsou všechny vazby znázorněny na obrázku 16.



Obrázek 16: Vztahy mezi objekty

Pro implementaci vlastní kreslící části diagramu byl použit „widget" pro plátno (canvas) z knihovny Tk. Každý obrazec (text, polygon, obdélník, elipsa atd.) zde může mít množinu značek (tags). Operace (posunutí, rušení, vyhledání atd.) s obrazci jde potom provádět najednou pro všechny obrazce se zadanou značkou. Např. posunutí prvku, který se skládá z obrysu (polygon), barevné výplně (obdélník), popisu (text), který je navíc označen (je kolem něho nakresleno osm čtverečků, navzájem propojených čárou), lze zajistit jediným příkazem. Realizace značek v knihovně Tk je pomocí hašovacích funkcí, nedochází tedy ani při velkém množství obrazců ke zpomalení operací s obrazci.

V diagramu rozlišujeme tři druhy obrazců:

- obrazce prvků diagramu,
- přívěsky prvků diagramu,
- obrazce objektů vybrání.

Obrazce prvků diagramu jsou základní části. Každý z nich má značku `obj` a `cl_`*třída*, kde *třída* je jméno třídy, do které prvek patří (box, line, conn). Pokud je prvek vybrán, má navíc značku `selected`. Další značky jsou odvozeny od akcí, které jsou pro daný prvek povoleny. Tyto značky mají tvar `f_`*akce*, kde *akce* je jméno požadované akce. Prvky objektu *entita* v uvedeném příkladu tedy budou mít značky: `obj`, `cl_box`, `f_move`, `f_edit`, `f_select`, `f_modify`.

Přívěsky prvků jsou obrazce, které doplňují vzhled prvku o další detaily. U zmíněné entity to je například název. Ten je tvořen textem umístěným do středu obdelníka. Každý přívěsek má značku ve tvaru `added_`*item*, kde *item* je jednoznačná číselná identifikace hlavního obrazce v plátně. Toto identifikační číslo je automaticky určeno při vytvoření obrazce a je uloženo v položce *item* každého prvku. Každý přívěsek má také všechny značky obrazce svého prvku, které začínají `f_`.

Obrazce objektů vybrání jsou dvojího druhu. Buď se jedná o obdélník, který má ovládací prvky na svých stranách a ve svých rozích , nebo se jedná o lomenou čáru, která má ovládací prvky v každém svém vrcholu. Každý obrazec vybrání má značku `sel_`*item*, kde *item* je opět identifikace hlavního obrazce v plátně.

Veškeré akce jsou definovány nad obrazci s danými značkami. V průběhu editace se mohou některým obrazcům značky přidávat, nebo odebírat. Například při výběru prvku diagramu se hlavnímu obrazci prvku přidá značka `selected`.

7 Možnosti rozšíření programu KREDI

Možná rozšíření lze rozdělit do tří skupin:

- přidání nových funkcí do vlastního programu,
- vytvoření nových podpůrných programů,
- přenesní programu na jiné platformy.

Protože je KREDI otevřený systém, lze velkou část rozšíření realizovat pouhou konfigurací systému. Jedná se především o vytvoření popisu různých tupů diagramů a jejich různých stylů zobrazení. I pro složitější objekty diagramů, které nelze realizovat pouhou instanciací předdefinovaných tříd (*kdBox*, *kdLine*, *kdConn*), je možno vytvořit nové třídy odvozené od některé předdefinované třídy. K tomu je potřeba podrobně znát nejen jazyk Tcl s knihovnou Tk, ale i knihovnu [incr Tcl] a strukturu základních tříd. Přesto se jedná pouze o rozšíření na uživatelské úrovni, není potřeba zasahovat do zdrojových textů programu.

Existuje ovšem několik vylepšení, které by zpříjemnily tvorbu diagramů a které vyžadují zásah do programu KREDI. Účelné například může být spouštění několika projektů z jediného programu. Tato úprava by umožnila přenášení diagramů, nebo jen částí jednoho diagramu mezi různými projekty.

Jiným rozšířením by bylo vytváření verzí diagramů. Dalo by se zaznamenat průběh veškerých změn diagramu (jeho historii). Implementace operací porovnání a sloučení jednotlivých verzí by umožnila práci celého týmu na jediném diagramu.

Vytvoření určitých externích programů se přímo nabízí. Zejména tvorba definice objektů diagramu vyžaduje rozsáhlou přípravu a zkoušení velikosti jednotlivých prvků na obrazovce. Užitečný by tedy byl nástroj, ve kterém by šlo navrhovat jednotlivé objekty a jejich styly interaktivně, s okamžitým zobrazením výsledku. Program by potom vygeneroval příslušné definice.

Jiný nástroj by mohl pomáhat při úpravách menu projektu a jednotlivých diagramů a při tvorbě ovládacích tlačítek diagramů. Tento nástroj ovšem není tak potřebný, protože zápis definice menu a tlačítek se zdá být dostatečně přehledný.

Na pomezí mezi externím programem a novou fukcí programu KREDI by byl nástroj pro trasování šablon a skriptů s tranformacemi diagramů. S prostředky Tcl by šlo realizovat obě varianty, optimální by asi byla kombinace obou. Jednalo by se o externí program, který by využíval nově vytvořené funkce v programu KREDI.

Díky přenositelnosti Tcl a databáze Postgres95 lze systém provozovat na většině dostupných variantách UNIXu. Další možností je použití jiné databáze než Postgres95. Znamenalo by to pravděpodobně zásah do zdrojového kódu a především implementaci funkcí, které poskytuje knihovna *pgtcl*, pro daný databázový systém.

Závěr

V teoretické části práce byly shrnuty formální podklady relačního datového modelu a entitně-relačních diagramů (kapitoly 1 a 2). Bylo navrženo několik rozšíření E-R diagramů, které usnadňují vývoj datových modelů pro informační systémy (kapitoly 3 a 4).

V praktické části byl navržen a implementován univerzální nástroj pro tvorbu diagramů, který umožňuje práci s libovolným typem diagramů (kapitoly 5 a 6). Jedná se o otevřený, plně konfigurovatelný systém s možností dalších rozšíření (kapitola 7).

Odkazy

[1] Peter Coad and Edward Yourdon, Object-Oriented Design, Prentice Hall, 1991.

[2] E. F. Codd, Further Normalization of Data Base Relational Model, in Data Base System, 1972, pp. 65–98.

[3] Tomáš Havlát, Pavel Hain, Organizace dat, přednáška KMI.

[4] Christian Kalus, Peter Dadam, Flexible Relations – Operational Support of Variant Relational Structures, 21st VLDB Conference, Switzerland 1995.

[5] James Martin, Principles of Object-Oriented Analysis and Design, Prentice Hall, 1993.

[6] James Martin, Recommended Diagramming Standards for Analysts and Programers, Prentice Hall, 1987.

[7] Vojtěch Merunka, Databáze dnes a zítra, Softwarové noviny 9/95.

[8] Edward Yourdon, Modern Structured Analysis, Prentice Hall, 1989.

[9] John K. Ousterhout, Tcl and Tk Toolkit, Addison Wesley Publishing, 1994.

[10] Brent Welch, Practical Programming in Tcl and Tk, Prentice Hall, 1995.

[11] Andrew Yu, Jolly Chen, The POSTGRES95 User Manual, University of California, Berkeley, 1995.