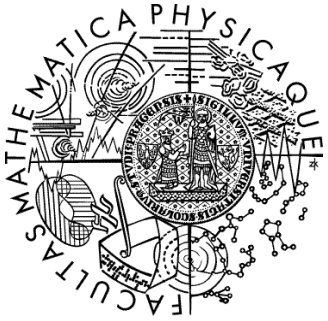
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Hana Kozelková

Editor ER diagramů s podporou převodu do relačního modelu

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Tomáš Skopal, Ph.D.

Studijní program: Informatika, programování

2006

Děkuji panu RNDr. Tomáši Skopalovi, Ph.D. za odborné vedení mé práce, za rady a za čas, který mi během vypracovávání této práce věnoval.

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 28.5.2006 Hana Kozelková

OBSAH

Název práce: Editor ER diagramů s podporou převodu do relačního modelu
Autor: Hana Kozelková
Katedra: Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Tomáš Skopal, Ph.D.
E-mail vedoucího: Tomas.Skopal@mff.cuni.cz
Abstrakt:
Předmětem práce je implementace editoru ER diagramů pro potřeby konceptuálního databázového modelování. ER diagramy jsou kresleny v notaci používané při výuce Databázových systémů na MFF UK. Editor byl vytvářen jako standardní CASE nástroj. Součástí programu je převod ER modelu do relačního modelu. Výsledkem převodu je skript v SQL-92 pro vytvoření tabulek v databázovém systému. Podporovány jsou mimo jiné ISA hierarchie, n-ární vztahy a několikanásobné atributy. Před převodem je provedena verifikace ER diagramu.
Práce obsahuje také teoretická východiska a porovnání s několika podobnými programy jak z komerční, tak akademické sféry.

Klíčová slova: návrh databází, konceptuální modelování, ER diagram, CASE nástroj

Title: An ER editor supporting conversions to relational model
Author: Hana Kozelková
Department: Department of Software Engineering
Supervisor: RNDr. Tomáš Skopal, Ph.D.
Supervisor's e-mail address: Tomas.Skopal@mff.cuni.cz
Abstract:
The task of the work is to implement an ER diagram editor for conceptual database modeling. ER diagrams are drawn by using notation which is taught in the course of Database systems at the Faculty of Mathematics and Physics of Charles University. The editor was created as a standard CASE tool. The program includes the mapping from the ER model to the relational model as well. The mapping results in a SQL-92 script for creation of tables in the database system. ISA hierarchy, n-ary relations and multivalued attributes are also supported. Before the mapping, the verification of the ER diagram is made.
The work also contains theoretical principles and a comparison with some similar programs from both commercial and academic sphere.

Keywords: database design, conceptual modeling, ER diagram, CASE tool

1 ÚVOD

Pojem databáze v dnešní technicky poměrně vyspělé době už zaslechl jistě každý. Ať se jedná o databázi telefonních čísel, knih či jízdních řádů, většina z nás je vnímá jen jako užitečné informace. Málokdo si už ale dokáže představit, jak jsou ve skutečnosti data v databázi uspořádána. Uspořádána tak, aby se dala snadno udržovat a zároveň se neplýtvalo prostředky kvůli uložení duplicitních informací. Při návrhu uspořádání databáze se vychází z reálných vlastností sledovaných objektů a vazeb mezi nimi. Během návrhu je popis objektů postupně upravován tak, aby nakonec odpovídal způsobu uložení ve zvoleném databázovém systému.

Po prvotním zkoumání modelovaných objektů je třeba vytvořit jejich popis. K formalizovanému, ale dostatečně srozumitelnému popisu slouží konceptuální datové modely. Mezi nejznámější z nich patří ER (z angl. Entity Relationship) model znázorňovaný pomocí tzv. ER diagramu. U těchto diagramů nejsou určena jednotná pravidla, jak mají vypadat. Předmětem této práce je vytvořit uživatelsky příjemný editor ER diagramů, který zobrazuje diagramy tak, jak se s nimi můžeme setkat při výuce předmětu Databázové systémy na MFF UK.

Zatímco ER model je jednoduchý a vhodný způsob popsání struktury dat, implementace dnešních databází je většinou postavena na jiném přístupu zvaném relační model. Relační model organizuje data do tabulek, tzv. relací, skládajících se z řádků a sloupců.. Přestože relační teorie má své přímé principy návrhu databáze, používá se často z důvodu jednoduchosti návrhu ER model, který se následně do relačního modelu transformuje.

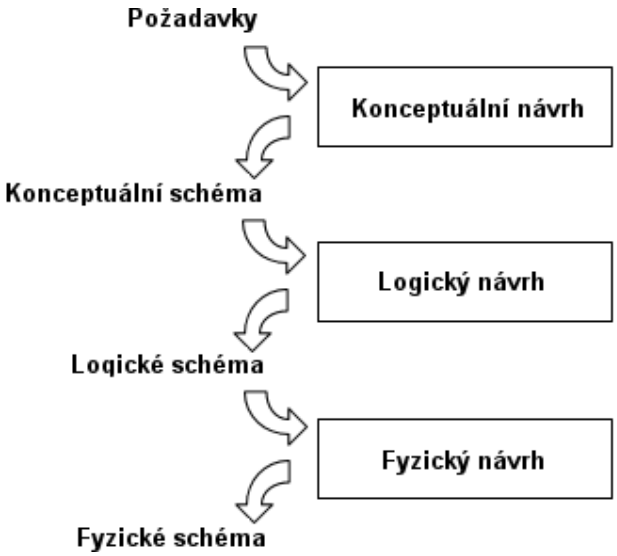
V rámci práce vznikl program ERTOS s názvem odvozeným ze slovního spojení ER TO Sql. Nedílnou součástí programu je funkce kontroly ER diagramu a jeho převodu do relačního schématu dat. Výsledkem převodu je SQL skript pro vytvoření potřebných tabulek v databázovém systému. Při převodu se vychází z teoretických základů vysvětlených v druhé kapitole. Třetí kapitola shrnuje požadavky, které by měl vytvářený program splňovat.

Čtvrtá kapitola obsahuje uživatelskou dokumentaci, která slouží jak pro orientaci při užívání programu, tak jako přehled všech funkcí programu. Na ni navazuje pátá a šestá kapitola, kde jsou popsány problémy, které se při implementaci programu vyskytly a jak byly vyřešeny. Závěrečné části práce ověřují použitelnost vygenerovaného SQL skriptu v různých databázových systémech a podrobně srovnávají ERTOS s konkurenčními programy. Text je prokládán obrázky ER diagramů, které byly z velké většiny převzaty přímo z aplikace ERTOS.

2 VÝCHOZÍ TEORIE

2.1 Návrh relační databáze

Navrhnout dobře strukturovanou relační databázi je nelehký úkol. Při návrhu se provádí datová analýza, která se skládá z několika fází (obr. 2.1). V jejich průběhu se postupně vytváří stále konkrétnější schémata, která znázorňují požadovaná data.



Obrázek 2.1 Fáze návrhu databáze

V první fázi se prostřednictvím konzultací s uživateli a zadavateli systému formulují a shromažďují přesné požadavky na to, co vše má být v databázi uloženo. Z takto získaných informací se vytvoří konceptuální model, který je výsledkem první fáze.

Konceptuální datový model popisuje data v databázi na abstraktní úrovni nezávisle na jejich fyzickém uložení. Proces tvorby konceptuálního modelu se nazývá konceptuální modelování. Jeho výsledkem je konceptuální model znázorněný jako konceptuální schéma nebo diagram, který má co nejvýstižněji zachycovat pohled člověka na danou část reálného světa [1]. Mezi nejznámější konceptuální datové modely patří ER model.

ER modelování lze metodologicky rozdělit do dvou kroků: [1]

- v prvním kroku se definují nezávislé entity, vztahy a závislé entity, to se opakuje do té doby, než se dospěje k souhlasu mezi zadavateli a analytiky;
- v druhém kroku se formulují atributy a klíče entit – klíčů může být v tomto stádiu i více.

V další fázi (zvané **logický návrh** [2]) se vytvoří relační model pomocí transformace objektů z ER diagramu. Následuje formulace definic tabulek v relačním databázovém systému obsahujících atributy, primární a cizí klíče.

V posledním stádiu (tzv. **fyzickém návrhu** [2]) se zabýváme přesnou definicí domén jednotlivých atributů, což zahrnuje i některá integritní omezení, vytváříme indexy pro rychlejší přístup k datům a provádíme další optimalizace.

2.1.1 Alternativy ER modelování

Relační schéma lze vytvořit také přímo z požadavků na aplikaci bez použití ER modelování. Relační schémata se takto vytvářela ještě před zavedením ER modelu (relační model byl zaveden Edgarem Coddem o šest let dříve než vznikl ER model [3]). Ze vstupních požadavků se určí relevantní atributy, které se přiřadí do jednotlivých relačních tabulek.

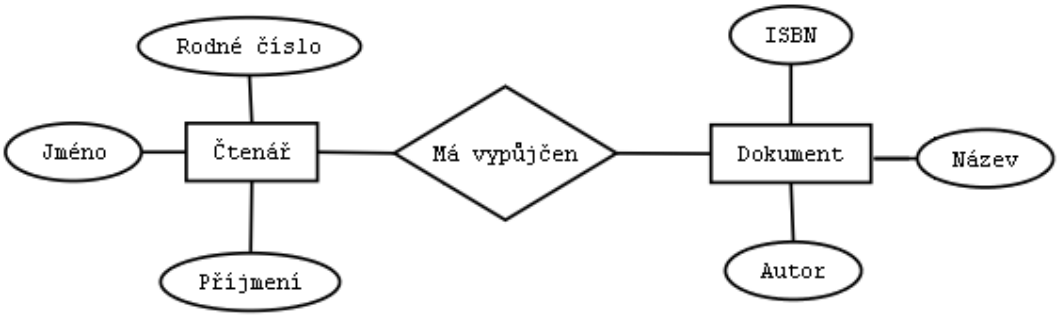
Tento proces se nazývá **normalizace**. Postupně vznikají relační schémata, která odpovídají zvyšujícím se stupňům tzv. **normálních forem**. Normální forma určuje úroveň normalizace schématu, u vyšších normálních forem se vyskytuje méně redundantních dat v tabulkách a ubývají anomálie při manipulacích s daty.

Narozdíl od konceptuálního návrhu, kdy je prováděn návrh shora-dolů (nejdříve se identifikují entity a pak se jim přiřadí atributy), se při normalizaci uplatňuje přístup opačný, nejdříve se najdou atributy a ty se přiřazují do tabulek, které v podstatě představují entity [3]. Zkušení systémoví analytici používají kombinaci obojího.

Zastánci objektové metodiky by k modelování dat doporučili **diagram tříd** modelovaný pomocí UML (z angl. Unified Modeling Language) [4]. Zopakujme, že navrhujeme relační databázi a data tedy modelujeme proto, abychom na základě toho vytvořili relační (a ne objektovou) databázi. V diagramu tříd lze sice nepřímo popsat entity, atributy a vztahy, ale například primární atributy nebo povinnost členství atributu už nelze (bez použití rozšíření) pomocí UML vyjádřit. Z výše uvedeného plyne, že využití staršího ER modelu je pro návrh relační databáze vhodnější.

2.2 ER diagram

ER diagram byl zaveden a poprvé použit Peterem Chenem v roce 1976 [3]. Brzy došlo k jeho rozšíření a stal se obecně uznávaným standardem. Diagram obsahuje typy entit a typy vztahů. **Entitní typ** představuje nezávisle existující objekt reálného světa. **Vztahový typ** je vazba mezi dvěma nebo více typy entit. Samotný pojem **entita** či **vztah** označuje výskyt (instanci) typu entity či typu vztahu. V dalším textu budeme pro zjednodušení většinou používat pojem entita a vztah i pro entitní a vztahový typ, význam by měl být zřejmý z kontextu.



Obrázek 2.2 Ukázka ER diagramu v Chenově notaci

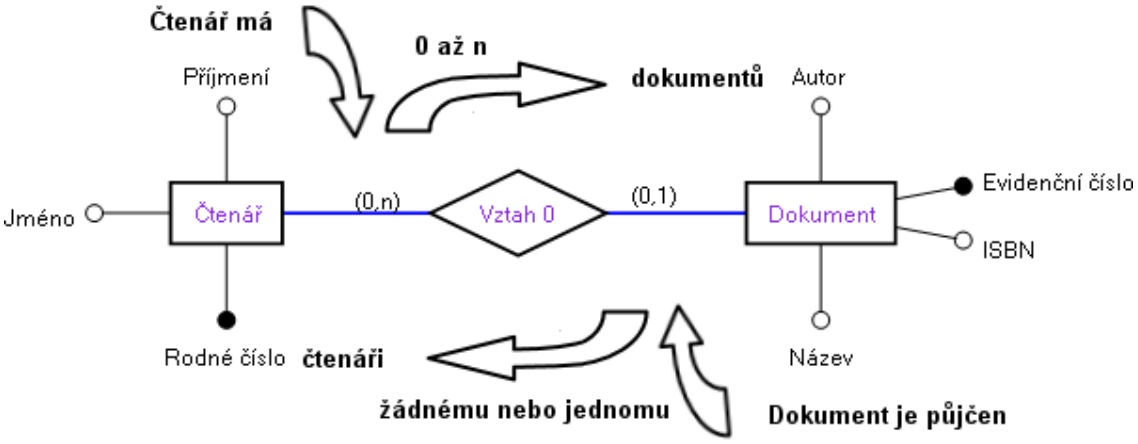
Entity se v těchto diagramech popisují pomocí obdélníků, vztahy se znázorňují pomocí kosočtverců. Entitám a vztahům lze přiřazovat určitou vlastnost označovanou jako **atribut**. Postupně vznikaly další způsoby, jak diagramy zakreslovat, původní notace byla totiž poměrně náročná na prostor (viz obr. 2.2).

Některé atributy jsou identifikační, tj. jsou součástí **identifikačního klíče** entity, ostatní jsou pouze popisné. Entity identifikovatelné pouze svými atributy jsou **silné entity**. Ostatní entity jsou označovány jako **slabé**. Pro jejich jednoznačnou identifikaci musí být použit i atribut jiné entity, tzv. **identifikačního vlastníka**. U atributu se kromě názvu upřesňuje množina hodnot, jakých může atribut nabývat (tzv. doména) a zda je atribut součástí klíče entity. Někdy se uvádí, zda atribut může mít prázdnou hodnotu *Null* a zda musí mít unikátní hodnotu (označováno jako *Unique*).

Vztah mezi entitami charakterizuje **kardinalita vztahu**. Kardinalita vztahu může nabývat těchto hodnot:

- 1 : 1 - každé entitě odpovídá maximálně jedna druhá entita.
- 1 : N - první entitě odpovídá více než jedna druhá entita, druhé entitě odpovídá maximálně jedna první entita. V tomto případě nazýváme druhou entitu **determinantem** první entity.
- M : N - první entitě odpovídá více než jedna druhá entita, druhé entitě odpovídá více než jedna první entita.

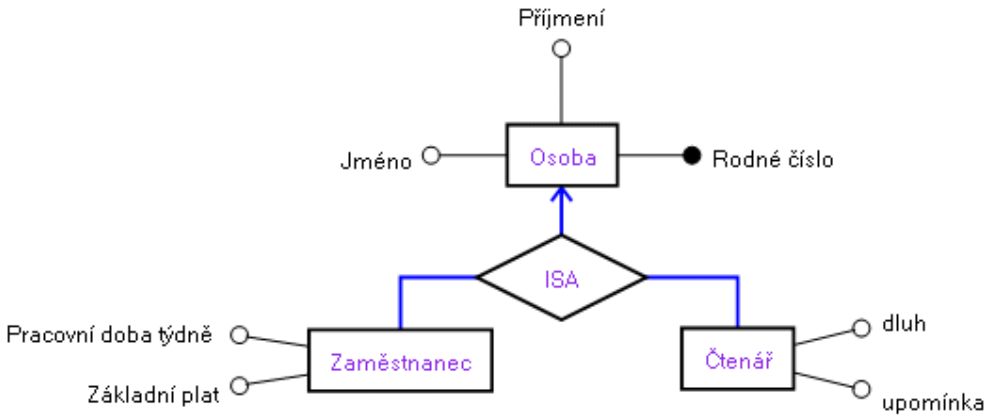
Jednotlivé entity, jejichž entitní typ se účastní vztahu, do vztahu nemusí vstupovat. Podle toho se rozlišuje povinné a nepovinné členství ve vztahu. Entitní typ, který má povinné členství ve vztahu, je **existenčně závislý** na druhém entitním typu.



Obrázek 2.3 Princip čtení kardinalit vztahu

Počet výskytů entity ve vztahu je ale často určen dvojicí hodnot, představujících minimum a maximum výskytů druhé entity. Taková kardinalita se tedy čte opačně než výše uvedený způsob zápisu. Například na obrázku 2.3 se čte kardinalita ve směru šipek a má tento význam: *Čtenář* má možnost si vypůjčit libovolný počet *dokumentů*, ale *dokument* může být vždy půjčen maximálně jednomu *čtenáři*.

Model může obsahovat také prvky pro vyjádření vztahu **generalizace-specializace** mezi entitami, tzv. **ISA hierarchii**. Využije se v případě, kdy máme obecnou entitu a několik dalších entit, které mají všechny vlastnosti obecné entity. Navíc se ale navzájem liší dalšími specializovanými vlastnostmi. Například vztah *Osoba ISA* (z angl. is a) *Zaměstnanec* nebo *Osoba ISA Čtenář* z obrázku 2.4 tvoří dva ISA vztahy. Specializované entity tak nemusíme vytvářet od začátku, protože zdědí všechny atributy a vztahy od svého předchůdce.



Obrázek 2.4 Příklad ISA hierarchie

Každou generalizaci můžeme charakterizovat dvojicí pojmů, první je buď totální nebo parciální, druhý vybíráme z dvojice výlučná a překrývající se [2]. ISA vztahy dohromady vytvářejí ISA hierarchii. Z důvodu jednoznačné identifikace se požaduje, aby v ISA hierarchii existoval pouze jeden uzel, z něhož nevychází žádná hrana, tzv. **zdroj ISA hierarchie**.

Model může být rozšířen o zvláštní druhy atributů. Příkladem **skupinového atributu** je atribut *Adresa*, dá se totiž dále dělit na název města, ulice a číslo domu. Skupinové atributy jsou užitečné, jestliže potřebujeme někdy přistupovat k jejich jednotlivým složkám a jindy nám vystačí atribut jako celek. **Vícehodnotové atributy** mohou zase obsahovat několik hodnot, ty ale musí být stejného typu.

2.2.1 Principy návrhu

Po schématu požadujeme, aby bylo smysluplné a jednoznačné co se týká sémantiky. Požadavky lze formulovat v definici **dobře definovaného schématu**: [1]

- Žádná entita nemá ve schématu více než jeden zdroj ISA hierarchie.
- ISA vztahy netvoří orientovaný cyklus.
- Identifikační vztahy netvoří orientovaný cyklus.
- Entita v ISA hierarchii, která není zdrojem ISA hierarchie, není identifikačně závislá na žádné entitě.
- Jména entit a vztahů jsou jednoznačná globální jména, jména atributů včetně zděděných jsou jednoznačná v rámci daného objektu.
- Vstupuje-li entita do vztahu vícekrát, je charakterizována různými rolemi.
- Zdroj ISA hierarchie (jsou jím i entity, které nevstupují do ISA vztahu) má identifikační klíč. Ostatní entity nemají identifikační klíč.
- Je-li identifikovaná slabá entita pro dva různé identifikační vztahy stejná, pak se buď vztahy liší v identifikačních zdrojích (obdoba ISA zdrojů pro hierarchii identifikačních vztahů) nebo druhé entity musí být odlišeny označením pomocí různých rolí .

Samotný popis prvků a definice korektnosti schématu nestačí. Je třeba se naučit, jak vytvářet dobrý návrh a čeho se vyvarovat. Návrh by měl věrně odpovídat specifikaci aplikace. Nic by ve schématu nemělo být řečeno dvakrát. Např. máme-li u entity *Kniha* atribut *Nakladatelství*, neměl už by být vytvořen mezi *Knihou* a *Nakladatelstvím* žádný vztah. Rozhodnutí, zda použít atribut nebo entitu se vztahem, není někdy jednoznačné (více viz [5]). Redundanci

způsobují i vztahy, které lze vyvodit z ostatních vztahů. Zahrnujeme jen ty elementy, které jsou pro aplikaci absolutně nutné.

Pokud jde o samotný vzhled diagramu, doporučují se dodržovat následující pravidla pro kreslení ER diagramů: [6]

- preferovat horizontálně kreslené vztahy (je obvyklé číst je zleva doprava);
- u kardinalit 1:N preferovat typ entity s N nalevo (od častějších entit k méně častým);
- minimalizovat šikmé čáry a křížení čar (křížení lze vyjádřit úhlem);
- nekreslit paralelní čáry těsně u sebe (zvyšuje to přehlednost čtení diagramu);
- pro konkrétní řešení nějakého problému používat podmnožiny ER diagramů;
- označit každý ER diagram jménem, datem a identifikací autora.

2.3 Transformace ER diagramu do relačního schématu

Výstupem této transformace je seznam tabulek, do kterých budou data uložena, obsahující názvy sloupců a integritní omezení. Nejdříve se postupně převádějí jednotlivé konstrukty ER. Poté se doporučuje výsledek podrobit analýze a vzniklé relace případně dále normalizovat, aby splňovaly požadovaný stupeň normální formy. Jinou možností je omezit možnosti ER diagramu nebo omezení vyřešit modifikací diagramu ještě před samotným převodem.

K mapování ER diagramů na databázové schéma existují dva přístupy [7]. Algoritmus plynoucí z prvního přístupu, o kterém se píše v [1], je vysvětlen v kapitole 2.3.1. Tento algoritmus byl použit při tvorbě programu ERTOS.

Druhý přístup je pro svá jednodušší výstupní schémata a lepší výkon při vyhodnocování dotazů databázovými systémy používán komerčními aplikacemi. Při druhém přístupu jsou navíc slévána schémata vzniklá ze vztahů s kardinalitou 1:N (bez ohledu na povinnou/nepovinnou účast determinantu) se schématy vzešlými z determinantu.

Jedná se o jednoduchou formu denormalizace. Vznikají při ní cizí klíče, které nejsou součástí primárního klíče, a hlavně je nutné povolit některým sloupcům hodnotu *Null*. Podle [1] by toto slévání v případě nepovinné účasti ve vztahu mělo proběhnout pouze tehdy, je-li identifikační klíč entity, která není determinantem, jednoatributový, což mnohdy není splněno.

Mechanická transformace nemusí nutně přinést optimální výsledek pro praktické použití. Některé tabulky se mohou následně spojit například z důvodu častých dotazů na jejich společná data [7].

2.3.1 Použitý algoritmus

Nejdříve budou popsány transformace jednotlivých části ER diagramu. Poté bude potřeba provést ještě určité úpravy, aby se zachovaly všechny závislosti a schémata odpovídala Boyce-Coddově normální formě (BCNF), což je stupeň normální formy, který budeme požadovat.

Reprezentace silného entitního typu není problém, převede se na tabulku se stejnými atributy (kromě vícehodnotových) a z identifikátoru entity se stane klíč tabulky. Nepovinný atribut se vyřeší sloupcem, který může obsahovat hodnoty *Null*. V případě, že je některý atribut vícehodnotový, budeme pro něj muset zřídit novou tabulku, která bude obsahovat

sloupce pro daný vícehodnotový atribut a atributy z klíče entity. Pokud byla kardinalita vztahu mezi entitou a vícehodnotovým atributem M:N, pak jsou klíčem nové tabulky všechny její sloupce.

Reprezentace slabého entitního typu bude vyřešena v rámci reprezentace vztahu s kardinalitou (0,N):(1,1) nebo (1,N):(1,1), protože slabý entitní typ se vyskytuje pouze v identifikační závislosti a identifikačně závislým se může stát jen v rámci vztahu s těmito kardinalitami.

Reprezentace binárních vztahů se liší podle příslušných kardinalit.

Kardinalita (1,1):(1,1)

Pro převod nám bude stačit jediná tabulka pro oba entitní typy. Dáme do ní všechny atributy obou entit a jejich vztahu, za klíč můžeme zvolit identifikační klíč buď jedné nebo druhé entity.

Kardinalita (0,1):(1,1)

Definujeme dvě schémata relací. Existenční závislost druhého entitního typu na prvním vyjádříme tak, že k jeho schématu přilepíme atributy odpovídající identifikačnímu klíči nezávislého entitního typu. Přidáme sem také atributy vztahu. Primárním klíčem může být opět libovolný z obou klíčů.

Kardinalita (0,1):(0,1)

Definujeme tři schémata, pro každý entitní typ a třetí pro vztah. Třetí schéma bude obsahovat identifikátory obou entitních typů, libovolný z nich může být primárním klíčem. Ostatní atributy v tabulce budou atributy vztahu.

Kardinalita (0 nebo 1,N):(1,1)

Definujeme dvě schémata, pro každý entitní typ jedno. Ke schématu entitního typu s kardinalitou (1,1) přidáme atributy odpovídající identifikačnímu klíči druhého entitního typu. Primárním klíčem tohoto schématu bude klíč entitního typu s kardinalitou (1,1).

Kardinalita (0 nebo 1,N):(0,1)

Definujeme tři schémata, pro každý entitní typ jedno, třetí pro vztah. Schéma pro vztah bude obsahovat atributy, které jsou identifikační u jednoho nebo druhého entitního typu a případně atributy vztahu. Primárním klíčem tohoto schématu bude klíč entitního typu s kardinalitou (0,1).

Kardinalita (0 nebo 1,N):(0 nebo 1,N) 

Podobně jako v předchozím případě definujeme tři schémata, pro každý entitní typ jedno, třetí pro vztah. Třetí schéma bude obsahovat identifikátory z entitních typů a vztahové atributy. Primárním klíčem schématu bude všechny atributy z identifikačních klíčů obou entitních typů.

Při **reprezentaci n-árních vztahů** definujeme bez ohledu na typ členství n+1 schémat, pro každý entitní typ jedno, poslední bude vztahové. Vztahové schéma bude obsahovat atributy tvořící identifikační klíč ve všech zúčastněných entitních typech a další atributy odpovídající atributům vztahu. Primární klíč ale obecně nemusí obsahovat všechny tyto identifikační klíče. V klíči nebudou figurovat identifikátory entit, které v kardinalitě vztahu mají přiřazenou hodnotu (0 nebo 1,1).

To byly základní postupy. Takto popsané s pomocí Chenovy notace je najdeme např. v [8] a [9]. Provedou se postupně metodou shora-dolů pro celý ER model. Poté ale, abychom přiblížili relační schéma jeho vzoru na konceptuální úrovni, musíme kromě primárních klíčů přidat další integritní omezení (doplňuje požadavky na vztahy mezi daty, které se nedají vyjádřit v ER modelu), a to referenční integritu. **Referenční integrita** popisuje vztah mezi atributy obsaženými ve dvou relacích pomocí cizích klíčů. Jde o atributy, jejichž hodnota je buď *Null* nebo musí být obsažena jako hodnota primárního klíče jiné relace.

Pro každý transformovaný vztah zavedeme integritní omezení říkající, že hodnoty všech cizích klíčů FK(z angl. Foreign Key) jsou podmnožinou hodnot sloupců K jiného schématu, kde FK tvoří primární klíč K. Je to proto, aby se do schématu, kde atributy tvoří cizí klíč, nedaly vkládat hodnoty, které ve skutečnosti v tabulce, kde jsou primárním klíčem, neexistují. Podobně pro slabé entitní typy, hodnoty cizích klíčů ze schématu odpovídající slabému entitnímu typu jsou podmnožinou hodnot příslušných primárních klíčů ze schémat jejich identifikačních vlastníků.

Protože chceme vytvořit co nejobecnější nástroj, budeme se snažit nezakazovat konstrukce, které se dají následně převést či obejít, i když to bude algoritmicky náročnější. Často jsou požadována tato omezení v ER modelu: žádné cyklické struktury, všechna jména globálně jednoznačná, zákaz vícehodnotových atributů. Cyklické struktury v ER diagramech jsou při převodu nežádoucí, protože nesplňují předpoklad jednoznačnosti vztahů mezi atributy. Problém se dá vyřešit přerušením cyklu tím, že jednu jeho hranu označíme jménem role a příslušný název role použijeme jako předponu cizího klíče při konstrukci relačního schématu.

S přejmenováváním je obecný problém, povolíme-li totiž např. jména atributů, která jsou jednoznačná jen v rámci jedné entity, může nastat při sdružení atributů dvou entit do jedné tabulky problém s jejich stejným pojmenováním. Pro vyřešení se nejdříve použije funkce *rename(O,A),* která bude pojmenovávat atributy pomocí tečkové notace tímto způsobem: je-li v konceptuálním schématu objekt *O* jediným, kde se vyskytuje atribut *A*, pak *rename(O,A) = A,* v opačném případě *rename(O, A) = O.A*. Následně proběhne níže popsaný algoritmus pro přejmenování cizích klíčů. Dá se ukázat, že po použití tohoto algoritmu (a dodržení výše popsaných pravidel) bude výsledné schéma v BCNF [1].

Algoritmus přejmenování cizích klíčů

Z ER diagramu vytvoříme graf spojení G = (V,H) tímto způsobem:

- Množina uzlů V obsahuje entitní typy (slabé i silné) a vztahové typy (ne identifikační).
- Množina hran H obsahuje neorientované hrany (V1,V2), kde pro uzly V1 a V2 platí jedna z následujících podmínek:
    - entitní typ V1 se účastní typu vztahu V2;
    - V1 a V2 jsou v identifikačním vztahu.

Následně pro graf spojení G provádíme:

1. Dokud v G existuje cyklus a žádná z hran není označená (některé hrany mohou být už na počátku označeny rolí, protože se jedná o vztah mezi entitami stejného typu), vybere se nějaká hrana a označí se jménem role.
2. Každý cizí klíč FK z relačního schématu databáze, které vzniklo transformací z ER diagramu, se přejmenuje podle jedné z následujících možností:
    - Je-li FK ve vztahové relaci odpovídající typu vztahu R a odpovídá-li identifikátoru entity E, pak každý A patřící do FK nahradíme L.A, existuje-li role L hrany (R,E) v grafu spojení. V opačném případě zůstanou atributy FK beze změny.
    - Je-li FK v relaci odpovídající slabému entitnímu typu E1 a odpovídá-li identifikátoru svého identifikačního vlastníka E, pak každý A patřící do FK nahradíme L.A, existuje-li role L hrany (E1,E) v grafu spojení. V opačném případě zůstanou atributy z FK beze změny.

Ve skutečnosti takový graf v aplikaci ERTOS nevzniká, tečková notace je nahrazena „podtržítkovou“, ale myšlenka tohoto algoritmu je dodržena.

3 POŽADAVKY NA PROGRAM

3.1 CASE nástroj

Nástroje, které podporují vývoj softwarových aplikací, se nazývají **CASE** (z angl. Computer Aided Software Engineering). CASE nástrojů existuje celá řada, liší se nejen podporovanou metodikou, ale také tím, v jaké fázi vývoje je nástroj používán. Pro standardní ohodnocení CASE nástroje byla vytvořena norma IEEE, která uvádí několik desítek kritérií. Součástí hodnocení CASE je také úroveň podpory určité metodiky [10].

Mezi důležité funkce a vlastnosti CASE obecně patří: [11]

- *Konzistentní grafické ovládací prostředí* (podle zásad tvorby GUI) – jednotný vzhled obrazovek, popisků, tlačítek, jednotné ovládání, použití symbolických ikon a podobně.
- *Centrální databáze* pro uchování informací o všech objektech (slovník).
- *Prostředky pro ověřování konzistentnosti dat a podpora normalizace dat* .
- *Textový editor* pro popis jednotlivých objektů – pro účely vytváření dokumentace systému, kterou je možno často ze systému generovat.
- *Export / import dat* – pro práci s modely a dokumentací, které byly vytvořeny v jiných programech nebo jsou v jiných programech dále využívány a zpracovávány.

Grafické rozhraní se skládá z předdefinovaných primitiv (kružnice, čtverce, přímky, křivky, šipky), které je možno uložit. Grafické rozhraní vlastně představuje elektronickou tabuli, na kterou analytik konstruuje grafy a diagramy. Úsilí, jaké je třeba k vytvoření diagramu, může být měřeno počtem stisků klávesy nebo kliknutí myši.

Jednou z vlastností grafického rozhraní je také to, že systém automaticky vymaže odkazy na mazaný objekt, aby byla zachována konzistence modelu. Grafické rozhraní by mělo rovněž umožňovat přemístění jednoho nebo více objektů. Z dalších požadovaných vlastností uveďme obnovu objektů do jejich předchozího stavu, a to i o více kroků, nebo změnu měřítka zobrazovaných objektů.

Důležitou vlastností CASE nástrojů je možnost **kontroly správnosti modelu**. Systém kontroluje tvořené modely a diagramy podle pravidel jejich vytváření a podle definovaných logických souvislostí. Ty jsou všechny uloženy ve slovníku, který se ke kontrolám používá.

Na složitých projektech, při kterých jsou v praxi využívány CASE nástroje, pracuje obvykle mnoho odborníků. Proto je třeba, aby nástroje CASE podporovaly práci v síti, výměnu dokumentů, správu verzí, různé formy komunikace a také použití centrálního slovníku.

CASE není samo o sobě metodologií, ale používá již zavedené metodologie. Použití CASE nástroje povede k dobrým výsledkům pouze v případě, že metody, na kterých je CASE založen, jsou již týmu blízké. Produktivita dosažená pomocí CASE není okamžitě zřejmá, ani by hned od začátku neměly být používány všechny funkce nástroje. Diagramy ulehčují komunikaci v týmu a pomáhají zpřesnit analýzu systému. Nástroj CASE slouží i zákazníkovi, kterému umožní najít případnou chybu v diagramech.

CASE nástroje pro analýzu problému a specifikaci požadavků podporují také **návrh ER diagramů**. Ty lze navrhovat od počátku (pomocí modelování), převodem z objektového modelu (diagramu tříd) nebo vygenerováním z existující databáze (tzv. reverse engineering).

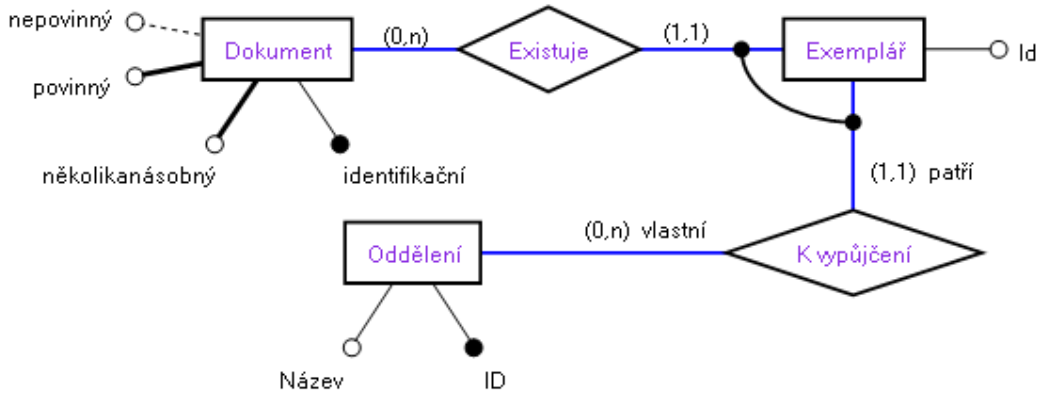
Z vytvořeného datového modelu pak umějí aplikace vygenerovat SQL skript. Často je program přímo propojen se zvoleným databázovým serverem a rovnou vytváří výsledné tabulky. Samozřejmě se jednotlivé nástroje odlišují škálou podporovaných databázových systémů.

3.2 Notace ER diagramu

Program ERTOS vytvářený v rámci práce má sloužit hlavně k výukovým účelům. Z tohoto důvodu byla zvolena notace používaná při výuce předmětu Databázové systémy na MFF UK, která vychází z notace použité ve skriptech [1]. Notace vypadá takto:

- **entita** je zobrazena pomocí obdélníku obsahujícího pouze název entity;
- **vztah** je zobrazen pomocí kosočtverce s názvem vztahu, který je propojen úsečkami s entitami, které se vztahu účastní; kolem příslušné úsečky bude vypsána kardinalita vztahu, případně také role, jsou-li zadány;
- **atribut** je zobrazen úsečkou vycházející z příslušné entity (nebo vztahu), úsečka je ukončena prázdným (bílým) kolečkem v případě obyčejného atributu a plným (černým) kolečkem v případě identifikátoru entity, poblíž kolečka je nápis názvu atributu;
- v případě, že se **identifikátor** skládá z více atributů nebo atributu a vztahu, bude každá část identifikátoru vyznačena černým kolečkem, která budou všechna navzájem spojena pomocí oblouku
- **ISA hierarchie** je zobrazena jako vztah, který je se zdrojem hierarchie propojen úsečkou se šipkou směřující ke zdroji (viz obrázek 2.2)

Mírné odlišnosti se vyskytují v zobrazení nepovinných a vícenásobných atributů. Ty jsou v programu zobrazeny pomocí čárkované v případě nepovinných a tučné čáry v případě vícenásobných atributů.



Obrázek 3.1 Ukázka ER diagramu ve zvolené notaci

Na obrázku 3.1 je ukázka části diagramu. Entita *Dokument* má jen ilustrativní atributy, aby se ukázalo zobrazení jednotlivých druhů atributů. Vztah *Existuje* má kardinalitu (0,N):(1,1). Entita *Exemplář* je identifikačně závislá na entitách *Dokument* a *Oddělení* (vyznačeno obloukem), *patří* a *vlastní* jsou názvy rolí u vztahu *K vypůjčení*.

4 UŽIVATELSKÁ DOKUMENTACE

4.1 Instalace programu

Program ERTOS je napsán pro operační systém Windows. Pro svůj běh vyžaduje Windows 98 nebo vyšší s instalovaným .NET Frameworkem 1.1. Pokud není nainstalován, je jeho instalace zajištěna během instalace programu.

Program je dodáván v podobě instalačního balíčku s názvem ErtosSetup.exe. Ten existuje ve dvou podobách. V prvním případě balíček obsahuje také soubor dotnetfx.exe obsahující .NET Framework 1.1. Tato instalace se hodí pro offline instalaci (například z CD). Pokud .NET Framework není na cílovém počítači nainstalován v případě druhého typu instalace, bude uživatel vyzván ke stáhnutí .NET Frameworku z internetu. S tímto typem instalace se setkáte zejména po stáhnutí balíčku z internetu.

Vše potřebné provede samotný průvodce instalací, který spustíte souborem ErtosSetup.exe. Můžete si zvolit umístění, kam chcete program instalovat, zda a kde chcete vytvořit zástupce v nabídce Start nebo na ploše. K běhu programu je potřeba knihovna MagicLibrary.dll, tuto knihovnu prosím ponechte v instalačním adresáři.

4.2 Spuštění programu

Program se spouští souborem ERTOS.exe, který se nachází v instalačním adresáři. Po spuštění se otevře hlavní okno aplikace (obr. 4.1). V jednom spuštěném programu může uživatel pracovat pouze s jedním diagramem. Je však možno spustit v jeden okamžik více instancí programu a mezi diagramy předávat data přes schránku Windows. Pokud jako parametr při spuštění uvedete název souboru s ER diagramem, bude soubor po spuštění otevřen. Pro spuštění souboru s uloženou historií změn je nutné za název souboru přidat ještě parametr *–h*.

K rychlému vyzkoušení můžete otevřít některý ze vzorových souborů, které se nacházejí v podadresáři *Priklady*. Na pracovní ploše se vykreslí celý diagram. Pokud přesahuje okno, můžete se v něm pohybovat kurzorovými šipkami nebo pomocí posuvných lišt. V horní části okna pod menu se nachází nástrojová lišta s ikonami pro rychlý přístup k často používaným položkám menu. V dalším textu se budeme odkazovat pouze na menu s tím, že po najetí myší na ikonu v nástrojové liště se zobrazí název ikony, který odpovídá položce v menu. Samozřejmě je použití ikony z nástrojové lišty rychlejší než hledání položky v menu.

V podadresáři instalačního adresáře *Napoveda* lze najít také referenční příručku s názornými obrázky pod názvem ErtosHelp.chm.

4.3 Editace ER diagramu

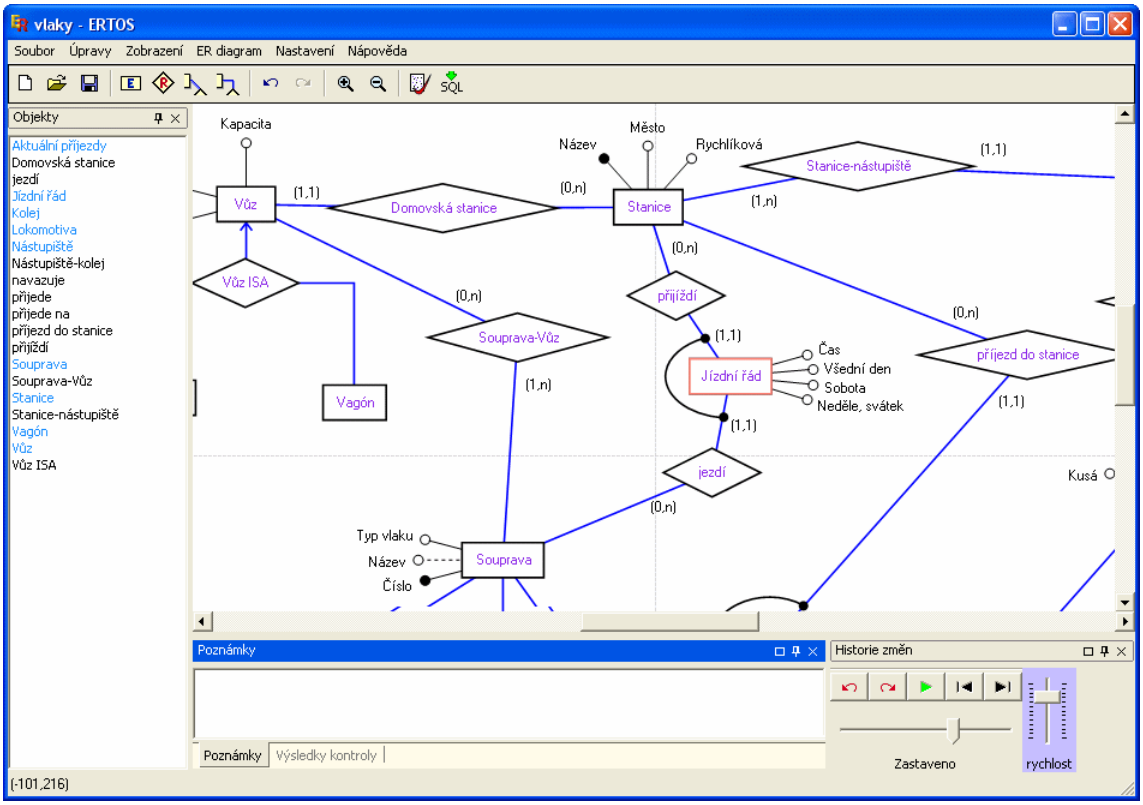
V programu je použita notace, která je popsána v kapitole 3.2 v této práci

4.3.1 Vytvoření

Po spuštění programu se vytvoří nový soubor pro vytváření ER diagramu. Nový soubor lze také vytvořit kdykoliv pomocí menu *Soubor* položka *Nový*. Pokud byly ve stávajícím souboru

provedeny nějaké změny, budete před otevřením nového souboru dotázáni, zda chcete změny uložit.

Nyní je možno přidat a umístit entity a vztahy. **Entity** lze přidávat po aktivování módu přidávání entit pomocí menu *ER diagram* položka *Přidávat entity*. Od tohoto okamžiku se po každém kliknutí myší na volné místo na pracovní ploše vytvoří nová entita. Vytváření entit se deaktivuje opětovným výběrem položky z menu nebo klávesou Escape. Dvojklikem myši na obrázku entity se otevře dialog pro editaci entity (obr. 4.3), v němž je možné si prohlédnout či editovat logický a fyzický název entity a hlavně její atributy.



Obrázek 4.1 Hlavní okno aplikace ERTOS

Propojit entity pomocí vztahů je možné dvěma způsoby. Snazší je použití **Automatického propojení**. Při něm je nejprve nutno vybrat z menu *ER diagram* položku *Přidávat vztahy*. Poté každé kliknutí myší na volné místo pracovní plochy způsobí přidání vztahu na místo, kam bylo kliknuto.

Nový vztah se zároveň automaticky propojí s nejbližšími dvěma entitami, pokud existují. Ty jsou před položením vztahu zvýrazněny žlutou barvou. Sledujte tedy už před přidáním vztahu žlutě zvýrazněné entity a vztah položte tak, aby byli zvýraznění požadovaní účastníci vztahu. I přes zvýraznění lze stále rozpoznat, zda je entita označená. Vytváření vztahů se deaktivuje opětovným výběrem položky z menu nebo klávesou Escape.

Vytvořená automatická propojení můžete následně zrušit v dialogu pro editaci vztahu (obr. 4.4), který se otevře po dvojkliku myší na obrázku vztahu. V části dialogu nazvané *Účastníci vztahu* je vedle názvu každého účastníka tlačítko *Smazat*, které smaže propojení entity se vztahem.

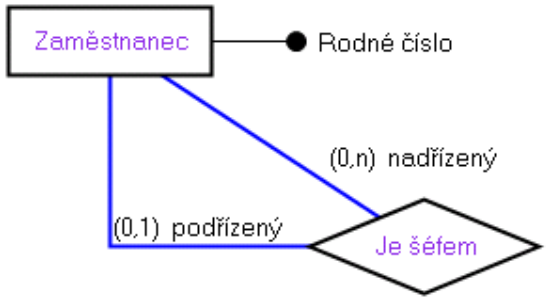
Když uživateli automatické propojení nevyhovuje nebo chce propojit vztahem více než dvě entity, nezbývá než použít druhého, tzv. **Ručního způsobu propojování**. K jeho použití je potřeba předem rozmístit nejen potřebné entity, ale i vztahy. Pro vytvoření vtahu musíte mít vybranou v menu *ER diagram* pouze položku *Přidávat vztahy*. Nesmí být zaškrtnuta žádná z položek *Propojovat přímkou* a *Propojovat lomeně*. Pokud je v dialogu *Nastavení* nastaveno, že se má preferovat automatické propojení, jedna z těchto položek se automaticky zaškrtne spolu s položkou *Přidávat vztahy*. Každé kliknutí na volné místo na pracovní ploše pak vytvoří nový vztah, který se nepropojí s žádnou entitou.

Nyní jestliže chcete propojit určitou entitu a vztah, vyberte jednu z položek *Propojovat přímkou* a *Propojovat lomeně* (nesmí být vybrána žádná jiná položka). Poté klikněte na entitu, přesuňte se nad vztah, od entity k myši by se měla natahovat propojovací čára. Propojení se vztahem dokončete kliknutím na vztah. Propojování můžete zrušit pomocí klávesy Escape nebo klikem pravým tlačítkem myši. Skutečnost, zda si vyberete propojení přímkou nebo lomeně, má vliv pouze na to, jaký styl čáry se bude mezi entitou a vztahem vykreslovat.

V případě, že chcete změnit typ již vytvořené propojovací čáry, znovu propojte příslušný vztah s entitou druhým typem čáry. Nevidíte-li nikde poblíž čáry vztahu zápis kardinality, zkuste změnit typ propojující čáry na přímou. V případě lomené čáry, která má celou jednu část lomené čáry schovanou pod některým z objektů, nemusí být kardinality viditelné. Při propojování lomenou čarou můžete pomocí kliknutí na volné místo na pracovní ploše změnit orientaci lomené čáry. V případě ISA hierarchie nelze změnit lomený styl čáry. Změnit typ všech čar najednou lze pomocí funkce v menu *ER diagram* položka *Změnit typ čar*.

4.3.2 Self relace

Chcete-li vytvořit vztah mezi dvěma stejnými entitami, je nutné použít vlastní (self) relaci. Pro její vytvoření klikněte na entitě pravým tlačítkem myši a vyberte příkaz *Vytvořit self relaci*. Vznikne obrázek vztahu, dvojklikem na něm aktivujete dialog pro editaci vztahu. V tomto dialogu je vhodné změnit názvy rolí. Pro odlišení dvou stejných entit v jednom vztahu je nutné zadat pro každou z entit jiný název role. Role jsou pak zapsány vedle propojujících čar mezi entitou a vztahem (viz obrázek 4.2).



Obrázek 4.2 Příklad self relace s názvy rolí

V případě změny typu čáry u self relace se před dokončením objeví jednoduchý dialog, ve kterém budete dotázáni, kterou z čar chcete změnit. Program nedovoluje vytvořit ternární self relaci.

4.3.3 ISA hierarchie

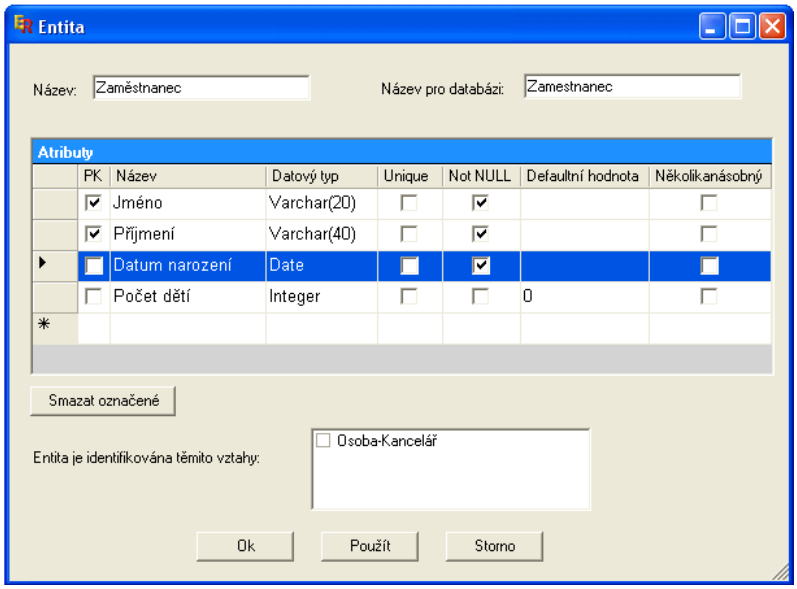
ISA hierarchie je způsobem vyjádření dědičnosti v ER diagramu. Jestliže chcete vytvořit ISA hierarchii pro určitou zdrojovou entitu, klikněte na entitě pravým tlačítkem myši a vyberte příkaz *Vytvořit ISA hierarchii*. Vznikne obrázek vztahu, který je připojen k entitě čárou se šipkou (obr. 2.4).

Pro připojení existující entity k ISA vztahu tak, aby se stala v hierarchii potomkem, se používá *Ruční propojení* entity se vztahem, které je popsáno v části o vytváření ER diagramů. Je respektováno pouze propojení lomenou čárou.

4.3.4 Editace entity

Entity lze libovolně přemísťovat tažením myši. Pro editaci informací o entitě otevřete dialog pro editaci vlastností entity (obr. 4.3). Kromě dvojkliku na obrázku entity lze dialog vyvolat také z kontextového menu entity, které se otevře po kliknutí pravým tlačítkem myši.

Každá entita má dva názvy. *Název* (logický název) se využívá všude při editaci schématu. Doporučuje se zde uvádět název, který entitu dostatečně názorně popíše. V *Názvu pro databázi* se pak ještě nahradí speciální znaky včetně mezer podle nastavení uživatele a název se použije ve výsledném SQL skriptu jako součást názvů tabulek a atributů. Po zadání *Názvu* se pokaždé upraví *Název pro databázi* podle pravidel, která se nastavují v dialogu *Nastavení*. Můžete si vybrat různé způsoby zkrácení názvu a zvolit, zda chcete odstranit diakritiku.



Obrázek 4.3 Dialog pro editaci entity

Významným obsahem dialogu je tabulka se všemi **atributy**, které entita má. Každý řádek představuje jeden atribut. Způsob setřídění atributů můžete změnit kliknutím na záhlaví sloupce, podle kterého chcete atributy setřídit. Nový atribut vytvoříte kliknutím do prvního následujícího prázdného řádku. U atributu lze specifikovat následující informace:

- **PK** - říká, zda má být atribut součástí primárního klíče. Všechny atributy, které mají zaškrtnuto *PK*, budou součástí primárního klíče.
- **Název** - název atributu.

- **Datový typ** - datový typ atributu můžete vybrat z množiny typů: *integer*, *float*, *char*, *varchar*, *date*. Po vybrání datového typu *varchar* budete dialogem vyzváni k zadání maximální délky řetězce.
- **Unique** - říká, zda je atribut unikátní, tj. jeho hodnota nesmí být stejná pro žádné dva atributy. *Unique* se týká vždy jen jednoho atributu, atributy se nesdružují jako u *PK*.
- **Not Null** - je-li zaškrtnuto, atribut nesmí mít hodnotu *Null*.
- **Defaultní hodnota** - výchozí hodnota pro atribut.
- **Několikanásobný** - je-li zaškrtnuto, může být jedné entitě přiřazeno více hodnot tohoto atributu. Při převodu vede na samostatnou tabulku.

Atribut, který je součástí primárního klíče, nesmí obsahovat hodnoty *Null*. Zaškrtnete-li tedy sloupec *PK*, zaškrtne se také *Not Null*. Atribut také nemůže být zároveň součástí primárního klíče a mít zaškrtnutý sloupec *Unique*, protože *Unique* se týká pouze jednotlivých atributů, které nejsou součástí primárního klíče.

Tlačítko *Smazat označené* smaže všechny atributy na vybraných (modře svítících) řádcích.

Atributy se samy vykreslují na vhodné místo v okolí obrázku příslušné entity. Jejich pozici nelze přímo nijak změnit. Atributy pouze uhýbají před propojujícími čarami a ostatními vztahy a entitami, pokud mají kam. Tato funkčnost odstiňuje uživatele od zdlouhavého umísťování jednotlivých atributů.

Některé entity mají identifikátor složen nejen ze svých atributů, ale jejich identifikátor obsahuje také identifikátor jiných entit. **Identifikační závislost** entity na jiné entitě lze vyjádřit zaškrtnutím vztahů, přes který je původní entita identifikována, v bílém obdélníku ve spodní části dialogu. Vztahy jsou v bílém obdélníku nabízeny pouze tehdy, je-li na straně původní entity v příslušném vztahu kardinalita (1,1).

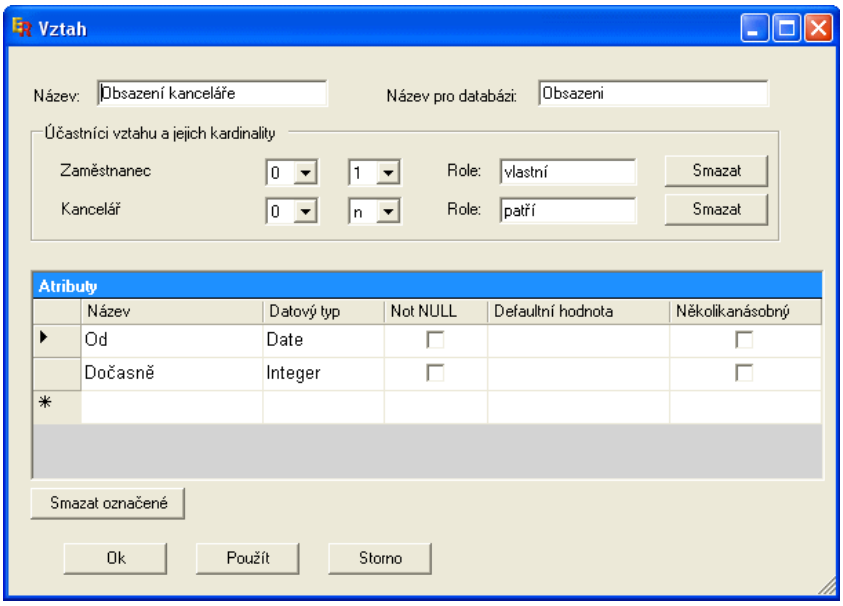
4.3.5 Editace vztahu

Podobně jako entity lze editovat vztahy. Spolu s posouváním obrázku entity nebo vztahu se posouvají také propojující čáry. Polohu propojujících čar nelze změnit přímo. Jejich poloha a směr je dána umístěním entity a vztahu, které čára propojuje. Také hodnoty kardinalit a názvy rolí se vypisují automaticky podle polohy čáry.

Pro editaci informací o vztahu otevřete dialog pro editaci vlastností vztahu (obr. 4.4). Kromě dvojkliku na obrázku entity lze dialog vyvolat také z kontextového menu vztahu, které se otevře po kliknutí pravým tlačítkem myši.

Problematika názvů a atributů je stejná jako u entit. Jediný rozdíl je v menším počtu sloupců v tabulce atributů, protože u vztahů nemá smysl uvažovat identifikující atributy. Chybí tedy sloupce *PK* a *Unique*. U vztahů taktéž nejsou podporovány několikanásobné atributy.

V dialogu se dají změnit **hodnoty kardinalit** účastníků vztahu. Dvojice čísel tvořící kardinalitu plní funkci minima a maxima, příslušná kardinalita je umístěná ve stejném řádku jako příslušná entita. Vedle dvojice polí pro zadávání kardinalit leží pole pro editaci názvu role. V této části dialogu lze také zrušit připojení jednotlivých entit ke vztahu tlačítkem *Smazat* ležícím na stejném řádku jako název entity.



Obrázek 4.4 Dialog pro editaci vztahu

Na příkladu z obrázku 4.4 kardinalita (0,1) u entity *Zaměstnanec* znamená, že ke každému *Zaměstnanci* existuje žádná nebo jedna *Kancelář*, do které *Zaměstnanec* patří. Opačně, v každé *Kanceláři* sídlí žádný nebo několik (n) *Zaměstnanců* (kardinalita (0,n)).

4.3.6 Označování objektů

Přesouvání objektů a také jejich kopírování, mazání a vyjímání se týká vždy všech označených objektů. Označených objektů může být více najednou. Jak se dají objekty označit? Jeden objekt se označí jednoduše kliknutím na jeho obrázek. Označení se projeví změnou barvy obrázku.

Jestliže chcete vybrat objekty ze souvislé oblasti schématu, klikněte levým tlačítkem a táhněte myší. Pod myší by se měl vykreslovat obdélník kreslený přerušovanou čarou. Tlačítko myši pusťte, jestliže je v obdélníku celá oblast, kterou jste chtěli vybrat. Výběr souvislé oblasti funguje jen tehdy, není-li klik myší spojen s nějakou akcí, tj. není aktivován mód přidávání entit ani vztahů.

Jestliže už je něco označeno a chcete přidat další označený objekt, klikněte myší na nově označovaný objekt spolu se stiskem klávesy Shift. Můžete také spolu se stiskem klávesy Shift přidat souvislou oblast způsobem uvedeným v předchozím odstavci. Pomocí klávesy Shift lze rovněž odznačovat objekty, jsou-li momentálně označeny.

4.3.7 Vlastnosti diagramu

Každý správný ER diagram by měl mít uvedeno, kdo je jeho autorem či ke kterému projektu patří. K zadávání metadat o schématu slouží dialog *Vlastnosti diagramu*. Můžete zde také zjistit datum a čas vytvoření a také poslední změny diagramu. Dialog vyvoláte pomocí menu *ER diagram* položka *Vlastnosti*. Jestliže jednotlivé položky vyplníte a při převodu do SQL zaškrtnete volbu *Přidat komentáře*, objeví se tyto informace ve výsledném skriptu.

4.4 Uložení do souboru

Pomocí menu *Soubor* položka *Uložit* nebo *Uložit jako* lze ER diagram uložit ve dvou formátech. Do formátu ER se uloží diagram klasicky bez historie změn. To znamená, že při jeho příštím otevření nebude možné vracet zpět změny provedené před uložením. Formát ERH naopak obsahuje celou historii změn. Lze se vracet k minulým stavům diagramu nebo si prohlédnout pomocí plovoucího okna *Historie změn* animaci, jak diagram vznikal.

Nechybí možnost uložení diagramu ve formě obrázku, provede se pomocí menu *Soubor* položka *Uložit jako obrázek*. Obrázky je pak možno prohlížet nebo vytisknout. Na výběr jsou formáty BMP, GIF a vektorový formát EMF. Uloží se vždy jen taková část digramu, která obsahuje nějaké objekty. Do formátu BMP a GIF se diagram uloží přesně tak, jak ho vidíte na obrazovce, respektive jak byste ho viděli, kdybyste měli obrazovku větší než diagram. To znamená, že je zohledněno zvolené měřítko.

4.5 Kontrola schématu

Již při vytváření schématu se kontroluje, zda se dva objekty, atributy nebo role v ER diagramu neshodují v názvu. Přitom pro názvy atributů a rolí stačí, že jsou jedinečné v rámci entity nebo vztahu, ke kterým přísluší.

Identifikační vztah lze vytvořit jen v případě, že je u vztahu na straně závislé entity kardinalita (1,1). Při existenci identifikačního vztahu tuto kardinalitu nelze změnit. Krátce před převodem do SQL skriptu se pak kontroluje, zda každá entita má identifikační atribut nebo aspoň identifikační vztah a zda identifikační vztahy netvoří orientovaný cyklus.

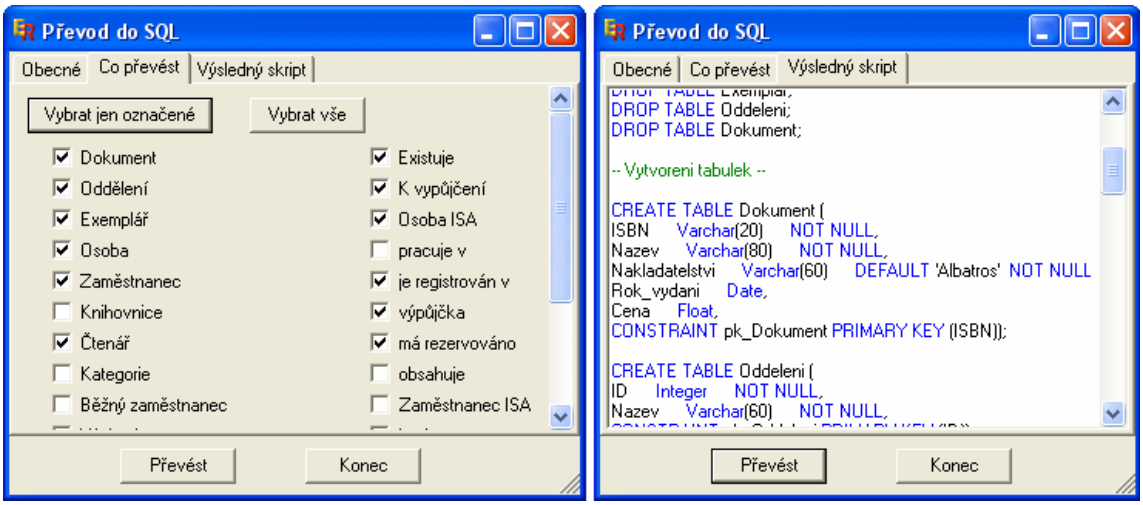
Zavedení **ISA hierarchie** s sebou přináší další kontroly. Kontrolujeme, zda platí, že členy ISA hierarchie kromě zdrojového členu nemají vlastní identifikátor. Zároveň nesmí mít žádná entita více zdrojů ISA hierarchie a potomek v ISA hierarchii nesmí být identifikačně závislý.

Kontrola se provádí automaticky před každým převodem do SQL. Můžete ji také vyvolat pomocí menu *ER diagram* položka *Zkontrolovat*. Případné chyby se objeví v plovoucím okně *Výsledky kontroly*.

4.6 Transformace schématu do SQL

Když máte aspoň část schématu hotovou, můžete ho převést do skriptu SQL, který po zadání do databázového klienta vytvoří příslušné tabulky databáze. Dialog pro převod do SQL (obr. 4.5) vyvoláte vybráním položky *Převést do SQL* v menu *ER diagram*.

Dialog obsahuje tři záložky. V záložce *Obecné* pomocí tlačítka *Vybrat* vyberete název souboru, do kterého se výsledný skript uloží. Převod můžete provést, i když nevyberete žádný výstupní soubor. Výsledek se vždy vypíše také do záložky *Výsledný skript*, ve které je podporováno zvýraznění syntaxe SQL. Je-li zaškrtnuto *Přidat SQL komentáře*, budou do výsledného skriptu přidány komentáře pro oddělení částí skriptu, čas vygenerování a také informace z dialogu *Vlastnosti diagramu*. Pole *Přidat skript pro smazání tabulek* přidá na začátek skriptu část, která obsahuje příkazy pro smazání všech vytvářených tabulek. Volba *Pojmenovat primární a cizí klíče* přidá do skriptu pojmenování primárních a cizích klíčů vygenerovanými názvy.



Obrázek 4.5 Vybrané záložky z dialogu pro Převod do SQL

Na záložce *Co převést* si můžete vybrat, které entity do výsledného SQL skriptu zařadit. Do výsledného skriptu budou zahrnuty jen zaškrtnuté entity a vztahy, které ve schématu sousedí s alespoň jednou zaškrtnutou entitou. Mohou být přidány také další entity potřebné pro zajištění identifikace zaškrtnutých entit. Při prvním otevření dialogu pro daný soubor jsou zaškrtnuty všechny entity a všechny vztahy. Při každém dalším otevření je zachován výběr z předchozího převodu do SQL. Můžete zde použít tlačítek pro rychlejší výběr všech objektů nebo označených objektů. Pokud tedy chcete mít zaškrtnuté jen některé entity a vztahy, tyto entity a vztahy označte ještě před otevřením dialogu pro převod do SQL a poté zmáčkněte tlačítko *Vybrat jen označené.*

4.7 Podrobný popis pracovního prostředí

4.7.1 Menu a toolbar

V horní části okna programu se nachází menu (obr.4.6), které obsahuje téměř všechny funkce programu. V menu nenajdeme jen položky pro přehrávání historie změn (dá se ovládat z plovoucího okna *Historie změn*) a funkce vytvoření ISA a self vztahů, které je nutno vyvolat v kontextovém menu určité entity. Nástrojová lišta (toolbar) ležící pod menu představuje rychlý přístup k nejpoužívanějším položkám menu. Po najetí myši nad některou z ikon v toolbaru se objeví název ikony. Název je stejný jako položka v menu, která vykonává stejnou funkci.

Nabídka **Soubor** obsahuje tyto položky:

- *Nový* - vytvoří nový prázdný soubor; jestliže jsou v aktuálně otevřeném souboru neuložené změny, program se uživatele dotáže, zda chce soubor uložit.
- *Otevřít* - otevře dialog pro otevření souboru, před otevřením se dotáže, zda chce uživatel uložit aktuální soubor.
- *Uložit* - uloží změny v souboru; jestliže soubor dosud nebyl uložen, zobrazí dialog pro výběr názvu souboru.
- *Uložit jako* - zobrazí dialog pro výběr typu a názvu souboru, do kterého bude diagram uložen.
- *Uložit jako obrázek* - uloží ER diagram do vámi zvoleného grafického formátu.

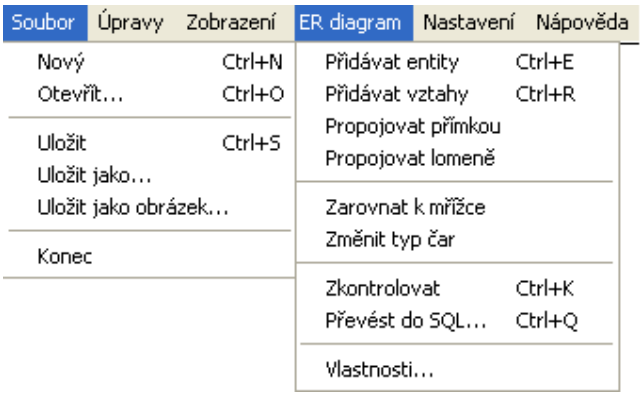
- *Konec* - ukončí program, před ukončením se dotáže, zda chce uživatel uložit aktuální soubor.

Nabídka **Úpravy** obsahuje tyto funkce:

- *Zpět* - vrátí zpět poslední změnu; uchovávaná historie operací není omezená, tedy veškeré změny lze vrátit zpět.
- *Znovu* - znovu vykoná akci, která byla jako poslední vrácena zpět a ještě nebyla vykonána znovu; uchovávaná historie operací není omezená.
- *Kopírovat, vložit, vyjmout, smazat* - klasické operace pracující se schránkou Windows; akce se vždy týká objektů, které jsou označeny.

Nabídka **Zobrazení** nabízí položky:

- *Zvětšit* - zvětší, zdetailní pohled na diagram.
- *Zmenšit* - zmenší, zpřehlední ER diagram.
- *Okna* – zde si může uživatel vybrat, která plovoucí okna chce mít zobrazeny.
- *Mřížka* – aktivuje/deaktivuje zobrazení čar, podle kterých se zarovnává umístění objektů; v mřížce je zvýrazněn původní střed diagramu.



Obrázek 4.6 Menu z programu ERTOS s rozbalenými nabídkami

Nabídka **ER diagram** obsahuje funkce pro vytváření a editaci ER diagramu. Podrobnější informace lze získat v kapitole 4.3 Editace ER diagramu.

- *Přidávat entity* - aktivuje mód přidávání entit.
- *Přidávat vztahy* - aktivuje mód přidávání vztahů.
- *Propojovat přímkou* - umožní propojovat entity a vztahy přímou čarou.
- *Propojovat lomeně* - umožní propojovat entity a vztahy lomenou čarou.
- *Zarovnat k mřížce* – jednorázově zarovná všechny objekty ve schématu k mřížce.
- *Změnit typ čar* - změní typ všech čar ve schématu z přímých na lomené a naopak.
- *Zkontrolovat* - provede kontrolu korektnosti schématu.
- *Generovat SQL* - otevře dialog pro převod schématu na skript v jazyce SQL.
- *Vlastnosti* - zobrazí dialog pro editaci informací o diagramu.

V nabídce **Nastavení** najdete:

- *Nastavení* - otevře dialog pro nastavení aplikace.

- *Uložit nastavení* - uloží nastavení do XML souboru, který si uživatel vybere.
- *Načíst nastavení* - načte nastavení z XML souboru.

Při nejasnostech se bude hodit poslední nabídka **Nápověda**. Položka *Obsah* otevře referenční příručku, která obsahuje informace o programu společně s názornými obrázky. Položka *O programu* informuje o verzi programu.

4.7.2 Ovládání

Program vyžaduje ovládání pomocí myši. Práci si však můžete urychlit použitím klávesnice. Mnohé položky v menu mají definovány klávesové zkratky. Odpovídající klávesy jsou v menu uvedeny vedle názvu položky. Kurzorové šipky posouvají s přesností pixelu označené objekty.

Maximální **velikost plochy ER diagramu** není omezena. Avšak posuvné lišty se posouvají vždy jen v určitém rozsahu. Rozsah a tím i dostupnou plochu ER diagramu zvětšíte tažením objektů směrem ven z diagramu přes některý z okrajů. Dostupná plocha diagramu se bude tímto směrem plynule zvětšovat. Dalším nepřímým způsobem zvětšení je zmenšit diagram tak, aby se do okna kromě diagramu vešlo i normálně nedostupné okolí diagramu a do tohoto okolí pak umístit objekt. Dostupná část diagramu se automaticky zvětší, aby obsahovala i tento objekt. Svislá posuvna lišta se ovládá také pohybem kolečka myši, současné zmáčknutí klávesy Shift posouvá obraz vodorovně.

4.7.3 Plovoucí okna

Plovoucí a vysouvací okna lze libovolně zasouvat, vysouvat nebo zrušit pomocí ikon v horní levé části okna. Zavřené okno otevřete pomocí menu *Úpravy* položka *Okna*. Tažením za titulkový pruh je lze přemístit k libovolnému okraji hlavního okna programu. Aplikace nabízí okna *Objekty*, *Historie změn*, *Poznámky* a *Výsledky kontroly*. Na obrázku 4.1 jsou zobrazeny v levé a spodní části okna.

Okno *Objekty* obsahuje seznam všech entit a vztahů řazených abecedně. Entity a vztahy jsou od sebe odlišeny barvou písma. Vybráním položky v seznamu dojde k označení objektu ve schématu a přesunutí pohledu na diagram tak, aby alespoň jeden z vybraných objektů byl vidět. V okně *Objekty* je podporován vícenásobný výběr, který způsobí výběr více objektů v diagramu. Dvojklik na objektu vyvolá dialog pro editaci vlastností objektu. Okno tedy slouží hlavně k rychlému vyhledání určitého objektu. Je užitečné také v případě, že se objekty překrývají a potřebujeme se dostat ke spodnímu z nich.

Okno *Historie změn* umožňuje narozdíl od položek *Zpět* a *Vpřed* v menu *Úpravy* vracet zpět veškeré změny, které jsou v souboru uloženy, tedy včetně těch, které proběhly před posledním otevřením souboru. Neméně důležitou funkcí je animované přehrávání tvorby diagramu.

Pro provedení jednoho kroku v historii změn slouží červená tlačítka (viz obr. 4.1 vpravo dole), černá tlačítka naopak posunou diagram úplně na začátek nebo na konec vývoje. V historii změn se dá přesunout do libovolného okamžiku nastavením pozice jezdce umístěného pod tlačítky.

Pro spuštění **přehrávání vývoje diagramu** zmáčkněte zelené tlačítko. Přehrávání se spustí z aktuálního umístění, takže pro přehrání celého vzniku diagramu je nejprve třeba stisknout černé tlačítko pro návrat na začátek. Rychlost přehrávání je určena pozicí jezdce ve fialovém

panelu a lze ji plynule měnit i během přehrávání. V průběhu přehrávání nelze provádět s diagramem žádné operace, přehrávání však lze přerušit, a to opětovným stiskem původně zeleného tlačítka.

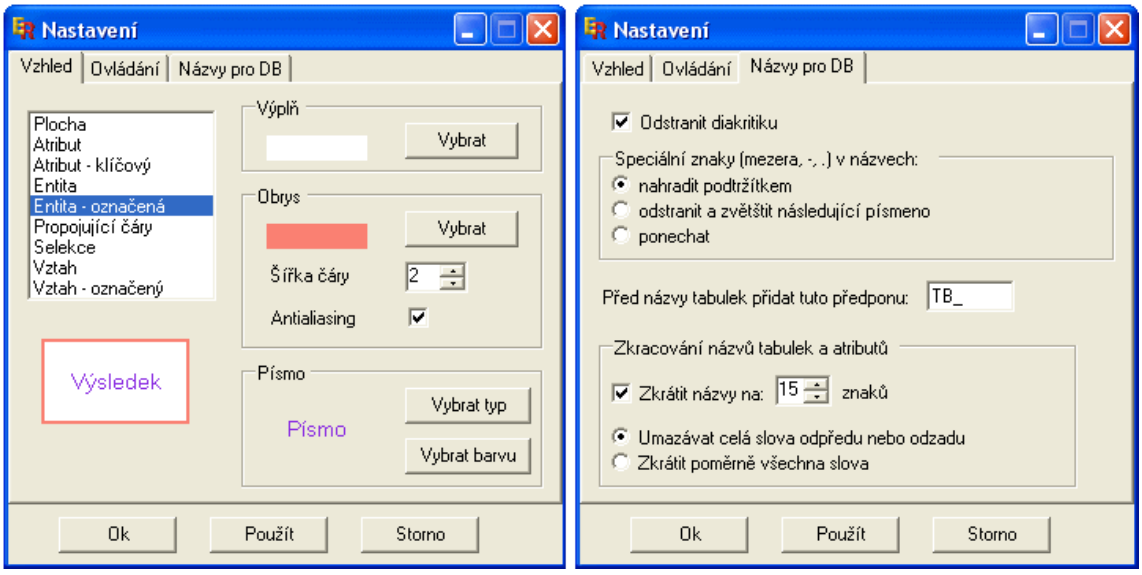
Okno *Poznámky* je tvořeno textovým polem, do kterého si můžete sepsat své poznámky. Ty budou uloženy společně s ER diagramem tak, aby byly při příštím otevření souboru k dispozici.

Okno *Výsledky kontroly* slouží k jedinému účelu, a to k vypisování chyb zjištěných při kontrole korektnosti schématu.

4.7.4 Nastavení

Dialog pro změnu nastavení aplikace lze vyvolat z menu *Nastavení* položka *Nastavení*. Dialog obsahuje několik záložek (některé jsou na obrázku 4.7).

Záložka *Vzhled* umožňuje nastavit barvy, druh a barvu písma a šířku čar pro většinu částí diagramu. Po vybrání objektu ze seznamu v levé části dialogu je možno měnit nastavení jeho grafických vlastností v pravé části dialogu. Vlevo dole se pak objevuje ukázka výsledného vzhledu. Zaškrtnutí položky *Antialiasing* vyhlazuje šikmé čáry a písmo na úkor sotva znatelného zpomalení vykreslování diagramu.



Obrázek 4.7 Vybrané záložky dialogu Nastavení

Jestliže zaškrtnete *Preferovat automatické propojování vztahu s entitami* na záložce *Ovládání*, dojde pokaždé při kliknutí na ikonu vztahu v nástrojové liště k současnému stisku ikony pro vytváření čar, a to konkrétně té, která byla naposledy stisknuta. Volba *Při vytvoření objektu otevřít dialog pro editaci* způsobí, že se ihned po každém vytvoření entity nebo vztahu otevře dialog, ve kterém je možno změnit název objektu a další vlastnosti. Zaškrtnutí možnosti *Zarovnávat středy objektů ke mřížce* zaručí, že od této chvíle každý nový objekt nebo posouvaný stávající objekt bude zarovnán ke mřížce. Zaškrtnutí této možnosti usnadní tvorbu rovných propojujících čar. Pro zarovnání stávajícího diagramu zvolte v menu *ER diagram* položku *Zarovnat ke mřížce*.

Záložka *Názvy pro DB* obsahuje několik voleb týkajících se úprav názvů entit, vztahů a jejich atributů a rolí před jejich zařazením do výsledného SQL skriptu. Jsou žádoucí, aby skript bylo možné přímo použít v databázovém systému.

Volba *Odstranit diakritiku* způsobí, že při převodu všech pojmenování na názvy pro databázi bude odstraněna diakritika. V části *Speciální znaky v názvech* si můžete vybrat, co se bude dít s mezerami, pomlčkami a tečkami v názvech při převodu názvů entit, vztahů, atributů a rolí na názvy pro databázi. Pojmenování jednotlivých možností jsou dostatečně vysvětlující. Před každý název tabulky může být nakonec přidána určitá předpona. Pokud chcete přidat předponu, napište ji do příslušného pole dialogu. Předpona se započítává do maximální délky názvu pro zkracování názvů.

Volba *Zkrátit název* například na 15 znaků způsobí, že délky všech identifikátorů ve skriptu nebudou delší než 15 znaků. Po zkrácení se znovu kontroluje unikátnost názvů a případně se názvy upraví očíslováním. Můžete si vybrat ze dvou způsobů, jak názvy zkracovat:

1) Způsob *Umazávat celá slova odpředu nebo odzadu* zkracuje u tabulek jména tak, aby se zachoval hlavně název objektu, ze kterého tabulka vznikla a u atributů umazává předpony tak, aby byl zachováno původní jméno atributu.

2) Možnost *Zkrátit poměrně všechna slova* ponechá ze všech slov takovou část, jejíž délka je úměrná původní délce slova. Tedy krátká slova zmizí úplně.

5 PROBLÉMY PŘI VYTVÁŘENÍ GUI EDITORU

Grafické uživatelské rozhraní (GUI) je u grafických editorů jednou ze stěžejních součástí. Aby bylo příjemné pro uživatele, je třeba implementovat řadu funkcí, což s sebou přináší nemálo problémů. Část se jich dá ušetřit volbou vhodného programovacího jazyka a vývojového prostředí pro daný operační systém. Program ERTOS byl napsán pro platformu .NET v programovacím jazyce C#. Bylo zvoleno prostředí .NET, protože už základní knihovna tříd .NET frameworku obsahuje třídy pro tvorbu grafiky a uživatelského rozhraní. V kombinaci s prostředím Microsoft Visual Studia 2003 tyto výhody zvítězily nad větší paměťovou náročností výsledného programu.

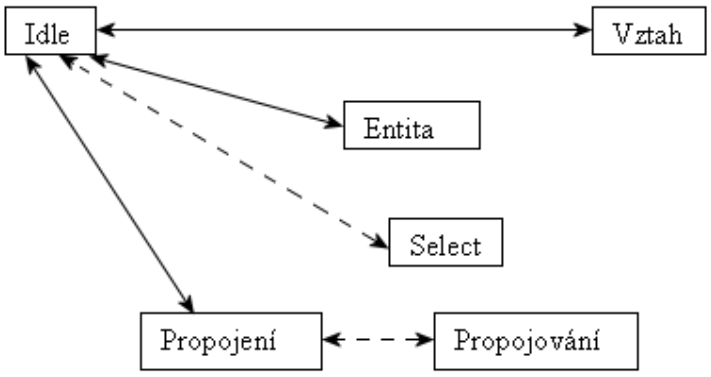
5.1 Uspořádání aplikace

Zadání svým charakterem vyžaduje formulářovou aplikaci, kde většinu formuláře zabírá editovaný diagram. Pro přesuny v rámci diagramu je použita svislá a vodorovná posuvná lišta. Pod vodorovnou lištou je stavový řádek, přidaný převážně z důvodu obvyklého vzhledu, obsahuje jen údaj o aktuální pozici myši ve schématu a měřítku zobrazení. V horní části okna jsou umístěny menu a toolbar. Pro pohodlnější práci je použito několik plovoucích oken. Na obrázku 4.1 je zobrazeno hlavní okno aplikace.

Aplikace nemá podporu více otevřených dokumentů najednou (Multiple Document Interface), protože lze spustit několik instancí programu zároveň a celé diagramy nebo jejich části je možno jednoduše kopírovat, vyjímat a vkládat i mezi těmito instancemi.

5.2 Editace diagramu

Pro editaci diagramu je třeba umět rozpoznat, zda klik myší na pracovní ploše proběhl nad nějakým objektem nebo nad volným místem. Podle toho a také stavu, ve kterém se aplikace nachází, se určí následná akce. Stav aplikace je uložen v proměnné výčtového typu a mění se podle obrázku 5.1. Změna při volbách uživatele v menu či v toolbaru je znázorněna plnou čárou. Změna na přechodný stav, ve kterém je aplikace po zahájení a zároveň před dokončením určité akce, je nakreslena čárkovanou čarou.



Obrázek 5.1 Přechody mezi stavy při editaci diagramu

Stav *Entita* nastane při aktivování módu přidávání entit, podobně stav *Vztah*. Jestliže uživatel v některém z těchto stavů klikne myší na volné místo v diagramu, přidá se do diagramu nová entita nebo vztah. Jestliže na místě kliku myši leží nějaký objekt, zachovají se stavy *Entita*

a *Vztah* stejně jako stav *Idle*. To znamená, že se daný objekt označí a uloží se pozice kliku myši, která bude potřeba v případě, že uživatel začne označenými objekty posouvat. Jestliže přijde zpráva o posunu myši se stále zmáčknutým levým tlačítkem, posunou se označené objekty o rozdíl aktuálních souřadnic myši a uložených souřadnic. Změna posunu se uloží do seznamu uchovávaného pro možnost vrátit změny zpět.

Při označování vztahu je třeba zjistit, zda klik proběhl uvnitř konvexního n-úhelníku. Má-li uživatel zapnuto automatické propojování vztahů s entitami, vyhledají se po přidání vztahu do schématu dvě entity, jejichž vzdálenost od vztahu je nejmenší a s nimi se vztah propojí čarou.

Jestliže ve stavu *Idle* klikne uživatel mimo objekt, aplikace přejde do stavu *Select* a opět se uloží původní souřadnice myši. S posunem myši se pak označuje souvislá oblast. Tady podobně jako u obyčejného označování hraje svou roli, zda je zmáčknuta klávesa Shift. Není-li zmáčknuta, jsou před novým označováním všechny objekty nejprve odznačeny.

Jestliže je aktivní mód propojování entit se vztahy, je program ve stavu *Propojení*. Klikne-li nyní uživatel na objekt, změní se stav na *Propojování* a čeká se, až uživatel klikem na objekt propojení dokončí nebo zruší např. klávesou Escape. Je třeba ověřit, zda je možno připojovaný objekt připojit k původnímu objektu.

Ve stavu *Idle*, *Entita* a *Vztah* způsobí dvojklik na objektu otevření dialogu pro editaci objektu. Dialog obsahuje hlavně tabulku jako instanci třídy *DataGrid* pro úpravu atributů, se kterou se při otevření dialogu propojí data, která jsou uložena u každého objektu. Technicky jsem řešila přidání combo boxu do sloupce tabulky pro výběr datového typu atributu a hlavně, aby jeho chování bylo v souladu s chováním tabulky. Jestliže totiž uživatel začne v *DataGridu* psát údaje do zatím posledního řádku, ihned se objeví následující prázdný řádek, který ale zatím není svázán s žádným řádkem dat v paměti.

Pomocí reakce na událost změny políčka v *DataGridu* se docílilo změny zaškrtnutí u sloupců tvořených zaškrtávacím políčkem na pouhé kliknutí myši a ne na dvojklik, jak to bylo původně. Hlídána je také událost kliku na *DataGrid*, následné zjištění, na kterou buňku uživatel klikl, spolu se zmapováním stavu ostatních sloupců zaručuje, že uživatel nezaškrtne nepovolenou kombinaci vlastností entity.

Souřadnice při stisku tlačítka myši, které přijdou ve zprávě od operačního systému, neodpovídají přímo souřadnicím, na kterých se nacházejí jednotlivé objekty. Místo, na které se nakreslí objekt, je ovlivněno polohou posuvných lišt a také měřítkem zvětšení. Souřadnice kliku myši projdou stejnou transformací, jakou prochází diagram před vykreslením, a teprve výsledek transformace je použit pro hledání, zda klik myši proběhl nad nějakým objektem.

Také se ukládá čas a souřadnice posledního kliku myši, aby se při následném kliku mohlo ověřit, zda tvoří s předchozím klikem dvojklik nebo se jedná o nový klik. U formuláře se sice dají odchytávat události kliku a dvojkliku myši, ty zase ale neposkytují údaje o poloze myši. Maximální časový i vzdálenostní rozdíl mezi kliknutími, aby se stále ještě jednalo o dvojklik, je systémový parametr.

5.3 Vykreslování diagramu

Při každé změně diagramu nebo při nutnosti překreslit okno aplikace je třeba překreslit diagram. Diagram se v prvním případě kreslí pokaždé celý, na rychlosti se to téměř neprojevuje. Aby se předešlo blikání, ke kterému by docházelo i při překreslování pouze

změněných částí okna, je použito **dvojité překreslení**. Při změně digramu se nejdříve celý diagram vykreslí do bitmapy, která je v paměti, a až hotový obrázek se celý nakopíruje do bitmapy v okně. Při překreslování okna aplikace stačí okopírovat z bitmapy v paměti zneplatněnou oblast do okna.

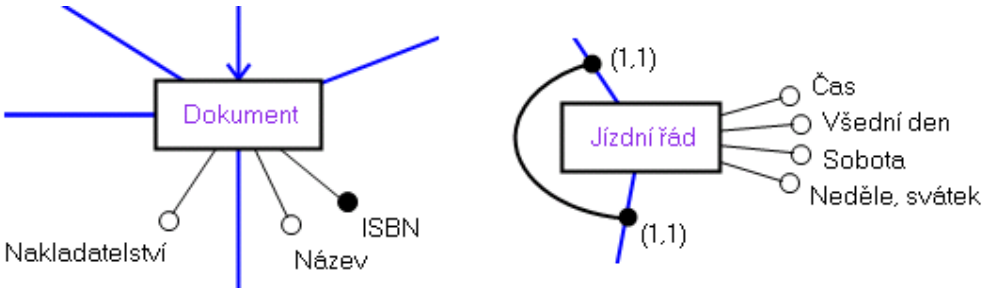
Barevné bitmapy zabírají značnou část paměti programu. Kdyby byla bitmapa stejně velká jako ER diagram, a ten může být velmi rozsáhlý, zbytečně by se spotřebovalo moc paměti. Místo toho stačí mít bitmapy velikosti okna a při kreslení kreslit jen viditelnou část diagramu.

Viditelná část diagramu je určena polohou posuvných lišt a měřítkem zvětšení. V paměťové bitmapě se nastaví příslušná transformace světových souřadnic pomocí metod TranslateTransform a ScaleTransform třídy Graphics [12]. Je důležité dodržet pořadí transformací. Jestliže by se nejdříve provedlo zvětšení a pak posun, musely by být hodnoty, o které se posouvá, vyděleny měřítkem zvětšení, aby se dosáhlo požadovaného efektu. Protože se objekty postupně vykreslují do bitmapy, která má nastavené transformace, mění objekty své rozměry vektorově a ani při větším zvětšení nejsou rozmazané.

V diagramu se kreslí entity a vztahy, které mají přesně stanoveny svou polohu, ale také atributy nebo popisy kardinalit a rolí, u kterých je třeba polohu určit. Názvy by měly být rozmístěny čitelně a také pravidelně v blízkosti objektu, který popisují. Nejsložitější bylo **rozmístit atributy** okolo entity, protože v případě, že je entita identifikována více atributy nebo i jiným vztahem, musí se propojit všechny identifikátory obloukem elipsy. Atributy je dobré umístit blízko sebe tak, aby oblouk nebyl zbytečně dlouhý a také nepřetínal mnoho neidentifikačních atributů.

Nejdříve se zjistí, pod jakými úhly vycházejí z entity propojující čáry. Zároveň se zaznamená, které úhly patří čarám vedoucím k entitám, na kterých je původní entita identifikačně závislá. Atributy jsou kresleny tak, že čáry vedoucí k zakončovacím kolečkům vycházejí ze středu obrázku objektu pod různými úhly. Délky těchto čar jsou voleny tak, aby zakončovací kolečka ležela na elipse, jejíž poměr délek poloos odpovídá poměru šířky a výšky obrázku objektu.

Čtyři strany obrázku objektu se seřadí podle počtu čar, které z nich vycházejí. Přitom se bere v úvahu, zda se blízko dané strany objektu nenachází jiný objekt. V tom případě se nebudou atributy na tuto stranu kreslit, pokud bude jinde dost volného místa. Pokud je entita identifikována právě jedním atributem, může být spolu s ostatními atributy rovnoměrně rozmístěn na volné strany entity. Úplně stejně se rozmístí atributy vztahu.



Obrázek 5.2 Příklad nakreslení atributů v některých situacích

Strany mají omezen počet atributů, které na ně lze nakreslit. Omezení je odlišné pro vodorovné a svislé strany z důvodu čitelnosti názvů atributů. Může se také stát, že už nebude

zbývat žádná volná sousední nebo dokonce vůbec žádná volná strana. V tom případě se vybere nejméně obsazená strana. S ohledem na čáry, které ze strany vycházejí, jsou atributy nakresleny do prostoru mezi tyto čáry (viz obr. 5.2 vlevo).

V případě více identifikačních atributů entity se namalují všechny tyto atributy na jednu stranu. Pokud se tam nevejdou, rozmístí se na sousední strany, aby mohly být jednoduše propojeny obloukem. Při kreslení oblouku je znám seznam úhlů čar, které mají být obloukem propojeny. Najde se největší mezera mezi čarami a oblouk se nakreslí mezi čarami, které tuto mezeru ohraničují. Volí se menší oblouk, nekreslí se tedy přes tuto největší mezeru (obr. 5.2 vpravo).

Při vykreslování se také podle délky názvu objektu upravují rozměry obrázku objektu. Dlouhý název objektu je rozdělen maximálně do tří řádků podle mezer, poté se upraví velikost obrázku a objekt posune tak, aby se nový a starý objekt shodovaly v souřadnicích středu. Při přejmenování pak objekt nepřirozeně „neuhne“ do strany.

5.4 Undo/Redo

Možnost vrátit zpět provedenou operaci (undo) a poté ji případně znovu vykonat (redo) patří k základním vlastnostem editoru. Editor ERTOS nemá omezenou historii ukládaných operací, tedy všechny změny by měly jít v mezích dostupné paměti vrátit zpět.

Princip provádění undo/redo operací je následující: Existují dva zásobníky *Undo* a *Redo*, které obsahují instance třídy *Zmena*. Jestliže **dojde k nějaké změně** v ER diagramu, která podléhá undo/redo operaci (až na změny v metadatech diagramu to jsou všechny), vytvoří se nová instance třídy *Zmena*. Ta obsahuje referenci na instanci, která byla změněna, a kopii instance s původními hodnotami. Protože změnám podléhají různé objekty, mají reference i hodnota datový typ *object*, od kterého jsou odvozeny všechny datové typy v C#. Instance třídy *Zmena* se pak vloží do zásobníku *Undo*. Některé třídy, například *Entita* a *Vztah*, mají metodu *Zmena*, kterou stačí před potřebnou operací zavolat.

Když uživatel zažádá o vrácení změny (**operace undo**), odebere se změna z vrcholu zásobníku *Undo* (viz. obr.5.3). Pak se vytvoří nová instance *Zmeny*, která bude přidána do zásobníku *Redo*. Ukazatel v nové instanci je stejný jako v původní instanci, položka nesoucí hodnotu se vytvoří kopií objektu, na který ukazuje ukazatel. Podle datového typu, na který ukazuje reference (v .NET se dá datový typ určit za běhu pomocí funkce *GetType*(), funkce je potřeba už pro vytvoření kopie objektu pro zásobník *Redo*) se dá rozlišit, o jakou změnu se jednalo a změnu vrátit zpět.

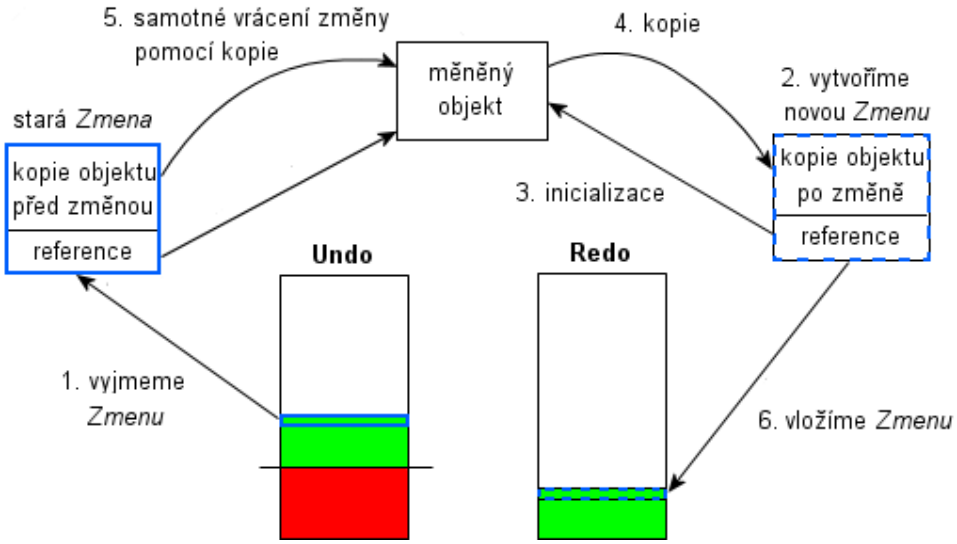
Ne vždy stačí jen do cíle, na který ukazuje reference, nakopírovat původní hodnotu. Třeba v případě přidávání objektu do schématu se ukládá do *Zmeny* pouze daný objekt, ne celý seznam *Objekty*, aby se zbytečně nezabralo moc paměti. Počet změn v zásobníku *Undo* totiž rychle přibývá a *Objekty* obsahují velkou část diagramu. Při vracení akce se pak zjišťuje, jestli v *Objektech* je objekt, na který ukazuje daná reference. Jestli ne, vytvoří se nový objekt s danými hodnotami. Jestliže tam je, ještě se zjistí, jestli se hodnoty objektu shodují s hodnotami objektu ve *Zmene*. Neshodují-li se, byla objevena podstata změny (například změna kardinality u vztahu). Jinak se celý objekt z *Objektů* vymaže.

Tímto ověřováním se řeší případy více možných operací s jedním datovým typem, u entity je to kupříkladu změna pozice nebo změna seznamu entit, na kterých je závislá. Některé časté operace, například změna souřadnic objektu jsou řešeny vytvořením třídy pro obalení

souřadnic objektu, aby se na danou třídu dala získat reference. Typ *int* je totiž v C# hodnotový a referenci na něj nelze získat [13].

Třída *Zmena* obsahuje také boolovskou hodnotu říkající, zda se má rovnou vrátit zpět i další změna ze zásobníku. To je potřeba v případě množiny operací, které jsou provedeny najednou, a je tedy třeba vrátit je v jednom kroku.

V případě **operace redo** probíhá vše podobně. Vyjme se změna ze zásobníku *Redo*, stejným způsobem se vytvoří nová instance *Zmeny* a přidá se do zásobníku *Undo*. Samotný průběh změny je naprosto stejný, skutečnost, ze kterého zásobníku jsme *Zmenu* odebrali, nemá na provedení operace změny vliv.



Obrázek 5.3 Provedení operace undo

Funkce **procházení historií změn** i před okamžik otevření souboru si vynutila přidání zarážky oddělující od sebe změny provedené před otevřením souboru (staré změny) a po otevření nebo vytvoření souboru (nové změny). Tato zarážka se po otevření nebo vytvoření souboru umístí na vrchol zásobníku *Undo*. Všechny nové změny budou umístěny nad zarážku. Na obrázku 5.3 je zarážka v zásobníku *Undo* zobrazena černou vodorovnou čárou, staré změny jsou zobrazeny červeně, nové změny zeleně.

Zarážka se spolu se změnami přemísťuje z jednoho zásobníku do druhého a na základě její polohy se stanovuje, zda budou aktivní tlačítka *Vpřed* a *Zpět* sloužící pro pohyb pouze v rámci nových změn. Jestliže se diagram nachází ve stavu mezi starými změnami a provede-li uživatel novou změnu, přesune se zarážka zevnitř zásobníku *Redo* na vrchol zásobníku *Undo* a teprve nad ní se umístí nově vzniklá změna. Došlo tedy k tomu, že se část starých změn přeměnila ve změny nové.

Přehrávání změn není nic jiného než opakované vykonávání redo operací v určitých časových intervalech. Přesun na začátek historie změn spočívá v provádění undo změn tak dlouho, dokud se zásobník *Undo* nevyprázdní, případně dokud nebude obsahovat pouze zarážku.

5.5 Kopírování, vkládání a vyjímání

Při kopírování, vkládání a vyjímání vybrané části schématu je využita schránka Windows. Kopírování a vkládání vyžaduje v první fázi vytvoření instance třídy, ve které je uložen celý diagram. Do této instance jsou poté nakopírovány všechny označené objekty a příslušná propojení. Před okopírováním se ještě najde nejmenší ohraničení označených objektů. Jednotlivým objektům jsou pak změněny souřadnice na relativní vzhledem k levému hornímu rohu ohraničení zvětšené o určité odsazení od okraje. Instance třídy se pak načte do schránky Windows pomocí třídy .NET *System.Windows.Forms.Clipboard*.

Při vkládání je naopak instance třídy ze schránky načtena a proběhne vložení objektů z této instance na předem vypočtené souřadnice. Událost vkládání je vyvolána z menu nebo klávesovou zkratkou a není tedy spojena s žádnými určitými souřadnicemi. V případě konfliktu jmen je nutno vkládané objekty přejmenovat, aby byl jejich název jedinečný. U všech těchto akcí se samozřejmě musí aktualizovat seznam objektů v plovoucím okně *Objekty* a také ukládat všechny změny do zásobníku *Undo*.

5.6 Ukládání, otevírání souborů

ER diagram lze uložit do formátu XML nebo jako obrázek ve třech různých formátech. Tisk diagramů se neřešil, protože obrázky lze následně vytisknout v libovolném prohlížeči obrázků.

Soubor uložený ve formátu XML lze v programu neomezeně otevírat a znovu ukládat. Soubor vzniká pomocí serializace (uložení instance do perzistentního úložiště zajišťované prostředím .NET [13]) třídy, ve které je uložen celý ER diagram. Je-li požadováno uložení souboru spolu s historií změn, je serializován také zásobník *Undo*. Serializace ukládá třídu ve formátu XML, jehož struktura je uzpůsobena pro komunikaci přes protokol SOAP. Soubor se dá editovat případně i ručně. Jestliže je soubor poškozen, oznámí se při otevírání pomocí deserializace chyba a soubor se neotevře.

Vektorový grafický soubor ve formátu EMF se vytváří pomocí stejné funkce, která se používá pro vykreslení diagramu na obrazovku. Nejdříve se vyhledá jen taková část diagramu, která obsahuje nějaké objekty, aby obrázek zbytečně neobsahoval velká prázdná místa. Poté se do instance vektorového souboru vykreslí to, co by se normálně při kreslení diagramu vykreslovalo do bitmapy v paměti. Ve formátu EMF se vykreslované objekty uchovají jako posloupnost příkazů, které je pak možno po uložení kdykoliv přehrát a soubor tak zobrazit v příslušném měřítku.

Rastrové formáty BMP a GIF vznikají podobně jako vektorový formát, jen je při jejich vzniku bráno v úvahu aktuální měřítko zobrazení. Tomu se musí přizpůsobit rozměry ukládaného obrázku.

5.7 Posuvné lišty

Posuvné lišty slouží k pohybu po schématu. Schéma má nastavenou svou aktuální velikost, kterou ale může uživatel kdykoliv změnit tažením objektu mimo diagram. Změnu rozměrů vyvolá také změna měřítka lupy nebo změna velikosti vysouvacích panelů. Je proto třeba, aby se na začátku a po každé takovéto změně znovu nastavily parametry posuvných lišt, aby poměr velikosti jezdce a délky celé posuvné lišty odpovídal poměru délky viditelné části

diagramu ku délce celého diagramu. Také musí být splněno, že posun prostřednictvím lišty nebude znamenat výjezd mimo diagram nebo naopak, že se do některých částí diagramu nedostane uživatel posunem vůbec.

Naopak při změně polohy jezdce na posuvné liště se musí překreslit diagram, aby se posunutí projevilo. Na začátku je jezdec nastaven přesně uprostřed na hodnotě 0. Minimum je nastaveno na zápornou hodnotu odpovídající velikosti levé, případně horní poloviny diagramu v pixelech. Maxima obdobně nabývají kladných hodnot. Pak lze rovnou posunovat celý obraz o opačnou hodnotu příslušné posuvné lišty. Odlišný je pouze případ, kdy bylo zmenšeno měřítko natolik, že se aspoň v jednom rozměru vejde celý diagram do okna, takže posuvná lišta není potřeba. V tomto případě se obrázek posune jen o polovinu rozměru okna, aby se zmenšoval právě uprostřed okna.

Jestliže si uživatel nechá v diagramu vyhledat objekt pomocí výběru objektu ze seznamu v plovoucím okně *Objekty*, dojde k přemístění v diagramu, aby se vybraný objekt nacházel v rámci rozměrů schématu co nejblíže středu. Spolu s přemístěním v diagramu se musí posunout i hodnoty posuvných lišt.

5.8 Plovoucí a vysouvací okna

Plovoucí a vysouvací okna, přesněji toolbary s vlastností dockování (přichycování k okrajům formulářových objektů [12]), umožňují uživateli zpřehlednit pracovní plochu a uspořádat si ji podle vlastního přání. To se hodí zejména v případě robustních aplikací. Ale i v případě programu ERTOS se vyplatilo využít knihovny pro tvorbu grafického uživatelského rozhraní MagicLibrary.dll. Použita byla verze 1.7.4, poslední verze, která je poskytována zadarmo a to i pro komerční použití [14]. Kromě těchto toolbarů knihovna poskytuje také mnoho jiných komponent pro GUI, ale z důvodu nedostatečné dokumentace nebyly použity části, které jsou obsaženy v knihovnách .NET.

Hlavní problém nastal, když bylo potřeba zjistit, v jaké poloze přesně se nacházejí jednotlivá okna, jestli jsou zasunuta a podobně. Tato informace ovlivňuje posunutí diagramu při vykreslování do okna. Žádné funkce z knihovny to neumožňují, takže jediným řešením bylo vyplnit celý zbytek okna, který nezabírají žádná plovoucí okna, panelem přichyceným pomocí dockování ke všem okrajům. Velikost panelu se pak sama roztahuje a přizpůsobuje okolí. Teprve na tento panel se vykresluje diagram.

Knihovna umožňuje také uložit a následně načíst uspořádání pracovní plochy do XML souboru, což se využívá při ukončování programu. Uspořádání plochy spolu s ostatním nastavením se na konci programu ukládá odděleně pro jednotlivé uživatele Windows.

5.9 Nastavení

Nastavení editoru zahrnuje hlavně změnu barev a typu čar, výplně a také písma u objektů diagramu. Po spuštění editoru se program pokouší načíst nastavení ze souboru uloženém v prostoru pro data aplikací. Je-li soubor smazán nebo poškozen, nastaví se výchozí hodnoty. Do hašovací tabulky se uloží barvy pro vybarvení pozadí, pera a písma. Při kreslení se nejdříve vyhledá příslušná informace nebo objekt v tabulce a poté se využije k vykreslení. Uživatel může nastavení měnit.

Je-li zvoleno, že se mají středy objektů zarovnávat k mřížce, musí se vždy při vytvoření nové entity nebo vztahu zaokrouhlit jejich poloha. Poloha se zaokrouhluje také při posouvání objektů. Protože se však objekty posouvají plynuje, to znamená, že během jednoho posunu se poloha zaokrouhluje několikrát, je potřeba si vždy pamatovat informaci o tom, o kolik pixelů byla poloha zaokrouhlena. Při příštím posunu se nejdříve toto číslo přičte k novým souřadnicím a až poté se souřadnice zaokrouhlí. Jinak dochází ke kumulaci zaokrouhlovacích chyb a objekty "ujíždí" jinam, než se pohybuje myš.

Nastavení se uloží spolu s polohou plovoucích oken při ukončování programu. Uživatel si může také sám kdykoliv ukládat a načítat dříve uložené nastavení do jím zvoleného souboru. V tomto případě se neukládá poloha plovoucích oken, protože ta je zajišťována cizí knihovnou a nepodařilo se spojit oba výstupy do jednoho souboru. K ukládání nastavení je opět použita serializace.

5.10 Testování

Program byl testován jak samostatně prostřednictvím editací mnoha diagramů, tak pomocí srovnávání výstupu s programem ER modelář.

I přes dosavadní testování se však mohou vyskytnout chyby. Aby byl dopad na uživatele co nejmenší, zachytává neošetřené výjimky speciální metoda, která se uživateli omluví a umožní mu vybrat si soubor, do kterého bude uložen právě editovaný soubor. Může se stát, že garbage collector stihne uklidit některé potřebné objekty dříve, než budou do souboru uloženy, ale během testování se to nestalo ani jednou. Na záchranu je vysoká šance, protože se nejdříve vše potřebné uloží do paměťového proudu a teprve poté jsou otevírány dialogy se zprávou a pro výběr souboru.

6 PROBLÉMY SPOJENÉ S VNITŘNÍM MODELOVÁNÍM ER

Při ukládání dat ani při provádění algoritmů nebyl kladen důraz na rychlost běhu. Aplikace má sloužit k výukovým účelům a nepředpokládá se tedy její velké zatížení. S běžnými daty rozsahu zápočtové práce pracuje program s okamžitou odezvou.

6.1 Uložení diagramu

Logickým základem programu je třída *ER*, která v sobě ukrývá všechny informace o editovaném schématu. Nejdůležitějšími členy třídy jsou seznam *objekty* obsahující entity a vztahy a seznam *propojeni* obsahující informace o tom, které vztahy jsou propojeny s kterými entitami. Co se týká metod, obsahuje třída vše, co je potřeba pro práci se schématem – metody pro vytváření nových objektů ve schématu, jejich propojování, provádění undo/redo změn, metody pro označování a přesuny objektů, pro hledání objektů ležících na dané pozici, pro provedení operací cut, copy, paste a další.

V seznamu *objekty* jsou uloženy instance tříd *Entita* a *Vztah*, které jsou obě potomky třídy *Objekt*. Právě proto jsou uchovávány entity i vztahy ve společném seznamu, aby se využilo jejich společných vlastností a zároveň se mohly volat virtuální funkce. Virtuální funkce jsou využity například pro vykreslování, označování a hledání objektů, které se liší z důvodu rozdílných tvarů objektů. Třída *Objekt* obsahuje informace jak o logickém názvu objektu vystupujícím ve schématu, tak o fyzickém názvu, který bude použit pro vytváření názvů tabulek a atributů pro databázi. Nechybí v ní také údaje o souřadnicích a rozměrech. Další důležitou součástí je paměťová tabulka atributů. V případě vztahu z ní nejsou využity sloupce související s identifikací entity a pro několikanásobné atributy.

Třída *Entita* přidává k členům třídy *Objekt* seznam vztahů, na kterých je instance *Entity* identifikačně závislá. Pomocí nalezení propojení s těmito vztahy v seznamu *propojeni* lze získat entity, na kterých je původní entita identifikačně závislá. Třídy *Entita* a *Vztah* obsahují metody pro přizpůsobení velikosti obrázku objektu délce názvu objektu a pro kreslení atributů včetně oblouků elips.

Při přidávání objektu do schématu se nová entita nebo vztah přidá na konec seznamu *objekty*, pořadí objektů však není důležité. Nově přidávaný objekt dostane automatický název, u kterého je nejprve třeba ověřit jednoznačnost. Je-li zapnuto automatické propojování, vzniknou v případě přidávání vztahu nová spojení přidávaného vztahu s dvěma nejbližšími entitami. Do seznamu *propojeni* se v tomto případě uloží dvě nové dvojice, které kromě informace o tom, které objekty spojují, obsahují také informaci o kardinalitách a názvech rolí.

ISA vztah je uložen jako dvě propojení v seznamu *propojeni* stejně jako ostatní vztahy. Odlišuje se však kardinalitami, které jsou navíc rozlišeny pro propojení s entitou předka a entitou následníka v ISA vztahu. Z těchto kardinalit lze zjistit směr ISA hierarchie.

6.2 Kontroly schématu

V prvé řadě je třeba zkontrolovat korektnost schématu. ER diagram je korektní, splňuje-li definici z kapitoly 2.2. **Jednoznačnost názvů** entit a vztahů se kontroluje už při přidávání objektů do schématu. Většinou se musí kontrolovat jak logické názvy objektů, tak názvy, které vzniknou po transformaci pro databázi. V každé množině názvů musí být zajištěna

unikátnost a to bez rozlišení velkých a malých písmen, protože SQL malá a velká písmena nerozlišuje. Následně se musí zkontrolovat při každé změně názvu uživatelem a také při převodu do relačního schématu, zda unikátnost nebyla porušena. Při převodu se rovněž pomocí přejmenovávání předřazováním názvů tabulek, ze kterých atribut pochází, docílí jednoznačnosti názvů zděděných atributů.

Před samotným převodem nebo na požádání se kontrolují ostatní body korektnosti. Při hledání, zda nějaká entita nemá více zdrojů **ISA hierarchie**, se stačí u každé entity podívat, zda nemá dva bezprostřední předchůdce v ISA hierarchii. Zároveň se při tomto procházení entit ověřuje, že entity, které nejsou zdrojem ISA hierarchie, nemají identifikační klíč. Nemůžou být také závislými entitami v identifikačním vztahu. Chybu taktéž způsobí, když ostatní entity nejsou ničím identifikovány.

Při prohledávání, zda **identifikační vztahy** nebo ISA vztahy netvoří cyklus, se používá hledání orientovaného cyklu v grafu. Vrcholy grafu tvoří entity a orientovaná hrana vede z vrcholu, který identifikuje, do vrcholu, který je identifikačně závislý (v případě ISA hierarchie z předka do potomka). Existuje-li v takovémto grafu cyklus, není schéma jednoznačné a kontrola skončí chybou.

Graf se ve skutečnosti z důvodu nepřehlednosti nevytváří, využívá se stávající struktura informací uložených v seznamu propojujících čar. Pro entitu je přímo dostupná informace, která entita ji identifikuje, takže hrany v grafu jsou ve skutečnosti zorientovány opačně. To ale nevadí, protože v orientovaném grafu existuje cyklus právě když existuje v opačně zorientovaném grafu. Cykly se hledají jednoduchým algoritmem prohledáváním do hloubky s obarvováním jednotlivých vrcholů.

Různé role pro více stejných entit ve vztahu nebo identifikačním vztahu jsou zajištěny při převodu do relačního modelu. Před názvy všech atributů z entit jsou předřazeny předpony tvořené buď názvem role v případě self relace nebo názvem vztahu u identifikačních vztahů. K tomu je potřeba zajistit kontrolu, že uživatel v případě self relace opravdu zadal neprázdné a různé názvy rolí.

Při vytváření schématu jsou kontrolovány také **další operace** prováděné uživatelem, aby se zamezilo chybám při generování skriptu SQL. Před potvrzením změn u editace atributů objektu se ověřuje, že všechny zadané atributy mají neprázdná jednoznačná jména a určen datový typ. U entit nesmí nastat, že několikanásobný atribut je zároveň primárním klíčem. Rovněž atribut, který je součástí primárního klíče, musí být *Not Null* a žádný atribut nesmí mít zaškrtnuto zároveň *Unique* a *Not Null*. Poslední dvě kontrolované skutečnosti předchází chybám ohlášeným ze strany databázového serveru. Jejich kontrola je zajišťována už při editaci atributů uživatelem např. znemožněním odškrtnutí sloupce *Not Null*, je-li ve stejném řádku zaškrtnuto *Primary Key*.

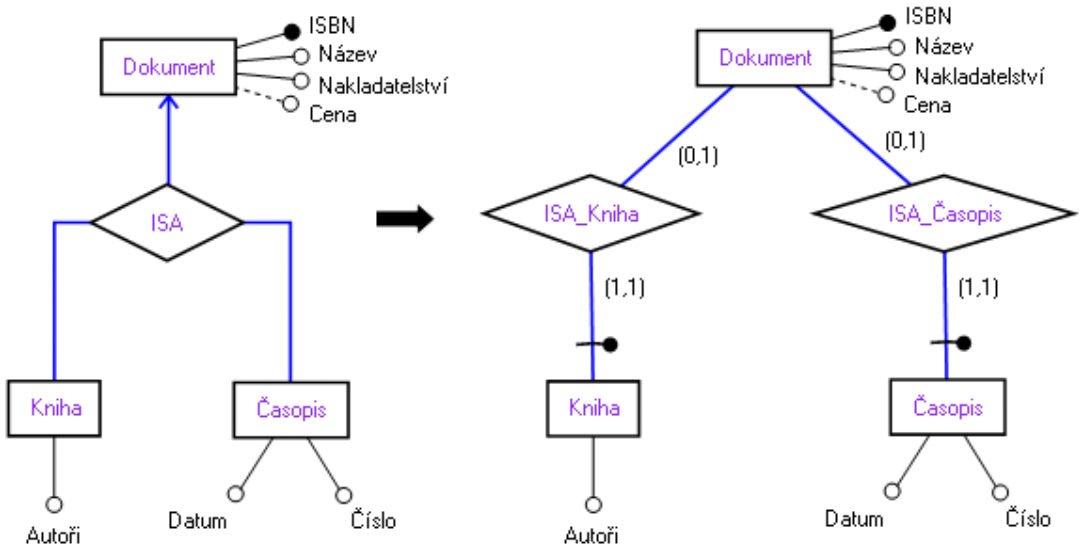
Identifikační vztah je možno vytvořit pouze v případě, že na straně závislé entity je zadána kardinalita (1,1). Aby mohla být tato kardinalita změněna v případě existence identifikačního vztahu, je nejprve nutné zrušit identifikační vztah.

6.3 Převod ISA hierarchie

Převést ER diagram obsahující ISA hierarchie na diagram bez těchto hierarchií lze několika způsoby. Z ISA vztahu je možné ponechat pouze předka s tím, že entita bude obsahovat atribut rozlišující, kterého z potomků entita reprezentuje. Bude také obsahovat všechny

atributy potomků a vystupovat v jejich vztazích, ve všech případech je ale nutné převést je z povinných na nepovinné. Z ISA vztahu lze ponechat také jen potomky. Potomci získají všechny atributy a účasti ve vztazích předka. Ve schématu se tak některé prvky znásobí.

Pro program ERTOS byl zvolen způsob třetí, který zachovává všechny entity z hierarchie. Převod spočívá v nahrazení ISA vztahu identifikačním vztahem (viz obr. 6.1). Před převodem do relačního modelu se tedy vytvoří nové vztahy s kardinalitami (0,1):(1,1), které se v paměti přidají do schématu. Po převodu se zase vymažou, aby s vícenásobným převodem do relačního modelu vztahy neustále nepřibývaly.



Obrázek 6.1 Příklad převodu ISA hierarchie

Možným rozšířením programu ERTOS je zobrazení diagramu po převodu ISA hierarchie. Vztahy se do schématu opravdu přidávají, ale jejich nakreslení vyžaduje vyřešení problému rozložení objektů tak, aby se nepřekrývaly a schéma zůstalo přehledné.

6.4 Převod do relačního modelu a výpis SQL skriptu

Převod do relačního modelu se opírá o algoritmus uvedený v kapitole 2.3. Nejdříve se vytvoří pro každou entitu jedna tabulka. Do tabulky jsou přímo nakopírovány existující řádky s informacemi o atributech uloženými uvnitř entity. Také dochází k úpravě názvů atributů, aby mohly být použity v SQL. Kopírují se všechny atributy kromě několikanásobných, ty budou vyřešeny později. Kromě toho se přidá do každé tabulky speciální nultý řádek obsahující název tabulky a také informaci, zda tabulka stále existuje. Některé tabulky mohou být při převodu zrušeny, ale nevymazávají se úplně z důvodu možnosti jejich hledání.

Kromě svých atributů obsahují identifikátory některých entit také **identifikační atributy** jiných entit. Ty se musí do tabulek přidat ještě před řešením problematiky vztahů. V tuto chvíli už také musí být převedena ISA hierarchie na identifikační vztahy. Identifikační atributy z jiných entit se propagují směrem dolů od identifikačních zdrojů k závislým entitám níže popsaným způsobem.

Na začátku se všechny entity odznačí. Pro každou neoznačenou entitu se zjistí, zda a na kterých entitách je identifikačně závislá. Není-li nějaký její identifikační vlastník závislý na

žádné entitě, mohou se jeho identifikační atributy přímo zkopírovat do závislé entity. Je-li vlastník závislý, provede se rekurzivně stejná akce pro entitu tohoto identifikačního vlastníka. Při návratu z rekurze se pak zpětně směrem dolů kopírují identifikační atributy. Každá navštívená entita se označí, aby se atributy neokopírovaly vícekrát. Atributy se nepropagují přes vztahy s kardinalitami (1,1):(1,1), protože se z tohoto vztahu a zúčastněných entit stane jedna tabulka a identifikační atributy by se vyskytovaly dvakrát.

Nyní se vyřeší **několikanásobné atributy** entit. Pro každý několikanásobný atribut se musí vytvořit nová tabulka, která bude obsahovat tento atribut a identifikační atributy entity (včetně zděděných od identifikačních vlastníků), které několikanásobný atribut patří. Všechny atributy nové tabulky budou jejím primárním klíčem. Vytvoří se také nový cizí klíč vedoucí z nové tabulky do původní přes identifikační atributy původní tabulky. Pro ukládání primárních a cizích klíčů slouží speciální třída, jejíž instance je vytvořena pro každou tabulku.

V další části je třeba vyřešit všechny vztahy jeden po druhém. Přitom se dbá na to, že výsledný SQL skript by měl mimo jiné splňovat požadavek, že příkazy pro vytvoření a smazání tabulek musí být v takovém pořadí, aby byly proveditelné. Jediné, co může v tomto způsobit chybu, jsou cizí klíče. Ty musí vést vždy do již vytvořené tabulky. Smazat tabulku lze zase jen tehdy, neexistují-li na ni žádné reference. Takže pořadí tabulek ve skriptu pro smazání tabulek je přesně opačné než ve skriptu pro jejich vytvoření. Stačí tedy správně stanovit pořadí, ve kterém se budou tabulky vytvářet. Samozřejmě by bylo možné až na konci před výpisem skriptu vytvořit graf referencí a topologicky ho setřídit. **Správné pořadí tabulek** však lze udržovat už během převodu, protože cizí klíče vznikají jen ve speciálních případech.

Tabulky jsou ve výsledném SQL skriptu vypisovány v pořadí, ve kterém se vyskytují v seznamu v programu. Nově vytvořená tabulka je pokaždé přidána na konec seznamu. Na začátku, když existují jen tabulky pro každou entitu, neexistují ještě žádné klíče. Následně musí být dodrženo pravidlo, že vytvářený cizí klíč může vést jen z tabulky, která vznikla později, do tabulky starší.

Z výše uvedených důvodů se musí nejdříve vyřešit všechny **vztahy s kardinalitami (1,1):(1,1)**. V tomto jediném případě dochází při převodu ke spojení tabulek a nastává problém s cizími klíči spojovaných tabulek. Ať už bychom umístili novou tabulku kamkoliv do seznamu, mohou vzniknout problémy s pořadím tabulek při vypisování skriptu. Naopak algoritmus nebrání tomu, že se nejdříve vyřeší jen určitý typ vztahů.

Při řešení vztahu s kardinalitami (1,1):(1,1) jsou obě původní tabulky označeny za neexistující a spojením původních tabulek se vytvoří tabulka nová. Zároveň se kontroluje, zda nebyly náhodou entity, které jsou propojeny tímto vztahem, stejné. To by znamenalo self relaci, u které musíme dbát na jiné pojmenování výsledné tabulky a hlavně na předřazení rolí před názvy atributů. V případě, že by tabulky, které máme spojovat, byly stejné, byl nalezen cyklus vztahů s kardinalitami (1,1):(1,1) a tento poslední vztah už není třeba nijak řešit. Ze všech klíčů původních tabulek je vybrán vždy jen jeden, který se stane klíčem primárním. Ostatní klíče (včetně kandidátních) se stanou novými kandidátními klíči a ve výsledném SQL skriptu jsou vypsány jako integritní omezení *Unique*.

Poté se vyřeší všechny **ostatní vztahy** podle pravidel popsaných v algoritmu. Rozlišují se jednotlivé případy podle arity vztahu a podle kardinalit. Všem atributům, které se do některé tabulky dostanou z jiné tabulky, je přitom předřazena předpona podle názvu původní tabulky nebo názvu vztahu, aby se žádné dva atributy v tabulce nejmenovaly stejně. U self relace se

používají názvy rolí. Také je vždy potřeba vytvořit cizí klíč ukazující z tabulky, ve které přibyl atribut, do tabulky původní. Cizí klíč musí vzniknout i při převodu self relace nebo ISA hierarchie.

Při **generování textu skriptu** se pak už jen v cyklu projdou všechny existující tabulky a jejich atributy a spolu s klíčovými slovy se ve správném pořadí vypíší. Vypisují se také primární a cizí klíče, které jsou jako druh integritního omezení pojmenovány pomocí klíčového slova *Constraint*. U těchto názvů se už neřeší jejich unikátnost. V případě ojedinělých konfliktů si může uživatel názvy klíčů přejmenovat sám nebo vypisování pojmenování klíčů vypnout. U cizích klíčů se sdružují dohromady všechny reference ukazující do stejné tabulky a zapíší se jako jeden cizí klíč.

Ještě před převodem si uživatel může vybrat, které objekty se mají převádět a které ne. Výběr uživatele je potřeba rozšířit tak, aby ve výběru byli všichni účastníci všech vybraných vztahů a aby každá entita měla ve výběru svůj zdroj ISA hierarchie a identifikační zdroj. Celý tento výběr pak prochází kontrolou a převodem.

6.5 Úpravy názvů pro databázi

Uživatel může během vytváření diagramu pojmenovávat objekty libovolně, tedy včetně speciálních znaků a diakritiky a jména mohou mít libovolnou délku. Je proto vhodné názvy následně upravit tak, aby SQL skript byl přijímán databázovými servery bez chyb. Veškeré úpravy může uživatel vypnout.

Názvy objektů se po každé změně uživatelem uloží a také se upraví na tvar pro databázi. Tento tvar může uživatel ještě před uložením pozměnit. Tyto názvy se pak použijí jako výchozí pro vytváření názvů tabulek. Po ukončení celého převodu, má-li uživatel zapnuto zkracování názvů identifikátorů, jsou znovu zkracovány celé názvy tabulek a atributů, protože mohly vzniknout spojením několika původních názvů. Poté se ještě zkontroluje unikátnost názvů a v případě konfliktu se nakonec doplní rozlišující číslice.

Při první úpravě názvů jsou všechny speciální znaky nahrazeny mezerami, protože později bude třeba rozlišit speciální znaky zapsané uživatelem a přidané programem. Program přidává znaky například při vytváření spojených názvů tabulek a atributů, kdy je jako spojovací znak použito podtržítko. Mají-li se pak jména zkracovat odsekáváním slov odpředu a odzadu, odsekávají se úseky vzniklé rozdělením řetězců právě podle znaku podtržítka. Teprve na závěr se všechny mezery definitivně nahradí podtržítky, případně se podle přání uživatele ponechají.

7 POUŽITELNOST V RŮZNÝCH DATABÁZÍCH

Použitelnost jsem zkoušela na následujícím SQL skriptu, který obsahuje všechny syntaktické konstrukce, které se mohou vyskytnout ve výstupu programu ERTOS.

```
-- Odstraneni tabulek --

DROP TABLE Kategorie_obsahuje_Dokument;
DROP TABLE Kategorie;
DROP TABLE Dokument;
DROP TABLE Zamestnanec_Automobil;

-- Vytvoreni tabulek --

CREATE TABLE Zamestnanec_Automobil (
Rodne_cislo     Varchar(10)     NOT NULL,
SPZ     Varchar(20)     NOT NULL,
Najeto      Float     DEFAULT '0.0' NOT NULL,
Garaz     Integer     DEFAULT '1',
odkdy     Date     DEFAULT '1.1.1990',
CONSTRAINT pk_Zamestnanec_Automobil PRIMARY KEY (Rodne_cislo));

CREATE TABLE Dokument (
ISBN     Integer     NOT NULL,
Nazev     Varchar(60)     NOT NULL,
Nakladatelstvi     Varchar(40)     DEFAULT 'Albatros',
CONSTRAINT pk_Dokument PRIMARY KEY (ISBN));

CREATE TABLE Kategorie (
Cislo     Integer     NOT NULL,
Nazev     Char     DEFAULT 'K' NOT NULL  UNIQUE,
CONSTRAINT pk_Kategorie PRIMARY KEY (Cislo));

CREATE TABLE Kategorie_obsahuje_Dokument (
Dokument_ISBN     Integer     NOT NULL,
Kategorie_Cislo     Integer     NOT NULL,
CONSTRAINT pk_Kategorie_obsahuje_Dokument PRIMARY KEY (Dokument_ISBN, Kategorie_Cislo),
CONSTRAINT fk_Kategorie_obsahuje_Dokument_Dokument FOREIGN KEY (Dokument_ISBN) REFERENCES
Dokument(ISBN),
FOREIGN KEY (Kategorie_Cislo) REFERENCES Kategorie(Cislo));

-- Konec skriptu --
```

Skript byl spuštěn na databázích uvedených níže. V každé z nich provedení příkazu *Drop table* na neexistující tabulku vyvolá chybu. V některých případech vykonávání pokračuje dále, v některých je to bráno jako fatální chyba, záleží také na použitém klientovi. Uživatel by proto v tomto případě měl před převodem do SQL odškrtnout volbu *Přidat skript pro smazání tabulek* nebo z výsledného kódu úsek s příkazy *Drop table* vyjmout a třeba ho uložit do jiného souboru určeného k mazání tabulek.

Po přidání podpory konkrétních databází do aplikace ERTOS by se mohly vypisovat specializované příkazy, které jsou ve většině databází obsaženy a které se tabulku, která neexistuje, nesnaží smazat.

7.1 Oracle 9.2

Databáze Oracle má omezenou maximální délku identifikátorů na 30 znaků. Doporučuje se proto nastavit zkrácení identifikátorů alespoň na tuto délku. S tímto nastavením byl skript přijat bez problémů.

7.2 Microsoft SQL Server 2000

Vyskytl se jediný problém, a to že databáze MS SQL nemá časový datový typ nazván *Date*, ale *Datetime*.

7.3 PostgreSQL 8.1

V případě databáze PostgreSQL skript proběhl úspěšně.

7.4 MySQL 4.0.21

Na databázovém serveru MySQL proběhl skript úspěšně.

7.5 Firebird 1.5 dialekt 3

Databáze Firebird má přesně dáno pořadí integritních omezení použitých v definicích atributů. To vyžaduje přítomnost klíčového slova *Default* až po slovech *Not Null* a *Unique*, tedy přesně naopak než požaduje např. databázový server Oracle. Protože Oracle je častěji používaným databázovým systémem, byla syntaxe přizpůsobena jemu. Pro správnou funkčnost v databázi Firebird je třeba všechna klíčová slova *Unique* včetně za nimi následujících hodnot umístit až nakonec řádku definujícího daný atribut.

Maximální povolená délka identifikátoru je 31 znaků, nastavte si proto zkrácení identifikátorů alespoň na tuto délku.

8 SROVNÁNÍ S PODOBNÝMI PROGRAMY

Existuje mnoho programů, které umožňují vytváření ER diagramů. Uživatel může použít programy pro kreslení nejrůznějších schémat, diagramů a plánků, které obsahují ER diagram jako jeden z podporovaných. Jako příklad uveďme aplikaci **Dia** s licencí GNU GPL [15]. Program obsahuje objekt entity, atributu a vztahu, u kterých je možno editovat název, u vztahu rovněž kardinalitu. Lze také označit entitu jako slabou, ta je pak zobrazena s dvojitým okrajem. Objekty se nasázejí na pracovní plochu a poté se musí spojit čarami. Čáry se k objektům přichytí, takže při posunu drží vše pohromadě. Tyto programy slouží jen k vizualizaci nebo tisku diagramů, ale neumožňují převod do relačního modelu.

Pro ER modelování s převodem do SQL skriptu slouží mnohé nástroje typu CASE. Většinou jsou komerční, ve verzi bez poplatku jsou omezeny počtem použitých entit nebo časově. Liší se tím, zda podporují jen fyzické nebo i logické datové modelování.

CASE nástroje podporující pouze fyzické datové modelování

Mezi tyto programy patří například CASE Studio [16], ER Creator [17] nebo XTG DataModeller [18]. Práce s CASE Studiem je popsána níže. Oba zbývající programy jsou principiálně velice podobné, ale méně propracované. Při modelování jsou data zobrazována tak, jak budou následně uložena do tabulek v databázi. Podporovaných databází je celá řada. Samozřejmostí je reverse engineering, který vizualizuje data uložená v databázi. Programy jsou oblíbené, a to zvláště pro menší projekty.

CASE nástroje podporující fyzické i logické datové modelování

Tato skupina programů obsahuje většinou robustnější a také cenově méně dostupné produkty. Z čistě databázových můžeme zmínit například ER/Studio [19] nebo ERwin [20]. Často se však jedná o komplexní CASE nástroje pro veškeré modelování při vývoji software. Patří sem například Rational Rose, Select Enterprise, Power Designer či Oracle Designer.

Výše uvedené nástroje nejsou příliš vhodné pro výuku. Pouze fyzické modelování neumožňuje použití logických prvků, například ISA hierarchie nebo vícenásobných atributů. Komplexnější nástroje jsou poměrně složité, drahé a navíc často podporují pouze jiný typ notace. K výuce jsou vhodnější existující nekomerční programy.

Nekomerční programy

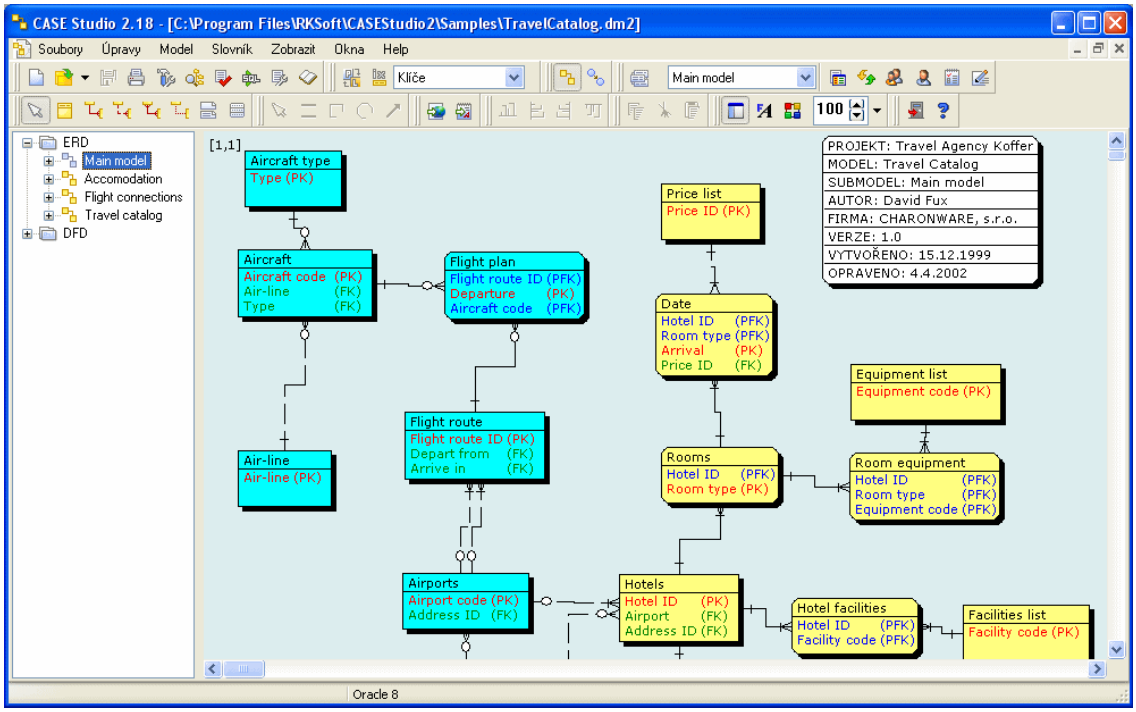
Volně dostupný (GNU GPL) je program Gerwin [21] popsaný níže. V jednoduché podobě podporuje jak logický, tak fyzický pohled. Vznikají také školní projekty, například v Javě napsané programy ER Modeller [22] a Sunicat [23] na ČVUT. Tyto programy jsou funkčně nejvíce podobné programu ERTOS.

8.1 Case Studio 2.18

Tento český program nabízí mnoho funkcí, podporuje velké množství databází včetně jejich různých verzí, kromě ER diagramů v něm lze vytvářet Data Flow diagramy, umožňuje proces reverse engineeringu s výběrem různých způsobů připojení k databázi, umí generovat HTML a RTF reporty, vytvářet tyto reporty i výsledný SQL skript podle uživatelem definovaných šablon. Testovala jsem demo verzi, která narozdíl od plné verze umožňuje uložit maximálně šest entit a neobsahuje správce verzí.

Vytvoření nového modelu a použitá notace

Před vytvořením nového souboru si uživatel musí vybrat cílovou databázi, které bude model přizpůsoben. Volba cílové databáze ovlivňuje, které objekty lze kromě entit, vztahů a atributů do schématu přidávat, týká se to například indexů, triggerů, uživatelských funkcí, a také co vše se u nich bude dát editovat. Podle databáze se liší také výběr datových typů, rozsah kontroly modelu a samozřejmě celý výsledný SQL skript. Zvolená databáze se dá později změnit. Možnost vytvářet model od začátku jen pro databázi, pro kterou je určen, zrychluje vývoj modelu a umožňuje od počátku využívat všech výhod vybraného databázového serveru.



Obrázek 8.1 Hlavní okno programu Case Studio 2.18

ER diagram se kreslí v notaci, kdy se atributy entit včetně datových typů vepisují dovnitř obdélníku entity (obr. 8.1). Rozlišují se vztahy identifikační, neidentifikační, M:N a informativní, které se navzájem liší svým zobrazením. Pouze vztah M:N může mít definovány atributy. Informační vztah jen informuje, že nějaká relace existuje a nevyvolává žádné akce. Obecně platí, že se ve schématu vůbec nevyskytují objekty typu vztah, ale pouze entity a propojující čáry (relace), což plyne už ze zvolené notace. Propojující čáry spojují entity lomenou čárou s mírným zlomem blízko konců čáry. Polohu čáry mezi dvěma entitami nelze ovlivňovat.

Druhy relací a jejich transformace

Při přidání **relace M:N** se rovnou místo vztahu přidá nová entita, do které se uloží primární klíče propojených entit. Ty v nové entitě tvoří zároveň primární i cizí klíč. Vztah už dále vůbec nevystupuje jako vztah, ale jako entita. Tedy například pozdější změnu typu vztahu nelze provést jinak než zrušením stávajícího vztahu a vytvoření nového vztahu žádaného typu. Vztah mezi více než dvěma entitami lze vytvořit jedině natáhnutím požadovaného množství relací z entity, která představuje vztah. Jako tuto entitu lze využít entitu vzniklou při dekompozici vztahu M:N. Pokud mají některé atributy v primárních klíčích propojovaných

entit shodný název, program relaci bez upozornění přidá, ale při kontrole schématu ohlásí chybu shodných názvů atributů nové entity. Automaticky tedy neřeší unikátnost názvů atributů.

Neidentifikační relace má výchozí kardinalitu (1,1) na straně entity, ve které bylo připojení zakončeno. Kardinalita se dá změnit v editačním dialogu relace nastavením nepovinné účasti ve vztahu na (0,1). Počáteční kardinalita u výchozí entity je (1,N), ta se dá změnit pomocí nepovinné účasti nebo ručním přepsáním druhé části kardinality na libovolné přirozené číslo. Kardinality se sice v diagramu změnit dají, ale na převod do výsledného SQL skriptu nemají žádný vliv. Do entity, ve které bylo propojení zakončeno, jsou vždy jen přidány všechny klíčové atributy první entity jako cizí klíč. U všech klíčových atributů ale zůstane klíčové slovo *Not null* i při nepovinné účasti, což jistě neodpovídá významu v diagramu. Úplně stejný výsledek navíc dostaneme také po převodu vztahu s kardinalitou (1,1):(1,1) .

Neidentifikační relace je zobrazována čárkovanou čárou s odlišnými konci. Nastavení nepovinné účasti přidá k příslušnému konci relace bílé kolečko, ruční změna kardinality se projeví zapsáním nové hodnoty do diagramu a také změnou tvaru zakončení čáry. Není jasné, proč existuje v diagramu textové i grafické vyjádření. Později už se nedá relace otočit, pro entitu, kterou jsme připojili jako druhou už nelze nastavit jinou kardinalitu než (0,1) nebo (1,1).

Identifikační relace se liší od neidentifikační zobrazením plnou čárou a hlavně tím, že entita, ve které jsme připojení ukončili, je druhou relací identifikována. To má za následek, že u druhé entity nelze nastavit nepovinnou účast ve vztahu. Automaticky se také do identifikované entity přidají klíčové atributy z druhé entity a statnou se zde primárním klíčem. Tato tzv. child entita je zobrazována odlišně pomocí zakulacení rohů. Identifikační relace je zobrazována plnou čarou.

Z relací je podporována ještě **self relace**. Nepodařilo se mi vygenerovat skript se self relací, aniž by předtím nebyla ohlášena chyba. Při vytvoření relace totiž program přidá do příslušné entity jako cizí klíč primární klíč z této entity. Názvy nejsou změněny, atributy klíčů se tedy jmenují stejně. Uživatel může změnit jedině názvy sloupců ve výsledné tabulce, takže SQL skript je v pořádku. Před výpisem se však vždy objeví chyba hlásící stejný název atributů ve schématu.

Fyzický model a jeho převod na SQL skript

Vzhled diagramu při fyzickém modelování je mnohem blíže relačnímu uspořádání dat než konceptuálnímu. Převod do relačního modelu probíhá už během vytváření diagramu. I když přímo není kladen požadavek na unikátní názvy atributů v rámci celého diagramu, Case Studio žádné přejmenování názvů při převodu do jiné entity neřeší. Každá entita představuje tabulku, ve které jsou vždy vypsány všechny atributy, které by se nyní do tabulky vkládaly. Generování SQL skriptu je pak už jen čisté vypsání všech entit včetně jejich atributů. Reverse engeneering z databáze do tohoto modelu probíhá poměrně přímočaře.

Tento způsob návrhu je přehledný, uživatel při pohledu na schéma ví, co bude obsahovat výsledný skript. Nejsou však podporovány prvky logického modelování jako ISA hierarchie nebo všechny druhy kardinalit.

Před generací skriptu si musí uživatel vybrat, co chce generovat. Implicitně není vybráno nic a vytvoří se prázdný skript. Program si výběr nepamatuje, pokud uživatel nestiskne tlačítko

Default, které se nachází na formuláři vpravo dole a je velice těžké zjistit jeho funkci. SQL skript se generuje do souboru. Dá se také prohlížet, ale v tomto případě se otevře další modální okno, které znemožní pohyb předchozího modálního okna. Pokud se chce uživatel podívat během čtení skriptu, co je pod tímto oknem, není to možné.

Přímé srovnání s programem ERTOS

Narozdíl od programu ERTOS aplikace Case Studio neumí vracet zpět změny, nemá ani jednonásobné undo. Proto se při každém mazání ptá, zda opravdu chcete objekt smazat, protože objekt se už zpět vrátit nedá. Také není podporován export do vektorového grafického formátu, i když je možno nastavit detaily uložení do tří různých rastrových formátů.

Case Studio se stalo inspirací k několika rozšířením programu ERTOS. Proto v programu ERTOS nalezneme například tyto funkce: výběr entit a vztahů, pro které se má generovat SQL skript, možnost generovat skript pro smazání tabulek, dialog pro editaci vlastností diagramu nebo pojmenování integritních omezení primárních a cizích klíčů. Case Studio však nabízí možnost parametricky nastavit přesné názvy jak integritních omezení, tak například i názvů tabulek. Z těchto myšlenek je v programu ERTOS zatím uplatněna jen možnost přidat předponu před název tabulky. Také myšlenka dvojích názvů pro objekty, logických a databázových, pochází z Case Studia. I když Case Studio například rušení české diakritiky při převodu moc nezvládá.

Jsou další věci v Case Studiu, o které by do budoucna ERTOS mohl být rozšířen. Patří mezi ně například generování HTML reportů o schématu, možnost obarvit jednotlivé entity různými barvami nebo přizpůsobení výsledného skriptu SQL různým databázím. Na vytvoření čeká také možnost definovat kandidátní klíče, tedy umožnit vytvořit *Unique* pro více atributů najednou. To s sebou přinese potřebu kontrolovat, zda neobsahuje primární klíč a *Unique* stejné atributy.

Case Studio vede k rychlému a pohodlnému návrhu databáze pro určitý databázový systém. Také pro správu stávajících databází se dá jedině doporučit. Blízkost k relačnímu modelu včetně možnosti rovnou definovat integritní omezení specifická pro určitou databázi však plně neumožňuje uvažovat a modelovat na konceptuální úrovni. A navíc, některé transformace příliš neodpovídají teorii, jak potvrzuje jeden z autorů programu: *„CASE Studio 2 vychází z požadavku trhu. Určité prvky teorie sice respektujeme, ale spíše než dodržet teoretická pravidla se snažíme vyhovět našim zákazníkům.“* [24]

8.2 ER/Studio Enterprise 7.0.1

Testovala jsem 14-denní trial verzi, kterou si lze stáhnout z internetu [19]. Součástí produktu je také verzovací systém s podporou týmové spolupráce. V ER/Studiu lze vytvářet logické modely a ty potom převádět na fyzické. Fyzický model lze vypsat v podobě SQL skriptu nebo se přímo připojit k databázi a tabulky si nechat vytvořit.

V programu jsou podporovány tři různé notace, z nichž ani jedna není podobná Chenově notaci. Dají se vkládat také různé další tvary, šipky či poznámky. Program nabízí několik možností pro automatické přerovnání objektů, které ale bohužel nelze vrátit zpět. Není k dispozici ani jednokroková operace undo. Lze vytvářet různé pohledy na hlavní model (submodely) sestávající jen z některých objektů, v jejich zobrazení se dají například skrýt nebo ukázat atributy. Program podporuje slévání těchto submodelů.

Logický model

Nový logický model lze vytvořit nakreslením, reverse engeneeringem z databáze nebo přeměnou textových skriptů včetně SQL. V logickém modelu je kromě entit a vztahů k dispozici objekt generalizace-specializace (ISA hierarchie). Je možné přidávat také objekty uživatelů, rolí a pohledů. Uživatelům lze nastavovat práva, např. které entity můžou číst a mazat, a také jim přidělovat role.

V průběhu editace modelu si uživatel volí následující činnost prostřednictvím nástrojové lišty nebo v kontextovém menu, které se otevře po kliku pravým tlačítkem myši. Poté se změní kurzor myši na obrázek představující objekt, který lze nyní vkládat. Dokud si uživatel v nástrojové liště nevybere jinou činnost nebo neklikne pravým tlačítkem myši někde na plochu, nelze vůbec nic jiného než vkládat entity dělat. Entita se vloží i při kliku na jinou entitu. Po umístění entity se dá ihned vepsat její název a po stisku klávesy Enter rovnou také atributy, což je příjemné.

Kromě typů vztahu, které se vyskytují v Case Studiu, existuje v ER/Studiu navíc ještě jeden vztah nazvaný jedna ku jedné. Zahrnuje ovšem jak kardinalitu (1,1), tak také (0,1), takže důsledkem přidání vztahu je pouze přemístění primárního klíče jedné entity do druhé entity. Při konfliktu se objeví dialog s dotazem, jak chce uživatel problémový atribut přejmenovat. Přemísťování klíčů probíhá už při vložení vztahu v rámci logického modelu, jen řešení vztahu M:N je ponecháno až pro proces vzniku fyzického modelu. Na propojovací čáru lze přidávat další zlomové body, takže čáry můžou mít libovolný lomený tvar.

Princip generalizace-specializace se realizuje podobně jako vztah, takže nejdříve musí schéma obsahovat potřebné entity, které se pak propojí ISA vztahem. Existují dva typy **ISA hierarchie**, které se liší grafickým zobrazením: úplná a neúplná. Úplná znamená, že pro každého příslušníka ISA hierarchie existuje specifický podtyp. V neúplné ISA hierarchii můžou existovat např. zaměstnanci (zdroj hierarchie), kteří nejsou ani programátoři ani účetní, což jsou jediní dva potomci v této hierarchii. Nicméně ve fyzickém modelu se oba druhy zobrazí stejně a to jako identifikační vztah s kardinalitou (1,1):(0,1), stejně jako v programu ERTOS.

Převod logického modelu na fyzický a následně do SQL

Při vytváření fyzického modelu z logického si uživatel musí vybrat cílovou databázi. Následně se převádí ISA hierarchie a M:N vztahy, mapují se datové typy, vznikají indexy, mění se názvy entit na názvy tabulek, názvy atributů na názvy sloupců a podobně. Nakonec se může diagram zkontrolovat. To vše se provádí podle uživatelem zvolených pravidel. Při převodu se dá mimo jiné zvolit, jak naložit se speciálními znaky v názvech, včetně mezer, a nakolik znaků se mají zkrátit názvy identifikátorů, podobně jako v programu ERTOS. Ve fyzickém modelu lze kromě ISA hierarchie vytvářet vše co v logickém a navíc přibyly objekty specifické pro zvolenou databázi, jako funkce, triggery, sekvence a další. Nic z toho, co se vytvoří ve fyzickém modelu, se už zpětně v logickém modelu neprojeví.

Z fyzického modelu probíhá už přímý převod do SQL skriptu. Opět je použita jednodušší verze převodu ER diagramu nevytvářející pro vztahy zvláštní tabulky. Narozdíl od Case Studia není u atributů, které tvoří příslušný cizí klíč u nepovinných vztahů, uvedeno integritní omezení *Not null*. Skript si lze zobrazit v náhledu pro různé varianty nastavení parametrů. Pro orientaci je užitečné okénko ukazující skript ve zmenšené podobě spolu s aktuálním výřezem. ER/Studiu nechybí možnost uložit model jako vektorový nebo rastrový obrázek.

Použití E/R studia k testování a k inspiraci

Zkoušela jsem zpětné vytvoření logického modelu z SQL skriptu. Vytvoří se stejný model s tím, že entity, které spolu původně tvořily ISA hierarchii, se objeví propojeny identifikační relací a vztah M:N už je zobrazen rozděleně. Lepší výsledek není možné očekávat, protože ISA hierarchii a vztah M:N z databáze rozpoznat nelze. V modelu vytvořeném z výstupního skriptu programu ERTOS chyběly některé relace. Příčinou byly chybějící cizí klíče, které měly být vytvořeny v rámci identifikačních vztahů a také u self relace. Zpětné modelování z SQL skriptu tedy pomohlo odhalit některé nedostatky v editoru ERTOS.

ERTOS by do budoucna mohl obsahovat kromě logického modelu, který obsahuje nyní, také fyzický model. Převod do něj je už nyní implementován, jen se hned vypíše v podobě skriptu. V ideálním případě by se ještě zobrazil a bylo by možné editovat názvy sloupců a třeba indexy. Zároveň by se tak implementovalo zobrazení ER diagramu po převodu ISA hierarchií vhodné pro potřeby výuky. Do fyzického modelu by mohl probíhat přímý reverse engeneering nebo převod ze skriptu, což je jen podmnožina reverse engeneeringu.

Dalším možným rozšířením je schopnost spojovat dohromady více modelů. Poslední dvě funkce by vyžadovaly také implementaci minimálně jednoho způsobu, jak rozmístit objekty a vztahy tak, aby se překrývaly co nejméně. V obou modelech by mohlo být na výběr z více notací nebo alespoň volba, zda zobrazovat všechny atributy, žádné atributy nebo jen klíče, a také způsob, jak zalomovat propojovací čáry na více místech.

ER/Studio je graficky velice hezky zpracovaný program. Podporuje sice méně databází než Case Studio, zato ale zobrazuje logické i fyzické modely, převádí je podle teorie a má více parametrických nastavení včetně makro jazyka. Je uzpůsoben práci v týmu na větších projektech.

8.3 Gerwin 0.6

Gerwin zaujme odlišným uspořádáním programu. Na jednotlivých záložkách jsou odděleně umístěny vlastnosti modelu, ER diagram, diagram tabulek a výstup do SQL. Akce prováděná na klik myši se mění pouze v jednoduché nástrojové liště, přitom tam najdeme i tak základní činnosti jako posun, editování a mazání. Činnosti se sice přizpůsobuje kurzor myši, ale i tak to může být pro uživatele matoucí.

Do ER diagramu lze vkládat entity a vztahy, které se zobrazují v podobě tabulek se seznamem atributů. U atributů je možné editovat pouze název, datový typ a údaj, zda je atribut klíčový. Objekty se editují ve spodní části obrazovky, kde se pro jednotlivé objekty vytvářejí záložky. Části diagramu nelze kopírovat, kombinace kláves Ctrl + C nezvykle zavírá soubor.

Je možné nastavit libovolné kardinality vztahu, ale převod probíhá opět úsporným způsobem. Nepovinná účast je řešena smazáním příslušné vlastnosti *Not null*. U relace nelze říci, že má být identifikační, takže všechny atributy kromě relace M:N jsou převáděny pouze jako cizí klíče. Gerwin přidává před názvy zděděných atributů předponu odvozenou od názvu entity, ze které pocházejí. Mezery v názvech nahrazuje znakem podtržítka.

Protože jsou diagram tabulek i SQL skript stále umístěny na záložkách, stačí je vždy jen aktualizovat. Pro SQL skript je na výběr SQL-92 a také dvě další databáze, ovšem výstup je pravděpodobně kvůli jednoduchosti programu ve všech případech shodný. Gerwin samozřejmě umožňuje ukládat a otevírat soubory ve svém formátu a disponuje také grafickým

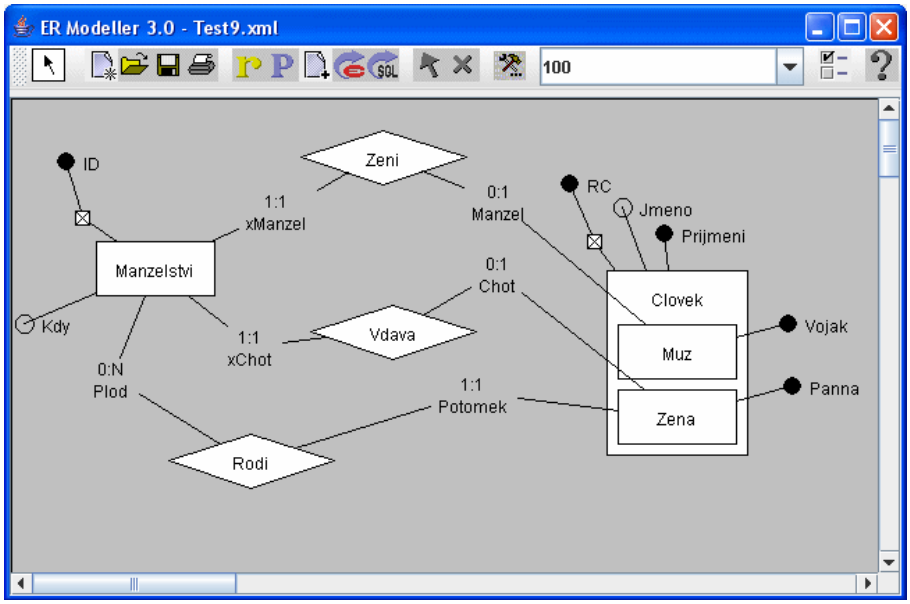
výstupem do PostScriptu. Vzhledem k tomu, že se program Gerwin neustále vyvíjí, může se z něj stát velice užitečný multiplatformní nástroj.

8.4 ER Modeller 3.0

Tento program byl vytvářen postupně několika diplomanty na ČVUT. Aplikace se neinstaluje, pouze je potřeba změnit ve spouštěcím souboru cestu k Java Runtime Environment. Verzi 3.0 zatím nelze spustit pod operačním systémem Linux a také příklady a nápověda jsou obsaženy pouze v balíčku předchozí verze programu.

Srovnání notací programů ER Modeller a ERTOS

ER Modeller obsahuje stejně jako program ERTOS pouze logické schéma, které se následně rovnou transformuje na SQL skript. Notace je v několika věcech velice podobná notaci používané v programu ERTOS, například v zobrazení entit, připojením atributů nebo v zapisování kardinalit a rolí vztahů. Úplně odlišně se zobrazuje ISA hierarchie (na obrázku 8.2 úplně vpravo) a typy atributů. Černé kolečko znamená povinnou účast, primární klíč se značí čtverečkem vyplněným křížkem, bez křížku se jedná o unikátní skupinu atributů.



Obrázek 8.2 Program ER Modeller 3.0 s ukázkou notace

Slabá entita je orámována dvojitě a vychází z ní lomené čáry ukazující na primární klíče identifikačních vlastníků. Tyto čáry nahrazují identifikační vztah, který se používá v programu ERTOS, takže v ER modeláři nemusí existovat mezi slabou entitou a jejím identifikačním vlastníkem žádný vztah. Jedním ze způsobů vytvoření slabé entity je dekompozice, kterou lze najít v kontextovém menu vztahu. Při ní je vztah nahrazen slabou entitou, která se stane závislá na všech entitách, se kterými byl vztah propojen.

Editace diagramu a ovládání

Nástrojová lišta slouží zejména k nahrazení menu, které v programu chybí. Nejvíce používané příkazy pro umísťování objektů do schématu a práci s nimi lze aktivovat pouze v kontextovém menu, které se objeví po kliku pravým tlačítkem myši. Vždy je potřeba vybrat

akci, kurzor se tím změní na křížek, a pak levým tlačítkem kliknout na místo, kam chce uživatel, aby se objekt umístil. Veškeré spojování, například vztahů s entitami, identifikačně závislých entit s jejich vlastníky, nebo přiřazování, ať už atributů do klíče nebo potomků do ISA hierarchie, se provádí tažením jednoho objektu na druhý. Tažení se musí provést pravým tlačítkem myši. Po chvíli se tlačítka myši můžou uživateli plést. Objekty se dají mazat jen v mazacím módu, který se zaktivuje v nástrojové liště.

Uživatel musí umístit každý atribut zvlášť. Tomu jsme se právě v programu ERTOS snažili vyhnout, protože to zpomaluje práci. Název atributu se vypisuje vždy doprava od zakončovacího kolečka, takže při umístění atributu doleva od objektu vede čára přes nápis názvu. U atributů lze změnit název, povinnou účast na nepovinnou a také datový typ. ER modelář ve verzi 3 podporuje uživatelské datové typy včetně polí, vnořených tabulek a objektů. Program je primárně určen pro připojení ke školnímu Oracle serveru, který tyto typy podporuje.

Kromě primárního klíče lze vytvářet kandidátní (unikátní) klíče, které se mohou skládat z více atributů. Kandidátní klíče mají svou značku v notaci, která je základem pro značku primárního klíče. Teprve vytvořený kandidátní klíč lze přeměnit v primární klíč. Kandidátní klíče jsou uvažovaným rozšířením programu ERTOS. Jediným problémem je chybějící podpora v notaci, možností jsou různě obarvená kolečka atributů podle příslušnosti k jednotlivým kandidátním klíčům. Jeden atribut by totiž nemělo jít přiřadit do více klíčů najednou. V ER modeláři se to dá učinit bez ohlášení chyby. V dokumentaci je ale napsáno, že by chyba měla být hlášena.

Editování diagramu v ER modeláři má své klady i zápory. Dá se libovolně měnit velikost entit, ale například předek ISA hierarchie po odebrání potomků zůstane v původní velikosti (příliš velký). Propojovací čára mezi entitou a vztahem se lomí v místech, kde je napsána kardinalita a role vztahu. Umožňuje to dát čáře libovolný lomený tvar, ale při posunu vztahu může docházet ke křížení těchto čar. Je tedy třeba přemístit i tyto zlomové body. Při editaci vlastností objektu neexistuje způsob, jak potvrdit změny, okno se pouze zavře, což vyvolává dojem, že se informace ztratí. Příjemná je možnost měnit kardinality vztahu už v kontextovém menu a ne až v dialogu pro editaci vlastností. Mezi nedostatky patří chybějící možnost vracet zpět provedené změny. Také není podporováno kopírování a vkládání částí schématu. Diagram má pevně určenou velikost, kterou nelze změnit.

Speciální funkce

Součástí programu je funkce slévání schémat s následnou možností řešit konflikty jak po jednom, tak po skupinách. Konflikty se zobrazí v přehledné tabulce, která se objevuje také po kontrole chyb ve schématu. Neobvyklou funkcí je dekomponování entity na entitu a slabou entitu nebo entity na dvě entity propojené relací s kardinalitami (1,1):(1,1). Opačný proces slévání entit a také vztahů je rovněž dovolen, dojde k přepojení atributů a účastí a při konfliktech se opět ukáže zmiňovaná tabulka.

Vytvořený diagram umí ER modelář uložit do formátu XML (přitom se pokaždé ptá na název souboru) nebo jako bitmapu se zvoleným rozlišením. Schéma lze rozdělit do několika obrázků a ty uložit do několika bitmap nebo přímo vytisknout na tiskárně.

Kontrola diagramu a výsledný skript

Program provádí kontrolu na požádání nebo před převodem do SQL skriptu. Kromě běžných kontrol vyžaduje existenci unikátních názvů rolí na obou stranách všech vztahů. Názvy rolí používá ve skriptu jako předponu zděděných atributů, takže nechá-li uživatel výchozí názvy rolí, nejsou názvy atributů moc hezké. Jako chyba jsou brány také české znaky v názvech, každý objekt má sice položku komentář, kam se dají psát libovolné znaky, ten se ale nikde nezobrazuje.

Do výsledného skriptu jsou vždy zahrnuty všechny objekty, nejdříve se objeví okno obsahující stromovou strukturu výsledných příkazů, ty se dají zobrazovat, ale pouze postupně po jednom. SQL skript se potom dá poslat do databáze pomocí nastaveného JDBC driveru nebo uložit do souboru. Před zděděné názvy atributů jsou přidány předpony, dá se zapnout jejich zkracování, to ale nechává z každé předpony jedno až dvě písmena, což není příliš čitelné. Pro atributy, které nebyly už přímo ve skriptu primárním klíčem a mají se stát klíčem po dekompozici vztahu, je vždy použito místo primárního klíče integritní omezení *Unique*. Uživatel si může také vybrat, zda chce do skriptu vložit část pro smazání tabulek.

ER modelář byl vyvíjen na míru školní výuce a to, že se v něm vytvářejí zápočtové práce, můžeme považovat za důkaz jeho úspěchu. Program podléhá stálému vývoji, dočkal se už několika verzí a určitě nabízí dobré možnosti pro konceptuální modelování pod operačním systémem Linux.

8.5 Sunicat 1.0

Program Sunicat se od ER modeláře liší lepším grafickým vzhledem s podporou tří notací včetně Chenovy. Atributy jsou ale vždy zobrazovány uvnitř obdélníku entity, přičemž se dá zobrazení neklíčových nebo všech atributů vypnout. Po kliknutí pravým tlačítkem myši na entitě nebo vztahu se otevře editační tabulka vlastností. U atributů lze zadat kromě běžných vlastností také *Unique* (obsahující maximálně jeden atribut), hodnotu *Default* a komentář. K objektům může uživatel přidat zkratku, kterou lze použít jako předponu pro databázové názvy, a také komentář. Pro vyjádření identifikačních vztahů slouží speciální objekt, u kterého lze měnit kardinality jen na jedné straně. Podporována je také ISA hierarchie.

Editace diagramu je někdy poněkud neobratná. Propojovací čáry entit a vztahů lze sice zlomit na více místech, přesto se někdy po několikerém kliknutí na vztah nic nestane. Ocitnou-li se dvě entity v poloze přes sebe, neexistuje způsob, jak je od sebe rozdělit. Označování objektů probíhá pouze při výběru oblasti tahem myši. Program má někdy problémy s překreslováním zneplatněných ploch. Stalo se také, že při návratu k programu Sunicat přepnutím z okna jiné aplikace program vůbec nereagoval. Bylo to způsobeno otevřeným dialogem programu, který byl skrytý za hlavním oknem aplikace a nebyl vidět. Ke kladům patří možnost změnit velikost plochy diagramu ručním zadáním rozměrů.

Program samozřejmě umí zkontrolovat schéma a převést ho do SQL skriptu, který lze přímo prohlížet nebo uložit do souboru. Je schopen také rovnou vytvořit tabulky v databázi Oracle. Před převodem si může uživatel vybrat, zda chce použít zkrácené názvy atributů a také, zda se mají slévat do jedné tabulky vztahy s kardinalitami (1,1):(1,1). Slévání není doporučeno při výskytu cyklu takovýchto vztahů.

Program zvládá ukládat diagramy do XML, do tří grafických formátů a také přímo tisknout. Generuje graficky hezky provedené HTML reporty s obrázkem schématu, tabulkovým

popisem entit a vyjmenováním všech relací. Oplývá množstvím nastavení, od volby vzhledu prostředí i diagramu, přes nastavení prefixů jednotlivých objektů až po nastavení viditelnosti mřížky. Po jejím aktivování jsou souřadnice polohy objektů zaokrouhlovány na určité násobky.

Program na první pohled vypadá propracovaněji než ER modelář, snaží se o modernější přístup, ale pro praktické použití zatím není ideální. Po opravě chyb se z něj ale může stát uživatelsky přívětivý program.

9 ZÁVĚR

Cílem této bakalářské práce bylo vytvořit editor ER diagramů s chováním standardního CASE nástroje podporující převod do relačního modelu. Vytvořila jsem program ERTOS, který umožňuje navrhovat ER diagramy v Chenově notaci a následně z diagramu vygenerovat SQL skript. Snažila jsem se uživateli editování co nejvíce zpříjemnit pomocí jednoduchého ovládání, zautomatizování kreslení atributů či možností kopírovat a vkládat části schématu. Důležitá je schopnost vracet zpět veškeré operace včetně těch provedených před minulým zavřením souboru a také unikátní možnost automatického přehrávání procesu vzniku diagramu.

Důležité bylo nezavádět zbytečná omezení při vytváření diagramu. Výsledkem je možnost použití ISA hierarchie, n-árních vztahů a několikanásobných atributů. Před převodem do relačního modelu se ověřuje, zda je ER diagram korektní. Samotný převod probíhá podle algoritmu uvedeného v druhé kapitole této práce.

Srovnání s jinými programy probírané v poslední kapitole mi vyjasnilo pohled na možnosti při návrhu databáze pomocí CASE nástrojů. Z množství komerčních programů zabývajících se touto problematikou plyne zájem programátorů o tyto produkty. Zajímavý je ovšem principiální rozdíl mezi implementací převodu ER diagramu do relačního modelu v těchto programech a v programech vznikajících v akademické sféře.

9.1 Budoucnost programu ERTOS

Program ERTOS by mohl být v budoucnu rozšířen o množství dalších funkcí:

- Podpora konkrétních databázových systémů. Podpora by se projevila jak odlišnými konstrukcemi ve výsledném SQL skriptu, tak možností se k databázi přímo připojit.
- Zavedení fyzického modelu, který by vznikal z logického modelu, a do kterého by mohl probíhat reverse engeneering z databáze či z SQL skriptu.
- Slévání více modelů dohromady.
- Reverse engeneering i slévání více modelů bude vyžadovat funkci automatického rozmísťování objektů (tzv. layout).
- Možnost definovat kandidátní klíče, tj. *Unique* pro více atributů najednou.
- Generování HTML reportů o diagramu.
- Parametrické nastavení názvů tabulek a integritních omezení, kontrola duplicit názvů integritních omezení.
- Podpora více notací, upřesnění typu ISA hierarchie.
- Detaily pro lepší editaci (obarvení objektů, možnost zadat k objektu komentář, okénko ukazující skript ve zmenšené podobě, zalomování propojovacích čar na více místech atd.).
- Tisk ER diagramu, popřípadě SQL skriptu.

ZDROJE

[1] Pokorný J., Halaška I.: *Databázové systémy: Vybrané kapitoly a cvičení*, Nakladatelství Karolinum UK, 1998.

[2] Batini C., Cero S., Navathe S. B.: *Conceptual database design*, The Benjamin/Cummings Publishing Company, 1992.

[3] Sanders G. L., *Data modeling*, boyd & fraser publishing company, 1995.

[4] Beneš M.: *Přehled OO metodik a notací*, diplomová práce VŠE, http://objekty.vse.cz/Objekty/MetodikyANotace.

[5] Garcia-Molina H., Ullman J. D., Widom J.: *Database systems: The complete book*, Prentice Hall, 2002.

[6] Pokorný J.: Prezentace k přednáškám, http://service.felk.cvut.cz/courses/36DB2/slides/koncept_analyza_a_design.pdf.

[7] Mannila H., Raiha K.J.: *The design of relational databases*, Addison Wesley, 1994.

[8] Ramakrishnan R., Gehrke J.: *Database management systems*, McGraw-Hill, 2003.

[9] O'Neil P., O'Neil B.: *Database: principles, programming, performance*, Morgan Kaufmann Publishers, 2001.

[10] Procházka J.: *CASE – jak vybrat?*, http://www.dbsvet.cz/view.php?cisloclanku=2004052402.

[11] Procházka J.: *Nástroje CASE? Co? Proč? Jak?*, http://www.dbsvet.cz/view.php?cisloclanku=2004052702.

[12] Prosise J.: *Programování v Microsoft .NET*, Nakladatelství Computer Press, 2003.

[13] Virius M.: *Od C++ k C#*, nakladatelství KOPP, 2002.

[14] MagicLibrary.dll, http://www.dotnetmagic.com/downloads/MagicInstall174.msi.

[15] Dia, http://dia-installer.sourceforge.net/.

[16] Case studio, http://www.casestudio.com/csy/.

[17] ERCreator, http://www.modelcreator.com/index.php?s=content&p=database.

[18] XTG Data Modeller, http://www.xtgsystems.com/xtgdm.php3.

[19] ER/Studio, http://www.embarcadero.com/products/erstudio/index.html.

[20] ERwin, http://www3.ca.com/solutions/Product.aspx?ID=260.

[21] Gerwin, http://www.gnu.org/software/ferret/project/what.html.

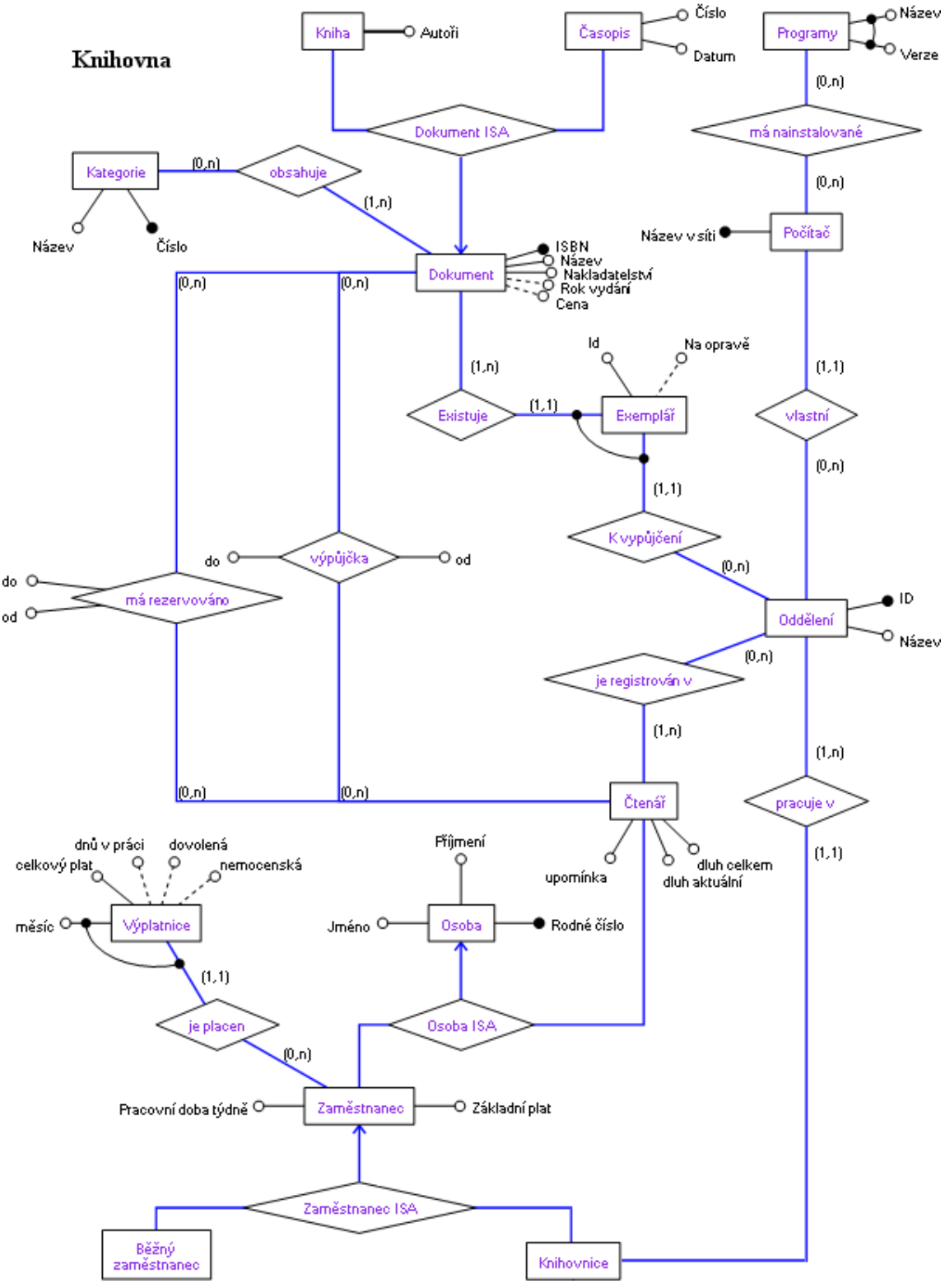
[22] ER modelář, http://service.felk.cvut.cz/courses/X36DBS/.

[23] Sunicat, http://martin.feld.cvut.cz/~molhanec/vyuka/36dbs/files/Sunicat.zip.

[24] Rozhovor s Radimem Kunzem, http://www.dbsvet.cz/view.php?cisloclanku=2004052604.

PŘÍLOHY

Příloha A – *ukázka ER diagramu nakresleného v programu ERTOS*



Příloha B – *SQL skript vygenerovaný z ER diagramu uvedeného v příloze A*

Skript byl vygenerován bez SQL komentářů a bez části pro smazání tabulek.

```
CREATE TABLE Dokument (
ISBN    Varchar(20)    NOT NULL,
Nazev    Varchar(80)    NOT NULL,
Nakladatelstvi    Varchar(60)    DEFAULT 'Albatros'  NOT NULL,
Rok_vydani    Date,
Cena    Float,
CONSTRAINT pk_Dokument PRIMARY KEY (ISBN));

CREATE TABLE Oddeleni (
ID    Integer    NOT NULL,
Nazev    Varchar(60)    NOT NULL,
CONSTRAINT pk_Oddeleni PRIMARY KEY (ID));

CREATE TABLE Exemplar (
Id    Integer    NOT NULL,
Na_oprave    Integer,
Existuje_ISBN    Varchar(20)    NOT NULL,
K_vypujceni_ID    Integer    NOT NULL,
CONSTRAINT pk_Exemplar PRIMARY KEY (Existuje_ISBN, K_vypujceni_ID),
CONSTRAINT fk_Exemplar_Dokument FOREIGN KEY (Existuje_ISBN) REFERENCES Dokument(ISBN),
CONSTRAINT fk_Exemplar_Oddeleni FOREIGN KEY (K_vypujceni_ID) REFERENCES Oddeleni(ID));

CREATE TABLE Osoba (
Rodne_cislo    Varchar(10)    NOT NULL,
Jmeno    Varchar(40)    NOT NULL,
Prijmeni    Varchar(40)    NOT NULL,
CONSTRAINT pk_Osoba PRIMARY KEY (Rodne_cislo));

CREATE TABLE Zamestnanec (
Zakladni_plat    Integer    NOT NULL,
Pracovni_doba    Integer    NOT NULL,
Rodne_cislo    Varchar(10)    NOT NULL,
CONSTRAINT pk_Zamestnanec PRIMARY KEY (Rodne_cislo),
CONSTRAINT fk_Zamestnanec_Osoba FOREIGN KEY (Rodne_cislo) REFERENCES Osoba(Rodne_cislo));

CREATE TABLE Knihovnice (
Rodne_cislo    Varchar(10)    NOT NULL,
Oddeleni_ID    Integer    NOT NULL,
CONSTRAINT pk_Knihovnice PRIMARY KEY (Rodne_cislo),
CONSTRAINT fk_Knihovnice_Oddeleni FOREIGN KEY (Oddeleni_ID) REFERENCES Oddeleni(ID),
CONSTRAINT fk_Knihovnice_Zamestnanec FOREIGN KEY (Rodne_cislo) REFERENCES Zamestnanec(Rodne_cislo));

CREATE TABLE Ctenar (
dluh_celkem    Float    NOT NULL,
dluh_aktualni    Float    NOT NULL,
upominka    Integer    NOT NULL,
Rodne_cislo    Varchar(10)    NOT NULL,
CONSTRAINT pk_Ctenar PRIMARY KEY (Rodne_cislo),
CONSTRAINT fk_Ctenar_Osoba FOREIGN KEY (Rodne_cislo) REFERENCES Osoba(Rodne_cislo));

CREATE TABLE Kategorie (
Cislo    Integer    NOT NULL,
Nazev    Varchar(60)    NOT NULL,
CONSTRAINT pk_Kategorie PRIMARY KEY (Cislo));

CREATE TABLE Bezny_zamestnanec (
Rodne_cislo    Varchar(10)    NOT NULL,
CONSTRAINT pk_Bezny_zamestnanec PRIMARY KEY (Rodne_cislo),
CONSTRAINT    fk_zamestnanec_Zamestnanec    FOREIGN    KEY    (Rodne_cislo)    REFERENCES
Zamestnanec(Rodne_cislo));

CREATE TABLE Vyplatnice (
mesic    Date    NOT NULL,
dnu_v_praci    Integer,
dovolena    Integer,
nemocenska    Integer,
premie    Integer,
celkovy_plat    Integer    NOT NULL,
```

```
socialni	Integer,
zdravotni	Integer,
dan	Integer,
Rodne_cislo	Varchar(10)	NOT NULL,
CONSTRAINT pk_Vyplatnice PRIMARY KEY (mesic, Rodne_cislo),
CONSTRAINT fk_Vyplatnice_Zamestnanec FOREIGN KEY (Rodne_cislo) REFERENCES Zamestnanec(Rodne_cislo));

CREATE TABLE Pocitac (
Nazev_v_siti	Varchar(20)	NOT NULL,
Oddeleni_ID	Integer	NOT NULL,
CONSTRAINT pk_Pocitac PRIMARY KEY (Nazev_v_siti),
CONSTRAINT fk_Pocitac_Oddeleni FOREIGN KEY (Oddeleni_ID) REFERENCES Oddeleni(ID));

CREATE TABLE Programy (
Nazev	Varchar(60)	NOT NULL,
Verze	Varchar(10)	NOT NULL,
CONSTRAINT pk_Programy PRIMARY KEY (Nazev, Verze));

CREATE TABLE Kniha (
Dokument_ISA_ISBN	Varchar(20)	NOT NULL,
CONSTRAINT pk_Kniha PRIMARY KEY (Dokument_ISA_ISBN),
CONSTRAINT fk_Kniha_Dokument FOREIGN KEY (Dokument_ISA_ISBN) REFERENCES Dokument(ISBN));

CREATE TABLE Casopis (
Cislo	Integer	NOT NULL,
Datum	Date	NOT NULL,
Dokument_ISA_ISBN	Varchar(20)	NOT NULL,
CONSTRAINT pk_Casopis PRIMARY KEY (Dokument_ISA_ISBN),
CONSTRAINT fk_Casopis_Dokument FOREIGN KEY (Dokument_ISA_ISBN) REFERENCES Dokument(ISBN));

CREATE TABLE je_registrovan_v (
Rodne_cislo	Varchar(10)	NOT NULL,
Oddeleni_ID	Integer	NOT NULL,
CONSTRAINT pk_je_registrovan_v PRIMARY KEY (Rodne_cislo, Oddeleni_ID),
CONSTRAINT fk_registrovan_v_Ctenar FOREIGN KEY (Rodne_cislo) REFERENCES Ctenar(Rodne_cislo),
CONSTRAINT fk_registrovan_v_Oddeleni FOREIGN KEY (Oddeleni_ID) REFERENCES Oddeleni(ID));

CREATE TABLE vypujcka (
od	Date	NOT NULL,
do	Date	NOT NULL,
Dokument_ISBN	Varchar(20)	NOT NULL,
Rodne_cislo	Varchar(10)	NOT NULL,
CONSTRAINT pk_vypujcka PRIMARY KEY (Dokument_ISBN, Rodne_cislo),
CONSTRAINT fk_vypujcka_Dokument FOREIGN KEY (Dokument_ISBN) REFERENCES Dokument(ISBN),
CONSTRAINT fk_vypujcka_Ctenar FOREIGN KEY (Rodne_cislo) REFERENCES Ctenar(Rodne_cislo));

CREATE TABLE ma_rezervovano (
od	Date	NOT NULL,
do	Date	NOT NULL,
Dokument_ISBN	Varchar(20)	NOT NULL,
Rodne_cislo	Varchar(10)	NOT NULL,
CONSTRAINT pk_ma_rezervovano PRIMARY KEY (Dokument_ISBN, Rodne_cislo),
CONSTRAINT fk_rezervovano_Dokument FOREIGN KEY (Dokument_ISBN) REFERENCES Dokument(ISBN),
CONSTRAINT fk_rezervovano_Ctenar FOREIGN KEY (Rodne_cislo) REFERENCES Ctenar(Rodne_cislo));

CREATE TABLE obsahuje (
Dokument_ISBN	Varchar(20)	NOT NULL,
Kategorie_Cislo	Integer	NOT NULL,
CONSTRAINT pk_obsahuje PRIMARY KEY (Dokument_ISBN, Kategorie_Cislo),
CONSTRAINT fk_obsahuje_Dokument FOREIGN KEY (Dokument_ISBN) REFERENCES Dokument(ISBN),
CONSTRAINT fk_obsahuje_Kategorie FOREIGN KEY (Kategorie_Cislo) REFERENCES Kategorie(Cislo));

CREATE TABLE ma_nainstalovane (
Nazev_v_siti	Varchar(20)	NOT NULL,
Programy_Nazev	Varchar(60)	NOT NULL,
Programy_Verze	Varchar(10)	NOT NULL,
CONSTRAINT pk_ma_nainstalovane PRIMARY KEY (Nazev_v_siti, Programy_Verze, Programy_Nazev),
CONSTRAINT fk_nainstalovane_Pocitac FOREIGN KEY (Nazev_v_siti) REFERENCES Pocitac(Nazev_v_siti),
CONSTRAINT fk_nainstalovane_Programy FOREIGN KEY (Programy_Nazev, Programy_Verze) REFERENCES
Programy(Nazev,Verze));
```