Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

Bakalářská práce

Dana Soukupová

Algoritmy logického návrhu relační databáze

Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Antonín Říha, CSc

Studijní program: Informatika

Děkuji vedoucímu této práce za trpělivost a cenné rady, které mi poskytoval.

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

V Praze dne 11. prosince 2004

Dana Soukupová

Obsah

1 ÚVOD 2

2 ALGORITMY 3

2.1 PROBLÉM NÁVRHU DATABÁZE 3
2.2 RELAČNÍ TEORIE 4
2.2.1 MAXIMÁLNÍ MNOŽINY 7

3 UŽIVATELSKÁ DOKUMENTACE 10

3.1 INSTALACE SYSTÉMU 10
3.2 OKNA APLIKACE 11
3.2.1 HLAVNÍ OKNO APLIKACE 11
3.2.2 ZADÁVÁNÍ ATRIBUTŮ 12
3.2.3 ZADÁVÁNÍ ZÁVISLOSTÍ 12
3.2.4 DATOVÉ SOUBORY 13
3.2.5 REDUNDANTNÍ ATRIBUTY 13
3.2.6 REDUNDANTNÍ ZÁVISLOSTI 13
3.2.7 KLÍČE 14
3.2.8 DEKOMPOZICE 15
3.2.9 SYNTÉZA 17
3.2.10 MAXIMÁLNÍ MNOŽINY 19
3.2.11 TESTOVACÍ MODUL 19

4 PROGRAMÁTORSKÁ DOKUMENTACE 23

4.1 ULOŽENÍ DAT 23
4.2 UNITY 24
4.2.1 ABC_INPUT, DEP_INPUT 24
4.2.2 FILE OPEN CLOSE 25
4.2.3 DEFINICE 25
4.2.4 MAIN FORM 26
4.2.5 CLOSURE 27
4.2.6 PRINT FORM 28
4.2.7 ABOUT_FORM 28
4.2.8 REDUNDANCE 28
4.2.9 REDUNDANT ATTRIBUTES FORM, REDUNDANT DEPENDENCY FORM 29
4.2.10 KEYS, KEYS_FORM 29
4.2.11 MAXSET FORM 30
4.2.12 NF 31
4.2.13 DECOMPOSITION FORM 2 31
4.2.14 SYNTEZA 2 32

4.2.15 TEST FORM 32

5 POROVNÁNÍ S JINÝM PROGRAMEM 34

5.1 DBE DESIGNER 34
5.1.1 POPIS PROGRAMU 34
5.1.2 DATA PRO POROVNÁNÍ 36
5.1.3 VÝSLEDEK POROVNÁNÍ 37
5.2 NORMALIZACE DATABÁZOVÝCH SCHÉMAT V PROLOGU 40

6 ZÁVĚR 44

LITERATURA 45

Název práce: Algoritmy logického návrhu relační databáze
Autor: Dana Soukupová
Katedra: Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Antonín Říha, CSc
e-mail vedoucího: Antonin.Riha@mff.cuni.cz

Abstrakt:
Vytvoření aplikace implementující algoritmy logického návrhu relační databáze. Tyto algoritmy zahrnují dekompozici, syntézu a algoritmy související – určení redundancí atributů a závislostí, klíčů, uzávěru, výpočet maximálních množin.
Zároveň program obsahuje část, která slouží uživateli k učení se algoritmů. Uživatel si zvolí vstupní data a program pak jeho odpověď vyhodnotí.
Vstupem jsou atributy a závislosti schématu, výstup je zobrazen uživateli. Navržená aplikace je vhodná pro menší databázová schémata. Předpokládá se, že uživatel nejprve použije ER návrh.
Tato práce obsahuje i porovnání programu Databázové algoritmy s programem DBE Designer a programem normalizace implementovaném v Prologu.

Klíčová slova: databázové algoritmy, implementace dekompozice, maximální množiny

Title: Algorithms of logical design for relational databases
Author: Dana Soukupová
Department: Department of Software Engineering
Supervisor: RNDr. Antonín Říha, CSc
Supervisor's e-mail address: Antonin.Riha@mff.cuni.cz

Abstract:
To create application with implemented algorithms of logical design for relational databases. These algorithms include decomposition, synthesis and algorithms related – redundant attributes and dependencies, keys, closure, maximal sets computation.
Program also includes part designed for users who want to learn database algorithms. User chooses data and algorithms he wants to be tested on. Program evaluates user's answer and gives correct one.
Input consists of attributes and dependencies of the database schema, output is shown to user. Application is designed for smaller database schemas. We assume that user applies ER design before this application is used.
This thesis also includes comparison of Database algorithms program with DBE Designer program and with normalization program implemented in Prolog.

Keywords: database algorithms, implementation of decomposition, maximal sets

1 Úvod

Cílem tohoto projektu je vytvořit aplikaci implementující algoritmy logického návrhu relační databáze.

Tyto algoritmy zahrnují dekompozici, syntézu a algoritmy související – určení redundancí, klíčů, uzávěru, výpočet maximálních množin.

Zároveň bude program obsahovat část, která bude sloužit uživateli k učení se algoritmů. Vstupem budou atributy a závislosti schématu, výstup bude zobrazen uživateli. Navržená aplikace bude vhodná pro menší databázová schémata. Předpokládá se, že uživatel nejprve použije ER návrh.

V tomto textu nejprve ve druhé kapitole popíši implementované algoritmy, třetí část má formu uživatelské dokumentace, která slouží k přehledu vlastností vytvořeného programu i jako návod pro jeho použití. Čtvrtá kapitola obsahuje programátorskou dokumentaci umožňující další vývoj aplikace. V páté, závěrečné, části porovnám svůj program Databázové algoritmy s programem DBE Designer a s programem normalizace implementovaném v Prologu.

2 Algoritmy

Tato kapitola obsahuje stručný přehled algoritmů logického návrhu relační databáze. Protože podrobné zkoumání algoritmů není cílem této práce, omezíme se pouze na potřebný základ. K bližšímu seznámení pak může posloužit literatura zmíněná v závěru.

2.1 Problém návrhu databáze

Navrhnout dobře strukturovanou relační databázi je náročný úkol, který vyžaduje jak znalost teorie databázového designu, tak obecné porozumění aplikaci, pro kterou je databáze navrhována.

Databáze by měla být navržena tak, aby mohla zpracovávat uživatelské požadavky, být prostorově efektivní a eliminovat vznik anomálií. Proces návrhu můžeme rozdělit na čtyři základní fáze: 1. analýza požadavků a specifikace, 2. konceptuální návrh, 3. logický návrh a 4. fyzický návrh [4].

1. Analýza požadavků

   Během této analýzy jsou shromažďovány detailní informace o aplikaci prostřednictvím konzultací s potenciálními uživateli a specialisty. Je důležité, aby specifikace byly kompletní, konzistentní a co možná nejsrozumitelnější, protože další proces záleží na nich. Chyby ve specifikaci mohou vyústit v návrh, který neuspokojuje původní požadavky.

2. Konceptuální návrh

   Konceptuální návrh zahrnuje mapování specifikace do sémantického modelu. Sémantické modely umožňují modelovat data tak, aby byla srozumitelná pro uživatele. Jako příklad můžeme citovat ER model.

3. Logický návrh

   Během logického návrhu jsou logické struktury definované v rámci konceptuálního návrhu převedeny do datového modelu. Logický návrh přímo ovlivňuje organizaci dat v tabulkách a tím i způsob, jak bude k datům logicky přistupováno. V průběhu logického návrhu se také zjišťuje, zda byly eliminovány redundance a provedena normalizace tabulek.

4. Fyzický návrh

V této konečné fázi jsou zvoleny vhodné datové struktury. Systém řízení databáze obvykle podporuje různé možnosti uložení datových struktur a různé přístupové metody. K určení optimálního fyzického návrhu je potřeba vědět, jak bude databáze používána uživatelem a znát relativní frekvenci uživatelských dotazů.

V každé fázi by se měli zapojit specialisté i databázoví návrháři. Rozhodnutí udělané v jedné fázi může ovlivnit další fáze a tím i konečný design.

2.2 Relační teorie

Relační model a první normální forma byly představeny Coddem v roce 1970[4]. Codd definoval funkční závislosti jako vztah mezi atributy relace. Codd poté také navrhnul druhou a třetí normální formu jako vyjádření toho, jak na základě vlastností funkčních závislostí mají vypadat správně formulované relace.

V relačním modelu jsou data uložena v tabulkách nazvaných 'relace'. Každá relace má fixní počet sloupců, nazvaných atributy, a dynamický, měnící se, počet řádek. Příkladem je relace popisující data zaměstnanců společnosti, viz níže:

Číslo	*Jméno*	*Věk*	*Číslo oddělení*	*Manager*
1	Jarda	21	2	Fric
2	Franta	45	3	Beneš
3	Josef	54	4	Brandejs
4	Tomáš	34	3	Beneš

Schématem relační databáze označujeme konečnou množinu relačních schémat. *Relačním schématem R* rozumíme konečnou množinu atributů spolu s konečnou množinou závislostí F.

Důležitým konceptem je koncept funkčních závislostí. Nechť $A = \{A_1,\ldots,A_n\}$ je relační schéma, X,Y podmnožiny A. Řekneme, že Y *funkčně závisí* na X, značíme $(X \rightarrow Y)$, jestliže pro každou instanci r nad A platí: pro všechna t_1, t_2 z r: $(t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y])$.

Nechť $A = \{A_1,\ldots,A_n\}$ je relační schéma. *Nadklíčem* schématu A je každá podmnožina množiny A, na níž A funkčně závisí. *Klíčem* schématu je pak takový nadklíč schématu A, jehož žádná vlastní podmnožina není nadklíčem A. Každý atribut, který je obsažen v alespoň jednom klíči se nazývá klíčový, ostatní atributy se označují jako neklíčové.

Ve výše uvedeném případě jsou funkční závislosti například (Číslo → Jméno, Číslo → Věk, Číslo → Číslo oddělení). Budeme-li předpokládat, že každé oddělení má právě jednoho manažera, platí také závislost Číslo oddělení → Manažer. Klíčem schématu je Číslo. Z poslední závislosti však vznikají určité anomálie:

- Není možné říci, kdo je manažerem oddělení, které nemá žádné zaměstnance.
- Kdykoli se změní manažer jednoho oddělení, je potřeba změnit více řádků tabulky.
- Smazání posledního zaměstnance oddělení způsobí ztrátu informace o tom, kdo je manažerem oddělení.

Aby se těmto anomáliím zabránilo, použijeme proces normalizace. Výše uvedené schéma by pak bylo uloženo jako:

Číslo	*Jméno*	*Věk*	*Číslo oddělení*
1	Jarda	21	2
2	Franta	45	3
3	Josef	54	4
4	Tomáš	34	3

Číslo oddělení	*Manažer*
1	Kocián
2	Fric
3	Beneš
4	Brandejs

Definované normální formy jsou následující:

První normální forma: Relační schéma R je v první normální formě, jestliže všem atributovým jménům jsou jako domény přiřazeny jednoduché datové typy.

Druhá normální forma: Relační schéma R je ve druhé normální formě, jestliže je v první normální formě a žádný neklíčový atribut není funkčně závislý na žádné vlastní podmnožině klíče.

Třetí normální forma: Relační schéma je ve třetí normální formě, jestliže je v první normální formě a žádný neklíčový atribut není tranzitivně závislý na žádném klíči.

Boyce-Coddova normální forma(BCNF): Relační schéma R je v BCNF jestliže pro každou funkční závislost $Y \rightarrow X$, kde $Y \subseteq R$ a X atribut z R, platí alespoň jedna z následujících podmínek:

- Závislost je triviální, tj. atribut X je obsažen v Y
- Y je nadklíč schématu R.

Správně navržené relace by měly být v co nejvyšší normální formě.

Dekompozice spočívá v náhradě výchozí relace dvěma jejími projekcemi. Schéma je rozděleno podle funkční závislosti, která porušuje požadovanou normální formu. Bezztrátovost dekompozice zaručuje věta o dekompozici funkčních závislostí [7]. Při dekompozici ale bohužel může dojít ke ztrátě některých závislostí. Tyto je pak na konci možno přidat jako samostatné schéma.

Před popisem algoritmu syntézy si nejprve objasněme pojmy uzávěr, redundantní závislost, redundantní atribut a ekvivalentní klíče.

Nechť F je množina závislostí relačního schématu R.

Uzávěr množiny atributů $X \subseteq R$, značíme X^+. Je definován jako množina atributů $A \in R$, pro které platí, že z F je odvoditelná závislost $X \rightarrow A$.

Závislost $X \rightarrow Y$ je *redundantní*, jestliže uzávěr X^+ její levé strany, vytvořený bez použití závislosti $X \rightarrow Y$, obsahuje Y; závislost je odvoditelná z ostatních závislostí v F

Atribut A je redundantní v levé straně některé závislosti $X \rightarrow Y$, jestliže $Y \subseteq (X - A)^+$.

Klíče K1 a K2 jsou ekvivalentní pokud platí, že $K1 \rightarrow K2$ a $K2 \rightarrow K1$.

Při syntéze se schémata generují ze závislostí. Generování schématu relační databáze metodou syntézy má tyto základní kroky:

- odstranění redundantních atributů a závislostí
- nalezení klíčů univerzálního schématu
- vytvoření skupin závislostí dle shody jejich levých stran
- vytvoření relačních schémat na základě skupin závislostí se shodnými levými stranami
- vyřešení schémat s ekvivalentními klíči nebo schémat, z nichž jedno je projekcí druhého.

Výsledné schéma je bezztrátové, pokud alespoň jeden klíč univerzálního schématu je obsažen v alespoň jednom vzniklém relačním schématu[1]. V opačném případě je nutné schéma určené univerzálním klíčem přidat. Při syntéze je v důsledku použité metody zaručeno pokrytí závislostí a třetí normální forma výsledného schématu.

S algoritmy dekompozice a syntézy úzce souvisí i algoritmy hledání klíčů schématu. Algoritmus pro nalezení klíčů dané množiny funkčních závislostí popsali Lucchesi a Osborn. Ti také ukázali, že rozhodnout, zda je atribut obsažen v klíči, je NP-úplný problém.

Lucchesi a Osborn algoritmus na hledání klíčů je následující. Nejprve se nalezne jakoukoli vhodnou metodou jeden klíč schématu. Například tak, že se množina atributů relačního schématu R postupně redukuje na minimum dle množiny závislostí F - odstraňují se redundantní atributy. Získaný klíč označme K. Poté procházíme závislosti F a pro každou závislost $X \rightarrow Y$ takovou, že průnik Y a K je neprázdný, vytvoříme množinu $(K \cup X) - Y$. Pokud tato množina není nadmnožinou dříve nalezeného klíče, je kandidátem na další klíč. Je však třeba ji redukovat na minimum. Pro každý nově nalezený klíč postup opakujeme. Hledání skončí, pokud v některém kroku nebyl žádný nový klíč nalezen.

2.2.1 Maximální množiny

Koncept maximálních množin byl představen z důvodu snížení exponenciální složitosti některých výše popsaných algoritmů. Maximální množinu definujeme jako množinu atributů, která je, pro daný atribut A, největší možnou množinou neurčující A. Maximální množiny jsou vlastně jinou reprezentací funkčních závislostí relačního schématu. Počet maximálních množin úzce souvisí s výpočetní složitostí algoritmů. Schémata ve vyšší normální formě mají méně maximálních množin, než nenormalizovaná schémata.[1]

Nechť F je množina závislostí relačního schématu R, $X \subseteq R$, a $A \in X$.

Pak max(F,X,A) je množina maximálních množin:

> $\max(F, X, A) = \{ Y \subseteq X$, Y je maximální množina $\mid Y \rightarrow A$ a pro všechna $Z \subseteq X$, takové, že $Y \subset Z$ platí, že v F je odvoditelná závislost $Z \rightarrow A \}$

Je-li F jasné z kontextu, píšeme pouze max(X, A). Pokud je X rovno R píšeme jenom max(F, A) nebo max (A). Sjednocení maximálních množin max (X,A) pro všechna $A \in F$ značíme MAX(X).

[1] Biskup, Dayal, Bernstein – „Synthesizing independent database schemas“, Proc. ACM SIGMOD Int. Symp. on Management of Data, Boston, 1979 [2]

V [1] byl navržen algoritmus pro výpočet maximálních množin schématu. Algoritmus je založen na tom, že testování, zda $Y \in \max(X, A)$ má složitost O (|X|*||F||), kde ||F|| je celkový počet atributů v závislostech F (jeden atribut může být v ||F|| obsažen vícekrát), |X| je počet atributů množiny X.[1]

Algoritmus 1 – algoritmus pro výpočet maximálních množin:

```
pro všechna A ∈ R do
    max (A): = { R \ {A}};
od
G = ∅;
pro všechna Y→Y' ∈ F do
    G:=G ∪ {Y→Y'};
    pro všechna A ∈ R do
        newmax(A):= max (A);
        pro všechna W ∈ max(A) do
            pokud neplatí, že W ∈ max (R,A) pak
                newmax(A) := newmax (A) \ {W};
                pro všechna B ∈ Y do
                    pro všechna Z ∈ max(B) do
                        if W ∩ Z ∈ max (R, A) pak
                        newmax(A) := newmax(A) ∪ {W∩Z};
                        fi;
                    od;
                od;
            fi;
        od;
    od;
    pro všechna A ∈ R do
        max(A) := newmax(A);
    od;
od;
```

Pro výpočet maximální množiny jednoho atributu však ve svém programu používám ještě jiný algoritmus.

Nejprve si definujme lhs (A) a cmax(A). Nechť R je relační schéma, F je množina závislostí schématu R, A je atribut z R. *lhs(A)* je minimální množina atributů $X \subseteq R$, pro

kterou platí, že $X \rightarrow A$ je odvoditelná z F. *cmax* je doplněk maximální množiny. $cmax(A) = \{R \setminus W \mid W \in max(A)\}$.

Množina cmax(A) je hypergrafem.[1] *Hypergraf* ́H je definován jako množina podmnožin R, kde pro všechna X, Y $\in$ ́H platí, že X, Y $\neq \emptyset$ a pokud $X \subseteq Y \Rightarrow X=Y$. Prvky ́H se nazývají vrcholy hypergrafu.

Nechť ́H je hypergraf na množině R. Pak *transversála* T hypergrafu ́H je podmnožina R, protínající všechny vrcholy ́H tj. pro všechny E$\in$ ́H platí, že $T \cap E \neq \emptyset$.

Minimální transversála hypergrafu ́H je trasnversála T, pro kterou platí, že žádná její podmnožina transversálou není. Množina minimálních transversál hypergrafu ́H se značí Tr(́H).

Protože platí, že cmax(A) = Tr(lhs(A))[1] je možné vypočítat cmax odvozením z lhs(A).

Pro výpočet množiny Tr(́H) je použit následující algoritmus:

Algoritmus 2 - výpočet množiny minimálních transversál pro jednoduchý hypergraf [1]

Tr = $\{\emptyset\}$;

pro všechna E $\in$ ́H do

Tr := $\{X \cup \{e\} \mid X \in Tr$ a $e \in E\}$;

Tr := $\{X \in TR \mid$ pokud neexistuje žádná množina $Y \in Tr$ taková, že $Y \cup X\}$;

od;

Abych ve své práci demonstrovala význam a použití maximálních množin, jsou naimplementovány i následující dva algoritmy – algoritmus testování zda je atribut klíčový na základě maximálních množin a určení toho, zda je schéma v BCNF.

Algoritmus 3 – testování klíčového atributu [1]

pokud pro nějaké W $\in$ max(R,A) platí, že $(WA)^+ = R$

pak je A klíčový atribut;

Algoritmus 4 – testování, zda je schéma v BCNF [1]

pro všechna W $\in$ MAX(X) do

pokud je W redundantní, tj. pro nějaké $V \subseteq W$ platí, že $V \rightarrow W$

schéma není v BCNF;

od;

3 Uživatelská dokumentace

Tato část se zabývá popisem programu "Databázové algoritmy", a jeho použitím z hlediska uživatelského. V první části je popsána instalace programu a parametry nutné pro jeho běh. V druhé části je dokumentace rozdělena podle jednotlivých oken aplikace s popisem nabídek a funkcí, které jsou uživateli k dispozici.

V jednom spuštěném programu může uživatel pracovat pouze s jedním hlavním relačním schématem. Je však možno spustit v jeden okamžik více oken aplikace.

Protože se nepředpokládá, že by bylo možno v tuto chvíli nějak využít výhod z toho plynoucích, není program síťový. Pokud by v budoucnu vznikla nutnost např. zadávání dat a řešení algoritmů prostřednictvím klient – server aplikace, je možno program rozšířit a doprogramovat příslušné unity.

Data, závislosti a atributy schématu, si uživatel může buď zadat sám při běhu programu nebo je může načíst ze souboru. V průběhu programu může uživatel kdykoli data hlavního schématu uložit.

Uživatel je omezen zadáním 15 atributů. Závislostí může být až 30. V případě závislostí je tu však omezení týkající se levé strany závislosti. Ta může obsahovat pouze 5 atributů. Pravá strana může při zadání obsahovat maximálně 3 atributy. Pokud by chtěl uživatel zadat na pravé straně atributů více, musí si závislost rozdělit na dvě a zadávat závislosti postupně.

Omezení jsou dána tím, že se předpokládá, že uživatel nejprve použije ER návrh a nebude proto program „Databázové algoritmy" aplikovat na příliš mnoho dat.

3.1 Instalace systému

Systém je vytvořen a odladěn na operačním systému Windows XP.

K běhu programu je zapotřebí dvou knihoven, borlndmm.dll a cc3250mt.dll. Tyto knihovny jsou také na přiloženém CD a měly by být uloženy ve stejném adresáři jako .exe soubor. Programu není třeba instalovat, stačí spustit exe soubor.

Vycházela jsem z předpokladu, že program bude nejčastěji využíván při studiu databázových algoritmů. Je dále vhodný i pro přednášejícího či cvičícího pro přípravu úloh.

Uživateli jsou k dispozici také vzorové soubory s daty v podadresáři \Soubory.

3.2 Okna aplikace

3.2.1 Hlavní okno aplikace

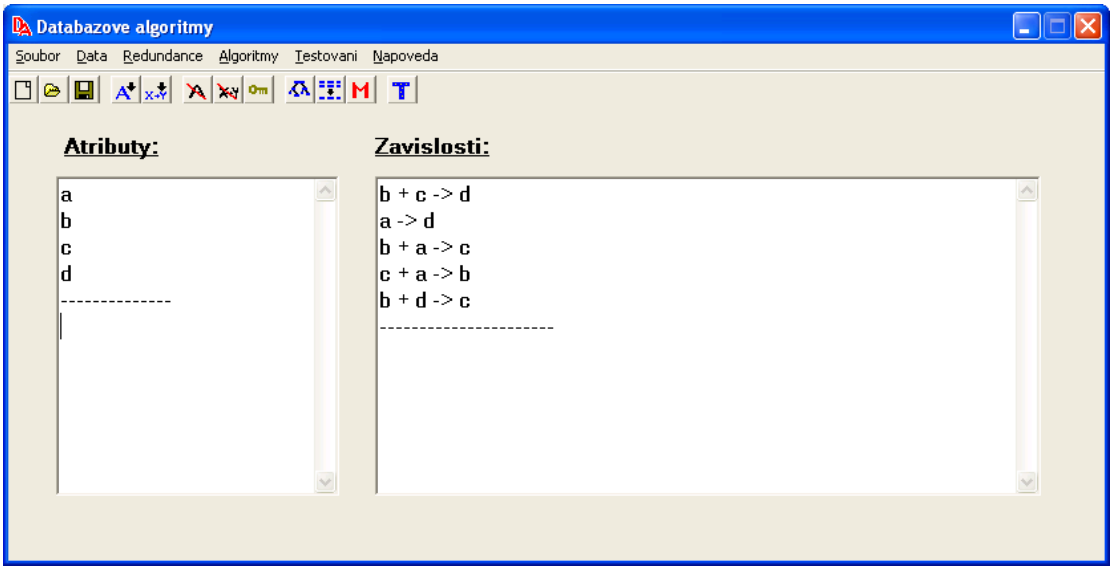
Když uživatel spustí aplikaci, otevře se mu okno „Databázové algoritmy“. Zavřením tohoto okna se pak aplikace ukončuje.

Pro rychlejší orientaci má uživatel k dispozici přehledné menu a toolbar. Volby dostupné z hlavního menu jsou:

- Soubor- Nový, Otevři, Ulož, Konec
- Data – Editovat atributy, Editovat závislosti
- Redundance – Redundantní atributy, Redundantní závislosti
- Algoritmy – Hledání klíčů, Dekompozice, Syntéza, Maximální množiny
- Testování – Testovací modul, Uzávěr, Klíče, Maximální množiny, Redundantní atributy, Redundantní závislosti, Dekompozice, Syntéza
- Nápověda – O programu

Hlavní okno aplikace obsahuje dva Memoboxy – jeden pro zobrazení atributů schématu, druhý pro zobrazení závislostí schématu. Pole se automaticky aktualizují pokud uživatel otevře či zavře soubor, smaže či přidá závislost nebo edituje atributy.

Všechny dále popsané formuláře, bod 3.2.2 – 3.2.11, jsou dostupné pouze z hlavního okna aplikace.



Obrázek 1 - Hlavní okno aplikace

3.2.2 Zadávání atributů

Jak již bylo uvedeno, může uživatel buď načíst atributy ze souboru nebo je může zadat přímo v aplikaci.

Okno pro zadávání atributů nabízí možnost zadat až 15 atributů. Uživatel zapíše nový atribut do editačního okna v horní části obrazovky. Po stisknutí tlačítka „*Přidat*“ se atribut objeví v jednom z editačních polí ve spodní části formuláře.

‘Aktivní‘ atribut je označen žlutě. Chce-li uživatel nějaký atribut měnit, stačí na něj dvakrát kliknout a atribut se opět načte do horního editačního okna.

Stejně tak je možno využít tlačítek „*Změnit*“ a „*Smazat*“ pro editaci či vymazání již zadaného atributu. Uzavře-li uživatel obrazovku jinak, než tlačítkem „*OK*“, všechny provedené změny jsou ztraceny.

Délka atributu může být až 20 znaků. Program nijak nelimituje data na vstupu – např. neupravuje mezery delší než jeden znak apod. Pouze se ověřuje, zda uživatel přidávaný atribut již nezadal.

Některá okna, např. dekompozice a syntéza, jsou však navržena tak, že se předpokládá, že uživatel bude zadávat atributy ve formě jednoho znaku. V případě delších názvů se některé výstupy mohou jevit jako nepřehledné.

3.2.3 Zadávání závislostí

Závislosti, stejně jako atributy schématu, může uživatel buď načíst ze souboru nebo si je manuálně zadat.

V horní části obrazovky, „Databázové algoritmy – Zadávání závislostí“ vidí uživatel comboboxy pro zadání levé a pravé strany závislosti.

Hodnoty do comboboxů jsou načteny podle již zadaných atributů schématu. Uživatel si může pouze vybírat z nabízených možností.Program kontroluje, zda přidávaná závislost není stejná jako závislost ve schématu již obsažená a zda levá či pravá strana závislosti není prázdná. Závislost se zobrazí ve spodní části okna po stisku tlačítka „*Přidat*“.

Jak již bylo uvedeno, uživatel má možnost zadat závislost, jejíž levá strana má maximálně pět atributů, pravá strana pak atributy tři. Z důvodu jednodušší implementace je pak závislost transformována na jednotlivé závislosti tak, aby na pravé straně byl vždy atribut právě jeden.

Uživatel nemá možnost již jednou zadanou závislost editovat, má pouze možnost ji smazat a zadat znovu. Smazání závislosti uživatel provede tak, že na ni nejprve dvakrát

klikne, tím se závislost označí žlutě – stane se aktivní. Poté stačí stisknou tlačítko „*Smazat*“. Program se ještě ubezpečí, zda skutečně chce uživatel závislost smazat.

Data se uloží po stisku tlačítka „*OK*“. Ukončí-li uživatel okno formuláře jinak, o zadané změny přijde.

3.2.4 Datové soubory

V hlavním menu má uživatel možnost otevřít nový, prázdný soubor. Může také otevřít již dříve vytvořený soubor či si uložit data hlavního souboru, se kterými právě pracuje.

Do souboru se uloží atributy a závislosti hlavního schématu. Nejprve jsou uloženy atributy oddělené středníkem. Poté následuje dvojtečka a závislosti ve formě: atributy levé strany ‘-‘ a pravá strana. Atributy závislostí jsou uloženy nikoli ve formě znakového řetězce, ale číselného indexu do pole atributů.

Např. relační schéma R= { A, B, C}; F = {A + B → C; C → A} bude uloženo jako

A;B;C;:0 1 -2;2 -0

3.2.5 Redundantní atributy

Okno „Databázové algoritmy - Redundantní atributy“ zobrazuje uživateli atribut, který je redundantní a také závislost, ve které k tomu dochází. Uživatel může stiskem příslušného tlačítka „*Smazat*“ zvolit, že chce daný atribut ze schématu vypustit. Při stisknutí tlačítka se systém ubezpečí o tom, zda uživatel skutečně chce atribut ze závislosti vymazat. Tato akce je nevratná. Uživateli ale stále zůstává možnost si data ručně zadat zpět.

Program využívá následující algoritmus pro hledání redundantních atributů: nechť závislost $(A + B \rightarrow C) \in F^+$. Při testování redundance atributu A program hledá zda $(B \rightarrow C) \in F^+$.

Po smazání některého redundantního atributu může dojít k tomu, že jiný zobrazený atribut již redundantní není. Program proto po každém smazání redundantního atributu ověřuje platnost ostatních zobrazených redundancí. Pokud některý atribut přestal být redundantní, uživatel je o tom uvědomen a atribut se barevně označí.

3.2.6 Redundantní závislosti

Formulářové okno zobrazí uživateli přehled závislostí, které jsou v daném schématu redundantní. Uživatel může stiskem tlačítka „*Smazat*“ danou závislost vypustit ze schématu. Tlačítko „Smazat“ je zobrazeno vedle každé závislosti.

Stejně jako u redundantních atributů se systém uživatele zeptá, zda chce skutečně závislost vymazat. Pokud uživatel potvrdí, a chtěl-li by zadat danou závislost zpět, musí ji opět přidat přes formulář zadávání závislostí.

Program využívá následující algoritmus. Nechť F je množina funkčních závislostí relačního schématu R, závislost D1: $(A + B \rightarrow C) \in F$. Pokud $C \in AB^+$ i bez použití závislosti D1, je závislost D1 redundantní.

Stejně jako v případě redundantních atributů se zde provádí ověření a aktualizace redundance zobrazených dat po smazání některé redundantní závislosti.

3.2.7 Klíče

Formulář pro hledání klíčů schématu je rozdělen na dvě části a to z důvodu rozdílných algoritmů implementovaných v pozadí hledání prvního a dalších klíčů.

Nejprve musí uživatel zadat, že chce najít první klíč schématu – tlačítko „*Zobraz první klíč*“.

Program zde hledá první klíč schématu následujícím algoritmem. Nechť F je množina funkčních závislostí relačního schématu R, A je množina atributů schématu R.

```
AtributyKlice := A
pro všechny závislosti ∈ F do
        if (LeváStrana závislosti je obsažena v poli AtributyKlice)
                AtributyKlice [PraváStrana] = false;
        fi;
od;
pro všechny atributy X v AtributyKlice do
        if X je redundantní
                AtributyKlice[PraváStrana] = false;
        fi;
od;
```

Uživateli se zobrazují jednotlivé klíče v read-only editačních polích.

Hledání dalších klíčů schématu je založeno na algoritmu Lucchesi, Osborn. Uživateli se zobrazí další klíče po stisku tlačítka „*Zobraz další klíče*“.

3.2.8 Dekompozice

Formulář dekompozice je rozdělen do dvou hlavních částí. Vrchní část zobrazuje strom dekompozice, ve spodní části je pak vidět detail jednotlivého vybraného schématu.

Při zobrazení formuláře má uživatel vyplněn pouze vrchol stromu dekompozice. Do editačního okna jsou načteny atributy hlavního schématu.

Schémata v dekompozičním stromě jsou barevně rozlišena podle normálních forem. Červená barva označuje schéma v první či druhé normální formě, žluté schéma je ve třetí normální formě, schéma v BCNF je obarveno šedě.

Dvojitým kliknutím na konkrétní editační pole s atributy schématu ve stromu dekompozice se ve spodní části zobrazí atributy schématu, závislosti a klíče.

Závislosti jsou opět barevně rozlišeny. Žluté závislosti porušují třetí normální formu, zelené závislosti porušují BCNF. Ostatní závislosti jsou šedé.

Uživatel si může vybrat, podle které závislosti chce schéma dekomponovat. Aktivní jsou závislosti obarvené žlutě a zeleně, podle šedých závislostí schéma rozkládat nelze. Po dvojitém kliknutí na závislost rozloží systém schéma na dvě. Schémata jsou dekomponována podle uzávěru levé strany zvolené závislosti. Závislost, kterou uživatel zvolil k dekompozici, se zobrazí i ve stromu dekompozice.

Uživatel má také možnost zkusit si schéma rozložit znovu, podle jiné závislosti. Systém se v tomto případě ujistí, zda uživatel skutečně chce akci provést a schéma rozloží.

Po dokončení dekompozice, nebo alespoň po dekomponování na minimálně dvě schémata, může uživatel zvolit tlačítko „*Nepokryté závislosti*“. Poté se do schématu, do okna pod strom dekompozice, přidají nepokryté závislosti ze stromu dekompozice.

Program prochází jednotlivé závislosti původního schématu. U každé závislosti se porovná uzávěr levé strany původního schématu s uzávěrem vypočteným použitím pouze závislostí v nově vzniklých schématech. Je-li některý atribut obsažen v uzávěru původního schématu, ale nikoli v nově vzniklých schématech, je tato závislost přidána jako nové schéma. Nové schéma obsahuje všechny atributy, o kterých se zjistilo, že nejsou pokryty. Poté přidám všechny další relevantní závislosti – projdu znovu všechny závislosti původního schématu a pokud nové schéma obsahuje všechny prvky závislosti, přidám také tuto závislost.

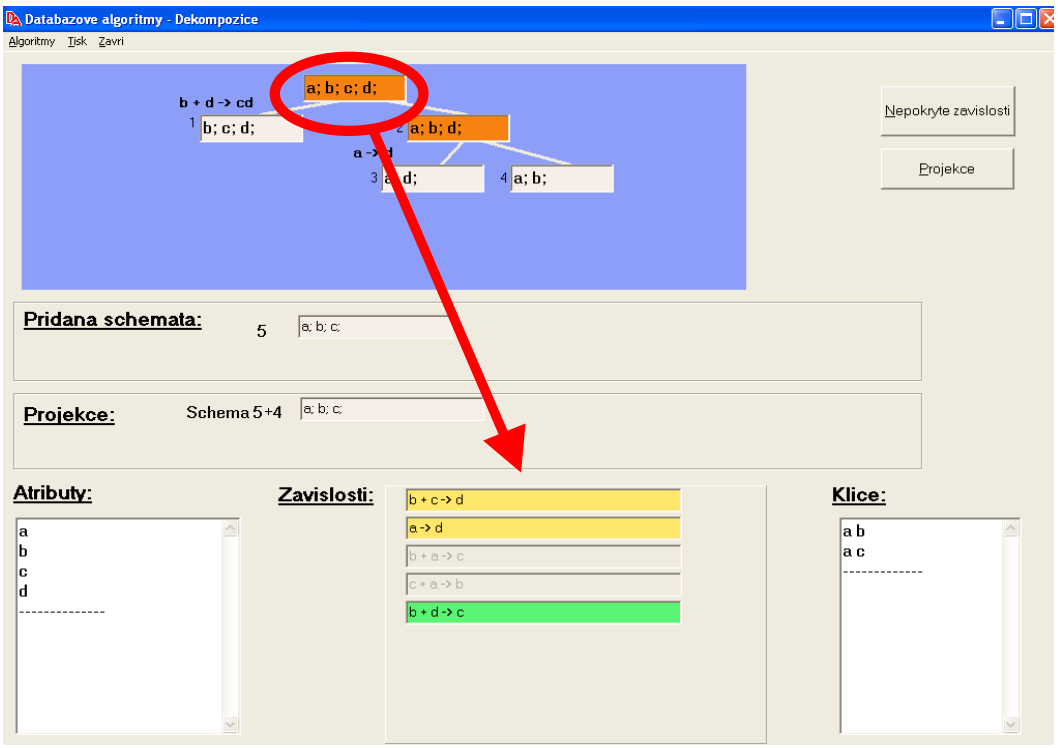
Po přidání nepokrytých závislostí může uživatel zvolit i možnost „*Projekce*“. Projekce projde všechna konečná schémata – schémata na konci nejnižších stupních stromu dekompozice. Pokud by některé bylo projekcí jiného, zobrazí informaci o tom v části pod stromem dekompozice. Uživatel vidí číslem označená schémata, která bylo

sloučena a schéma nové. Schéma je opět barvou označeno podle toho, v jaké normální formě je.

Pro provedení projekce prochází program postupně jednotlivá schémata a zkoumá, zda atributy jiného schématu nejsou v testovaném obsaženy. Pokud ano, schémata sloučí.

V případě přidávání nepokrytých závislostí či projekce program přidá pouze schéma, které je alespoň ve třetí normální formě.

Program předpokládá, že se bude postupovat při dekompozici tak, že se nejprve rozloží schéma ve stromu dekompozice, poté se přidají nepokryté závislosti a nakonec, podle volby, se sloučí schémata, která by byla projekcí. Uživatel přesto má možnost zobrazovat si nepokryté závislosti po každém kroku, či zvolit projekci bez přidání nepokrytých závislostí. Aby však byla zachována určitá kontinuita a v jednotlivých oknech formuláře byla zobrazována data aktuální, logicky se na sebe navazující, vrátí-li se uživatel o krok zpět, výše, data logicky následující jsou smazána. Např. bude-li uživatel dále dekomponovat schéma poté, co si již zobrazil nepokryté závislosti, schéma se dekomponuje, schémata dříve vytvořená přidáním nepokrytých závislostí se však ztratí.



Obrázek 2 - Okno "Databázové algoritmy - Dekompozice"

Uzavřením formuláře se ztratí provedené úpravy. Program si neukládá rozložená schémata.

Uživatel má však možnost si data vytisknout či uložit. Tato volba je dostupná z menu – „*Tisk*“. K dispozici jsou dvě možnosti – buď pracovat pouze s výslednými schématy nebo si nechat zobrazit všechna schémata tak, jak vznikala v průběhu procesu dekompozice. Po volbě jedné z variant se otevře formulářové okno s daty. Okno „Databázové algoritmy – Tiskový formulář“ si uživatel může buď vytisknout nebo vyexportovat do .txt souboru. Datový výstup je zvolen velice jednoduše. Pokud bude chtít uživatel data prezentovat v lepší grafické úpravě, může využít textový editor. Lepší grafický výstup, možnost uživatele si výstup upravit přímo v programu Databázové algoritmy, by mohl být samozřejmě možným rozšířením aplikace.

3.2.9 Syntéza

Grafické provedení okna „Databázové algoritmy – Syntéza“ je podobné oknu „Dekompozice“.

V horní části obrazovky vidí uživatel přehled schémat, tak jak vznikají v průběhu syntézy. Dvojitým kliknutím na konkrétní schéma se pak, stejně jako v případě dekompozice, do spodní části obrazovky načtou a zobrazí jednotlivé atributy, závislosti a klíče schématu. Schémata jsou opět barevně odlišena podle toho, v jaké normální formě schéma je. Červená jsou schémata v první nebo druhé normální formě, žlutě označená schémata jsou ve třetí normální formě, šedá schémata jsou v BCNF. I barevné označení závislostí je stejné jako u dekompozice.

Při zobrazení okna je vyplněno nejvyšší editační pole, kde jsou zobrazeny atributy původního schématu.

Rozložení podle levých stran závislostí provede uživatel stiskem tlačítka „*Rozložení podle stejných levých stran*“. Schémata, vzniklá rozložením, se zobrazí v groupboxu „*Stejné levé strany*“.

Program postupně prochází závislosti původního schématu. Pro každou závislost se založí nové schéma. Po vytvoření nového schématu se projdou zbylé závislosti a pokud mají stejnou levou stranu jako testovaná závislost, přidají se do schématu. Po vytvoření všech schémat, jejich atributů a závislostí, se ještě řeší otázka klíčů. Klíče každého schématu jsou vytvořeny Lucchesi a Osborn algoritmem popsaným výše. Kvůli zajištění bezztrátovosti se eventuelně jako další tabulka přidá také univerzální klíč, není-li obsažen.

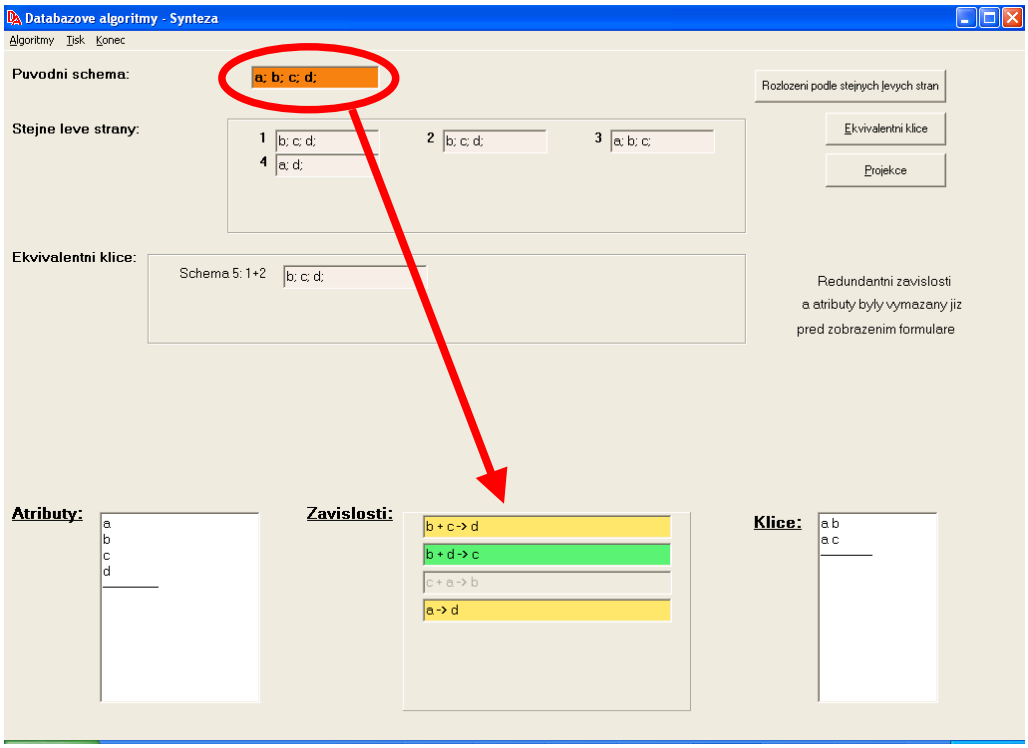
Dále má uživatel možnost sloučit schémata, která mají ekvivalentní klíče. K vyvolání funkce slouží tlačítko „*Ekvivalentní klíče*“. Pod skupinou schémat rozložených podle

levých stran se zobrazí schémata vzniklá sloučením schémat s ekvivalentními klíči. Uživatel vidí číslem označeno, z jakých dvou schémat vzniklo nové.

Při generování schémat s ekvivalentními klíči prochází program jednotlivá schémata a testuje, zda pro klíč K1 schématu S1 a klíč K2 schématu S2 platí, že K1→K2 a K2→K1. Pokud ano, schémata jsou sloučena.

Tlačítko „*Projekce*“ umožňuje uživateli sloučit schémata, která jsou projekcí. Opět se zobrazí nově vzniklé schéma, číslem je označeno, z jakých schémat vzniklo. Implementovaný algoritmus je stejný jako v případě dekompozice.

U syntézy musí uživatel nejprve rozložit schéma podle levých stran. Dále se předpokládá, že uživatel vygeneruje schémata s ekvivalentními klíči a nakonec zvolí projekci. Má i možnost nejprve určit projekci a pak se vrátit k ekvivalentním klíčům. V tomto případě jsou však z důvodu konzistence dat předchozí schémata vytvořená projekcí smazána.



Obrázek 3 - Okno "Databázové algoritmy - Syntéza"

Stejně jako v případě dekompozice se po uzavření formuláře ztratí data vzniklá provedením dekompozice. Uživatel má však možnost si data vytisknout či uložit volbou „*Tisk*“ z menu. Postup je stejný jako u dekompozice.

3.2.10 Maximální množiny

V okně maximálních množin je uživateli umožněno nejen vypočítat maximální množinu k jednomu konkrétnímu vybranému atributu, ale také najít všechny množiny schématu. Důvodem pro implementaci těchto dvou rozdílných funkčností je využití dvou různých algoritmů. Pokud hledáme maximální množinu jednoho konkrétního atributu je využit algoritmus 2 popsaný v kapitole 2.2.1. Pro hledání všech maximálních množin schématu je z důvodu větší efektivity výpočtu naimplementovaný algoritmus 1 popsaný v kapitole 2.2.1.

Pokud uživatel chce zobrazit maximální množinu pouze jednoho atributu, zvolí si jej v comboboxu v horní polovině obrazovky. Stiskem tlačítka *„Maximální množina (A)“* se pak zobrazí jak maximální množina, max (A), tak i lhs(A) a Tr(lhs(A)).Uživatel tak může lépe sledovat průběh algoritmu. Chce-li zvolit další atribut, stačí pouze vybrat jiný prvek v comboboxu a stisknout „maximální množina“.

Naimplementovaný je také algoritmus 3 popsaný v kapitole 2.2.1. - test, zda je atribut A klíčový atribut na základě výpočtu maximálních množin. Uživatel se výsledek algoritmu dozví po stisku tlačítka „ *Je A klíčový atribut?*“. Toto tlačítko se zviditelní až po výpočtu maximálních množin nějakého atributu.

Všechny množiny schématu je možno získat stiskem tlačítka „*Maximální množiny*“. V dolní části panelu se pak zobrazí všechny maximální množiny, vypsané postupně podle jednotlivých atributů schématu.

Na základě vypočtených maximálních množin se může uživatel i dozvědět, zda je schéma v BCNF: Implementovaný algoritmus 3 popsaný v kapitole 2.2.1. se spustí tlačítkem „*Je schéma v BCNF*?“, které se zviditelní po vygenerování maximálních množin.

Uzavřením formuláře se vypočtená data ztratí. Uživatel nemá v tomto případě možnost si data uložit. Uložení, či ve vhodné formě vytištění, dat by opět mohlo být rozšířením aplikace.

3.2.11 Testovací modul

Testovací část slouží pro uživatele, kteří se chtějí učit databázové algoritmy. Data zadání – atributy a závislosti se přebírají z původního schématu hlavní aplikace. Pokud uživatel chce, může jak již bylo zmíněno výše, využít předdefinovaných souborů v adresáři \Soubory.

Konkrétní atributy, např. pro výpočet uzávěru, si uživatel volí sám. Program ho v nějaké formě nechá zadat odpověď. Poté jeho odpověď vyhodnotí a řekne, zda byla správná či špatná. Pokud špatná, zobrazí se uživateli pro porovnání i správné řešení.

Detailnější a lépe graficky zpracovaný testovací modul by mohl být navržen jako rozšíření aplikace. V případě programu Databázové algoritmy bylo spíše snahou nastínit a implementovat různé možnosti testování uživatele. Vycházím z toho, že tento modul budou používat spíše studenti, kteří se s problematikou algoritmů logického návrhu schématu relační databáze teprve seznamují. Hodně algoritmů, hlavně v případě dekompozice a syntézy, si může uživatel zkoušet na papír sám, během toho, jak postupně prochází jednotlivými kroky v oknech Dekompozice a Syntéza. Před tím, než se správnou odpověď od programu, stisknutím příslušného tlačítka, dozví.

Hlavní okno testovací části je dostupné z okna „Databázové algoritmy“. Uživatel má k dispozici tlačítko na toolbaru nebo volbu v menu „*Testování*“. Z menu může uživatel zobrazit přímo i konkrétní testovací okno.

V okně formuláře „Databázové algoritmy – Testovací modul“ jsou velká tlačítka s následujícími odkazy: uzávěr, redundantní atributy, redundantní závislosti, klíče, maximální množiny, dekompozice.

Nyní popíši jednotlivé podformuláře:

3.2.11.1 Uzávěr - testování

Formulář má tři hlavní části. V levé části si uživatel zaškrtne, uzávěr jaké neprázdné množiny atributů by chtěl počítat. Připomínám, že atributy a závislosti se načítají podle hlavního schématu, které má uživatel otevřené.

Po stisknutí tlačítka „*OK*“ se v prostředním panelu objeví checkboxy pro jednotlivé atributy. Uživatel zaškrtne, jaké atributy patří do uzávěru množiny, kterou si zvolil v úvodu.

Tlačítkem „*Správná odpověď*“ se zobrazí třetí, pravý panel. V panelu jsou zaškrtnuty checkboxy u těch atributů, které patří do uzávěru. Pokud uživatel zadal všechna data do prostředního panelu správně, program ho pochválí. Udělal-li chybu, jsou červeně zvýrazněny atributy, které byly zadány chybně.

Volbou „*Nové zadání*“ se zviditelní opět pouze levý panel s možností zvolit si další atributy pro výpočet uzávěru.

Stiskne-li uživatel jednou tlačítko „*OK*“ a chtěl-li by zvolit jiné zadání, musí přesto projít oknem „*Správná odpověď*“ a až pak zvolit „*Nové zadání*“.

Pokud by uživatel chtěl testovat uzávěr na jiných datech, musí formulářové okno zavřít a v hlavním okně aplikace otevřít jiný soubor.

3.2.11.2 Maximální množiny - testování

Formulář pro testování maximálních množin vypadá podobně jako formulář pro testování uzávěru.

Opět je okno rozděleno na tři části. V levé jsou radiobuttony s jednotlivými atributy schématu. Zde si uživatel zvolí maximální množinu jakého atributu chce počítat. Po stisku tlačítka „*OK*“ se zviditelní prostřední část okna s checkboxy – opět jeden checkbox pro jeden atribut.

Uživatel zaškrtne ten atribut, který je obsažen v alespoň jedné maximální množině zvoleného atributu. Po stisku tlačítka „*Správná odpověď*“ se uživatelova odpověď vyhodnotí a opět následuje pochvala nebo upozornění, že v odpovědi jsou chyby. Správná odpověď se zobrazí v pravé části okna.

Také se zviditelní tlačítko „*Nové zadání*“, kterým může uživatel pokračovat a zvolit si jiný atribut.

Testování maximálních množin touto formou je pouze jednou z možností, které se nabízejí. Další možností by například bylo testovat maximální množiny podobně jako klíče, viz níže.

3.2.11.3 Klíče - testování

V okně „Testovací modul – Klíče“ vidí uživatel v pravé části závislosti schématu. V levé části jsou potom tři checkboxy s nabídkami klíče. Správná je alespoň jedna odpověď.

Uživatel zaškrtne některou z možností a stiskne tlačítko „*Správná odpověď*“. Odpověď se vyhodnotí a také se ve střední části formuláře zobrazí správné řešení.

Možnosti, ze kterých uživatel vybírá jsou generovány následovně: nejprve se náhodně vybere jeden z klíčů schématu, který se vloží na náhodnou ze tří možných pozic v odpovědi. Ostatní dvě možnosti jsou generovány náhodně. Ještě se však otestuje, zda byl či nebyl náhodně vygenerován klíč.

3.2.11.4 Redundantní atributy - testování

Okno “Testovací modul – Redundantní atributy“ zobrazuje jednotlivé atributy ve formě tlačítek. Stiskem tlačítka se přepne jeho zobrazení přepne z modu ‘*OK*’ – zelené zaškrtnutí na mod ‘*No*’ – červený křížek a obráceně. Uživatel tímto způsobem, přepnutím na mod ‘*No*’, označí atributy, o kterých se domnívá, že jsou redundantní.

Po zvolení tlačítka „*Správná odpověď*“ se jeho odpověď vyhodnotí. Pokud odpověď není správná, program o tom uvědomí uživatele a ujistí se, zda chce uživatel pokračovat a dozvědět se správné řešení nebo zda chce ještě zkusit svou odpověď opravit.

Správná odpověď se zobrazí v pravé části okna, kde jsou zapsány závislosti. Uživatel si tak může porovnat svou odpověď se správným řešením.

3.2.11.5 Redundantní závislosti - testování

Testování redundantních závislostí je řešeno podobně jako testování redundantních atributů. Uživateli jsou zobrazeny závislosti schématu a on zaškrtnutím příslušného checkboxu volí, zda je závislost redundantní či nikoli.

Po stisku tlačítka „*Správná odpověď*“ program opět odpověď vyhodnotí. V případě, že jsou v uživatelově řešení chyby, uvědomí ho o tom a dá mu možnost buď se dozvědět odpověd správnou či zkusit ještě vstup opravit.

Tlačítkem „*Zkusit znovu*“ zmizí správná odpověď a uživatel může zkusit odpovědět znovu. Jak již bylo zmíněno výše, aby se změnila data, atributy a závislosti, je potřeba zavřít okno a otevřít jiný soubor v hlavním okně aplikace.

3.2.11.6 Dekompozice – testování

V sekci dekompozice je pouze nastaveno testování závislostí – zda porušují některou normální formu. Rozložení jednotlivých schémat podle zvolené závislosti si může uživatel zkoušet v okně „Databázové algoritmy – Dekompozice“, když postupně vytváří strom dekompozice.

Z tohoto důvodu není naimplementováno ani testování syntézy. Doporučuji uživateli zkoušet si zachytit na papír jednotlivé kroky předtím, než zvolí příslušné tlačítko.

Formulář testování dekompozice je stejně jako v předchozím případech rozdělen do několika částí. V levé části jsou zobrazeny závislosti schématu. Pokud se uživatel domnívá, že závislost porušuje třetí normální formu nebo BCNF, zaškrtne příslušný checkbox. Správné řešení se opět dozví stiskem tlačítka „*Správná odpověď*“. Pokud jsou v jeho řešení chyby, je mu dána možnost si řešení ještě opravit. Po zobrazení správné odpovědi se zviditelní tlačítko „*Zkusit znovu*“. Po jeho stisku zmizí správné řešení a uživatel může zkusit své znalosti znovu.

4 Programátorská dokumentace

Hned na úvod bych chtěla poznamenat, že program Databázové algoritmy má sloužit hlavně jako výukový program demonstrující základní algoritmy logického návrhu relační databáze. Cílem této práce není jejich co nejefektivnější implementace. Program si pouze klade za cíl na datech v rozsahu 15 atributů a 30 závislostí poskytovat uživateli odezvu v rozumném, uživatelsky příjemném, čase.

Programátorská dokumentace je napsána tak, aby umožnila jiným programátorům v případě potřeby eventuelní doplnění aplikace o další moduly, utility, funkčnosti.

4.1 Uložení dat

Jak již bylo zmíněno v úvodu, uložení dat je zvoleno tak, aby dovolovalo snadnou a dostatečně efektivní manipulaci s daty.

Atributy hlavního schématu jsou uloženy v poli *String ABC_ARRAY[ABC_SIZE]* unity Main_Form.h. Konstanta ABC_SIZE je předdefinována na 15. Pokud s atributy pracují v průběhu výpočtů algoritmů jiné unity, zpravidla se používá pole ve formě int ABC_ARRAY[ABC_SIZE]. Pouze pro zobrazení výstupů uživateli se používají data, která se načtou z pole Stringů. Ostatní unity se 'název' atributu dozví voláním funkce unity Main_Form.h - *String Return_Attribute(int i).*

Závislosti hlavního schématu jsou uloženy v seznamu. Prvním prvkem seznamu je proměnná *dependency *First_Dependency* opět unity Main_Form.h. Pokud chce se závislostmi pracovat jiná unita, zjistí si je pomocí funkce *dependency *Return_First_Dependency(void)*. Datová struktura dependency má následující tvar:

```
struct dependency {
        int LS[ABC_SIZE];
        dependency *Next;
        PS_Struct *PS;
        int Tested;
                };
```

LS slouží k uložení levé strany závislosti. Atributy levé strany jsou uloženy za sebou, od pozice [0] ve formě indexu do pole ABC_ARRAY. Hodnota -1 znamená, že již žádný další atribut není na levé straně obsažen.

Next ukazatel na další závislost.

PS uložení pravé strany závislosti.

```
struct PS_Struct {
    int Value;
    int Closure;
  PS_Struct *Next;
    };
```

Value atribut pravé strany závislosti. Stejně jako v případě levé strany je i zde atribut uložen jako index do pole ABC_ARRAY.

Closure, Next při úvodním navrhování datových struktur se uvažovalo o využití uchování dat např. z výpočtu uzávěru či o implementaci závislostí, které by měly více než jeden atribut na pravé straně. Z tohoto důvodu má struktura PS také parametry Next – ukazatel na další atribut na pravé straně a Closure – pomocná proměnná, uchovávající např. informace o tom, zda-li je atribut výsledkem výpočtu uzávěru, či je přímo zadán uživatelem. Vzhledem k tomu, že množina naimplementovaných algoritmů není jistě úplná, a existuje zde možnost jistého využití v budoucnu, ponechávám datovou representaci pravé strany závislosti v této podobě. V mnou použitým algoritmech má však pravá strana vždy podobu Next = NULL.

Tested pomocná proměnná, která se používá v průběhu např. generování uzávěru

Další údaje o použitých datových typech jsou uvedeny v následujících podkapitolách, při popisu jednotlivých unit aplikace.

4.2 Unity

Program se skládá z několika unit. Jednotlivé unity jsou popsány níže:

4.2.1 ABC_Input, Dep_Input

Tato unita obsahuje data pro editaci atributů uživatelem. Unita se v průběhu programu volá pouze z Main_Form.cpp.

Před zobrazením formuláře se načtou data právě otevřeného souboru. Inicializace okna před zobrazením se provede voláním metody formuláře *void Fill_In_Data(void)*. Atributy jsou načteny z hlavního formuláře pomocí výše popsané funkce *String*

Return_Attribute(int i). Unita ABC_Input si z hlavního formuláře načítá i závislosti. A to z důvodu kontroly, zda uživatel nemaže atribut obsažený v některé závislosti.

Podobný systém inicializace formuláře se aplikuje na všechny formuláře, které pracují s daty – atributy a závislostmi. Před zobrazením je potřeba volat metodu formuláře *void Fill_In_Data(void)*, která příslušně nastaví data a zobrazení komponent.

V průběhu toho, jak uživatel data ve formuláři „Databázové algoritmy – Zadávání atributů“ mění – přidává, maže – se žádné změny neukládají. Až když uživatel ukončí editaci stiskem tlačítka „*OK*“, změny se uloží prostřednictvím volání metody implementované v unitě Main_Form *void Add_New_ABC(String ABC_ARRAY[ABC_SIZE])*.

Unita Dep_Input obsahuje formulář pro editaci závislostí uživatelem. Před zobrazením formuláře je ho třeba také inicializovat voláním metody *void Fill_In_Data(void).*

Změny provedené uživatelem se opět uloží až když uživatel ukončí editaci stiskem tlačítka „*OK*“ voláním funkce unity Main_Form - *int Add_New_Dependencies(dependency *D1)*.

4.2.2 File Open Close

Unita, která slouží k ukládání dat do souboru. Obsahuje následující funkce:

*int Close_File(dependency *First_Dependency, String ABC_ARRAY[ABC_SIZE]);*

Uloží a uzavře soubor. Soubor bude mít příponu .dbc.

int Open_File(void);

Zobrazí dialog pro otevření souboru. Název souboru je kontrolován, zda končí příponou .dbc. Pokud ano, program načítá data – atributy a závislosti schématu. Pokud není úspěšný, uživateli se zobrazí chybový výstup.

*int Save_File(dependency *First_Dependency, String ABC_ARRAY[ABC_SIZE]);*

Uloží data, soubor zůstává otevřen, uživatel může s daty dále pracovat.

void New_File(int New);

Vytvoří nový soubor s prázdnou množinou atributů a závislostí. Před uzavřením předchozího souboru se zeptá, zda chce uživatel data uložit.

4.2.3 Definice

Tato unita obsahuje uložení definovaných konstant a datových struktur.

Konstanty:

ABC_SIZE určuje maximální možný počet atributů schématu

SCHEMATES definuje maximální počet schémat, na které je možno původní schéma rozložit pomocí dekompozice, syntézy

DEP_NO konstanta určující maximální počet závislostí, které může uživatel zadat

Další konstanty využívané v algoritmech jako např. *BNCF* (=4), *NF3* (=3), *NF2* (=2), *NF1* (=1)

Struktury:

Struktury *PS_struct* a *dependency* byly již popsány v úvodu.

*struct key {int Key[ABC_SIZE]; int Tested; key *Next;};*

Klíč je uložen jako struktura, kde v prvním parametru je obsažen klíč, reprezentovaný polem 0,1. Pole je indexováno stejně jako pole ABC_ARRAY. Key[i] = 0 znamená že i-tý atribut není v klíči obsažen. Parametr Tested je opět pouze pomocná proměnná používaná při výpočtu algoritmů. Ukazatel Next ukazuje na další klíč, pokud existuje.

*struct Schema {int ABC [ABC_SIZE]; dependency *First_Dependency; key *First_Key; int Normal_Form; int Tested; };*

Schema je struktura, která se využívá při syntéze, dekompozici. Jednotlivá schémata jsou uložena v průběhu výpočtu ve formě této struktury. Atributy schématu jsou ve formě 0,1 uloženy v poli ABC. First_Dependency ukazuje na seznam závislostí schématu, First_Key na seznam klíčů. Normal_Form obsahuje informaci o tom, v jaké normální formě schéma je. Parametr Tested je využíván opět pouze pro pomocné účely.

4.2.4 Main Form

Formulář Main_Form obsahuje hlavní okno aplikace. Jednotlivé funkčnosti, otevírání a inicializace formulářů jsou uloženy jako akce: open_file, a_save_file, a_attributes, a_dependencies, a_red_attributes, a_red_dependencies, a_keys, a_decomposition, a_synthesis, a_about, a_maxset, a_test, a_clear_data.

Metody formuláře:

int File_Data_Changed_Status(bool STATUS);

Pokud uživatel změní atributy nebo závislosti, je nutno nastavit proměnnou FILE_DATA_CHANGED. Soubor se potom při otevření nového nebo prázdného souboru zeptá, zda chce uživatel data uložit.

*dependency *Return_First_Dependency(void);* - funkce vrací ukazatel na seznam závislostí schématu.

*key *Return_First_Key(void);* - funkce vrátí ukazatel na seznam s klíči schématu.

String Return_Attribute(int i); - vrací název atributu uloženého na pozici i v poli atributů

*int Set_First_Dependency(dependency *First_Dependency_S);* - nastaví závislostí schématu podle seznamu v parametru First_Dependency_S

int Set_Attributes(String ABC_ARRAY_S[ABC_SIZE]); - nastaví atributy schématu podle pole, které je předávané jako parametr

4.2.5 Closure

Tato unit obsahuje pouze jedinou metodu:

*void Closure_Computation (int (*Set)[ABC_SIZE],int Search, dependency *D1, dependency *D2);*

Metoda slouží k výpočtu uzávěru určitého atributu, skupiny atributů. V prvním parametru je pole obsahující nuly a jedničky. Pro atributy, jejichž uzávěr hledáme, je v poli 1. Na všech ostatních pozicích je 0. Parametr *Search* je použit, pokud chceme pouze určit, zda je daný prvek obsažen v uzávěru či nikoli. Pokud pak v průběhu algoritmu zjistím, že hledaný prvek je v uzávěru obsažen, algoritmus končí, nevypočítává se celý uzávěr. Parametr *D1* udává ukazatel na seznam závislostí. Parametr *D2* určuje závislost, která se nemá při výpočtu uzávěru použít. Toho se využívá např. při testování redundantních závislostí.

<u>Schéma algoritmu:</u>

```
pro všechny závislosti do
    Tested = false;
od;
Added= true; Uzaver = Set;
While (Added)
    Added = false;
    pro všechny závislosti do
        if (not Tested)
            if celá levá strana podmnožinou již vytvořeného uzávěru
                přidej pravou stranu do uzávěru;
                Tested = true; Added = true;
            fi;
        fi;
```

od;

Wend;

4.2.6 Print Form

Unita Print Form obsahuje formulář, do kterého se načtou data z dekompozice, syntézy, a uživatel má možnost si je uložit do textového souboru nebo vytisknout.

Se zobrazováním dat souvisí i unita Data_Statement_2, která umožňuje výpis závislostí nebo atributů do jednotlivých komponent formulářů.

Obsahuje tyto základní funkce:

*void Write_ABC(String ABC_ARRAY[ABC_SIZE], TMemo *Memo);*

*void Write_Dep(dependency *D1,String ABC_ARRAY[ABC_SIZE], TMemo *Memo);*
*void Write_Dep_5(dependency *D1,key *First_Key,String, ABC_ARRAY[ABC_SIZE],TEdit *Edit);*

*void Write_Key_2(key *K1, String ABC_ARRAY[ABC_SIZE], TMemo *Memo);*

4.2.7 About_Form

Unita obsahující formulářové okno „O programu“.

4.2.8 Redundance

Unita obsahuje funkce pro testování redundantních atributů a redundantních závislostí.

Funkce:

*int Redundant_Dependencies (dependency **First_Dependency, key **Key, String ABC_ARRAY[ABC_SIZE]);*

*int Redundant_Attributes (dependency **First_Dependency, key **First_Key, String ABC_ARRAY[ABC_SIZE]);*

V unitě jsou implementovány ještě 2 metody, které jsou volány při syntéze a na rozdíl od předchozích dvou jsou redundance rovnou vymazávány.

*void Redundant_Dependencies_Decomposition (dependency **First_Dependency, key **Key);*

*void Redundant_Attributes_Decomposition (dependency **First_Dependency, key **First_Key);*

<u>Redundantní závislosti – schéma algoritmu:</u>

Předávané parametry udávají: *First Dependency* – závislosti schématu, *Key* - klíče a *ABC_ARRAY* - pole atributů, které slouží pouze pro vypisování závislostí.

```
pro všechny závislosti do
    vypočti uzávěr levé strany bez použití této závislosti;
    if uzávěr obsahuje i pravou stranu závislosti
            závislost je redundantní;
    fi;
od;
```

Redundantní atributy – schéma algoritmu

Předávané parametry udávají: *First Dependency* – závislosti schématu, *Key* - klíče a *ABC_ARRAY* - pole atributů, které slouží pro vypisování závislostí.

```
pro všechny závislosti do
    pro všechny atributy levé strany do
            vypočti uzávěr levé strany bez použití testovaného atributu;
            if uzávěr obsahuje i testovaný atribut
                    atribut je redundantní;
            fi;
    od;
od;
```

4.2.9 Redundant Attributes Form, Redundant Dependency Form

Formuláře pro zobrazení redundantních atributů, závislostí. Pro zjištění redundancí jsou, ještě před zobrazením formuláře, použity metody popsané v předchozím odstavci.

Pokud se uživatel rozhodne redundance smazat, a stiskne tlačítko „*Smazat*“, data jsou vymazána. Nikoli až po zavření formuláře jako např. u editace atributů.

4.2.10 Keys, Keys_Form

Unita s algoritmy hledání klíčů.

Obsahuje dvě funkce:

*key *Find_First_Key (dependency *First_Dependency, String ABC_ARRAY[ABC_SIZE]);*

Prvním parametrem je ukazatel na seznam závislostí schématu, druhým parametrem je pole atributů. Funkce vrací ukazatel na nalezený klíč. Algoritmus hledání prvního klíče byl již popsán výše, v rámci uživatelské dokumentaci.

*void Lucchesi_Osborn(key **First_Key, dependency *First_Dependency);*

Tato metoda využívá k hledání dalších klíčů algoritmus Lucchesi a Osborn.

Předávané parametry – první klíč schématu nalezený prostřednictvím funkce Find_First_Key, ukazatel na seznam závislostí. Nově nalezené klíče se přidávají přímo do seznamu za First_Key.

<u>Schéma algoritmu:</u>

```
K1 = First_Key; Added = true;
While (Added)
    Aded = false;
    pro všechny závislosti do
        if PraváStrana má neprázdný průnik s klíče K1
            vytvoř množinu M = (K∪LS) – PS;
            if M není nadmnožinou nebo rovna již vytvořenému klíči
                vymaž z M redundantní atributy;
                přidej M do seznamu klíčů;
                Added = true;
            fi;
        fi;
    od;
    K1 = K1→Next;
Wend;
```

Unita **Keys_Form** obsahuje formulář pro zobrazení klíčů. Pro hledání klíčů se využívá algoritmů z unity Keys.h

4.2.11 MaxSet Form

Unita s formulářem a algoritmy souvisejícími s výpočtem maximálních množin.

a_maxsetcomputation akce, implementovaný algoritmus 1, popsaný v kapitole 2.2.1., výpočtu všech maximálních množin schématu.

S výpočtem všech maximálních množin schématu souvisí i algoritmus testující, zda je schéma v BCNF.Ten je obsahem akce *a_BCNF*.

a_maxAcomputation algoritmus výpočtu maximálních množin jednoho vybraného atributu. Algoritmus rozhodující na základě maximálních množin zda je atribut klíčový, je naprogramován jako akce *a_prime*.

Vzhledem k tomu, že algoritmus výpočtu max(a) se používá i ve formuláři Maximální množiny testovacího modulu, je naimplementována i metoda

*void MaxSet_A(int A, int (*Max_Set)[ABC_SIZE], dependency *First_Dependency, String ABC_ARRAY[ABC_SIZE]);* - parametr A udává prvek, jehož maximální množina se má vygenerovat, v parametru Max_Set se vrací informace o tom, jaké prvky jsou obsaženy v max(A), závislosti a atributy jsou v dalších dvou parametrech.

4.2.12 NF

Unita obsahuje funkce, metody, které souvisí se zjištěním normální formy schématu.

*void Non_Key_Attributes (key *First_Key, int (*non_key_attributes)[ABC_SIZE]);*

Zjištění neklíčových atributů schématu. Funkce projde všechny klíče a atributy, které jsou v nich obsaženy označí jako klíčové.

*int Dependency_Test(dependency *D1, key * First_Key, int (*non_key_attributes)[ABC_SIZE]);*

Parametry udávají – testovanou závislost, ukazatel na seznam klíčů a pole neklíčových atributů schématu.

Funkce otestuje, zda závislost splňuje jednu ze tří podmínek:

- Na pravé straně je klíčový atribut – pokud ano, vrací konstantu BCNF.
- Závislost je triviální, tj. pravá strana je podmnožinou levé strany, je vrácena konstanta BCNF.
- Levá strana závislosti je nadklíčem schématu – vrátí se konstanta 3NF.

Pokud není splněna ani jedna z podmínek, funkce vrací konstantu 2NF.

4.2.13 Decomposition Form 2

Obsahuje formulář a algoritmy dekompozice.

Uživatelský formulář je rozdělen na několik částí: V levé horní části uživatel vidí strom dekompozice. U každého schématu jsou zobrazeny atributy a uživatel vidí i závislost, podle které schéma dekomponoval. V dolní části formuláře je vidět detail schématu.

Strom dekompozice je implementován jako frame obsažený v unitě **Frame_Decomposition_Tree**. Jedinou metodou Frame_Decomposition_Tree je *void Draw_Line(int Number,TColor Color)*, která barvou, druhý parametr, vykreslí příslušnou větev stromu dekompozice.

Formuláře Decomposition_2 obsahuje následující akce:

Projekce	přidá schémata, která jsou projekcí.
NepokrytéZávislosti	přidá nepokryté závislosti.
Dekomposice	provede dekompozici schématu podle zvolené závislosti. Vlastní rozdělení schémat je implementováno v metodě Decompose_2, která je popsána níže.
Print_Selected	zobrazí formulář pro tisk pouze pro výsledná schémata.
Printed_All	zobrazí pro tisk, export všechna schémata.
Konec	uzavře formulář a smaže data.

void Decompose_2 (int Dependency, int Schema); - vlatní provedení dekompozice

Jednotlivá schémata jsou uložena v poli o velikosti definované proměnnou SCHEMATES. Předávané parametry mi udávají závislost, podle které budu schéma dekomponovat a ukazatel na místo v poli schémat, tj. určuje, jaké schéma budu dekomponovat.

4.2.14 Synteza 2

Unita Synteza_2 obsahuje okno a algoritmy syntézy. Jednotlivé algoritmy jsou implementovány jsou akce formuláře.

Levé strany	algoritmus rozdělení schématu podle stejných levých stran.
Ekvivalentní klíče	algoritmus sjednocení schémat s ekvivalentními klíči.
Projekce	sjednotí schémata, která jsou projekcí.
Print_Selected	zobrazí formulář Tisk pro výsledná schémata syntézy.
Print_All	do formuláře Tisk jsou načtena všechna vzniklá schémata.
Konec	uzavře formulář a vymaže data.

4.2.15 Test Form

Unita obsahuje hlavní okno formuláře testovacího modulu. Ostatní formuláře testovacího modulu jsou uloženy v unitách

- Test_Form_Closure
- Test_Form_decomposition
- Test_Form_Keys
- Test_Form_maxSet
- Test_Form_RedAtr

- Test_Form_RedDep

Každý z formulářů je třeba, jak již bylo i zmíněno výše, inicializovat voláním metody *void Fill_In_Data(void)*.

5 Porovnání s jiným programem

5.1 DBE designer

K otestování jsme použili program DBE Designer[2]. Design a algoritmy, použité v tomto programu jsou popsány v knize [1].

5.1.1 Popis programu

Tradiční přístup dekompozice a syntézy bývá někdy kritizován kvůli obtížnosti správného určení vstupních dat pro algoritmy. Tento problém se snažili vyřešit autoři programu DBE Designer. Program využívá možnost náhledu, zobrazení příkladu databázové relace uživateli. Zobrazená data by měla usnadnit identifikaci eventuelně vzniklých anomálií. Program také umí generovat z dané relace množinu funkčních závislostí. Přidáním nebo změnou řádek v relaci tak může návrhář získat další funkční závislosti, které chyběly v původním návrhu. Generování příkladu je založeno na Armstrongových relacích (více [1]).

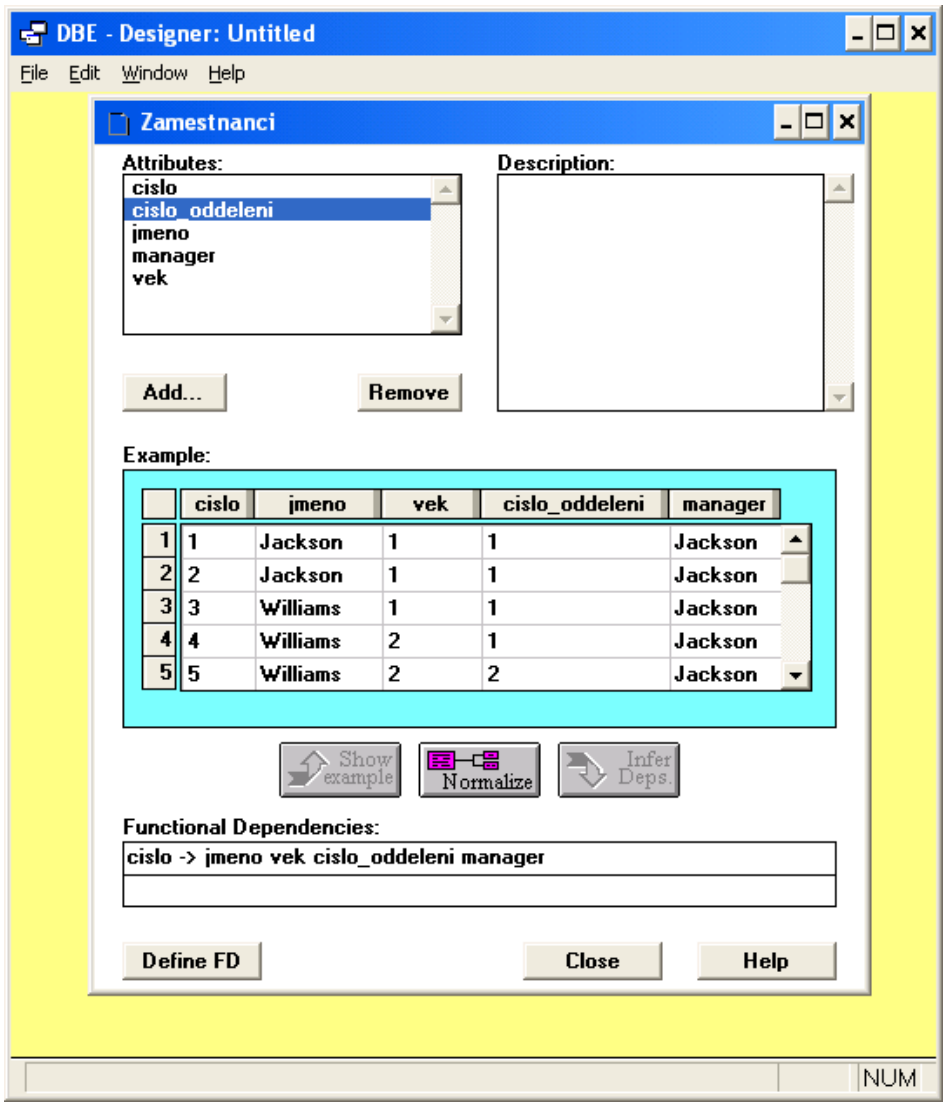
Kritika dekompozice a syntézy se také opírá o to, že tyto algoritmy nakonec vlastně neposkytují uživateli informaci o tom, které výsledné schéma bude nejlepší v praxi. V DBE Designeru je schéma generováno uživatelem. Ten sám řídí celý proces, program mu pomáhá vyhnout se anomáliím.

DBE Designer používá menší formulářová okna, která je možno zobrazovat i najednou. V programu je možné mít zároveň otevřeno více schémat.

Atributy se zadávají v okně Attributes. Uživatel zvolí název atributu a jeho datový typ (číslo, datum, text). V okně Tables pak definuje novou relaci, do které zadá některé z předdefinovaných atributů. Použití DBE Designeru si demonstrujeme na následujícím příkladě.

Použijeme stejnou relaci jako v kapitole 2.2. Nejprve zadáme atributy, a necháme si vygenerovat příklad relace. Získaná tabulka má 6 řádek. Po stisku „*Infer Dep*“, generování závislostí ze zobrazeného příkladu, nejsou vypsány žádné funkční závislosti. Změníme-li data tak, aby ve sloupci číslo bylo toto vždy jedinečné, vygeneruje program následující funkční závislost Číslo→Věk + Jméno + Číslo oddělení + Manažer.

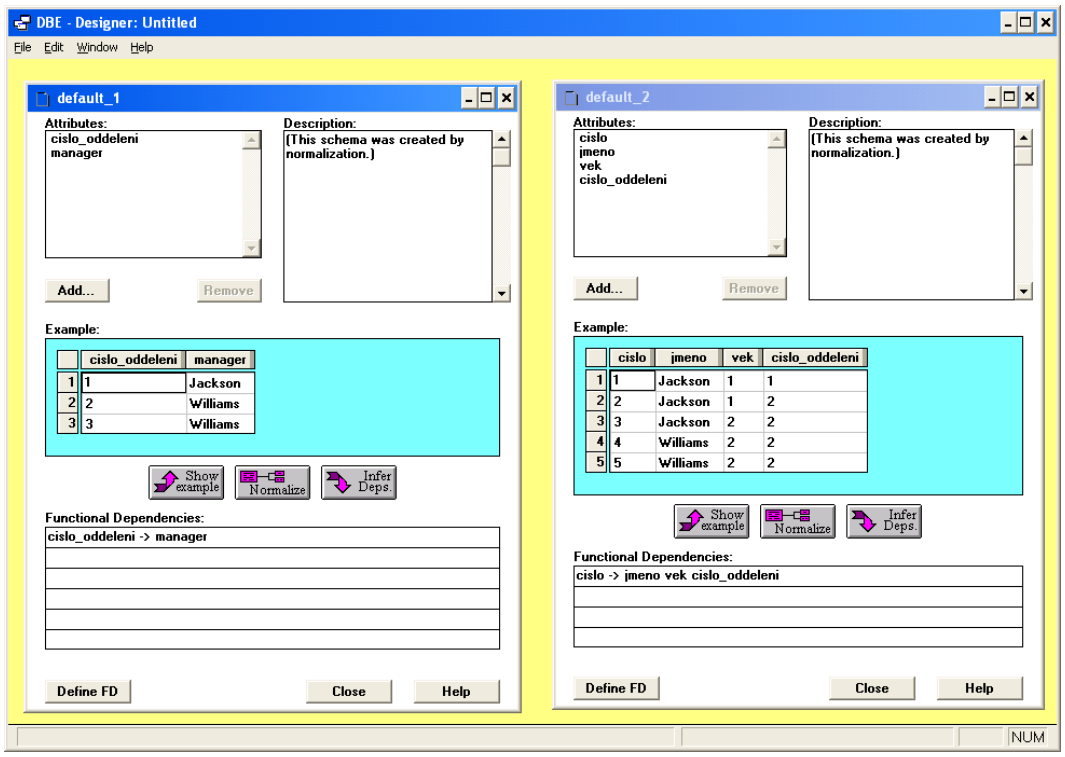
[2] http://www.cs.uta.fi/~hs/dbe.html



obrázek 4 - DBE Designer - Tabulka zaměstnanci

Přidáme další funkční závislost Číslo oddělení→Manažer a smažeme závislost Číslo→Manažer. Necháme si aktualizovat příklad relace. Data se změní, vyhovují novým, námi zadaným funkčním závislostem.

Nyní přejdeme k normalizaci. Po stisknutí tlačítka „*Normalize*“ se uživateli zobrazí okno, popisující následnou normalizaci. Tu může uživatel buď přijmou nebo zamítnout. Nemá bohužel možnost v této verzi programu ji ovlivňovat. Program normalizuje tabulky přímo do Boyce-Coddovy normální formy.



obrázek 5 - DBE Designer - Normalizace relačního schématu

Uživatel nevidí klíče, ani normální formu schématu.

Program Databázové algoritmy slouží především k demonstraci jednotlivých algoritmů. Možnost ovlivňovat průběh procesu dekompozice bohužel u DBE Designeru není. Tento program se spíše než pro výukové účely hodí pro konkrétní návrhy databáze, kdy zobrazení příkladu relace může být pro návrháře jistě velmi přínosné.

5.1.2 Data pro porovnání

Pro srovnání DBE Designeru a programu Databázové algoritmy byla použita čtyři vzorová schémata a porovnán výsledek normalizace v obou výše uvedených programech. Protože v programu Databázové algoritmy lze použít jak dekompozici tak syntézu, je dokumentované rozdělení podle obou těchto metod.

První vybrané schéma bylo ve druhé normální formě, neobsahovalo však ani redundantní závislosti, ani redundantní atributy.

Druhé schéma je také ve druhé normální formě a obsahuje redundantní atribut, třetí schéma ve druhé normální formě a má redundantní závislosti.

Čtvrté schéma je ve třetí normální formě.

5.1.3 Výsledek porovnání

1. schéma

Atributy schématu: u, v, w, x

Závislosti schématu: w + x→u, v + x →w

DBE Designer rozložil schéma následovně:

1. schéma - atributy u, w, x

   závislost w + x →u

2. schéma - atributy v, x, w

   závislost v + x →w

Program Databázové algoritmy rozložil schéma stejně a to jak v případě dekompozice, tak syntézy.

2. schéma

Atributy schématu: i, j, k, l, m

Závislosti schématu: m→k, l + m→j, i + j + k→l, i →m, l→i, l→k

DBE Designer rozložil schéma následovně:

1. schéma - atributy: i, m

   závislost i→m

2. schéma - atributy: i, k

   závislost i→k

3. schéma - atributy: i, j, l

   závislosti l→ i+j, i + j→l

Po rozložení podle stejných levých stran, přidání schémat s ekvivalentními klíči a schémat, která jsou projekcí, vznikla programem Databázové algoritmy následující schémata:

1. schéma - atributy k, m

   závislost m→k

2. schéma ve třetí normální formě - atributy i, k, l

   závislost l→i+k

3. schéma ve třetí normální formě - atributy i, j, l, m

závislosti l→i, l+m→j, i+j→l, j→m

Dekompozice podle m→k, poté jsou přidány nepokryté závislosti a odstraněna schémata, která jsou projekcí:

1. schéma - atributy k, m

závislost m→k

2. schéma ve třetí normální formě - atributy j, l, m

závislosti l + m→j, j→m

3. schéma ve třetí normální formě - atributy i, j, k, l

závislosti i + j + k→l, l→i + k

Dekompozice podle j→m + k a l→i + k dávají výsledná schémata stejná jako dekompozice podle m→k.

V případě tohoto schématu se v Databázových algoritmech nepovedlo nasimulovat stejné rozložení jako u DBE Designeru.

DBE Designer ve svém rozložení nepokrývá závislosti m→k a l + m→j. Databázové algoritmy oproti tomu nenabízí pokrytí závislostí i→m + k a l→j.

3. schéma

Atributy schématu: a, b, c, d

Závislosti schématu: b + c→d, a→d, b + a→c, c + a→b, b + d→c

DBE rozložil schéma následovně:

1. schéma - atributy a, d

závislost a→d

2. schéma - atributy a, b, c

závislosti: a+c→b, a+b→c.

Program Databázové algoritmy schéma pomocí syntézy rozložil na tři schémata:

1. schéma - atributy b,c,d

závislosti: b + c→d, b + d→c

2. schéma - atributy a, b, c

závislost c + a→b

3. schéma - atributy a,d

závislost a→d

Dekompozicí lze schéma rozdělit několika způsoby:

1. dekompozice schématu podle závislosti b + c→d, tím vzniknou 2. schémata

1. schéma - atributy: b, c, d

závislosti: b + c→d, b + d→c

2. schéma - atributy: a, b, c

závislosti: a + b→c, c + a→b

2. dekompozicí podle závislosti a→d vzniknou stejná schémata jako u DBE Designeru

3. dekompozice podle závislosti b + d→c

1.schéma - atributy b, c, d

závislosti: b + c→d , b + d→c

2. schéma - atributy a, d

závislost a→d

3. schéma - atributy a b

závislosti žádné

V tomto případě byla jedna z možností, kterou jsme získali dekompozicí, stejná jako u programu DBE Designer. V případě syntézy bylo navíc přidáno schéma b,c,d, kde jsou pokryty závislosti ztracené v případě rozkladu DBE Designeru.

4. schéma

Atributy: p, q, r,s

Závislosti: r + s→q, q→s, r→p, p + q→r

DBE Designer rozložil schéma následovně:

1.schéma - atributy q, s

závislost q→s

2. schéma - atributy p, r

závislost r→p

3. schéma - atributy q→r,

závislosti žádné

Syntézou, bez přidání schémat s ekvivalentními klíči a projekcí, v programu Databázové algoritmy vzniknou následující schémata:

1. schéma - atributy q, r,s

   závislost r + s→q

2. schéma - atributy p, q, r

   závislost p + q→r

3. schéma - atributy p, r

   závislost r→p

4. schéma - atributy q, s

   závislost q→s

Oproti rozkladu nabízeného v DBE Designeru pokrývá syntéza programu Databázové algoritmy i závislosti r + s→q a p + q→r.

Dekompozicí podle q→s i r→p vzniknou stejná schémata jako u DBE Designeru.

Vzhledem k tomu, že možností jak rozložit schéma je většinou více, je těžké udělat z provedených pokusů nějaký všeobecný závěr. Ve většině případů byl alespoň jeden výsledek programu Databázové algoritmy stejný jako u DBE Designeru.

Výhodou mého programu je větší možnost uživatele zasahovat do procesu, hlavně k případě dekompozice a tím si regulovat a zkoušet různé možnost výsledného rozkladu.

5.2 Normalizace databázových schémat v Prologu

Program v Prologu publikovaný Cerim a Gottlobem v [3] implementuje řadu algoritmů, které jsou použity ve, v této práci předkládané, aplikaci Databázové algoritmy. Na rozdíl od přístupu systému DBE je tedy Databázovým algoritmům blízký, neposkytuje však žádné vhodné uživatelské rozhraní. Program pracuje dávkově, nikoli interaktivně; neumožňuje tedy uživateli do procesu zasahovat či vybírat si některou z alternativních cest. Neobsahuje zpracování maximálních množin ani část testující studenta. Některé dílčí algoritmy (např. uzávěr množiny atributů či hledání klíčů) lze spouštět i samostatně, ale pouze prostřednictvím programovacího jazyka.

Uveďme ukázku vstupních dat a výstupu tohoto programu (výsledky se shodují s výsledky předkládané práce).

1. schéma:

```
schema(rel,[p,q,r,s,t]).
fd(rel,[p],[q]).
fd(rel,[q,r],[s]).
fd(rel,[s],[q]).
fd(rel,[s],[r]).
fd(rel,[r],[p]).
```

?- thirdnfdecomp(rel).

```
decomposition of rel into third normal form step1
making the cover of rel nonredundant
dependency [q, r]---->[s]: new left hand side = [r]
all extraneous attributes eliminated
elimination of redundant functional dependencies
redundant fd[s]-->[q]eliminated

decomposition of rel into third normal form, step2
partitioning the cover into groups
group based on lhs:[r]
group based on lhs:[s]
group based on lhs:[p]
partition into groups completed

decomposition of rel into third normal form, step3
merging groups with equivalent keys
equivalent keys discovered: [s]<-->[r]
all equivalent keys discovered

decomposition of relinto third normal form, step4
elimination of transitive dependencies
 * transitive dependencies eliminated - definitive groups
group: [[s], [r]]
group: [[p]]

decomposition of relinto third normal form, step5
construction of relations

RELATION: rel_a SCHEMA: [p, r, s] in third normal form
SOME KEYS: [s] [r]
RELATION: rel_b SCHEMA: [p, q] in third normal form
SOME KEYS: [p]
```

testing lossless join of decomposition of rel
testing lossless join of decomposition of rel_a
testing lossless join of decomposition of rel_b
the decomposition is NOT losless
the relation rel_key with schema[s, t] is added to the decomposition of rel

2. schéma:

schema(rel7,[a,b,c,d]).
fd(rel7,[b,c],[d]).
fd(rel7,[a],[d]).
fd(rel7,[a,b],[c]).
fd(rel7,[a,c],[b]).
fd(rel7,[b,d],[c]).

?- thirdnfdecomp(rel7).
decomposition of rel7 into third normal form step1
making the cover of rel7 nonredundant
all extraneous attributes eliminated
redundant fd[a, b]-->[c]eliminated
*all redundant functional dependencies eliminated
decomposition of rel7 into third normal form, step2
partitioning the cover into groups
group based on lhs:[b, d]
group based on lhs:[a, c]
group based on lhs:[a]
group based on lhs:[b, c]
* partition into groups completed
decomposition of rel7 into third normal form, step3
merging groups with equivalent keys
equivalent keys discovered: [b, c]<-->[b, d]
* all equivalent keys discovered
decomposition of rel7 into third normal form, step4
elimination of transitive dependencies
[b, d]-->[c]eliminated
[b, c]-->[d]eliminated
* transitive dependencies eliminated - definitive groups
group: [[b, c], [b, d]]
group: [[a]]
group: [[a, c]]
decomposition of rel7 into third normal form, step5

construction of relations
RELATION: rel7_aSCHEMA: [b,c,d] in third normal form
SOME KEYS: [b, c] [b, d]
RELATION: rel7_bSCHEMA: [a, d] in third normal form
SOME KEYS: [a]
RELATION: rel7_cSCHEMA: [a, b, c] in third normal form
SOME KEYS: [a, c]

6 Závěr

Cílem této práce bylo vytvořit program s naimplementovanými algoritmy logického návrhu relační databáze – dekompozicí, syntézou a algoritmy souvisejícími. Program měl sloužit spíše pro menší databázové systémy.

Za účelem řešení byl vytvořen program Databázové algoritmy. Tento program umožňuje uživateli hledat klíče schématu, určit redundantní atributy a závislosti, obsahuje i část věnující se maximálním množinám. Začátečník v oblasti algoritmů logického návrhu databáze může využít testovací část programu.

Celý program byl vyzkoušen a odladěn na více než desítce relačních schémat. Výsledkům čtyř pokusů a jejich porovnání s výstupem s DBE Designeru se věnuje kapitola 5.

V budoucnu by bylo možné aplikaci rozšířit tak, aby mohla zpracovávat větší schémata, pracovat s více závislostmi a atributy. Rozšířit počet atributů jak na levé, tak na pravé straně závislosti. V souvislosti s tím by také bylo nutno upravit zobrazení některých formulářů, zejména u syntézy a dekompozice, tak, aby byly přehledné i v případě většího počtu dat. Další možností rozšíření je i pracovat s více schématy najednou.

Nabízí se i lepší grafický výstup dekompozice a syntézy, navíc s možností ho uživatelsky upravit, možnost tisku či exportu i dalších obrazovek aplikace.

Jako další rozšíření aplikace by bylo možno doimplementovat i další algoritmy zmíněné například v [4] a [1].

Literatura

[1] Mannila Heikki, Raiha Kari–Jouko, *The design of relational databases*, Addison Wesley, 1992.

[2] Mannila Heikki, Raiha Kari–Jouko, *Design-By-Examples: A design tool for relational databases*, Department of Computer Science, Cornell University, Ithaca, July 1985, TR 85-692.

[3] Ceri S., Gottlob G., *Normalization of relations and Prolog*, Communication of ACM, June 1986, Volume 29.

[4] Torgersen Solveig, *Automatic design of relational databases*, Department of Computer Science, Cornell University, September 1989, TR 89-1038.

[5] Mannila Heikki, Raiha Kari-Jouko, *Using feedback in logical design of relational databases*, Department of Computer Science, University of Helsinki, June 1987, Report C-1987-43.

[6] http://www.cs.uta.fi/~hs/dbe.html

[7] Antonín Říha, *Databázové systémy,* http://www.ms.mff.cuni.cz/%7Eriha