Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Relační uživatelské rozhraní

2024

Denis Březina

Vedoucí práce:
Mgr. Jan Laštovička, Ph.D.

Studijní program: Informatika,
Specializace: Programování a vývoj software

Bibliografické údaje

Autor:	Denis Březina
Název práce:	Relační uživatelské rozhraní
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2024
Studijní program:	Informatika, Specializace: Programování a vývoj software
Vedoucí práce:	Mgr. Jan Laštovička, Ph.D.
Počet stran:	24
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Denis Březina
Title:	Relational user interface
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2024
Study program:	Computer Science, Specialization: Programming and Software Development
Supervisor:	Mgr. Jan Laštovička, Ph.D.
Page count:	24
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Ve své práci se věnuji základní teorii relačního modelu, reprezentaci a tvorbě rozhraní aplikace pomocí relační algebry a implementaci knihovny. Výsledkem je knihovna umožňující reprezentovat jednoduchou aplikaci a vykreslit ji pomocí knihovny JavaFX.

Synopsis

In my thesis, I focus on the basic theory of the relational model, the representation and creation of the application interface using relational algebra, and the implementation of the library. The result is a library that allows user to represent a simple application and render it using the JavaFX library.

Klíčová slova: relační algebra; Java; uživatelské rozhraní; JavaFX

Keywords: relational algebra; Java; user interface; JavaFX

Poděkování
Tímto bych chtěl poděkovat vedoucímu práce Mgr. Janu Laštovičkovi, Ph.D. za množství cenných rad a podporu během psaní bakalářské práce.

Obsah

Úvod

Svoji bakalářskou práci na téma *Relační uživatelské rozhraní* jsem psal s cílem vytvořit knihovnu pro reprezentaci a tvorbu uživatelského rozhraní aplikace pomocí relační algebry. Důvodem pro výběr tématu, bylo především to, že součástí práce byl grafický výstup a to v podobě uživatelského rozhraní aplikace.

V teoretické části práce se nejprve věnuji vysvětlení základního pojmu relačního modelu, termínům jako je relace, relační algebra, relační proměnná, relační výraz a také propagaci změn.

V praktické části bylo mým cílem popsat postup způsobu reprezentace a tvorby uživatelského rozhraní pomocí relační algebry a nakonec implementaci knihovny v jazyce Java, který jsem pro práci zvolil.

1 Relační model

V této kapitole se věnuji relační algebře, pohledům v relačních databázích a propagaci změn. Vycházím z knihy *An Introduction to Database Systems* Date [1]

1.1 Relace

Atribut A je uspořádaná dvojice $\langle N, T \rangle$, kde N je název atributu a T je název *typu*. Typ chápeme jako množinu hodnot, například typ *Integer* je množina celých čísel (respektive všech celých čísel reprezentovatelných v počítači), typem může být také například datum, text, ...

Hodnota daného typu je individuální konstanta, například číslo 2. Hodnota nemůže být změněna. *Proměnná* uchovává hodnotu a na rozdíl od hodnot může být změněna, tedy lze jí přiřadit jinou hodnotu. Hodnoty i proměnné jsou vždy nějakého typu, který se nemění. Každá hodnota, kterou lze přiřadit nějaké proměnné musí být stejného typu jako proměnná.

Mějme soubor atributů A_i $(i = 1, 2, ..., n)$, atributy $A_1, ..., A_n$ mají po dvou různé názvy. *Řádkem* r na těchto atributech rozumíme množinu uspořádaných dvojic $\langle A_i, v_i \rangle$, kde v_i hodnota typu atributu A_i. Dále definujeme:

- Hodnota n je *stupeň* nebo *arita* řádku.
- Uspořádaná dvojice $\langle A_i, v_i \rangle$ je *komponenta* řádku.
- Množina všech atributů je *hlavička* řádku r.

Relační hodnota rel zkráceně *relace* se skládá z *těla* a *hlavičky*, kde:

- Hlavička relace *rel* je hlavička řádku. Relace *rel* má stejné atributy a tedy i stejný stupeň jako hlavička.

- Tělo relace *rel* je množina řádků, které mají všechny stejnou hlavičku jako relace *rel*.

Kardinalita relace je počet řádků v jejím těle. Typ relace je určen její hlavičkou. Relace mají následující vlastnosti:

- Každý řádek obsahuje právě jednu hodnotu pro každý atribut.
- Atributy nemají žádné uspořádání zprava doleva.
- Řádky nemají uspořádání shora dolů.
- Nemá duplicitní řádky.

Relace *rel* může mít omezení na hodnoty atributů. Například pro atribut A typu *integer* může být omezení že jeho hodnota je větší než 3. Všechny řádky relace *rel* pak tohle omezení splňují.

V textu práce používám tabulkový zápis relace, kde první řádek obsahuje názvy atributů a ostatní řádky pak hodnoty jednotlivých atributů.

1.2 Relační algebra

Jelikož relace obsahuje množinu řádků, lze na ni aplikovat množinové operace. Dále máme také další speciální relační operace. V textu jsou definovány pouze operace, které byly použity, není definován rozdíl a průnik relace je zahrnut jako speciální případ spojení. Pro příklad využiji relace A a B určeny tabulkami 1 a 2

widget	row	col
label	0	0
button	1	0

Relace 1: Relace A

rowSpan	colSpan
1	1
1	2

Relace 2: Relace B

1. *Přejmenování*

   Na vstupu operace je relace R a na výstupu je relace R', která je se vstupní relací identická až na to, že jeden atribut má jiné jméno. Takže pro relaci A (tabulka 1) po přejmenování atributu *widget* na *item* dostaneme relaci 3.

   Zápis: *R RENAME oldName AS newName.*

item	row	col
label	0	0
button	1	0

Relace 3: Přejmenování atributu widget na item

2. *Sjednocení*

   Na vstupu operace jsou dvě relace R a P jejichž hlavičky se shodují. Na výstupu je relace Q, která obsahuje všechny řádky, které se vyskytují v R nebo v P. Podmínka, aby hlavičky sjednocovaných relací byly shodné, zajišťuje, že výsledkem sjednocení je také relace. Sjednocením relací A 1 a D 4 dostaneme relaci 5

widget	row	col
textArea	0	1

Relace 4: Relace D

widget	row	col
label	0	0
button	1	0
textArea	0	1

Relace 5: Sjednocení relací A a D

   Zápis: R *UNION* P

3. *Kartézský součin*

   Na vstupu operace jsou dvě relace R a P, takové že, R a P se neshodují v žádném názvu atributu. Na výstupu je relace Q, jejíž hlavička je sjednocení hlaviček relací R a P a tělo je tvořeno všemi řádky r ,které vzniknou sjednocením řádku v R s řádkem v P. Kardinalita relace Q je součin kardinalit relací R a P a stupeň relace Q je součet stupňů relací R a P. Výsledkem skalárního součinu relace A 1 s relace B 2 je relace 6.

widget	row	col	rowSpan	colSpan
label	0	0	1	1
label	0	0	1	2
button	1	0	1	1
button	1	0	1	2

Relace 6: Kartézský součin A a B.

   Zápis: R *TIMES* P.

4. *Projekce*

   Na vstupu operace je relace R a atributy $X, Y, ..., Z$ z hlavičky R. Na výstupu relace s hlavičkou $\{X, Y, ..., Z\}$. a tělem tvořeným řádky $\{< X, x > , < Y, y >, ..., < Z, z >\}$, což jsou všechny řádky v R, které mají pro atribut X hodnotu x, pro atribut Y hodnotu y, ... a pro atribut Z hodnotu z.

   Projekcí relace A 1 na atributy $widget, col$ (*A PROJECT {widget, col}*) je relce 7

   Zápis: *R PROJECT {X, Y, ..., Z }*

widget	col
label	0
button	0

Relace 7: Projekce relace A na atributy $widget, col$

5. *Přirozené spojení*

   Na vstupu operace jsou dvě relace $R1$ a $R2$ a na výstupu je relace V. Nechť relace $R1$ má atributy $X1, X2, ..., Xm, Y1, Y2, ..., Yn$ a relace $R2$ má atributy $Y1, Y2, ..., Yn, Z1, Z2, ..., Zp$, atributy $Y1, Y2, ..., Yn$ se v obou relacích shodují. Výsledná relace V má hlavičku $\{X1, X2, ..., Xm, Y1, Y2, ..., Yn, Z1, Z2, ..., Zp\}$. Pro všechny řádky v_i v jejím těle platí, že v relaci $R1$ je řádek $r1_j$, který má v atributech $X1, X2, ..., Xm, Y1, Y2, ...,$ stejné hodnoty jako v_i a v relaci $R2$ je řádek $r2_k$, který má v atributech $Y1, Y2, ..., Yn, Z1, Z2, ..., Zp$ stejné hodnoty jako v_i. Pokud $R1$ a $R2$ nemají žádné shodné atributy jedná se o kartézský součin. Spojení relace A 1 s relací E 8 je následující relace 9.

widget	rowSpan	colSpan
label	1	1
textArea	1	1

Relace 8: Relace E

widget	row	col	rowSpan	colSpan
label	0	0	1	1

Relace 9: přirozené spojení A a E.

   Zápis: *R1 JOIN R2.*

6. *Rozšíření*

   Na vstupu operace je relace R a na výstupu relace R', která je shodná s R až na to, že má přidaný atribut y s hodnotou, která byla získána vyhodnocením skalárního výrazu *expression*. Skalární výraz je výraz obsahující

názvy atributů relace, operace a případně konstanty. Pokud rozšíříme relaci F 10 o atribut $newData$ s hodnotou danou skalárním výrazem $data + 1$ bude výsledkem relace 11.

widget	data
label	1
textArea	10

Relace 10: Relace F

widget	data	newData
label	1	2
textArea	10	11

Relace 11: rozšíření relace F o atribut $newData$ s hodnotou $data + 1$

Zápis: *EXTEND R ADD (expression) AS y*

1.3 Relační proměnné

Rozlišujeme dva typy *relačních proměnných*. Prvním jsou *základní relační proměnné* a druhým *pohledy*. Relační proměnné mají stejně jako relační hodnoty hlavičku. Hodnotou relační proměnné je relace.

Všechny možné hodnoty dané relační proměnné jsou stejného relačního typu (typ relace je specifikován při definici relační proměnné) a tedy mají stejnou hlavičku. Když je relační proměnná definována je jí dána výchozí hodnota, kterou je prázdná relace daného typu. Předpokládáme, že máme způsob, jak specifikovat výchozí hodnoty atributů pro základní relační proměnné. Tyto hodnoty se použijí pokud při přidávání řádku nejsou specifikovány hodnoty pro všechny atributy.

Relační proměnné lze na rozdíl od relačních hodnot upravovat, přesněji lze jí přiřadit jinou hodnotu. Základní relační proměnné lze upravit přímo. Konkrétně lze přidávat řádky, odstraňovat řádky a měnit hodnoty atributů v jednotlivých řádcích. Vložení, odstranění a změna řádku jsou množinové operace, takže pokud upravujeme jeden řádek, tak ve skutečnosti upravujeme množinu řádků s kardinalitou jedna. Takže máme-li relační proměnnou q, tak přidání řádku znamená, že do proměnné q přiřadíme relaci, která vznikne sjednocením původní hodnoty s relací obsahující přidávaný řádek.

Jelikož řádky relační proměnné jsou hodnoty, tak nemohou být měněny. Změnou řádku t na t' myslíme tedy nahrazení řádku t jiným řádkem t'. Stejný postup se aplikuje pokud měníme hodnotu atributu A v řádku.

Stejně jako relace mohou mít omezení i relační proměnné. Všechny hodnoty, které lze do proměnné přiřadit musí splňovat její omezení.

1.4 Relační výrazy

Relační výraz je:

- relační proměnná,
- aplikace relační operace na relační výraz.

Hodnotou relačního výrazu je relace.

Pohled v relační databázi je pouze pojmenovaný výraz relační algebry. Při vytvoření pohledu není výraz vyhodnocen, ale je zapamatován pod daným jménem. Nicméně z pohledu uživatele se bude databáze chovat, jako by v ní skutečně existovala relační proměnná daného jména. Název určuje odvozenou a virtuální relační proměnnou, jejíž hodnotou je relace, která by vznikla vyhodnocením pohled-definujícího výrazu v daný moment.

Pohled není oddělená kopie dat, je to pouze okno do podkladových dat. Jakákoliv změna v podkladových datech bude v pohledu okamžitě a automaticky viditelná. Rovněž každá změna dat v pohledu bude okamžitě a automaticky aplikována na podkladová data a tedy rovněž viditelná v pohledu. Uživatel tak může pracovat s pohledem jako se základní relační proměnnou.

Také výraz může mít omezení. Tohle omezení pak má i relace, která je jeho výsledkem.

1.5 Propagace změn

Pohledy jsou relační proměnné a tak mohou být měněny. Pokud máme určitou změnu na určitém pohledu, je potřeba zjistit, jakou změnu je nutné provést na podkladových datech, aby se provedla původní změna v pohledu. Podkladová data mohou být buď základní relační proměnné, nebo pohledy. Pravidla změny musí být aplikovatelná v obou případech. To znamená, že pravidla musí být možné aplikovat rekurzivně. Pokud změna podkladové proměnné selže, tak selže i původní změna pohledu.

Mějme pohled P s omezením O a relační proměnné P_1 a P_2 s omezeními O_1 a O_2. Změna v P se provede na relačním výrazu, který tuto proměnnou určuje v závislosti na relační operaci:

- Sjednocení *($P = P_1$ UNION P_2)*
  - Omezení: $O = O_1\ OR\ O_2$.
  - Vložení: Pokud vkládaný řádek r splňuje omezení pro P_1 je r vloženo do P_1, pokud r splňuje omezení pro P_2 je r vloženo do P_2.
  - Odstranění: Pokud se odstraňovaný řádek nachází v P_1 je z P_1 odstraněn, pokud se nachází P_2 je z P_2 odstraněn.
  - Změna: Při změně řádku r na r' se nejdříve provede odstranění řádku r z P a poté přidání řádku r' do P.

- Rozšíření ($P = EXTEND\ P_1\ ADD\ (expression)\ AS\ y$)
  - Omezení: $O = O_1\, AND\, P.y = experssion(P_1)$, kde $P.y$ značí hodnoty atributu y v relaci P.
  - Vložení: Vkládaný řádek r musí splňovat omezení pro P. Řádek a, který vznikne odstraněním přidaného atributu y je přidán do P_1.
  - Odstranění: Nechť je r řádek k odstranění z P. Z P_1 je odstraněn řádek a, který vznikne odstraněním přidaného atributu y.
  - Změna: Při změně řádku r na r' se nejdříve provede odstranění řádku r z P a poté přidání řádku r' do P.
- Přirozené spojení ($P = P_1\ JOIN\ P_2$)
  - Omezení: $O = O_1(r(P_1))\ AND\ O_2(r(P_2))$, kde r je řádek relace P a $O_1(r(P_1))$ značí, že P_1 část řádku r splňuje omezení O_1. P_1 část řádku r je řádek, který vznikne projekcí řádku r na všechny atributy P_1.
  - Vložení: Nový řádek j musí splňovat omezení pro P. Pokud se P_1 část j nevyskytuje v P_1 je tato část vložena do P_1. Pokud se P_2 část j nevyskytuje v P_2 je tato část vložena do P_2.
  - Odstranění: P_1 část odstraňovaného řádku j je odstraněna z P_1 a P_2 část je odstraněna z P_2.
  - Změna: Při změně řádku r na r' se nejdříve provede odstranění řádku r z P a poté přidání řádku r' do P.

Změna pro projekci zde není definována jelikož ji nevyužívám k tvorbě rozhraní aplikace. Využívám ji pouze při změnách v aplikaci.

2 Reprezentace rozhraní aplikace

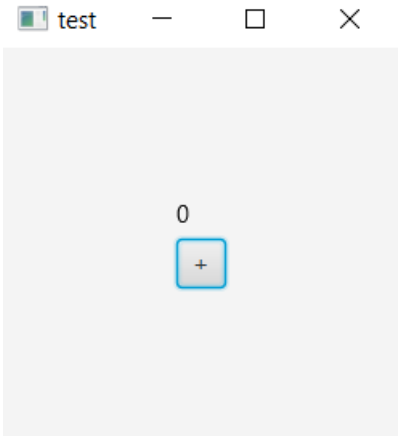
V této kapitole popisuji postup vytvoření uživatelského rozhraní aplikace za použití relačních operací.

2.1 Proces tvorby rozhraní aplikace

Jako první příklad jsem zvolil počítadlo, tedy aplikaci zobrazující pouze číslo a tlačítko, po jehož stisknutí se hodnota čísla zvedne o jedna. Ukázka rozhraní je na obrázku 1. Data jsou reprezentována relační proměnnou 12 nazvanou *cntr*. Rozhraní aplikace je reprezentováno relační proměnnou *app* 13.

counter
0

Relace 12: Úvodní hodnota proměnné *cntr*



Obrázek 1: Ukázka rozhraní aplikace

id	widget	row	col	rowSpan	colSpan	data	newData	text
1	label	0	0	1	1	0		0
2	button	1	0	1	1	0	1	+

Relace 13: Rozhraní aplikace

Relační proměnná *app* má atributy *id*, *widget*, *row*, *col*, *rowSpan*, *colSpan*, *data*, *newData* a *text*. Atribut *id* je jednoznačný identifikátor prvku, *widget* označuje typ prvku, *row* je číslo prvního řádku a *col* číslo prvního sloupce, na které se prvek vykreslí. Dále atributy *rowSpan* a *colSpan* definují, přes kolik řádků respektive sloupců se daný prvek vykreslí. Atribut *data* označuje momentální data prvku, atribut *newData* má smysl použít pouze pro objekt typu tlačítko, jelikož udává jaká má být hodnota dat po stisku tlačítka. Atribut *text* označuje text, který se vykreslí v daném prvku. Hodnota atributu *newData*, pro jiné prvky než je tlačítko, může být prázdná hodnota, jelikož nemá žádný vliv na aplikaci.

Popis relace zní: prvek *id* je typu *widget*, začíná na řádku *row* a ve sloupci *col*, zabírá *rowSpan* řádků a *colSpan* sloupců, jeho data jsou *data*, je-li prvek tlačítko, tak se po jeho stisku změní data na *newData* a při vykreslení se zobrazí hodnota *text*.

Při vytváření rozhraní aplikace se nejprve vytvoří základní relace pro GUI *l*1 14 a *b*1 15 obsahující identifikátor prvku *id*, typ prvku *widget*, pozici prvku *row* a *col*, a velikost prvku *rowSpan* a *colSpan*. Pro tento příklad to jsou následující relace:

id	widget	row	col	rowSpan	colSpan
1	label	0	0	1	1

Relace 14: Relace l1 – úvodní hodnota proměnné pro popisek

id	widget	row	col	rowSpan	colSpan
2	button	1	0	1	1

Relace 15: Relace b1 – úvodní hodnota proměnné pro tlačítko

Nad relační proměnnou *cntr* 12 se nejdříve vytvoří pohled pojmenovaný dt, ve kterém je pomocí operace přejmenování

$$dt = cntr\ RENAME\, widget\ AS\, data \tag{1}$$

změněn název hodnoty z *counter* na *data*. Poté se pomocí přirozeného spojení

$$\begin{aligned} l2 &= l1\ JOIN\, dt \\ b2 &= b1\ JOIN\, dt \end{aligned} \tag{2}$$

vytvoří pohledy $l2$ a $b2$ reprezentovány tabulkami 16 a 17, kde je pohled *Data* spojen s relačními proměnnými pro popisek $l1$, respektive tlačítko $b1$.

id	widget	row	col	rowSpan	colSpan	data
1	label	0	0	1	1	0

Relace 16: Pohled l2 – popisek spojen s pohledem dt

id	widget	row	col	rowSpan	colSpan	data
2	button	1	0	1	1	0

Relace 17: Pohled b2 – tlačítko spojeno s pohledem dt

Použitím operace rozšíření

$$b3 = EXTEND\, b2\ ADD\, (data + 1)\ AS\, newData \tag{3}$$

se vytvoří pohled $b3$ 18, kde je tlačítko rozšířeno o hodnotu $newData$ odpovídající hodnotě $data + 1$.

Pohledy $l2$ i $b3$ se pomocí operace přirozené spojení nebo rozšíření doplní o hodnotu *text*, která obsahuje text, jenž se má vykreslit v daném objektu. Vznikají pohledy $b4$ 19 a $l3$ 20. Při použití operace přirozeného spojení je potřeba nejdříve vytvořit relační proměnné respektive pohledy, které obsahují atribut *text* s požadovanou hodnotou. Operaci přirozené spojení lze v případě, že spojované relační proměnné mají disjunktní množiny atributů nahradit operací skalární součin.

Pohledy $b4$ 19 a $l3$ 20 se poté pomocí operace sjednocení

$$app = b4\ UNION\, l3 \tag{4}$$

sjednotí do jednoho pohledu nazvaného *app* reprezentovaného relací 13, jenž reprezentuje celkové rozhraní aplikace. Jelikož pohledy $b4$ 19 a $l3$ 20 nemají shodné

id	widget	row	col	rowSpan	colSpan	data	newData
2	button	1	0	1	1	0	1

Relace 18: Pohled b3 – tlačítko rozšířeno a atribut *newData*

id	widget	row	col	rowSpan	colSpan	data	newData	text
2	button	1	0	1	1	0	1	+

Relace 19: Pohled b4 – kompletní reprezentace tlačítka

množiny atributů, je pohled $l3$ 20 nejdříve doplněn o chybějící atribut s hodnotou prázdné hodnoty a teprve poté se sjednotí.

Po stisku tlačítka v aplikaci dojde k výměně hodnoty *data* za hodnotu *newData* a spustí se zpětná propagace změny.

2.2 Aplikace změny

Při vytváření reprezentace vzniká struktura pohledů, a právě tuto strukturu lze využít při zpětné propagaci změn. Změna probíhá tak, že se odstraní původní řádek a vloží se nový řádek. Respektive pokud je proměnná pohled, tak se hledá jaký řádek se má odstranit a čím se má nahradit v podkladových datech pohledu. V tomhle případě se po stisknutí tlačítka nahradí řádek 21 řádkem 22.

Dle pravidel pro sjednocení se změna provede na pohledu $b4$. Zde se v závislosti na použité operaci pokračuje podle pravidel pro přirozené spojení nebo rozšíření na pohledu $b3$, kde se provede následující změna: 23. Podle pravidel pro rozšíření se pokračuje na pohledu $b2$, kde se provede změna nastíněná tabulkou 24. Dle pravidel pro přirozené spojení se pokračuje na pohledu dt, kde je provedeno nahrazení řádku 25 řádkem 26. Nakonec se provede změna na základní proměnné *cntr*, kde již dojde k nahrazení řádku 27 řádkem 28.

3 Implementace

V této kapitole se věnuji implementaci mnou vytvořené knihovny. Pro implementaci jsem použil jazyk *Java* společně s knihovnou *JavaFX* pro vykreslování aplikace. Při programování jsem použil následující dokumentaci: *Java® Platform, Standard Edition & Java Development Kit Version 21 API Specification* [2].

3.1 Reprezentace relace

Relaci reprezentuji jako instanci třídy *Relation*. Instance obsahuje seznam atributů relace reprezentující hlavičku a seznam řádků reprezentující tělo relace. Třída Relation definuje základní metody na přidání a odebrání řádku. Instance třídy Relation je při vytvoření prázdná a tyto metody slouží pro vytvoření požadované relační hodnoty. Ty se používají také při aplikaci změny v relační pro-

id	widget	row	col	rowSpan	colSpan	data	text
1	label	0	0	1	1	0	0

Relace 20: Pohled l3 – kompletní reprezentace popisku

id	widget	row	col	rowSpan	colSpan	data	newData	text
2	button	1	0	1	1	0	1	+

Relace 21: Řádek k odstranění

měnné. Třída Relation definuje také metodu na vytvoření kopie relace. Při vytvoření kopie vzniká nová relace, která je naplněna kopiemi řádků z původní relace.

Řádek relace reprezentuji instancí třídy *RelationItem*. Instance obsahuje *hašovací tabulku* s hodnotami v řádku uloženými ve formátu *atribut-hodnota* a seznam atributů, který používám pro vyhledávání v hašovací tabulce. Třída RelationItem definuje metody pro přidání prvku a odstranění prvku. Tyto metody se používají při vytváření požadovaných řádků pro vložení do relace a také při většině relačních operací. Dále třída definuje metodu porovnání na shodu s jiným řádkem, kterou používám při kontrole duplicit, při sjednocení. Třída také definuje metodu na vytvoření kopie řádku, kterou používám při vytváření kopie relace. Při vytvoření kopie řádku vzniká nový řádek, takže i nový seznam atributů a nová hašovací tabulka, tyto jsou ovšem naplněny původními hodnotami. Tento postup jsem zvolil, jelikož hodnoty v řádku jsou neměnné. Nedochází tedy ke změně hodnoty, ale může dojít jen k vyměnění hodnoty. Jako hodnotu atributu lze použít instanci třídy s rozhraním *Value*. Momentálně mám třídy pro reprezentaci hodnot typu *Integer* a *String*.

Množina atributů řádku je podmnožinou množiny atributů relace, do které řádek patří. To znamená, že řádek nemusí mít všechny atributy, které má relace do níž patří. Výsledkem dotazu na hodnotu atributu, který řádek nemá, je prázdná hodnota. Tohle rozšíření používám při sjednocování relací, kdy pak není potřeba kontrolovat shodu množin argumentů sjednocovaných relací.

Relační proměnná je reprezentována jako instance třídy *Variable*, která používá rozhraní *Expression*. Instance Variable obsahuje název relační proměnné. Třída definuje metodu pro provedení změny a metodu na vyhodnocení, která vrátí hodnotu proměnné (relaci). Databázi představuje hašovací tabulka obsahující záznamy typu *název-relace*.

V ukázce kódu 1 je ukázáno vytvoření základních relací použitých při tvorbě uživatelského rozhraní aplikace zobrazeného na obrázku 1. V ukázce byla použita pomocná třída *PrefixDispenser*, která slouží k vytvoření úvodních relací pro grafické prvky.

id	widget	row	col	rowSpan	colSpan	data	newData	text
2	button	1	0	1	1	1	2	+

Relace 22: Řádek ke vložení

id	widget	row	col	rowSpan	colSpan	data	newData
2	button	1	0	1	1	0	1

↓

id	widget	row	col	rowSpan	colSpan	data	newData
2	button	1	0	1	1	1	2

Relace 23: Změna prováděná na pohledu $b3$. Tabulka nahoře představuje původní řádek a tabulka dole nový řádek.

3.2 Reprezentace operací

Každá operace je reprezentována samostatnou třídou, jejíž instance pak odpovídá relačnímu výrazu, který danou operaci realizuje. Třídy pro operace mají rozhraní Expression. Instance libovolné třídy s rozhraním Expression je dále označována jako *objekt typu Expression* nebo také *výraz*. Každá třída má metodu pro vyhodnocení, která má na vstupu databázi a vrací relační hodnotu (instanci třídy `Relation`)a metodu pro propagaci změny, která má na vstupu původní řádek relace, nový řádek relace a databázi. Implementoval jsem následující třídy:

- *Rename* pro operaci přejmenování,
- *Product* pro operaci kartézský součin,
- *NaturalJoin* pro operaci přirozené spojení,
- *Union* pro operaci sjednocení,
- *Extend* pro operaci rozšíření.

Výraz **přejmenování** má na vstupu objekt typu Expression, původní a nový název přejmenovávaného atributu. Při vyhodnocení se nejdříve vytvoří nová relace, do které jsou vloženy kopie řádků, kde byla původní hodnota odstraněna, a poté vložena pod novým názvem.

Při změně se v odstraňovaném i vkládaném řádku provede zpětné přejmenování. To znamená, že nový název atributu je nahrazen původním a provede se změna na vstupním výrazu.

Výrazy **kartézský součin** a **přirozené spojení** mají oba na vstupu dva objekty typu Expression. Při vyhodnocení přirozeného spojení se nejdříve zjistí shodné atributy hodnot vstupních výrazů. Pokud je průnik množin atributů neprázdný, tak se pro každý řádek z hodnoty prvního výrazu hledá řádek z hodnoty druhého výrazu, se kterým se ve společných atributech shoduje hodnotou. Pokud je takový řádek nalezen, vytvoří se jejich sjednocení, které je vloženo do výsledku

id	widget	row	col	rowSpan	colSpan	data
2	button	1	0	1	1	0

↓

id	widget	row	col	rowSpan	colSpan	data
2	button	1	0	1	1	1

Relace 24: Změna prováděná na pohledu $b2$. Tabulka nahoře představuje původní řádek a tabulka dole nový řádek.

data
0

Relace 25: Řádek k odstranění v pohledu *dt*

data
1

Relace 26: Řádek ke vložení v pohledu *dt*

couter
0

Relace 27: Řádek k odstranění v proměnné *cntr*

couter
1

Relace 28: Řádek ke vložení v proměnné *cntr*

a pokračuje se v hledání, dokud pro každý řádek hodnoty prvního výrazu nejsou prošlé všechny řádky hodnoty druhého výrazu. Pokud je průnik množin atributů prázdný, vytvoří se instance kartézského součinu a vrátí se její výsledek.

Při vyhodnocení kartézského součinu se vytvoří nová relace, která obsahuje všechna sjednocení řádků z prvního výrazu s řádky z druhého výrazu. Pokud je instance kartézského součinu vytvořena uživatelem tak se předpokládá, že vstupní výrazy nemají shodné názvy atributů.

Změna je pro kartézský součin i přirozené spojení identická. Původní a nový řádek se rozdělí projekcí na vstupní výrazy a aplikuje se metoda změny ve vstupních výrazech.

Výraz **sjednocení** má na vstupu dva objekty typu Expression. Při vyhodnocení vzniká nová relace, která je kopií hodnoty prvního vstupního výrazu, kam poté byly vloženy všechny řádky hodnoty druhého výrazu neobsažené v prvním výrazu.

Při změně se hledá, který ze vstupních výrazů obsahuje měněný řádek. Pokud byl měněný řádek nalezen v hodnotě prvního výrazu, pak je aplikována metoda

```
1  Relation counter =new Relation();
2  Database database =new Database();
3  RelationItem counterItem =new RelationItem();
4  counterItem.add("counter",new IntValue(0));
5  counter.insert(counterItem);
6  database.insert("counter",counter);
7
8  PrefixDispenser disp= new PrefixDispenser();
9  database.insert("lbl",disp.getLabelPrefix(0,0));
10 database.insert("btn",disp.getButtonPrefix(1,0));
```

Zdrojový kód 1: Ukázka použití knihovny při tvorbě úvodních relací

změny prvního výrazu. Jinak je aplikována metoda změny druhého výrazu.

Výraz **rozšíření** má na vstupu objekt typu Expression, název nového atributu, objekt typu *ScalarExpression* (dále skalární výraz), což je třída obsahující funkci následujícího tvaru `Function<ArrayList<Value>,Value> function`, která je definována uživatelem. Nakonec má na vstupu argumenty, které se ukládají do seznamu.

Při vyhodnocení se vytvoří kopie relace vzniklé vyhodnocením vstupního výrazu. Poté jsou pro každý řádek vyhodnoceny argumenty skalárního výrazu, následně je k řádku přidán nový atribut s hodnotou, která byla vrácena po vyhodnocení skalárního výrazu. Při změně je z měněného řádku odstraněn atribut, který byl přidán, a je aplikována metoda změny vstupního výrazu.

Mějme relaci *counter* a dále relace *btn* a *lbl* vytvořené v ukázce 1. V následujícím kódu 2 je ukázáno vytvoření reprezentace uživatelského rozhraní aplikace zobrazené na obrázku 1.

3.3 Vykreslení rozhraní aplikace

Pro vytvoření vykreslované reprezentace uživatelského rozhraní aplikace slouží třída *SimpleDrawer*, jejíž instance obsahuje objekt typu Expression reprezentující rozhraní, databázi, ve které je objekt typu Expression vyhodnocen, a seznam vykreslovaných prvků. Ve třídě *SimpleDrawer* jsou definovány vnořené (nested) třídy pro jednotlivé grafické objekty reprezentace. V současné verzi to jsou tlačítko popisek a textové pole.

Třída definuje metodu `createScene`, která vrací objekt typu `Scene`, což je objekt vykreslitelný pomocí knihovny JavaFX. Při vytvoření reprezentace v JavaFX vzniká instance třídy *GridPane* což je objekt, který se nakonec vykreslí. Po vytvoření objektu `GridPane` se vyhodnotí vstupní výraz v dané databázi. Tím se vytvoří pohled reprezentovaný objektem typu `Relation`, ve kterém se prochází všechny řádky. Pro každý řádek se vytvoří objekt typu, který je daný atributem *widget*. Tento objekt se umístí do objektu `GridPane` na místo určené atributy *row* a *col* a nastaví se mu velikost daná atributy *rowSpan* a *colSpan*.

```
1  // funkce počítající součet čísel
2  Function<ArrayList<Value>,Value> fun =(list)->{
3     int res=0;
4     for(int i=0;i< list.size();i++){
5        IntValue v = (IntValue) list.get(i);
6        res+=v.eval();
7     }
8     return new IntValue(res);
9  };
10
11 //funkce vracející konstantní String s hodnotou "+"
12 Function<ArrayList<Value>,Value> fun2 =(list)-> new StringValue("+")
       ;
13
14 Rename rename =new Rename(new Variable("counter"),"counter","data");
15 Product p1 =new Product(new Variable("lbl"),rename);
16
17 Extend ex =new Extend(rename,"text",new ScalarExpression(fun2));
18 NaturalJoin p2= new NaturalJoin(new Variable("btn"),ex);
19
20 ScalarExpression increment =new ScalarExpression(fun);
21 Extend e1=new Extend(p2,"newData",increment,new ScalarConstant(new
       IntValue(1)),new ScalarVariable("data"));
22 Union app = new Union(p1,e1);
```

Zdrojový kód 2: Ukázka použití knihovny při tvorbě uživatelského rozhraní aplikace

Poté se objekt vloží do seznamu objektů instance. Vytvořený objekt typu `Scene` (třída knihovny JavaFX) se nakonec vykreslí pomocí metod knihovny JavaFX.

Při změně se nejdříve spustí propagace změny, následně se vstupní objekt typu Expression vyhodnotí znovu. Všem objektům v seznamu objektů jsou aktualizována data (objekty v seznamu objektů se se svou reprezentací ve vyhodnocení objektu typu expression párují přes atribut *id*) a nakonec je aktualizováno rozhraní aplikace.

Vykreslení aplikace reprezentované relací *app* vyhodnocovanou v databázi *database* se provede následující částí kódu 3

```
1  @Override
2  public void start(Stage stage) throws Exception {
3  .
4  //Vytvoření relace app z předchozího příkladu.
5  .
6  stage.setTitle("test");
7  stage.setScene(new SimpleDrawer(app,database).createScene());
8  stage.show();
9  }
10 public static void main(String[] args) throws Exception {
11
12    launch(args);
13 }
```

Zdrojový kód 3: Ukázka použití knihovny při vykreslení uživatelského rozhraní aplikace

Závěr

Cílem práce bylo navrhnout a naprogramovat knihovnu pro reprezentaci a tvorbu uživatelského rozhraní aplikace pomocí relačních vztahů. Vytvořená knihovna obsahuje základní operace pro tvorbu rozhraní a jednoduchou knihovnu pro vykreslování pomocí knihovny JavaFX. Pokud by to bylo třeba, lze knihovnu snadno rozšířit o další operace. Pokud bychom uvažovali o rozšíření o další datové typy hodnot, jednalo by se již o složitější proces.

Knihovna nepodporuje možnost vyskakovacích oken. Vyskakovací okna by bylo možné implementovat jako speciální tlačítko, které by místo změny v datech otevřelo okno definované jeho daty. Pro tuto variantu by bylo potřeba mít možnost použít relaci jako hodnotu atributu.

A Obsah elektronických dat

Elektronická data obsahují zdrojový kód knihovny. To znamená: třídy pro reprezentaci uživatelského rozhraní a třídu pro převod reprezentace rozhraní na data knihovny JavaFX. Knihovna JavaFX není přiložena. Součástí dat jsou také vzorové aplikace, které jsem použil při testování.

Literatura

[1] Date, C.J. *An Introduction to Database Systems*. 8th. 2004. 983 s. ISBN 0-321-19784-4.

[2] *Java® Platform, Standard Edition & Java Development Kit Version 21 API Specification*. USA: Oracle, 2023. Dostupný z: ⟨https://docs.oracle.com/en/java/javase/21/docs/api/index.html⟩.