VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

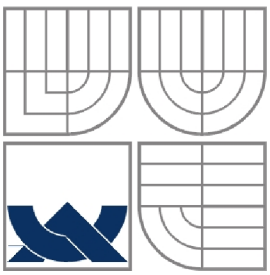
NÁVRH GUI PRO PRÁCI S DATABÁZEMI POSTGRESQL

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

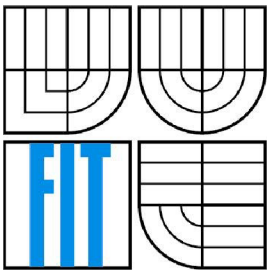
AUTOR PRÁCE
AUTHOR

ONDŘEJ DVOŘÁČEK

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NÁVRH GUI PRO PRÁCI S DATABÁZEMI POSTGRESQL

DESIGN OF GUI FOR DATABASE POSTGRESQL

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

ONDŘEJ DVOŘÁČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ROMAN LUKÁŠ, Ph.D.

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Dvořáček Ondřej**
Obor: Informační technologie
Téma: **Návrh GUI pro práci s databázemi PostgreSQL**
Kategorie: Databáze

Pokyny:

1. Seznamte se se stávajícími grafickými rozhraními pro databáze.
2. Navrhněte vlastní grafické rozhraní pro databázi PostgreSQL.
3. Implementujte toto navržené grafické uživatelské rozhraní pro databáze PostgreSQL. Výsledná aplikace by měla umět základní a rozšířené operace s tabulkami.
4. Srovnejte funkcionality s dostud existujícími aplikacemi.
5. Zhodnoťte dosažené výsledky a uveďte možnosti rozšíření této aplikace do budoucnosti.

Literatura:

- Dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1) a 2).

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Lukáš Roman, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Ondřej Dvořáček**
Id studenta: 84413
Bytem: Blatenská 34, 539 01 Hlinsko
Narozen: 14. 04. 1985, Chrudim
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Návrh GUI pro práci s databázemi PostgreSQL
Vedoucí/školitel VŠKP: Lukáš Roman, Ing., Ph.D.
Ústav: Ústav informačních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Bakalářská práce popisuje návrh a implementaci grafického uživatelského rozhraní pro práci s databázovým systémem PostgreSQL. Výsledná aplikace umožňuje uživateli provádět základní a rozšířené operace s tabulkami. Návrh rozhraní je proveden pomocí programu Glade, který slouží k navrhování aplikací pro grafickou knihovnu GTK+. K implementaci aplikace je pak použit programovací jazyk C a knihovna pro přístup k databázovému serveru libpq. Cílem této práce je především vytvoření volně šiřitelného a přenositelného grafického rozhraní, které bude možné dále rozšiřovat.

Klíčová slova

grafické uživatelské rozhraní, GUI, databáze, PostgreSQL, postgres

Abstract

The bachelor's thesis describes the design and implementation of graphical user interface working with database system PostgreSQL. The final application allows users to execute basic and extended operations with tables. The design of this interface was built-up with Glade, programme which enables the development of user interfaces for the library GTK+. For the implementation of this application were used programming language C and library for accessing database server libpq. The main aim of this work is to develop free and multiplatform graphical interface that might be extended later.

Keywords

graphical user interface, GUI, database, PostgreSQL, postgres

Citace

Ondřej Dvořáček: Návrh GUI pro práci s databázemi PostgreSQL, bakalářská práce, Brno, FIT VUT v Brně, 2007

Návrh GUI pro práci s databázemi PostgreSQL

Prohlášení

Prohlašuji, že jsem pod vedením Ing. Romana Lukáše, Ph.D., vypracoval tuto bakalářskou práci samostatně a uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Dvořáček
15. 5. 2007

Poděkování

Děkuji panu Ing. Romanu Lukášovi, Ph.D., za jeho rady a pomoc při tvorbě této bakalářské práce, a také společnosti Red Hat Czech s.r.o., zejména panu Ing. Radku Vokálovi, za poskytnuté konzultace a prostředí pro tvorbu této práce.

Obsah

Úvod

Grafické uživatelské rozhraní je jedním ze způsobů komunikace člověka s počítačem. Z počátečních písmen anglického překladu tohoto spojení, Graphical User Interface, vznikla všeobecně používaná zkratka GUI. Název této bakalářské práce – Návrh GUI pro práci s databázemi PostgreSQL – tedy napovídá, že cílem práce bude navrhnout a následně implementovat grafické uživatelské rozhraní, které bude schopné realizovat základní operace pro práci s databázemi PostgreSQL.

V první kapitole začneme nejdříve tím, že se zmíníme o způsobech, jakými lze vyměňovat informace mezi člověkem a počítačem, a na základě toho pak uvedeme hlavní důvody vzniku grafických uživatelských rozhraní a základní pravidla pro jejich návrh.

Seznámení s již existujícími rozhraními pro práci s databázemi je věnována kapitola druhá. Zde se jednak podíváme na různé databázové systémy a pro ně dostupná administrační či vývojová grafická rozhraní, ale hlavně podrobněji prozkoumáme funkce nabízené rozhraními se zaměřením na databáze PostgreSQL.

Kapitola třetí pak představuje základní vlastnosti databázového systému PostgreSQL a přehled objektů a funkcionalit, které tato databáze implementuje.

V kapitole čtvrté je popsán vlastní návrh grafického uživatelského rozhraní. Jsou zde také specifikovány funkce, které by aplikace měla být schopna provádět, a možnosti přístupu k těmto funkcím pomocí navrženého rozhraní.

Konkrétní implementaci navrženého rozhraní je věnována kapitola čtvrtá. V několika podkapitolách jsou zde popsány použité nástroje a způsoby, jakými bylo docíleno některých základních funkcí výsledné aplikace.

Nakonec se pokusíme v poslední kapitole srovnat výslednou aplikaci s již existujícími rozhraními, která jsou podrobněji popsána v kapitole druhé. Jsou zde uvedeny i náměty na další rozšíření aplikace a možnosti jejího dalšího vývoje.

1 Grafická uživatelská rozhraní

Komunikace člověka s počítačem může být vysvětlena jako obousměrný přenos informací mezi člověkem a strojem. Tuto výměnu zajišťují, při toku informací od počítače k člověku, výstupní periferní zařízení a vstupní periferní zařízení, pro přenos informací od člověka k počítači. Člověk a počítač jsou tedy navzájem propojeni takzvanými komunikačními kanály, kterými proudí jednotlivé informace. Při studiu těchto kanálů nás především zajímá, zda-li je pomocí nich možné přenášet určitou užitečnou informaci a zda je tato informace přenesena dostatečně rychle a pro člověka přijatelně. Mírným prohloubením této problematiky se v následujících dvou kapitolách pokusíme vysvětlit základní důvody vzniku grafických uživatelských rozhraní.

1.1 Komunikační kanály vhodné pro přenos informací mezi člověkem a počítačem

Při tvorbě rozhraní pro práci člověka se strojem je třeba se zaměřit na ty kanály, pomocí nichž lze dané informace přenášet v reálném čase, reprodukovatelně, a pokud je to možné, s co nejmenším zkreslením. Jedním z dalších významných požadavků je také technická realizovatelnost a spolehlivost přenosu.

Člověk je schopen přijímat informace od počítače výhradně pomocí svých smyslů. V současné době jsou však pro komunikaci vhodné jen některé z nich, a to:

- Zrak – Přenosovým médiem je obraz. Díky své vysoké informační propustnosti a možnosti náhodného přístupu pozorovatele k jednotlivým informacím, které daný obraz obsahuje, se jedná o nejvýhodnější komunikační kanál.
- Sluch – Jako médium je použit zvuk. Je vhodný pro přenos menšího množství informací. Pro svou nižší propustnost se používá převážně jako doplnění obrazové komunikace.
- Hmat – V současné době je využíván výjimečně, zejména jako prostředek pro dorozumívání nevidomých se stroji.

Při komunikaci opačným směrem, tedy od člověka ke stroji, jsou opět využívány lidské smysly, avšak jejich využití je odlišné:

- Hmat – Jedná se o nejobvyklejší způsob přenosu informací. Ke komunikaci jsou zde nejčastěji použity klávesnice v nejrůznějších podobách a spousta dalších doplňkových zařízení, například myš.
- Řeč – Jako přenosové médium se používá zvuk. V dnešní době patří mezi velice perspektivní způsoby komunikace člověka k počítači, který by postupem času mohl silně

konkurovat klávesnicím. Dosud však není dostatečně vyřešeno porozumění strojů lidské řeči.

- Gesta – Informace pomocí gest je přenášena obrazovým médiem. Jedná se však o nejméně vyvinutý způsob komunikace a jeho nasazení k běžnému používání je zatím velmi vzdálené.

1.2 Grafická rozhraní

Grafická rozhraní jsou v současné době nejpoužívanějším způsobem předávání informací mezi uživatelem a počítačem. Důvod jejich masivního rozšíření plyne z poznatků předchozí kapitoly 1.1, ve které je obrazový komunikační kanál od počítače k člověku označen jako informačně nejpropustnější.

V důsledku používání grafických uživatelských rozhraní se tak práce s počítačovými programy stala snadnější a efektivnější, což významně napomohlo rozšíření počítačů mezi širší okruh veřejnosti.

Protože jde u těchto rozhraní především o urychlení práce s aplikacemi, je třeba správně spojit grafickou podobu aplikace s její funkčností tak, aby práce s ní byla co nejjednodušší a aby byl uživatel schopný se v ní bezproblémově orientovat.

Kromě těchto požadovaných vlastností je třeba brát v úvahu i jiné aspekty, mezi které patří zejména vhodná volba barev. Můžeme tak docílit například uklidnění uživatele klidnými barvami (například zelenou), nebo naopak jeho zalarmování (například barvou červenou). Nevhodnou kombinací barev však můžeme vyvolat nervozitu uživatele a tím snížit jeho efektivitu práce.

Návrh grafických rozhraní je tedy vcelku složitý proces, který zaslouží nemalou pozornost. Správně navržené rozhraní je totiž významným měřítkem kvality grafických aplikací.

2 Současná grafická rozhraní pro práci s databázemi

Grafická uživatelská rozhraní pro práci s databázemi jsou počítačové programy, které jsou schopné spravovat metadata[1] databázových systémů. Některá rozhraní umožňují provádět i základní a rozšířené operace nad daty uloženými v databázích (například výběr či vkládání dat). Takových grafických rozhraní je velké množství a jsou rozdělena především podle databázového systému, který administrují. Z toho plyne, že jsou to většinou rozhraní specializovaná pouze na konkrétní databázový systém.

Počítačových programů, které jsou nezávislé na určitém databázovém systému, je velice málo. Proto vyvinutí grafického rozhraní, které by pracovalo s jakýmkoliv databázovým systémem se stejnou rychlostí a souborem poskytovaných operací jako specializované rozhraní, by mohlo být námětem do budoucnosti. Velkým přínosem takového univerzálního rozhraní by bylo určité sjednocení operací nad databázovými systémy a struktury získaných informací o metadatech těchto systémů.

2.1 Příklady rozhraní pro některé databázové systémy

V této kapitole jsou uvedeny příklady databázových systémů a stručný přehled grafických rozhraní pro jejich administraci. Z výčtu těchto aplikací je patrná silná převaha komerčních nástrojů.

2.1.1 Oracle

Oracle je komerční relační databázový systém vyvíjený firmou Oracle Corporation. Pracuje pod všemi operačními systémy. Dostupná rozhraní jsou například:

- Aqua Data Studio for Oracle – komerční, více na [13]
- DB Tools for Oracle – komerční, více na [14]
- Oracle Developer Tool – komerční, více na [15]
- SQL Manager for Oracle – komerční, více na [8]
- a další

[1] metadata – informace o datech

2.1.2 DB2

Relační databázový systém DB2 od firmy IBM je také komerční a pracuje na operačních systémech Unix, Windows a Linux. Příklady grafických rozhraní jsou:

- Aqua Data Studio for DB2 – komerční, více na [13]
- a další

2.1.3 SQL Server

SQL Server je jedním z dalších komerčních relačních databázových systémů. Je vyvíjen firmou Microsoft a pracuje pouze pod operačním systémem Microsoft Windows. Existující rozhraní jsou:

- Aqua Data Studio for SQL Server – komerční, více na [13]
- SQL Manager for SQL Server – komerční, více na [8]
- a další

2.1.4 MySQL

Volně šiřitelný relační databázový systém MySQL firmy MySQL AB pracuje pod všemi operačními systémy. Mezi grafická rozhraní patří například:

- Aqua Data Studio for MySQL – komerční, více [13]
- SQL Manager for MySQL – komerční, více na [8]
- phpMyAdmin – volně šiřitelný s otevřeným zdrojovým kódem, více na [16]
- a další

2.2 Příklady rozhraní pro databázový systém PostgreSQL

PostgreSQL je objektově-relační databázový systém, který je volně šiřitelný. Hlubšímu poznání toho systému je věnována kapitola 3. Nyní si v jednotlivých podkapitolách pouze rozebereme některá z dostupných grafických rozhraní pro práci s tímto databázovým systémem.

2.2.1 SQL Manager for PostgreSQL

SQL Manager for PostgreSQL je komerční grafické uživatelské rozhraní pro operační systém Microsoft Windows. Hlavními vlastnostmi této aplikace jsou:

- efektivní grafické a textové nástroje pro tvorbu SQL dotazů
- rychlá správa databáze a navigace mezi databázovými objekty
- rozšířené nástroje pro manipulaci s daty

- plná podpora PostgreSQL do verze 8.1
- jednoduchá správa všech databázových objektů
- efektivní správa bezpečnosti
- exportování a importování dat
- grafický nástroj pro návrh databáze
- připojování k databázi pomocí SSH tunelu
- přístup k databázi pomocí protokolu HTTP

Více informací je možné získat na internetových stránkách programu [8].

2.2.2 PG Explorer

Volně šiřitelné grafické rozhraní PG Explorer je implementováno rovněž pouze pro operační systém Microsoft Windows. Jeho hlavními vlastnostmi jsou:

- strom obsahující databáze a všechny databázové objekty
- zobrazení SQL kódu u jednotlivých objektů
- průvodce pro vytvoření SQL dotazů na různé objekty
- zvýrazňování syntaxe
- doplňování kódu SQL dotazů

Více informací je možné se dozvědět na internetové stránce [9].

2.2.3 Navicat

Navicat je silný administrační a vývojový nástroj. Je však komerční a pracuje pouze pod operačními systémy Microsoft Windows a Mac OS X. Mezi jeho přednosti patří:

- připojení pomocí SSH a HTTP tunelu
- prohlížení databáze
- vytváření databázových objektů
- vytváření a provádění SQL dotazů
- spravování uživatelských oprávnění
- přenos databáze z jednoho PostgreSQL serveru na druhý
- importování a exportování dat
- zálohování a obnova databáze

Pro více informací lze navštívit internetovou stránku [10].

2.2.4 pgAdmin III

Administrační a vývojový nástroj pgAdmin III je volně šiřitelný s otevřeným zdrojovým kódem. Pracuje pod operačními systémy Microsoft Windows, Linux, FreeBSD, Mac OS X, OpenBSD a Solaris. Nejdůležitější vlastnosti jsou:

- podpora PostgreSQL 7.3 a vyšší
- obsáhlá dokumentace
- přes tucet úplných překladů a dvacet pět částečných
- kvalitní nástroj pro tvorbu SQL dotazů se zvýrazňováním syntaxe
- možnost upravování konfiguračních souborů
- přístup ke všem databázovým objektům

Další vlastnosti jsou dostupné na internetové stránce [11]. Jelikož je tento nástroj pro správu databáze PostgreSQL jediným newebovým, volně šiřitelným a multiplatformním[2] grafickým rozhraním, budeme ho v dalším textu používat jako program referenční.

2.2.5 phpPgAdmin

Webový administrační nástroj phpPgAdmin je volně šiřitelný a nezávislý na operačním systému. Poskytuje řadu užitečných vlastností:

- administrace více serverů
- podpora PostgreSQL od verze 7.0.x do 8.2.x
- správa všech databázových objektů, uživatelů a skupin, ...
- jednoduchá manipulace s daty: procházení tabulek a pohledů
- vykonávání SQL dotazů
- importování SQL skriptů
- vynikající jazyková podpora: dostupný ve dvaceti sedmi jazycích
- jednoduchá instalace a konfigurace

Doplňující informace jsou k dispozici na internetové stránce [12].

[2] multiplatformní – program pracující pod více operačními systémy

3 Databáze PostgreSQL

3.1 Úvodem

Databáze PostgreSQL vznikla jako následník systému Ingres (Interactive Graphics and Retrieval System), který byl vyvinut na kalifornské univerzitě v Berkley. Tento projekt byl pak později rozvíjen pod názvem Postgres.

Jméno PostgreSQL se objevilo poprvé na konci roku 1996, kdy byl přejmenován projekt Postgres95, který vznikl rozšířením původního Postgresu o podporu jazyka SQL. Tyto projekty už pak byly dále vyvíjeny s otevřeným zdrojovým kódem (anglicky open source).

Postupným přidáváním nových funkcí a vlastností se PostgreSQL stalo produktem patřícím mezi špičky databázových systémů.

3.2 Vlastnosti PostgreSQL

PostgreSQL je plnohodnotný objektově-relační databázový systém s otevřeným zdrojovým kódem. K jeho vynikající pověsti přispívá nejen více než patnáct let aktivního vývoje, ale hlavně jeho spolehlivost a bezpečnost.

Nesporným důkazem kvality tohoto databázového systému je také jeho široké spektrum přenositelnosti. Jeho funkcí mohou vyžívat uživatelé na téměř všech rozšířených operačních systémech včetně Linuxu, různých distribucí UNIXů (např. AIX, BSD, Mac OS X, Solaris, ...) a Microsoft Windows.

3.2.1 Podpora transakcí

Při hodnocení databázového sytému se rovněž posuzuje jakým způsobem splňuje základní vlastnosti transakce označené *ACID*. Zkratka vznikla spojením počátečních písmen anglických názvů čtveřice vlastností transakce:

- *Atomicity* (atomičnost) – zajišťuje, že transakce představuje z hlediska provedení nedělitelný celek.
- *Consistency* (konzistence) – znamená, že transakce, která není ovlivněna žádnými jinými souběžně probíhajícími transakcemi (tzv. izolovaná transakce), neporušuje konzistenci databáze.
- *Isolation* (izolovanost) – zajišťuje, že při průběhu několika souběžných transakcí budou tyto transakce provedeny jakoby probíhaly jedna za druhou a tím udržuje databázi v konzistentním stavu.

- *Durability* (trvanlivost) – znamená, že po skončení transakce budou mít všechny změny, které transakce provedla, trvalý charakter a to i při výpadku databázového systému.

U PostgreSQL jsou tyto podmínky splněny stoprocentně, čímž se řadí mezi kvalitní databázové systémy.

3.2.2 Funkce

Funkce jsou ve skutečnosti určité bloky příkazů, které mohou být provedeny databázovým serverem. Tyto části kódu mohou být také psány pomocí jazyka SQL. Avšak nedostatek základních programovacích operací, jako je větvení nebo použití smyček, vedl k umožnění použití i ostatních programovacích jazyků ve funkcích, mezi které patří:

- Vestavěný jazyk nazvaný PL/pgSQL (připomíná procedurální jazyk databázového systému Oracle, PL/SQL).
- Skriptovací jazyky (např. PL/Perl, plPHP, PL/Python, PL/Ruby, PL/sh, PL/Tcl nebo PL/Scheme).
- Jazyky kompilované (překládané, zejména C, C++ a Java, popřípadě PL/Java).
- Statistický jazyk R včetně své modifikace pro PostgreSQL, PL/R.

3.2.3 Indexy

Databázové indexy jsou datové struktury, které výrazně zrychlují operace s tabulkami. U serveru PostgreSQL jsou podporovány následující vlastnosti indexů:

- Indexy je možné prohledávat pozpátku, kdykoliv je to nutné, tudíž systém nepotřebuje vytvářet pomocné indexy (např. pro podporu seřazení záznamů pozpátku příkazem `ORDER BY field DESC`).
- Výrazový index (anglicky expression index) je databázový index, který není postavený na seznamu sloupců tabulek, ale na obecném výrazu. To dovoluje definovat indexy závislé na datech v tabulce, která zde však ve skutečnosti nemusí být uložena.
- Částečný index (anglicky partial index) může být vytvořen pomocí přidání podmínky `WHERE` na konec příkazu `CREATE INDEX`. Tím lze docílit, že index bude ukazovat pouze na tu část tabulky, která nás zajímá a tím se zmenší i velikost nově vytvořeného indexu.

Mezi další výhody patří například schopnost vytvářet uživatelem definované indexové metody. Lze však také využít již předdefinovaných vestavěných funkcí (např. B-tree, hash table či GiST).

3.2.4 Triggery

Triggery jsou události spouštěné nějakou akcí SQL dotazu. Například při vkládání dat do databáze pomocí příkazu `INSERT` mohou být kontrolována vstupní data.

V PostgreSQL jsou triggery plně podporovány a mohou být připojovány k tabulkám. U pohledů to však možné není, zde jsou ale k dispozici jiné prostředky, kterými lze docílit podobných výsledků (např. rules). Vícenásobné triggery jsou spouštěny v abecedním pořadí.

Kromě toho, že triggery umožňují volat funkce napsané v nativním jazyku PL/pgSQL, mohou volat i funkce jiných jazyků (např. PL/Perl).

3.2.5 Multi-Version Concurrency Control

Konkurentní přístup k databázi PostgreSQL je zajišťován systémem zvaným Multi-Version Concurrency Control (MVCC). Uživatelům jsou předloženy takzvané „snímky“ (anglicky snapshots) databáze, ve kterých mohou provádět různé změny. Provedené úpravy pak musí být potvrzeny, jinak zůstanou skryty ostatním uživatelům. Tím je nejen rapidně snížen počet čtecích zámků, ale také zaručeno dodržení principů *ACID* (viz kapitola 3.2.1).

3.2.6 Rules

Rules jsou určitá pravidla pro vykonávání SQL dotazů, která umožňují přepisovat vykonávaný SQL dotaz. Této vlastnosti lze především využít při implementování aktualizovatelných pohledů (anglicky views).

3.2.7 Datové typy

Množina vestavěných datových typů je v PostgreSQL velice široká, zahrnující například i:

- texty neomezené délky
- základní geometrické prvky
- adresy IPv4 a IPv6
- hardwarové adresy (tzv. MAC adresy)
- pole

Navíc má uživatel možnost definovat své vlastní datové typy.

3.2.8 Uživatelem definované objekty

Nové typy téměř všech databázových objektů poskytovaných PostgreSQL mohou být definovány uživatelem. Patří mezi ně například indexy, operátory, agregáty nebo přetypování.

3.2.9 Dědičnost

Tabulky v PostgreSQL mohou být děděny z ostatních tabulek. Pokud tomu tak je, jsou veškerá data mezi původní a zděděnou tabulkou sdílena. To znamená, že pokud jsou do (z) jedné tabulky data vložena (smazána), jsou automaticky vložena (smazána) i v tabulce druhé. Také přidání sloupce

v jedné tabulce se projeví přidáním stejného sloupce do tabulky druhé. V tomto případě však ne zcela přesně, neboť v současné době PostgreSQL nepodporuje dědění omezení (anglicky constraints).

3.2.10 Ostatní vlastnosti

Mezi další podporované vlastnosti patří například podpora integritních omezení, včetně cizích klíčů, dále pohledů, spojování tabulek, šifrovaného připojení pomocí SSL, ukládání binárních a textových objektů, tvorby domén a mnoho dalších. Podrobnější informace lze získat například na této internetové stránce [7].

4 Návrh vlastního grafického uživatelského rozhraní

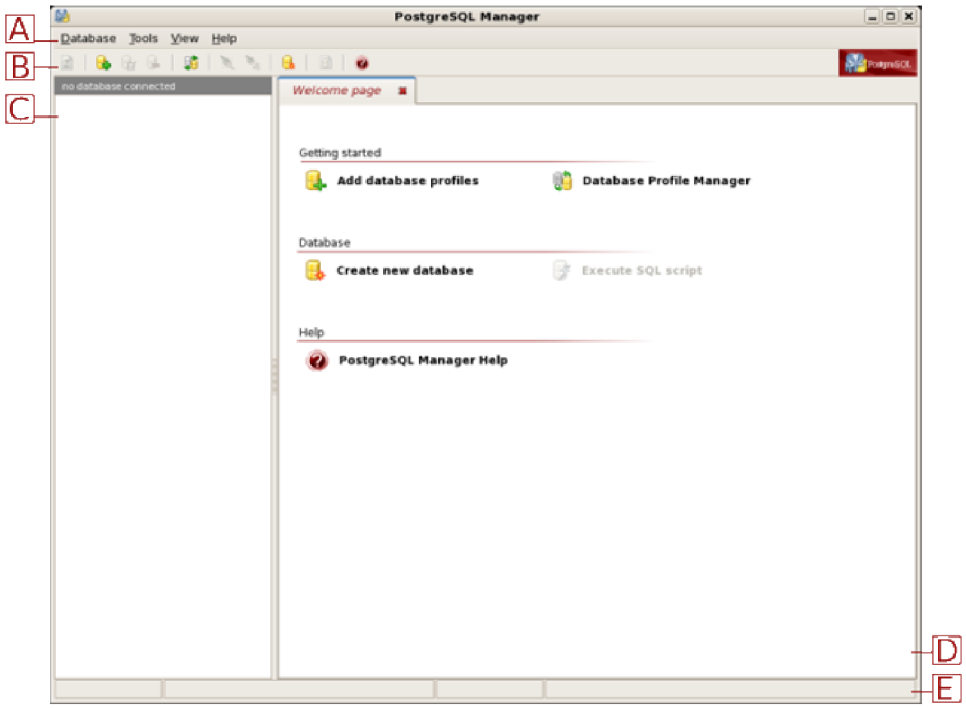
Při zkoumání současných grafických uživatelských rozhraní v kapitole 2 jsme zjistili, jaké grafické prvky a možnosti práce s databázemi by měly být zahrnuty do plnohodnotného a uživatelsky příjemného rozhraní.

Pro implementaci výsledné aplikace byla použita grafická knihovna GTK+ (viz kapitola 5.1.1) a program pro návrh rozhraní pomocí této knihovny – Glade (viz kapitola 5.1.2). Její použití bylo zvoleno především pro její spolehlivou přenositelnost.

Při následujícím návrhu je tedy třeba brát v úvahu možnosti, které nám dává grafická knihovna k dispozici a na základě toho vytvořit výsledný vzhled aplikace.

4.1 Základní rozložení aplikace

Zvolením správného rozložení aplikace bychom měli docílit přehledné a efektivní práce s rozhraním. Bylo tedy použito rozložení založené na jednom hlavním okně s případnými dalšími dialogy pro doplňující funkce. V tomto hlavním okně aplikace musí být možné jednak intuitivně přistupovat k jednotlivým funkcím aplikace a dále snadno prohlížet databázové objekty, spravovat jejich vlastnosti a provádět ostatní operace s nimi související.



Obr. 4.1: Základní rozložení aplikace

Výsledný návrh rozložení tak obsahuje sekci s hlavním menu (část A na obr. 4.1), dále lištu základních a často používaných nástrojů (část B na obr. 4.1), část se stromem databázových objektů (část C na obr. 4.1) a hlavní okno (část D na obr. 4.1), ve kterém budou zobrazovány informace o databázových objektech a rozhraní doplňkových funkcí. Pro případné rozšíření je zde přidána i stavová lišta zobrazující důležité informace o stavu aplikace (část E na obr. 4.1).

4.2 Hlavní menu

Hlavní menu bylo navrženo tak, abychom se s jeho pomocí dostali k základním funkcím aplikace. V následujícím seznamu jsou zobrazeny jednotlivé položky a stručný popis jejich významu:

- *Database* – nabídka funkcí pro práci s databázemi
  - *Create Database Profiles* – spustí průvodce vytvoření nových databázových profilů
  - *Edit Database Profile* – spustí *Database Profile Manager*, nástroj pro správu a upravování databázových profilů
  - *Remove Database Profile* – odstraní vybraný databázový profil
  - *Connect To Database* – připojí se k vybrané databázi a získá informace o jejích objektech
  - *Disconnect From Database* – odpojí vybranou databázi
  - *Create New Database* – spustí průvodce vytvořením nové databáze
  - *Drop Database* – odstraní databázi
  - *Exit* – ukončí aplikaci
- *Tools* – nabídka nástrojů
  - *Database Profile Manager* – spustí nástroj pro správu databázových profilů
  - *SQL Editor* – spustí SQL editor
- *View* – nabídka možných úprav vzhledu a zobrazení určitých prvků rozhraní
  - *Refresh* – obnoví informace o datech vybraného databázového profilu
  - *Show Tool Panel* – skryje či zobrazí lištu nástrojů
  - *Show Welcome Page* – skryje či zobrazí uvítací obrazovku
  - *Show Status Bar* – skryje či zobrazí stavovou lištu
- *Help* – nabídka nápovědy
  - *PostgreSQL Manager Help* – zobrazí programovou nápovědu
  - *About* – zobrazí základní informace o aplikaci

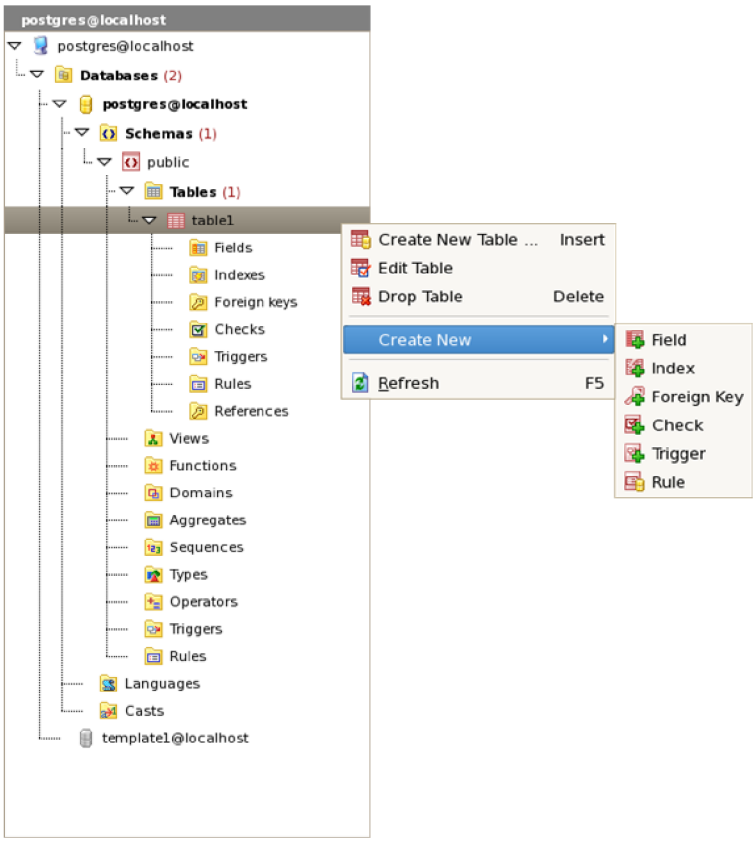
4.3 Lišta nástrojů

Lišta nástrojů by měla umožňovat o něco rychlejší přístup k základním funkcím aplikace než pomocí hlavního menu. Zde byly však vybrány pouze některé (nejčastěji používané) funkce: *Refresh*, *Create Database Profiles*, *Edit Database profile*, *Remove Database profile*, *Database Profile Manager*, *Connect To Database*, *Disconnect From Database*, *Create New Database*, *SQL Editor* a *PostgreSQL Manager Help*. Operace, které pod sebou skrývají jsou totožné se stejnojmennými položkami hlavního menu. Výsledný vzhled je zobrazen na následujícím obrázku 4.2.

Obr. 4.2: Lišta nástrojů

4.4 Strom databázových objektů

Pro zobrazení hierarchie objektů v databázi a navigaci mezi nimi byl navržen stromový prohlížeč (viz obr. 4.3). Jednotlivé objekty jsou zde organizovány do složek, které jsou navíc doplněny informací o počtu objektů v nich obsažených. Pro dosažení lepší orientace jsou navíc jména složek a objektů doplněna o specifické ikony vysvětlující jejich význam a jména těch složek, ve kterých je obsažen alespoň jeden databázový objekt, jsou zvýrazněna tučným fontem.



Obr. 4.3: Strom databázových objektů

Nad prohlížečem byla navíc navržena lišta ukazující jméno aktuálně připojené databáze, čehož může uživatel využít především při správě rozsáhlých stromů objektů (například při provádění operací nad databází jako celkem).

Pro databázový systém PostgreSQL byla navržena tato hierarchie složek databázových objektů:

- *„server“* – jméno konkrétního serveru, na kterém je databáze uložena
  - *Databases* – databáze
    - *Schemas* – schémata
      - *Tables* – tabulky
        - *Fields* – sloupce v tabulce
        - *Indexes* – indexy
        - *Foreign Keys* – cizí klíče
        - *Checks* – omezení
        - *Triggers* – databázové triggery
        - *Rules* – pravidla pro vykonávání SQL dotazů nad tabulkou
        - *References* – seznam cizích klíčů, které se odkazují na tabulku
      - *Views* – pohledy
        - *Fields* – sloupce pohledu
        - *Rules* – pravidla pro vykonávání SQL dotazů nad pohledem
      - *Functions* – funkce
      - *Domains* – domény
        - *Checks* – omezení
      - *Aggregates* – agregáty
      - *Sequences* – číselné sekvence
      - *Types* – datové typy
      - *Operators* – operátory
      - *Triggers* – seznam databázových triggerů ve všech tabulkách
      - *Rules* – seznam pravidel ve všech tabulkách a pohledech
    - *Languages* – jazyky, například pro psaní definic funkcí
    - *Casts* – přetypování mezi datovými typy

4.4.1 Vytahovací menu pro položky ve stromu

Ke spravování každého z objektů obsaženého ve stromovém prohlížeči byla vytvořena vytahovací menu. S jejich pomocí lze upravovat nebo rušit vybraný databázový objekt, vytvořit zcela nový objekt stejného typu jako je vybraný nebo aktualizovat informace o objektech z databázového systému.

Některá menu mají přidána několik dalších položek a to zejména pro vytváření svých podobjektů. Příklad takového menu je na obrázku 4.3.

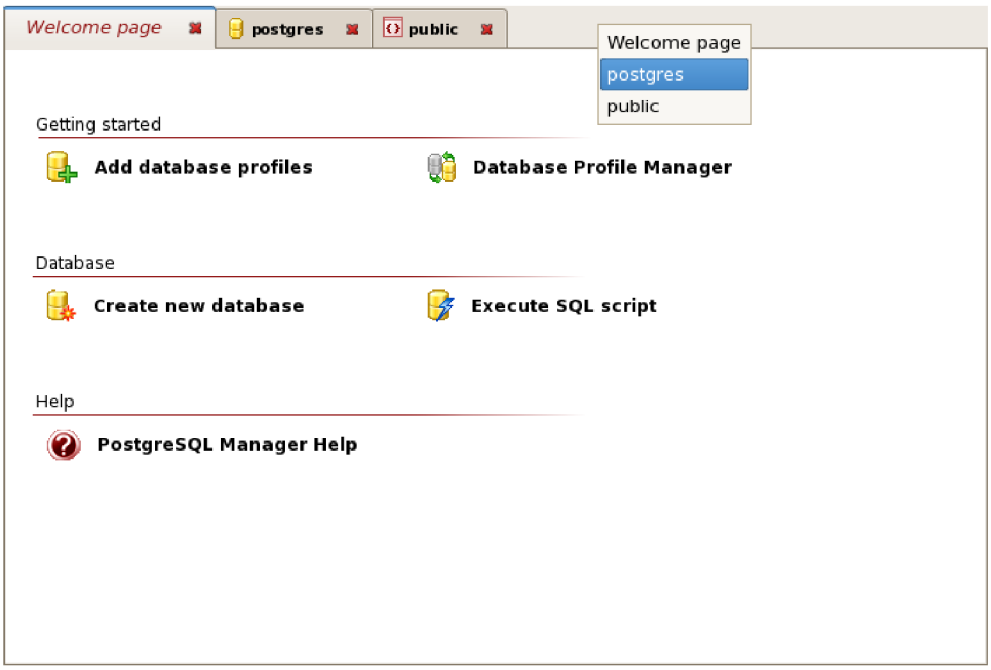
4.5 Hlavní okno aplikace

Jedná se o nejdůležitější část aplikace. V hlavním okně jsou zobrazovány veškeré informace o objektech, data tabulek, případně doplňkové funkce programu, jako je *Database Profile Manager* a *SQL Editor*.

Podokna těchto objektů a nástrojů jsou organizována v podobě záložek (viz obr. 4.4), jejichž nadpisy byly pro zlepšení přehlednosti doplněny o patřičné ikony.

K snadnějšímu pohybu mezi jednotlivými panely bylo přidáno vytahovací menu obsahující odkazy na všechna otevřená podokna. Tím je uživateli umožněn rychlý přechod na požadovanou záložku i v případě velkého množství panelů.

Otevřená okna lze potom zavírat tlačítkem umístěným vedle nadpisu každého z panelů.



Obr. 4.4: Rozložení záložek a uvítací obrazovky

4.5.1 Uvítací obrazovka

Navržením uvítací obrazovky (v aplikaci „*Welcome page*“) je uživatelům dána možnost snadnější orientace v aplikaci, především v začátcích používání tohoto grafického rozhraní (viz obr. 4.4).

Obrazovka nabízí odkazy na operace a nástroje rozdělené do následujících tří skupin:

- *Getting started* – soubor funkcí pro úplný začátek práce s programem
  - *Add database profiles* – spustí průvodce vytvořením nových databázových profilů
  - *Database Profile Manager* – spustí nástroj pro správu databázových profilů

- *Database* – soubor základních funkcí týkající se databáze
  - *Create new database* – spustí průvodce vytvořením nové databáze
  - *Execute SQL script* – spustí SQL editor, umožňující vykonávat SQL příkazy
- *Help* – soubor odkazů k nápovědám o aplikaci
  - *PostgreSQL Manager Help* – zobrazí nápovědu

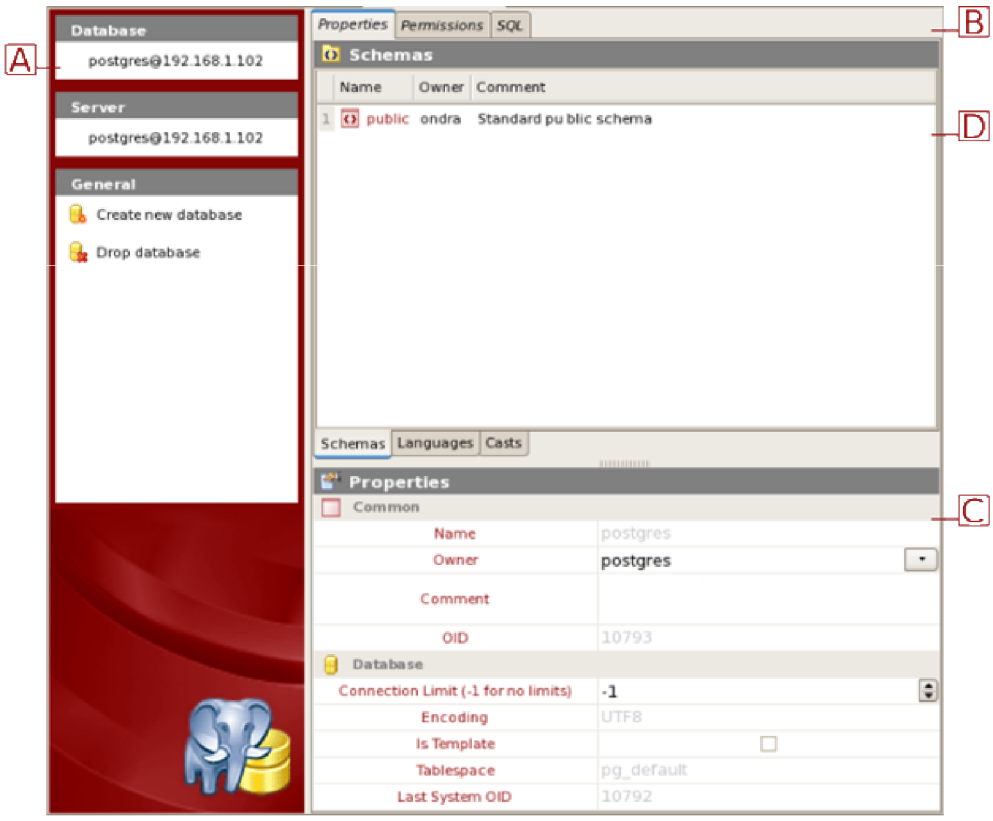
Pomocí zavíracího tlačítka nebo položky v hlavním menu lze tuto obrazovku skrývat a zobrazovat podle potřeby.

4.5.2 Panely vlastností jednotlivých databázových objektů

Informace získané o databázových objektech je potřeba uživateli zpřístupnit. Proto byl navržen panel vlastností jednotlivých objektů, který dává úplný přehled o vybraném objektu, jeho umístění v databázi a jeho vztahu k ostatním objektům.

Pomocí tohoto panelu má uživatel také možnost tyto vlastnosti upravovat a na základě potvrzení promítnout provedené změny do databáze.

V jedné z dalších částí panelu jsou zobrazeny ve stromových prohlížečích seznamy objektů, na kterých je objekt závislý, a naopak které objekty jsou závislé na prohlíženém objektu. Tyto stromové prohlížeče jsou uspořádány stejným způsobem jako hlavní prohlížeč objektů. Avšak jsou doplněny informací o typu závislosti.



Obr. 4.5: Panel vlastností databázových objektů

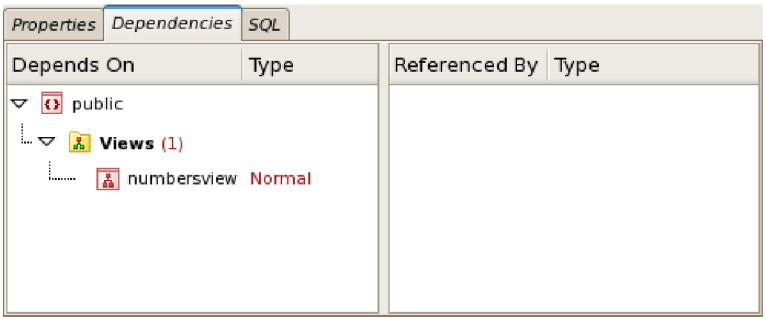
Pro uživatele vyvíjející aplikace s využitím databázového systému PostgreSQL, je přidán grafický prvek zobrazující se zvýrazněnou syntaxí SQL kód, pomocí něhož lze daný objekt vytvořit včetně všech jeho zobrazených vlastností.

Jako případné rozšíření zde byla také navržena část schopná zobrazovat oprávnění objektů.

4.5.2.1 Rozložení panelu

Panel vlastností je rozdělen na dvě základní části. V první je zobrazeno menu (část A na obr. 4.5), umožňující rušit zobrazený objekt, vytvářet nové, či případně aplikovat provedené změny. Uživatel je zde také informován o jménu daného objektu a jménu objektu, v němž je obsažen.

Druhá část je pak uspořádána v podobě záložek (část B na obr. 4.5). Mezi základní, které jsou zobrazeny u všech objektů patří *Properties* (vlastnosti) a *SQL*. V panelu *Properties* jsou zobrazeny vlastnosti objektů (část C na obr. 4.5) a může zde navíc přibýt grafický prvek s přehledem podobjektů prohlíženého objektu (část D na obr. 4.5). Záložka *SQL* pak zobrazuje výše zmíněný SQL kód pro vytvoření daného objektu. Dalšími záložkami jsou *Dependencies* (závislosti) a *Permissions* (oprávnění). Jejich zobrazení se řídí typem vybraného objektu. Příklad vzhledu záložky *Dependencies* je na obr. 4.6. Panel *Permissions* je zde pak přidán, jak už bylo napsáno dříve, pro možné rozšíření aplikace.



Obr. 4.6: Zobrazení závislostí databázového objektu

4.5.3 Správce databázových profilů

Jednotlivá připojení k databázím jsou v aplikaci spravována pod pojmem *databázový profil* (anglicky database profile). Jména těchto profilů jsou uživatelem definované aliasy[3]. Aby bylo možné měnit vytvořené profily, nebo je rušit, byl navržen speciální nástroj – *Database Profile Manager* (viz obr. 4.7).

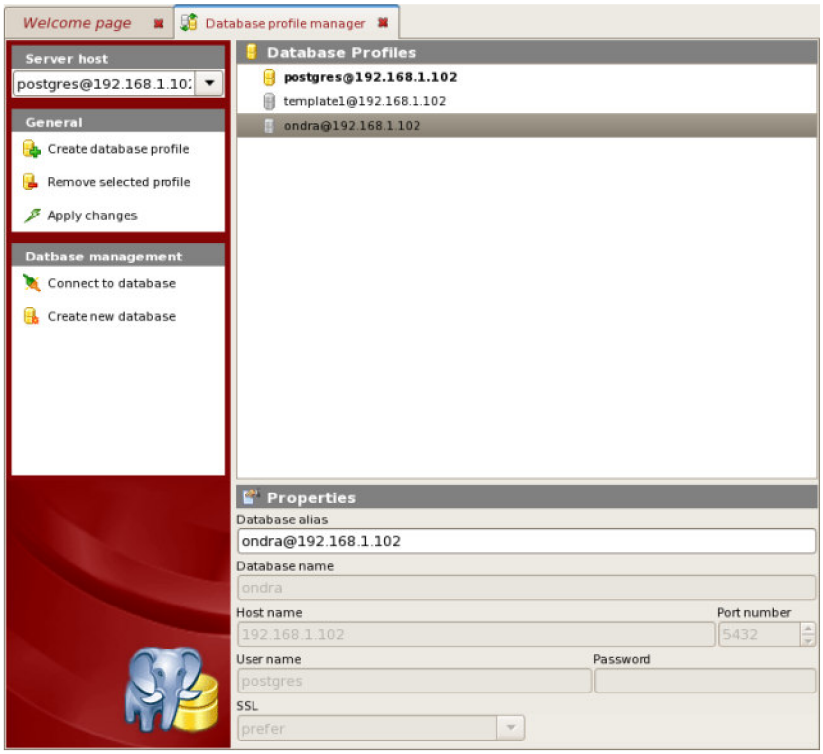
4.5.4 SQL editor

Pro případ, že by uživatel potřeboval provést nějaký svůj vlastní SQL kód, byl do aplikace přidán nástroj *SQL Editor* (viz obr. 4.8).

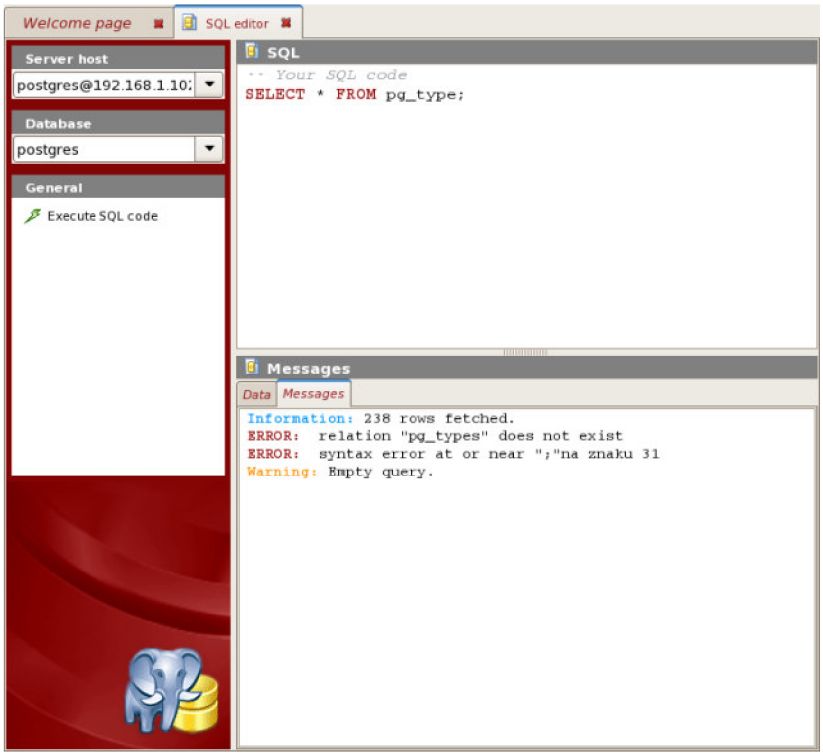
---

[3] alias – jméno, pod kterým chce uživatel spravovat databázi

Jedná se o jednoduchý editor se zvýrazněnou syntaxí, který provádí zadaný kód. Výsledky vykonání příkazu zobrazuje *SQL Editor* do speciálních záložek, jednak pro zprávy vrácené databázovým serverem (barevně odlišené podle naléhavosti) a jednak pro zobrazení vybraných dat z databáze.



Obr. 4.7: Nástroj pro správu databázových profilů – Database Profile Manager



Obr. 4.8: SQL Editor

4.6 Dialog vlastností objektů tabulek

Pro objekty, které jsou nedílnou součástí jiných objektů, byly vytvořeny speciální jednoduché dialogy (viz obr. 4.9). Obsahují pouze přehled základních informací o objektu, přičemž má uživatel možnost tyto údaje měnit.

Mezi takto zobrazované objekty patří *Fields* (sloupce tabulek), *Indexes* (indexy), *Foreign Keys* (cizí klíče) a *Checks* (omezení).

Obr. 4.9: Dialog vlastností objektů tabulek

4.7 Průvodce vytváření databázových objektů

Navržením průvodců je umožněno i méně zkušeným uživatelům vytvářet všechny databázové objekty. Průvodce vytvoří SQL kód, který pak lze prohlížet či upravit a následně provést. Příklad první strany takového průvodce je na obrázku 4.10.

Obr. 4.10: První strana průvodce vytvoření databázových profilů

4.8 Ostatní prvky rozhraní

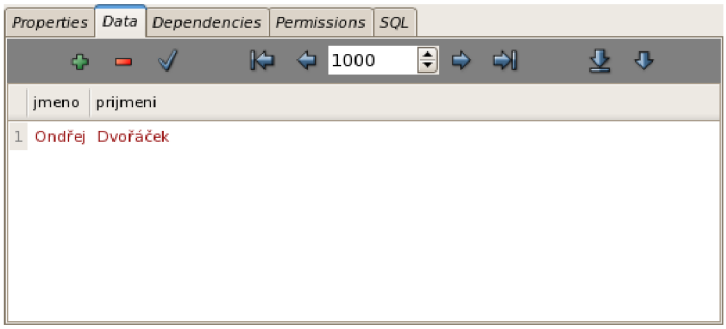
Celé navržené grafické rozhraní má ještě několik dalších prvků potřebných k správnému chodu aplikace. Jsou to především potvrzovací dialogy, dialog zadávání hesla pro připojení k databázi a grafická rozhraní potřebná k prohlížení dat v tabulkách, jejich filtrování, vkládání, mazání či upravování.

4.8.1 Prohlížení dat v tabulkách

Prohlížení dat uložených v jednotlivých tabulkách je přístupné z panelů vlastností těchto tabulek (viz obr. 4.11). Uživatel zde má k dispozici jednak možnost výběru všech záznamů nebo jen těch, které splňují určitou podmínku zadanou pomocí dialogu pro filtrování dat.

Vybraná data jsou pak stránkována po určitém, uživatelem zadaném, počtu záznamů. Mezi nimi se pak lze postupně pohybovat.

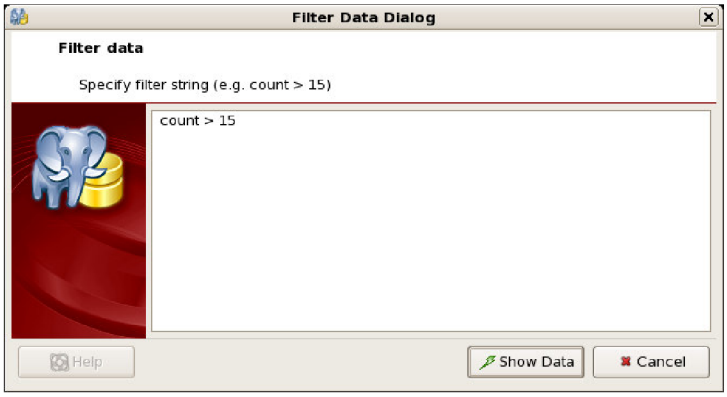
Pro vkládání a upravování dat je navržen další dialog stejného charakteru, jako jsou dialogy vlastností objektů.



Obr. 4.11: Zobrazení dat v tabulkách

4.8.1.1 Filtrování dat

Pomocí filtrování dat má uživatel možnost vybírat ta data z tabulek, která ho zajímají. Slouží k tomu speciální editor podmínky se zvýrazněnou syntaxí (viz obr. 4.12).



Obr. 4.12: Dialog pro filtrování zobrazených dat tabulek

5 Implementace navrženého rozhraní

5.1 Použité nástroje

Při volbě nástrojů pro tvorbu tohoto grafického rozhraní byla nejdůležitějším požadavkem přenositelnost výsledné aplikace. Z tohoto důvodu byla zvolena knihovna pro tvorbu grafických aplikací GTK+ a program pro návrh grafických rozhraní Glade.

Celá aplikace je implementována v programovacím jazyku C, který byl vybrán jednak pro jeho nezávislost na operačním systému a jednak proto, že je v něm napsána použitá grafická knihovna.

V několika následujících podkapitolách jsou popsány základní vlastnosti použitých nástrojů.

5.1.1 Grafická knihovna GTK+

Grafická knihovna GTK+ je multiplatformní nástroj pro tvorbu grafických uživatelských rozhraní. Nabízí kompletní sadu grafických prvků, využitelných při tvorbě malých i velice rozsáhlých aplikací.

GTK+ je volně šiřitelný software šířený pod licencí GNU LGPL, což umožňuje, aby byla tato knihovna použita i pro komerční využití bez jakýchkoli licenčních poplatků.

Je založena na následujících třech knihovnách vyvíjených týmem GTK+: GLib, Pango, ATK. Byla navíc od začátku tvořena nejen pro programovací jazyky C/C++, ale i pro jazyky jako jsou například Perl nebo Python.

Pro získání více informací je možné navštívit internetovou stránku [4].

5.1.2 Glade GUI builder

Program Glade je nástroj pro rychlou a jednoduchou tvorbu grafických uživatelských rozhraní pro GTK+ a GNOME. Je volně šiřitelný pod licencí GNU GPL s otevřeným zdrojovým kódem.

Rozhraní navržená v programu Glade jsou ukládána ve formátu XML a užitím knihovny libglade pak mohou být do aplikací načítány dynamicky podle potřeby.

Další informace jsou na internetové stránce [5].

5.1.3 Knihovna libpq

Pro komunikaci implementované aplikace s databázovým serverem PostgreSQL byla použita knihovna libpq. Jedná se o sadu knihovních funkcí v programovacím jazyce C, která dovoluje posílat SQL dotazy na databázový server a získávat zpět výsledky těchto operací.

Pro správné použití knihovny je třeba do programu vložit hlavičkový soubor libpq-fe.h.

Bližší informace o nabízených funkcích a způsobu práce s touto knihovnou lze nastudovat na internetové stránce [6].

5.2 Základní struktura aplikace

Celá aplikace je tvořena, jak už bylo zmíněno v předešlém návrhu, hlavním oknem, na jehož pozadí běží hlavní smyčka programu řízená knihovnou GTK+. Ta se stará především o zpracovávání signálů a událostí přijímaných aplikací (např. zmáčknutí klávesy, kliknutí myši apod.). Pro rozšířené zpracování signálů (např. spuštění správné operace při stisku klávesy) bylo nutné naimplementovat takzvané „callback“ funkce, tedy funkce vykonané v případě, kdy aplikaci přijde určitý signál, který je potřeba zaznamenat. Tyto funkce jsou tedy navázány na veškeré kontrolní prvky programu a starají se o spuštění správných akcí implementujících uživatelem očekávané funkcionality aplikace.

Po startu aplikace je třeba nejdříve provést počáteční inicializaci programu. Načítá se grafické rozhraní (uložené v XML souboru) knihovnou libglade, upravují se některé grafické prvky, nastavují globální proměnné a napojují se výše zmíněné „callback“ funkce na jednotlivé grafické prvky rozhraní.

Jakmile je aplikace inicializována, dostává se do ustáleného stavu a je schopna plnit požadavky uživatele.

5.3 Získávání informací z databáze

Komunikace mezi aplikací a databázovým serverem je implementována pomocí knihovny libpq. Díky poskytovaným funkcím touto knihovnou lze z databáze získat informace, které je potřeba zobrazit u jednotlivých databázových objektů.

K získání dat z databázového systému nestačí pouze knihovna, ale musíme vytvořit pro každý databázový objekt SQL dotaz, který dokáže vybrat hodnoty jeho vlastností z databázových katalogů či informačních schémat. Při tvorbě těchto dotazů bylo použito dokumentace databázového systému PostgreSQL 8.1 (viz [3]). Jejich přehled je možné shlédnout v příloze 1.

Po zajištění přesunu informací ze serveru do aplikace, je nutné zajistit také na straně klienta nějaké úložiště nově získaných dat. K tomuto účelu byly nadefinovány speciální struktury uchovávající v paměti programu vlastnosti existujících databázových objektů. Tyto datové struktury jsou pak dynamicky alokovány přesně podle příchozích informací do podoby navržené stromové hierarchie objektů popsané v kapitole 4.4.

Při běhu aplikace jsou informace o objektech získávány pouze v případě připojení grafického rozhraní k databázi nebo spuštění funkce zajišťující obnovení načtených dat (*Refresh*).

Mimo tyto dva případy se získávají informace z databáze zcela výjimečně, a to například v průvodcích vytváření nových objektů pro nabídnutí všech možných hodnot určité vlastnosti objektu uživateli. Tyto přístupy k databázi jsou však minimální.

5.4 Tvorba, upravování a mazání databázových objektů

Tvorba nových databázových objektů probíhá, jak už bylo popsáno, pomocí průvodců vytvoření nových objektů. Na jejich konci je vytvořen takový SQL dotaz, který je schopný zcela přesně nadefinovat nový objekt v právě připojené databázi podle požadavků uživatele.

Tento kód je pak poslán speciální funkci, která dotaz nejdříve zobrazí uživateli, aby si ho mohl prohlédnout, případně zkopírovat a použít kdekoliv jinde, nebo aby měl možnost si tento dotaz upravit podle svých představ. Výsledný kód je pak po potvrzení uživatelem poslán na databázový server a proveden. V případě chyby při vykonávaní SQL dotazu je uživateli zobrazeno chybové hlášení a dialog se zobrazeným kódem zůstává otevřený.

Upravování vlastností databázových objektů je prováděno téměř stejným způsobem jako jejich vytváření. Celý proces se tak liší pouze způsobem tvorby SQL dotazu. V případě, že uživatel změní určitou vlastnost v panelu vlastností v hlavním okně nebo v dialogu vlastností některých dalších objektů, jsou jednotlivé upravitelné vlastnosti prozkoumány a při tvorbě dotazu jsou brány v úvahu pouze ty, které se liší od jejich stávajících hodnot. Sestavený SQL kód je pak stejně jako při vytváření objektů zobrazen uživateli a následně proveden.

Při mazání objektů je potřeba brát v úvahu možnost pochybení uživatele a z bezpečnostních důvodů zobrazit dialog, kterým potvrdí své rozhodnutí daný objekt opravdu odstranit. Do tohoto dialogu byla navíc přidána volba umožňující smazání také všech závislých objektů. V případě, že i teď by došlo k pochybení uživatele, je opět nejdříve zobrazen SQL dotaz a až na základě dalšího potvrzení je tato operace provedena.

Ve výsledné aplikaci je tento způsob tvorby, upravování a mazání objektů implementovaný pouze pro některé databázové objekty. Jsou to následující objekty týkající se především práce s tabulkami: *Databases* (databáze), *Schemas* (schémata), *Tables* (tabulky), *Fields* (sloupce v tabulkách), *Indexes* (indexy), *Foreign Keys* (cizí klíče), *Checks* (omezení), *Triggers* (databázové triggery) a *Rules* (pravidla pro vykonávání SQL dotazů). Objektů, které nabízí celý databázový systém, je totiž velké množství a implementace těchto možností úprav by značně přesáhla rozsah a zadání této bakalářské práce.

5.5 Implementace SQL editoru

SQL editor byl zahrnut do výsledné implementace především proto, aby měl uživatel možnost provádět své vlastní dotazy a získal tak v přijatelné formě výsledky, které potřebuje.

Editor je složen ze tří základních částí: textového editoru, záložky pro zobrazení výsledků a záložky pro zobrazení zpráv posílaných databázovým serverem po vykonání určitého SQL kódu.

Textový editor byl naprogramován s velice jednoduchým, avšak pro běžné dotazy postačujícím, zvýrazňováním syntaxe. Uživatel může tedy vytvářet vlastní kód, který pak tlačítkem *Execute SQL code* odešle na databázový server ke zpracování.

Případná data vrácená serverem jako výsledek operace jsou zobrazena v záložce *Data*. Tento panel je pak automaticky zobrazen jako hlavní. Jistou nevýhodou může být, že jsou zde zobrazena data pouze posledního příkazu, který byl serverem proveden.

Ostatní zprávy (informace, upozornění a chybové hlášky) jsou uživateli zobrazeny v druhé záložce nazvané *Messages*. Tato záložka je zobrazena jako hlavní ve všech případech kromě případu popsaného výše, kdy jsou jako výsledek operace vrácena nějaká data.

6 Porovnání s existujícími rozhraními a náměty pro další vývoj aplikace

Z kapitoly 2 je patrné, že administračních nástrojů pro databázové systémy je celá řada. Jsou to však většinou aplikace komerční a nebo ne dostatečně přenositelné. Z tohoto faktu také vznikla myšlenka vytvoření takového grafického rozhraní, které bude volně šiřitelné a spolehlivě přenositelné. Přitom by ale mělo být schopné konkurovat komerčním aplikacím nejen nabízenými funkcionalitami, ale i svým vzhledem.

Protože jedinou volně šiřitelnou aplikací stejného charakteru jako toto nově navržené rozhraní je pgAdmin III (viz kapitola 2.2.4), budeme srovnání provádět výhradně s tímto grafickým uživatelským rozhraním.

Z předchozího textu je patrné, že jsou obě grafická administrační rozhraní určena pro databázový systém PostgreSQL. Zatímco pgAdmin III podporuje všechny verze tohoto sytému od verze 7.3, rozhraní navržené v této bakalářské práci je kompatibilní s verzí 8.1. Návrh a následná implementace přístupu k databázovému serveru však byla zohledněna pro možné rozšíření pro ostatní verze. Bylo tak dosaženo oddělením všech přístupů k databázi do zvláštní vrstvy aplikace.

Rozložení grafických rozhraní těchto nástrojů jsou mírně odlišná. Nicméně základní model hlavního okna s případnými dialogy je zachován v obou aplikacích. Rozdílnosti jsou především ve vzhledu a rozmístění grafických prvků.

Lišta nástrojů je u programu pgAdmin III tvořena příliš velkými ikonami, čímž ubírá pracovní plochu důležitějším částem aplikace. V navrženém rozhraní jsou ikony mnohem menší, avšak neubírají na efektivitě práce s tímto grafickým prvkem. K tomu, aby se uživatel snadněji orientoval v aplikaci, byla navíc přidána uvítací obrazovka s odkazy na základní funkce, která jde však v případě potřeby skrýt.

Další rozdílnost můžeme najít při implementaci stromu databázových objektů. V navrženém rozhraní má uživatel možnost výběru databází, které potřebuje, kdežto v aplikaci pgAdmin III jsou zobrazovány všechny dostupné databáze. To však může být v případě velkého počtu databází nežádoucí efekt. Navíc byla v navrženém rozhraní přidána lišta podávající uživateli užitečnou informaci o aktuálně připojené databázi. V pgAdmin III nic takového neexistuje.

Ve výsledné implementované aplikaci (narozdíl od pgAdmin III) ale nejsou zobrazeny všechny databázové objekty. To je způsobeno vysokou časovou náročností implementace celé aplikace a hlavně to nebylo předmětem zadání této práce. Návrh rozhraní je však přizpůsoben pro jejich snadné přidání.

Zcela jiným způsobem jsou také zpřístupněny informace o jednotlivých objektech. V pgAdmin III jsou tyto informace zobrazovány do jednoho okna společného pro všechny objekty

nezávisle na jejich typu. U navrženého rozhraní je naopak dynamicky vytvořeno okno s vlastnostmi, které se přímo týkají daného typu objektu, čímž není uživateli zobrazeno ani více ani méně než potřebuje o objektu vědět. Grafická podoba zobrazovaných informací je navíc v pgAdmin III velice jednoduše, však nepříliš uživatelsky příjemně vyřešena. Nežádoucí vlastností je také nutnost spuštění speciálního dialogu pro editaci těchto údajů. V navrženém rozhraní byly proto použity takové grafické prvky, které informace zpřehledňují, převádí do formy srozumitelné uživateli a umožňují zároveň jejich úpravu. V konečné implementaci navrženého rozhraní lze však editovat pouze některé objekty (viz kapitola 5.4). S ostatními je zatím počítáno pouze v návrhu aplikace a jejich doimplementování je možným námětem dalšího rozšíření.

Tvorba objektů v navrženém rozhraní je realizována pomocí speciálních průvodců, čímž je umožněna jednoduchá práce s aplikací i méně zkušeným uživatelům. Naproti tomu jsou v pgAdmin III objekty vytvářeny pomocí jednoduchých dialogů, ve kterých se začátečníci nemusí správně orientovat. Obě aplikace pak také nabízejí uživateli zobrazení a úpravu SQL dotazu, který danou operaci provede.

Získávání dat z tabulek, jejich přidávání, úprava, mazání či filtrování, je v obou rozhraních implementováno téměř stejně. Avšak v navrženém rozhraní bylo zobrazení těchto dat přesunuto do speciální záložky panelu vlastností určité tabulky. Tím je uživateli dovoleno prohlížení dat více tabulek a jasný přehled o tom, která data patří ke které tabulce.

Celé grafické rozhraní aplikace pgAdmin III samozřejmě nabízí více funkcionalit. Je to dáno především tím, že se na jejím vývoji podílí spousta uživatelů databázového systému PostgreSQL z celého světa. Cílem této bakalářské práce však nebylo vytvořit plně funkční aplikaci, což by ani nebylo v tak krátkém čase a jedním člověkem možné, ale ukázat, že i volně šiřitelná a přenositelná rozhraní lze navrhnout uživatelsky příjemně a s takovým zpracováním grafického vzhledu, které je schopné konkurovat komerčním nástrojům. Tímto se také potvrdila myšlenka z kapitoly 1.2, že pokud chceme vytvořit kvalitní grafickou aplikaci, je třeba věnovat více času správnému návrhu uživatelského rozhraní. Navíc pokud pak tato aplikace, v tomto případě konkrétně pro správu databázového systému, bude kladně přijata uživateli, docílíme i rozšíření použití spravovaného databázového systému.

7 Závěr

Výsledné grafické uživatelské rozhraní plní nejen požadavky zadání bakalářské práce, ale bylo rozšířeno o další užitečné funkce, které by měla být taková aplikace schopna provádět. Kromě zobrazení informací o všech databázových objektech, zde byl například přidán i jednoduchý editor SQL dotazů se zvýrazněnou syntaxí.

Velkým přínosem tvorby tohoto rozhraní bylo zejména podrobné prostudování grafické knihovny GTK+, programu Glade pro návrh rozhraní pomocí této knihovny a hlavně poznání databázového systému PostgreSQL, jeho systémových katalogů a informačních schémat.

Implementovaná aplikace byla testována na operačních systémech Fedora Core 6 a Red Hat Enterprise Linux 5. Oba systémy jsou linuxového typu, ale díky použitým nástrojům, které jsou samy o sobě spolehlivě přenositelné, je možné toto rozhraní použít i na ostatních operačních systémech (např. Microsoft Windows).

V průběhu návrhu a implementace rozhraní byla neustále brána v úvahu možnost dalšího rozšíření aplikace. Z tohoto důvodu byla také vytvořena určitá vrstva, která odděluje grafické rozhraní od veškerých přístupů k databázi. Tohoto modelu je pak možné využít v případném rozšíření aplikace o podporu ostatních databázových systémů například v podobě zásuvných modulů. Tyto moduly by pak musely doimplementovat pouze komunikaci s databází.

Dalším zajímavým rozšířením by mohlo být také vytvoření nástroje pro grafický návrh databází nebo ostatních databázových objektů.

Literatura

[1] Zemčík, P. *Tvorba uživatelských rozhraní.* Studijní opora. Brno, UPGM FIT VUT v Brně, 2006.

[2] Zendulka, J., Rudolfová, I. *Databázové systémy.* Studijní opora. Brno, UIFS FIT VUT v Brně, 2006.

[3] *PostgreSQL: Documentation: Manuals: PostgreSQL 8.1: PostgreSQL 8.1.9 Documentation* [online]. Poslední modifikace: 23. dubna 2007 [cit. 2007-05-15]. Dostupné z: <http://www.postgresql.org/docs/8.1/interactive>.

[4] *GTK+ - The GIMP Toolkit* [online]. Poslední modifikace: 28. dubna 2007 [cit. 2007-05-15]. Dostupné z: <http://www.gtk.org>.

[5] *Glade User Interaface Builder* [online]. Poslední modifikace: 14. března 2007 [cit. 2007-05-15]. Dostupné z: <http://glade.gnome.org>.

[6] *PostgreSQL: Documentation: Manuals: PostgreSQL 8.1: libpq – C Library* [online]. Poslední modifikace: 23. dubna 2007 [cit. 2007-05-15]. Dostupné z: <http://www.postgresql.org/docs/8.1/interactive/libpq.html>.

[7] *PostgreSQL – Wikipedia, the free encyklopedia* [online]. Poslední modifikace: 29. dubna 2007 [cit. 2007-05-15]. Dostupné z: <http://en.wikipedia.org/wiki/PostgreSQL>.

[8] *SQL Manager: Database Management Tools for MySQL, SQL Server, PostgreSQL, InterBase, Firebird* [online]. [cit. 2007-05-15]. Dostupné z: <http://www.sqlmanager.net>.

[9] *PG Explorer GUI development tool for postgres (PostgreSQL frontend client)* [online]. [cit. 2007-05-15]. Dostupné z: <http://www.pgexplorer.com>.

[10] *Navicat – the World's Best PostgreSQL GUI for Windows & Mac OS X* [online]. [cit. 2007-05-15]. Dostupné z: <http://pgsql.navicat.com>.

[11] *pgAdmin III: PostgreSQL administration and management tools* [online]. Poslední modifikace: 26. března 2007 [cit. 2007-05-15]. Dostupné z: <http://www.pgadmin.org >.

[12] *phpPgAdmin :: Web Based PostgreSQL Administration Tool* [online]. Poslední modifikace: 24. března 2007 [cit. 2007-05-15]. Dostupné z: <http://phppgadmin.sourceforge.net >.

[13] *Database query tool – Aqua Data Studio is the query analyzer that works with Oracle, DB2, Sybase, MySQL and more* [online]. Poslední modifikace: 16. ledna 2007 [cit. 2007-05-15]. Dostupné z: <http://www.aquafold.com>.

[14] *DB Tools for Oracle* [online]. Poslední modifikace: 10. dubna 2007 [cit. 2007-05-15]. Dostupné z: <http://www.softtreetech.com/dbtools>.

[15] *Oracle Developer Tools – Future Tek Solutions, Inc.* [online]. [cit. 2007-05-15]. Dostupné z: <http://www.oracledevelopertools.com>.

[16] *phpMyAdmin | MySQL Database Administration Tool | www.phpmyadmin.net* [online]. Poslední modifikace: 24. února 2007 [cit. 2007-05-15]. Dostupné z: <http://www.phpmyadmin.net>.

Seznam obrázků

Seznam příloh

Příloha 1 – Přehled SQL dotazů pro získání informací o databázových objektech

```
// Database info
SELECT datname AS name, rolname AS owner, description AS comment,
      pg_database.oid, datconnlimit AS conn_limit,
      pg_encoding_to_char(encoding) AS encoding,
      CASE WHEN datistemplate THEN 'true' ELSE 'false' END AS is_template,
      spcname AS tablespace, datlastsysoid AS lastoid, datacl AS acl
FROM pg_database
      LEFT OUTER JOIN pg_authid ON pg_database.datdba = pg_authid.oid
      LEFT OUTER JOIN pg_tablespace
            ON pg_database.dattablespace = pg_tablespace.oid
      LEFT OUTER JOIN pg_description
            ON pg_database.oid = pg_description.objoid
WHERE datname = 'name';

// Tablespaces
SELECT spcname AS name, rolname AS owner, pg_tablespace.oid AS oid,
      spcacl as acl
FROM pg_tablespace
      LEFT OUTER JOIN pg_authid ON pg_tablespace.spcowner = pg_authid.oid
ORDER BY name;

// Schemas
SELECT CASE WHEN nspname LIKE 'pg\\_temp\\_%%' THEN 1
      WHEN (nspname LIKE 'pg\\_%' OR nspname = 'information_schema') THEN 0
      ELSE 3 END AS nsptyp,
      nspname AS name, rolname AS owner, description AS comment,
      pg_namespace.oid AS oid, nspacl AS acl
FROM pg_namespace
      LEFT OUTER JOIN pg_authid ON pg_namespace.nspowner = pg_authid.oid
      LEFT OUTER JOIN pg_description
            ON pg_namespace.oid = pg_description.objoid
ORDER BY name;

// Languages
SELECT lanname AS name, description AS comment, pg_language.oid AS oid,
      CASE WHEN lanpltrusted THEN 'true' ELSE 'false' END AS trusted,
      han.proname AS handler, val.proname AS validator, lanacl AS acl
```

```
FROM pg_language
      LEFT OUTER JOIN pg_proc han ON pg_language.lanplcallfoid = han.oid
      LEFT OUTER JOIN pg_proc val ON pg_language.lanvalidator = val.oid
      LEFT OUTER JOIN pg_description
            ON pg_language.oid = pg_description.objoid
WHERE lanispl=true
ORDER BY name;

// Casts
SELECT description AS comment, pg_cast.oid AS oid,
      stype.typname AS source_type, ttype.typname AS target_type,
      func.proname AS function,
      CASE WHEN castcontext='a' THEN 'Assignment'
            WHEN castcontext='e' THEN 'Explicit'
            WHEN castcontext='i' THEN 'Implicit' END AS context
FROM pg_cast
      LEFT OUTER JOIN pg_type stype ON pg_cast.castsource = stype.oid
      LEFT OUTER JOIN pg_type ttype ON pg_cast.casttarget = ttype.oid
      LEFT OUTER JOIN pg_proc func ON pg_cast.castfunc = func.oid
      LEFT OUTER JOIN pg_description ON pg_cast.oid = pg_description.objoid
WHERE pg_cast.oid > number
ORDER BY source_type, target_type;

// Tables
SELECT tablename AS name, tableowner AS owner, description AS comment,
      class.oid AS oid,
      CASE WHEN class.relhasoids THEN 'true' ELSE 'false' END AS with_oids,
      inhf.relname AS inherited_from,
      inhfsch.nspname AS inherited_from_schema, tablespace,
      class.relacl AS acl
FROM pg_tables
      LEFT OUTER JOIN pg_class class ON pg_tables.tablename = class.relname
      LEFT OUTER JOIN pg_inherits inher ON inher.inhrelid = class.oid
      LEFT OUTER JOIN pg_class inhf ON inher.inhparent = inhf.oid
      LEFT OUTER JOIN pg_namespace inhfsch
            ON inhf.relnamespace = inhfsch.oid
      LEFT OUTER JOIN pg_description
            ON class.oid = pg_description.objoid
            AND pg_description.objsubid = 0
WHERE schemaname = 'name'
ORDER BY tablename;
```

```
// Views
SELECT viewname AS name, viewowner AS owner, description AS comment,
      class.oid AS oid, definition AS body, class.relacl AS acl
FROM pg_views
      LEFT OUTER JOIN pg_class class ON pg_views.viewname = class.relname
      LEFT OUTER JOIN pg_description ON class.oid = pg_description.objoid
WHERE schemaname = 'name'
ORDER BY viewname;

// Functions
SELECT proname AS name,  rolname AS owner, description AS comment,
      pg_proc.oid as oid,
      CASE WHEN proretset THEN 'true' ELSE 'false' END AS return_set,
      rettype.typname AS return_type, lan.lanname AS language,
      CASE WHEN proisstrict THEN 'true' ELSE 'false' END AS strict,
      CASE WHEN prosecdef THEN 'Definer'
            ELSE 'Invoker' END AS execution_privilegies,
      CASE WHEN provolatile='i' THEN 'Immutable'
            WHEN provolatile='s' THEN 'Stable'
            WHEN provolatile='v' THEN 'Volatile' END AS stability,
      pronargs AS n_args, proargtypes, proallargtypes, proargmodes,
      proargnames, prosrc AS definition, proacl AS acl
FROM pg_proc
      LEFT OUTER JOIN pg_authid ON pg_proc.proowner = pg_authid.oid
      LEFT OUTER JOIN pg_type rettype ON pg_proc.prorettype = rettype.oid
      LEFT OUTER JOIN pg_language lan ON pg_proc.prolang = lan.oid
      LEFT OUTER JOIN pg_namespace
            ON pg_proc.pronamespace = pg_namespace.oid
      LEFT OUTER JOIN pg_description ON pg_proc.oid = pg_description.objoid
WHERE nspname = 'name' AND NOT proisagg
ORDER BY name;

// Domains
SELECT dom.typname AS name, rolname AS owner, description AS comment,
      dom.oid AS oid, dtype.typname AS data_type,
      CASE WHEN dom.typnotnull THEN 'true' ELSE 'false' END AS not_null,
      dom.typdefault AS default_value
FROM pg_type dom
      LEFT OUTER JOIN pg_type dtype ON dom.typbasetype = dtype.oid
      LEFT OUTER JOIN pg_authid ON dom.typowner = pg_authid.oid
      LEFT OUTER JOIN pg_namespace nsp ON nsp.oid = dom.typnamespace
```

```
      LEFT OUTER JOIN pg_description ON dom.oid = pg_description.objoid
WHERE nsp.nspname='name' AND dom.typtype = 'd'
ORDER BY dom.typname;

// Aggregates
SELECT agg.proname AS name,  rolname AS owner, description AS comment,
      agg.oid as oid, btype.typname AS base_type,
      stype.typname AS state_type, sfunc.proname AS state_function,
      ftype.typname AS final_type, ffunc.proname AS final_function,
      agginitval AS initial_condition
FROM pg_proc agg
      LEFT OUTER JOIN pg_authid ON agg.proowner = pg_authid.oid
      LEFT OUTER JOIN pg_type btype ON agg.proargtypes[0] = btype.oid
      LEFT OUTER JOIN pg_type ftype ON agg.prorettype = ftype.oid
      LEFT OUTER JOIN pg_aggregate ON agg.oid = pg_aggregate.aggfnoid
      LEFT OUTER JOIN pg_type stype ON aggtranstype = stype.oid
      LEFT OUTER JOIN pg_proc sfunc ON aggtransfn = sfunc.oid
      LEFT OUTER JOIN pg_proc ffunc ON aggfinalfn = ffunc.oid
      LEFT OUTER JOIN pg_namespace ON agg.pronamespace = pg_namespace.oid
      LEFT OUTER JOIN pg_description ON agg.oid = pg_description.objoid
WHERE nspname = 'name' AND agg.proisagg
ORDER BY name;

// Sequences
SELECT relname AS name, rolname AS owner, description AS comment,
      pg_class.oid, relacl AS acl
FROM pg_class
      LEFT OUTER JOIN pg_authid ON pg_class.relowner = pg_authid.oid
      LEFT OUTER JOIN pg_namespace
            ON pg_class.relnamespace = pg_namespace.oid
      LEFT OUTER JOIN pg_description
            ON pg_class.oid = pg_description.objoid
WHERE nspname = 'name' AND relkind = 'S'
ORDER BY name;

SELECT last_value AS value,increment_by AS increment, max_value,
      min_value, cache_value as cache,
      CASE WHEN is_cycled THEN 'true' ELSE 'false' END AS cycle
FROM name;
```

```
// Types
SELECT typ.typtype, typ.typname AS name, rolname AS owner,
      description AS comment, typ.oid AS oid,
      ifunc.proname AS input_function, ofunc.proname AS output_function,
      rfunc.proname AS return_function, sfunc.proname AS send_function,
      afunc.proname AS analyze_function, typ.typlen AS internal_length,
      typ.typdefault AS Default, elem.typname AS element,
      typ.typdelim AS delimiter,
      CASE WHEN typ.typalign='c' THEN 'Char'
            WHEN typ.typalign='s' THEN 'Short'
            WHEN typ.typalign='i' THEN 'Int'
            WHEN typ.typalign='d' THEN 'Double' END AS alignment,
      CASE WHEN typ.typbyval THEN 'true'
            ELSE 'false' END AS passed_by_value,
      CASE WHEN typ.typstorage='p' THEN 'Plain'
            WHEN typ.typstorage='e' THEN 'Secondary relation'
            WHEN typ.typstorage='m' THEN 'Compressed inline'
            WHEN typ.typstorage='x'
                  THEN 'Compressed inline or secondary relation'
            END AS storage
FROM pg_type typ
      LEFT OUTER JOIN pg_proc ifunc ON typ.typinput = ifunc.oid
      LEFT OUTER JOIN pg_proc ofunc ON typ.typoutput = ofunc.oid
      LEFT OUTER JOIN pg_proc rfunc ON typ.typreceive = rfunc.oid
      LEFT OUTER JOIN pg_proc sfunc ON typ.typsend = sfunc.oid
      LEFT OUTER JOIN pg_proc afunc ON typ.typanalyze = afunc.oid
      LEFT OUTER JOIN pg_type elem ON typ.typelem = elem.oid
      LEFT OUTER JOIN pg_authid ON typ.typowner = pg_authid.oid
      LEFT OUTER JOIN pg_namespace ON typ.typnamespace = pg_namespace.oid
      LEFT OUTER JOIN pg_description ON typ.oid = pg_description.objoid
WHERE typ.typtype='b' AND nspname='name'
ORDER BY typ.typname;

// Operators
SELECT oper.oprname AS name, rolname AS owner, description AS comment,
      oper.oid,
      CASE WHEN oper.oprkind='b' THEN 'Infix both'
            WHEN oper.oprkind='l' THEN 'Prefix left'
            WHEN oper.oprkind='r' THEN 'Postfix right'
            WHEN oper.oprkind='d' THEN 'Double' END AS kind,
      func.proname AS function, leftop.typname AS left_operand_type,
```

```
      rightop.typname AS right_operand_type, com.oprname AS commutator,
      neg.oprname AS negator, rfunc.proname AS restrict_function,
      jfunc.proname AS join_function, lsort.oprname AS left_sort_operator,
      rsort.oprname AS right_sort_operator,
      lcmp.oprname AS less_than_operator,
      rcmp.oprname AS grater_than_operator,
      CASE WHEN oper.oprcanhash THEN 'true'
           ELSE 'false' END AS supports_hash_joins
FROM pg_operator oper
      LEFT OUTER JOIN pg_proc func ON oper.oprcode = func.oid
      LEFT OUTER JOIN pg_type leftop ON oper.oprleft = leftop.oid
      LEFT OUTER JOIN pg_type rightop ON oper.oprright = rightop.oid
      LEFT OUTER JOIN pg_operator com ON oper.oprcom = com.oid
      LEFT OUTER JOIN pg_operator neg ON oper.oprnegate = com.oid
      LEFT OUTER JOIN pg_proc rfunc ON oper.oprrest = rfunc.oid
      LEFT OUTER JOIN pg_proc jfunc ON oper.oprjoin = jfunc.oid
      LEFT OUTER JOIN pg_operator lsort ON oper.oprlsortop = com.oid
      LEFT OUTER JOIN pg_operator rsort ON oper.oprrsortop = com.oid
      LEFT OUTER JOIN pg_operator lcmp ON oper.oprltcmpop = com.oid
      LEFT OUTER JOIN pg_operator rcmp ON oper.oprgtcmpop = com.oid
      LEFT OUTER JOIN pg_authid ON oper.oprowner = pg_authid.oid
      LEFT OUTER JOIN pg_namespace ON oper.oprnamespace = pg_namespace.oid
      LEFT OUTER JOIN pg_description ON oper.oid = pg_description.objoid
WHERE nspname='name'
ORDER BY oper.oprname;

// Triggers
SELECT tgname AS name, description AS comment, pg_trigger.oid AS oid,
      CASE WHEN tgenabled THEN 'true' ELSE 'false' END AS enabled,
      CASE WHEN instg.condition_timing='BEFORE' THEN 'Before'
           ELSE 'After' END AS type,
      CASE WHEN instg.action_orientation='ROW' THEN 'Row'
           ELSE 'Statement' END AS for_each,
      CASE WHEN instg.event_manipulation='INSERT' THEN 'Insert'
           ELSE '' END AS insert_event,
      CASE WHEN deltg.event_manipulation='DELETE' THEN 'Delete'
           ELSE '' END AS delete_event,
      CASE WHEN updtg.event_manipulation='UPDATE' THEN 'Update'
           ELSE '' END AS update_event,
      proname AS function, tgnargs, tgargs AS args,
      pnsp.nspname AS func_nsp
```

```
FROM pg_trigger
      LEFT OUTER JOIN
            (SELECT * FROM information_schema.triggers
                  WHERE event_manipulation='INSERT') AS instg
            ON tgname = instg.trigger_name
      LEFT OUTER JOIN
            (SELECT * FROM information_schema.triggers
                  WHERE event_manipulation='DELETE') AS deltg
            ON tgname = deltg.trigger_name
      LEFT OUTER JOIN
            (SELECT * FROM information_schema.triggers
                  WHERE event_manipulation='UPDATE')AS updtg
            ON tgname = updtg.trigger_name
      LEFT OUTER JOIN pg_class class ON pg_trigger.tgrelid = class.oid
      LEFT OUTER JOIN pg_namespace nsp ON class.relnamespace = nsp.oid
      LEFT OUTER JOIN pg_proc ON pg_trigger.tgfoid = pg_proc.oid
      LEFT OUTER JOIN pg_namespace pnsp ON pronamespace = pnsp.oid
      LEFT OUTER JOIN pg_description
            ON pg_trigger.oid = pg_description.objoid
WHERE nspname='name1' AND class.relname ='name2' AND NOT tgisconstraint
ORDER BY name;

// Rules
SELECT rul.rulename AS name, description AS comment, rew.oid AS oid,
      CASE WHEN rew.ev_type='1' THEN 'Select'
            WHEN rew.ev_type='2' THEN 'Update'
            WHEN rew.ev_type='3' THEN 'Insert'
            WHEN rew.ev_type='4' THEN 'Delete' END AS event,
      CASE WHEN rew.is_instead THEN 'true' ELSE 'false' END AS instead,
      CASE WHEN rew.is_instead THEN
            substring(pg_get_ruledef(rew.oid, true)
                        from '%INSTEAD #"%#";' for '#')
            ELSE substring(pg_get_ruledef(rew.oid, true)
                              from '%DO #"%#";' for '#')END AS definition,
      substring(pg_get_ruledef(rew.oid, true)
                  from '%WHERE #"%#" DO%' for '#'), rul.tablename
FROM pg_rules rul
      LEFT OUTER JOIN pg_rewrite rew ON rul.rulename = rew.rulename
      LEFT OUTER JOIN pg_description ON rew.oid = pg_description.objoid
      LEFT OUTER JOIN pg_class rew_tab ON rew.ev_class = rew_tab.oid
WHERE schemaname='name1'
```

```
      AND rul.tablename='name2' AND rew_tab.relname = 'name2'
ORDER BY name;

// Fields
SELECT fields.attname AS name, description AS comment,
      CASE WHEN cols.udt_name LIKE '\\\\_%' THEN
                  substring(cols.udt_name from '_#\"%#\"' for '#')
            ELSE cols.udt_name END AS type,
      CASE WHEN cols.udt_name LIKE '\\\\_%' THEN 'true'
            ELSE 'false' END AS is_array,
      cols.character_maximum_length AS size,
      cols.numeric_precision AS precision,
      CASE WHEN cons_p.contype ='p' THEN 'true' ELSE 'false' END AS pkey,
      CASE WHEN cons_u.contype ='u' THEN 'true' ELSE 'false' END AS unique,
      CASE WHEN attnotnull THEN 'true' ELSE 'false' END AS not_null,
      fieldef.adsrc AS default_value, fields.attnum AS column_number,
      fields.attndims AS dimensions,
      CASE WHEN fields.attinhcount > 0
            THEN 'true' ELSE 'false' END AS inherited
FROM pg_attribute fields
      LEFT OUTER JOIN pg_class tab ON tab.oid = fields.attrelid
      LEFT OUTER JOIN pg_attrdef fieldef
            ON fields.attrelid = fieldef.adrelid
            AND fields.attnum = fieldef.adnum
      LEFT OUTER JOIN pg_constraint cons_p
            ON tab.oid = cons_p.conrelid
            AND fields.attnum = ANY (cons_p.conkey)
            AND cons_p.contype = 'p'
      LEFT OUTER JOIN pg_constraint cons_u
            ON tab.oid = cons_u.conrelid
            AND fields.attnum = ANY (cons_u.conkey)
            AND cons_u.contype = 'u'
      LEFT OUTER JOIN pg_namespace ON tab.relnamespace = pg_namespace.oid
      LEFT OUTER JOIN information_schema.columns cols
            ON nspname = cols.table_schema
            AND tab.relname = cols.table_name
            AND fields.attname = cols.column_name
      LEFT OUTER JOIN pg_description ON tab.oid = pg_description.objoid
            AND pg_description.objsubid = fields.attnum
WHERE nspname='name1' AND tab.relname='name2'
      AND fields.attnum > 0 AND attisdropped = FALSE
```

```
ORDER BY fields.attnum;

// Indexes
SELECT  ind.relname AS name, description AS comment, ind.oid AS oid,
      indkey AS columns,
      CASE WHEN indisprimary THEN 'true'
            ELSE 'false' END AS pkey_constraint,
      CASE WHEN con.contype = 'u' THEN 'true'
            ELSE 'false' END AS unique_constraint,
      CASE WHEN (indisprimary OR con.contype = 'u' OR indisunique = false)
            THEN 'false' ELSE 'true' END AS unique,
      CASE WHEN indisclustered THEN 'true' ELSE 'false' END AS clustered,
      am.amname AS access_method, spcname AS tablespace,
      indpred AS predicate
FROM pg_index
      LEFT OUTER JOIN pg_class ind ON pg_index.indexrelid = ind.oid
      LEFT OUTER JOIN pg_class tab ON pg_index.indrelid = tab.oid
      LEFT OUTER JOIN pg_namespace ON tab.relnamespace = pg_namespace.oid
      LEFT OUTER JOIN pg_tablespace
            ON ind.reltablespace = pg_tablespace.oid
      LEFT OUTER JOIN pg_am am ON ind.relam = am.oid
      LEFT OUTER JOIN pg_constraint con
            ON indrelid = con.conrelid AND ind.relname = con.conname
      LEFT OUTER JOIN pg_description ON ind.oid = pg_description.objoid
WHERE nspname = 'name1' AND tab.relname = 'name2'
ORDER BY ind.relname;

// Foreign Keys
SELECT con.conname AS name, description AS comment, con.oid AS oid,
      con.conkey AS columns, ref_nsp.nspname AS ref_schema,
      ref_tab.relname AS ref_table, con.confkey AS ref_columns,
      CASE WHEN con.confupdtype='a' THEN 'No Action'
            WHEN con.confupdtype='r' THEN 'Restrict'
            WHEN con.confupdtype='c' THEN 'Cascade'
            WHEN con.confupdtype='n' THEN 'Set Null'
            WHEN con.confupdtype='d' THEN 'Set Default'
            ELSE 'Unknown' END AS on_update_rule,
      CASE WHEN con.confdeltype='a' THEN 'No Action'
            WHEN con.confdeltype='r' THEN 'Restrict'
            WHEN con.confdeltype='c' THEN 'Cascade'
            WHEN con.confdeltype='n' THEN 'Set Null'
```

```
            WHEN con.confdeltype='d' THEN 'Set Default'
            ELSE 'Unknown' END AS on_delete_rule,
      CASE WHEN con.confmatchtype='u' THEN 'Simple'
            WHEN con.confmatchtype='f' THEN 'Full'
            ELSE 'Unknown' END AS match_type,
      CASE WHEN con.condeferrable THEN 'true'
            ELSE 'false' END AS deferrable,
      CASE WHEN con.condeferred THEN 'At the end of the transaction'
            ELSE 'After each statement' END AS check_time
FROM pg_constraint con
      LEFT OUTER JOIN pg_class tab ON con.conrelid = tab.oid
      LEFT OUTER JOIN pg_namespace nsp ON tab.relnamespace = nsp.oid
      LEFT OUTER JOIN pg_class ref_tab ON con.confrelid = ref_tab.oid
      LEFT OUTER JOIN pg_namespace ref_nsp
            ON ref_tab.relnamespace = ref_nsp.oid
      LEFT OUTER JOIN pg_description ON con.oid = pg_description.objoid
WHERE con.contype = 'f' AND nsp.nspname = 'name1'
      AND tab.relname = 'name2'
ORDER BY name;

// Foreign Key References
SELECT con.conname AS name, con.confkey AS columns,
      nsp.nspname AS ref_schema, tab.relname AS ref_table,
      con.conkey AS ref_columns
FROM pg_constraint con
      LEFT OUTER JOIN pg_class tab ON con.conrelid = tab.oid
      LEFT OUTER JOIN pg_namespace nsp ON tab.relnamespace = nsp.oid
      LEFT OUTER JOIN pg_class ref_tab ON con.confrelid = ref_tab.oid
      LEFT OUTER JOIN pg_namespace ref_nsp
            ON ref_tab.relnamespace = ref_nsp.oid
      LEFT OUTER JOIN pg_description ON con.oid = pg_description.objoid
WHERE con.contype = 'f' AND ref_nsp.nspname = 'name1'
      AND ref_tab.relname = 'name2'
ORDER BY name;

// Checks
SELECT conname AS name, description AS comment, pg_constraint.oid,
      consrc AS condition
FROM pg_constraint
      LEFT OUTER JOIN pg_class ON conrelid = pg_class.oid
      LEFT OUTER JOIN pg_namespace ON connamespace = pg_namespace.oid
```

```
      LEFT OUTER JOIN pg_description
            ON pg_constraint.oid = pg_description.objoid
WHERE contype='c' AND nspname = 'name1' AND relname='name2'
ORDER BY name;
```