Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Databázové systémy ve WIRINGu

Martin Kropáč

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačního inženýrství

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Kropáč Martin

Informatika

Název práce

Databázové systémy ve WIRINGu

Anglický název

Database systems in WIRING

Cíle práce

Bakalářská práce je tématicky zaměřená na využití databázové technologie v evidenci propojení spojových bodů na zařízení DDF. Cílem práce je:

a) vymezit teoretické principy relačně databázové technologie a WIRINGu,

b) zmapovat současnou úroveň využívání db technologie v této záležitosti a identifikovat bariéry jejiího širšího uplatnění,

c) navrhnout odstranění identifikovaných bariér,

d) navržené záležitosti ověřit a demonstrovat na konkrétním řešení,

e) ověřené záležitosti zobecnit pro další možná použití.

Metodika

Použitá metodika zadané bakalářské práce bude založena na studiu a analýze dostupných informačních zdrojů. Navrhované řešení bude realizováno formou praktické aplikace, která bude respektovat identifikované bariéry. Na podkladě syntézy teoretických poznatků a dosažených výsledků budou formulovány závěry této bakalářské práce a následně zobecněny pro další možná použití.

Harmonogram zpracování

Vymezení teoretických principů relačně db technologie a WIRINGU - předmět 1. zápočtu z BP za 2. ročník studia: 06/2012-09/2012.

Zmapování současné úrovně využívání db technologie v této záležitosti a identifikace existujících bariér: 09/2012-11/2012

Navržení konkrétního řešení - předmět 1. zápočtu z BP za 3. ročník: 11/2012-01/2013.

Ověření navrženého řešení a jeho zobecnění - předmět 2. zápočtu z BP za 3. ročník: 02/2013-03/2013.

Rozsah textové části

40-50 stran

Klíčová slova

DBS, WIRING, spojovací bod, síť spojovacích bodů, DDF

Doporučené zdroje informací

CASTAGNETO, J., SCHUMAN, H. R., SCOLLO, C.: Programujeme PHP profesionálně. Brno: Computer Press, 2001. ISBN 80? 7226-310-2.
KOFLER, M., ÖGGL,B.: PHP 5 a MySQL 5 - Průvodce webového programátora. Computer Press, 2007. ISBN: 978-80-251-1813-9
KOSEK, J.: PHP-tvorba interaktivních internetových aplikací. Grada Publishing, spol. s r.o., 1999; ISBN 80-7169-373-1
PONKR, M.: PHP a MySQL bez předchozích znalostí Brno. Computer Press a. s. 2011. ISBN 978-80-251-1758-3

Vedoucí práce

Vostrovský Václav, doc. Ing., Ph.D.

Termín odevzdání

březen 2013

Ing. Martin Pelikán, Ph.D.
Vedoucí katedry

prof. Ing. Jan Hron, DrSc., dr.h.c.
Děkan fakulty

V Praze dne 5.10.2012

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Databázové systémy ve WIRINGu" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.11.2013 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Václavu Vostrovskému, Ph.D. za jeho odborné vedení práce a přínosné konzultace, které mi pomohly při zpracování práce.

Databázové systémy ve WIRINGu

Database Systems in WIRING

Souhrn

Předmětem bakalářské práce je uplatnění malého databázového systému v praxi telekomunikační společnosti. Vysvětluje, proč jsou databáze nepostradatelnou součástí podnikového systému informační technologie jako způsob uchovávání a účelné organizace důležitých údajů a dat obecně. V teoretické části popisuje relační databázové systémy jako celek. Způsob ukládání a organizace dat v relačních databázových systémech, jejich praktického hledání a vyvolávání. V praktické části se zaměřuje na malou lokální relační databázi WIRING, používanou v provozu telekomunikační společnosti Telefónica Czech Republic a.s. Popisuje účel provozu této databáze, její praktickou implementaci do systému informačních technologií firmy a postupy užití tohoto informačního nástroje. Poslední část bakalářské práce se zabývá identifikací strukturálních nedostatků databáze WIRING a návrhem možných řešení těchto nedostatků.

Klíčová slova: databázový systém, WIRING, spojovací bod, SQL, DDF, Digital Distribution Frame, lokální databáze, relace, tabulka, data, PHP, MySQL

Summary

Subject of this thesis is to implement a small database system in practice Telecommunication Company. It explains why databases are indispensable components of enterprise system information technology as a way to preserve and efficient organization of relevant information and data in general. The theoretical part describes relational database systems as a whole. Way of storing and organizing data in relational database systems, their practical search and retrieval. In the practical part focuses on a small local relational database WIRING, used in service Telecommunications Company Telefónica Czech Republic co. Describes the purpose of the operation of the database, its practical implementation in the IT system of the company and processes using this information tool. In the last part, the work seeks to identify structural weaknesses WIRING database and propose possible solutions to these shortcomings.

Keywords: Database System, WIRING, Connection Point, SQL, DDF, Digital Distribution Frame, Local Database, Relation, Table, Data, PHP, MySQL

Obsah

1 Úvod

Současná podniková praxe se stejně jako mnoho dalších oborů neobejde bez ukládání, údržby a použití velkého množství dat různého druhu. Pod pojmem data si můžeme představit veškeré informace o technologiích použitých pro předmět podnikání, finančních tocích podniku, ale i o pracovnících dané organizace a mnoho dalších. Účelné použití velké většiny výpočetních systémů je závislé na zdroji dat. Takovým zdrojem dat, který je využíván ve velkém množství společností působících na trhu, nezřídka bývá databázový systém.

Jedním z databázových systémů je i databázový nástroj WIRING, který je užíván v praxi vybrané telekomunikační společnosti pro evidenci propojení spojových bodů, které jsou tvořeny zakončeními vstupů a výstupů použité technologie.

Praxe potvrdila, že užití lokální databáze je nejvýhodnější, neboť je tím zajištěn aktuální obraz skutečného stavu propojení spojovacích bodů. Při zápisu informací o propojení přímo do globálního informačního systému, dochází často k potížím s aktualizací datové báze systému způsobenou velikým objemem dat. Při použití vhodné struktury báze dat lokálního systému WIRING je možné jej propojit s rozsáhlejším informačním systémem podniku a poskytovat tak aktuální informace širokému okruhu pracovníků, kteří pak mohou tato data používat pro svou pracovní činnost.

Předkládaná bakalářská práce se zabývá teoretickými pravidly struktury relační databáze, která byla pro organizaci dat systému WIRING použita, popisem stávající struktury databáze a po identifikaci jejích nedostatků návrhem struktury, která bude výhodnější a bude využívat novějších prostředků poskytovaných systémem řízení báze dat MySQL.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem předkládané bakalářské práce je definovat princip a účel malé lokální databáze WIRING ve fungování telekomunikačních společností. Účelem databáze WIRING je evidence propojení spojových bodů, tedy vstupně-výstupních rozhraní technologie instalované v dané lokalitě, na zařízení Digital Distribution Frame (DDF). Cílem teoretické části bakalářské práce je objasnění konceptů použitých pro vytvoření tohoto databázového nástroje a popis současného stavu užití databázové technologie. V části věnované vlastnímu zpracování se bakalářská práce zabývá specifikací nedostatků ve struktuře uložení dat, navržením změn v systému organizace údajů a následným vytvořením nové struktury databáze. Dále jsou v bakalářské práci, za pomoci popsaných teoretických východisek, objasněny důvody provedených změn a navrženy možné alternativy pro využití databáze WIRING.

Účelem navržených řešení je racionalizace databázového nástroje WIRING s využitím normálních forem a umožnění jeho začlenění do širších databázových struktur.

2.2 Metodika práce

Bakalářská práce je rozdělena na dvě stěžejní části, část teoretickou a část věnovanou vlastnímu zpracování. Teoretická východiska jsou podkladem pro praktickou část předkládané bakalářské práce. K vypracování bakalářské práce byla použita specifická technika zkoumání, studium odborné literatury a internetových zdrojů. Informace pro zpracování bakalářské práce byly čerpány z vlastní zkušenosti, z doporučené odborné literatury a z dalších publikací, které souvisí tématicky s obsahem předkládané práce.

V bakalářské práci je využita analýza teoretických principů tvorby datových bází, které byly popsány v doporučené odborné literatuře. Dále struktury a algoritmy dotazovacích jazyků použitých v současném návrhu databázového systému WIRING.

V části zabývající se vlastním zpracováním byly navrženy změny struktury databáze na praktickém návrhu, který vyhovuje požadavkům datové normalizace. Ten byl vytvořen na základě analýzy současné struktury databáze WIRING. Tabulky byly navrženy tak, aby obsahovaly věcně příbuzná data. Pro zápis struktury relací bylo použito dotazovacího jazyku SQL. V praktickém návrhu byla použita technika návrhu struktury tabulek a také tvorba pohledů pro zajištění ucelených informačních výstupů. Dále bylo užito rutin spouštěných z SQL pro možnost zadávání dat tak, aby byla ve struktuře databáze uložena na místa, kam skutečně patří.

Závěr bakalářské práce byl formulován na základě syntézy poznatků z teoretické části i části, která se věnuje vlastnímu zpracování. V závěru byly formulovány obecné principy databáze WIRING a navrženy další možnosti jejího využití i v jiných oborech, než pro který byla vytvořena.

3 Teoretické principy databázových systémů

Databázové systémy v posledních dvaceti letech postupovaly hlavně směrem rozvoje relačních systémů, kde jsou data ukládána prostřednictvím tabulek. V 90. letech nastupují objektové databáze jako další možnost, jak budovat databázové systémy. Tyto systémy navazují na objektově orientované programování a reflektují přirozený pohled uživatelů na databáze, kteří mohou (nebo chtějí) vidět data v souvislosti s objektem, který reprezentují. Zatímco v relačním systému hovoříme o řádku dat, který reprezentuje určitý objekt, v objektových systémech manipulujeme přímo s abstraktní reprezentací objektu pomocí jeho atributů a programů (metod), které modelují chování objektu. Tyto trendy vedly ke vzniku nového, objektově orientovaného databázového softwaru, který dnes rychle dohání klasický relační model. [5]

3.1 Relační databáze

V současnosti je nejpoužívanějším typem tzv. relační databázový model navržený Edgarem Frankem Coddem v sedmdesátých letech dvacátého století.

3.2 Struktura relace

V relačním modelu jsou data uložena v relacích. Z uživatelského hlediska jsou vnímány jako dvourozměrné tabulky a tvoří samostatné soubory dat. Každá relace je tvořena záhlavím, kde jsou specifikovány názvy sloupců (též nazývané atributy tabulky) a záznamy tvořícími řádky tabulky. Jednotlivé záznamy musí být jednoznačně identifikovány atributem, který je určen jako primární klíč. Pokud nelze zajistit jedinečnost primárního klíče, musí být doplněn o sekundární klíčový atribut. V takovém případě zajišťuje jednoznačné určení záznamu konjunkce obou klíčů.

- Názvy atributů musí být jednoznačné. Každý sloupec obsahuje hodnoty určitého typu a obor těchto hodnot, které se v sloupci mohou vyskytovat, se nazývá doménou sloupce nebo atributu.
- Všechny hodnoty v tabulce musí být elementární, tj. dále nedělitelné na další údaje.
- Pořadí jednotlivých sloupců ani řádků není významné.

- Obor hodnot každého sloupce musí být stejný, údaje musí být stejného druhu.
- Každý řádek musí být jednoznačně rozlišitelný, v tabulce nesmí být dva řádky, jejichž hodnoty jsou ve všech sloupcích stejné. [1]

S relacemi se dají provádět relační operace.
Operace projekce (project) – vytváří se nová projekční relace, obsahující vybrané sloupce.
Operace selekce (select) – vzniká nová relace, která obsahuje pouze záznamy splňující uživatelem stanovenou podmínku.
Operace spojení (join) – spojením dvou relací pomocí klíčů vzniká nová relace obsahující kombinace záznamů vyhovující zadaným podmínkám. [1]

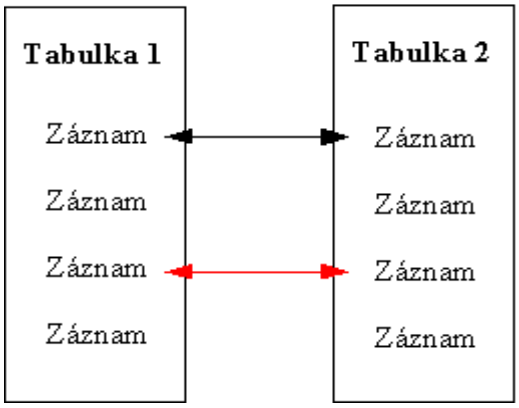
3.2.1 Spojení (join) mezi databázovými tabulkami

Spojení popisují vztahy mezi objekty popisovanými tabulkami. V jednotlivých tabulkách jsou definovány atributy, obsahující primární nebo sekundární klíč odkazující na záznam v připojené tabulce. Takovýto sloupec obsahuje tzv. cizí klíč. [2]

3.2.1.1 Relace jedna ku jedné (1:1, one-to-one)

Každý řádek jedné tabulky odkazuje na právě jeden záznam související tabulky. Záznamy z obou tabulek jsou v tomto typu relace rovnocenné. Relace tohoto typu lze vyjádřit příkladem „vztah manželů“. Manžel má jen jednu ženu a žena má jen jednoho manžela, pokud jde o relaci v jednom časovém okamžiku. [1]

Obrázek 1 - Vztah jedna ku jedné

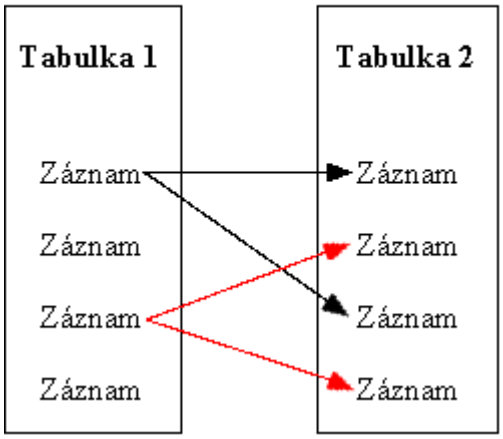


Zdroj: [1]

3.2.1.2 Relace jedna ku více (1:N, one-to-many)

Každý řádek primární tabulky odkazuje na jeden nebo více řádků. Tuto relaci vyjadřuje příklad zákazníka a objednávky. Jeden zákazník může mít více objednávek, ale každá objednávka přísluší jen jednomu zákazníkovi. [1]

Obrázek 2 - Vztah jedna ku více

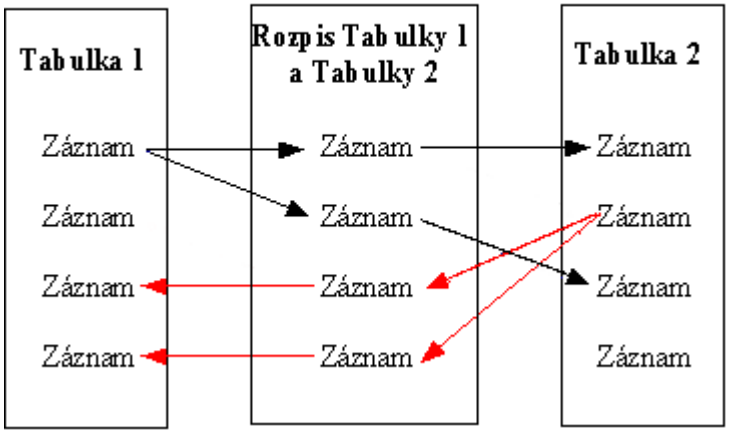


Zdroj: [1]

3.2.1.3 Relace více ku více (N:M, many-to-many)

Více řádků primární tabulky může být svázáno s více záznamy sekundární tabulky. Prakticky lze tuto relaci docílit pomocí spojovací tabulky, která rozloží relaci M:N na dvě relace 1:N. Relací M:N lze vyjádřit vztah zboží a objednávka. Každá objednávka může obsahovat více typů zboží stejně jako typ zboží může být ve více objednávkách. [1]

Obrázek 3 – Relace více ku více



Zdroj: [1]

3.2.2 Datová normalizace

Normalizace je proces, kterým se odstraňují nadbytečná data, snižuje se složitost a zjednodušuje se aktualizace dat. Normalizované tabulky obsahují všechna potřebná data a lze je snadno udržovat a efektivně se na ně dotazovat. K tomu se často využívá operace spojení.

Operace spojení (JOIN) je jednou z hlavních operací relační algebry. Pomocí této operace lze funkčně propojit tabulky tak, aby v nich obsažené údaje byly snadno dostupné v jedné aplikaci. [1]

3.2.2.1 První normální forma (1NF)

Relace nesmí obsahovat násobná data.

1NF je nezbytným předpokladem relačního datového modelu vymezeného původními definicemi E. F. Codda. Pokud má být databáze v souladu s požadavky relačního datového modelu, tak musí být alespoň v 1NF vymezené pravidlem, že v tabulce nesmějí být obsaženy multizávislosti. To znamená, že pokud v tabulce existují závislosti více hodnot na jedné klíčové hodnotě, tak je třeba tuto tabulku rozdělit na dvě tabulky, které jsou propojené s klíčovou hodnotou relací 1:N. [1]

3.2.2.2 Druhá normální forma (2NF)

Všechna neklíčová data relace musí funkčně záviset na celém primárním klíči. Druhá normální forma se týká tabulek se složeným klíčem (alespoň ze dvou atributů) a takové tabulky jsou v 2NF jen tehdy, pokud jsou všechny hodnoty v tabulkách závislé na celém klíči tabulky. Pokud tomu tak není, je třeba tabulku rozdělit do dvou tabulek, kde budou údaje funkčně závislé na příslušné části složeného klíče. [1]

3.2.2.3 Třetí normální forma (3NF)

Všechna neklíčová data musí záviset pouze na klíčových hodnotách a ne mezi sebou navzájem.

3NF se týká vzájemných závislostí mezi daty v tabulce. Aby byla tabulka v 3NF, musí dílčí hodnoty v ní obsažené záviset pouze na klíčových hodnotách a ne mezi sebou. V případě, že tabulka nevyhovuje 3NF, je nutno tabulku rozdělit tak, aby atribut, na kterém jsou některé hodnoty závislé, byl v nové tabulce klíčem a původní tabulka se na něj odkazovala jako na cizí klíč. [1]

Tyto tři normální formy jsou považovány za základní. Splňuje-li je databáze, lze ji považovat za normovanou. Pokud jsou tyto normální formy beze zbytku splněny, pak bylo dosaženo i dalších tří formulovaných normálních forem. [1]

3.2.2.4 Čtvrtá normální forma (4NF)

Tato normální forma se týká pouze databází, jejichž relace obsahují složený primární klíč. Jedná se tedy o velice specifický okruh bází dat.

Složený primární klíč nesmí být tvořen z nezávislých dat.

Pokud složený klíč tvoří nezávislé hodnoty, jejich spojením ve složeném primárním klíči vzniká falešná souvislost mezi těmito hodnotami a nemohou existovat nezávisle na sobě, což není v souladu s modelovanou realitou. Z těchto důvodů je vyžadováno, aby klíč byl tvořen hodnotami, které mají skutečnou vzájemnou souvislost. [1]

3.2.2.5 Pátá normální forma (5NF)

Složený (3 a více) primární klíč nesmí obsahovat párové cyklické závislosti. [1]

Omezení vyplývající z dodržení této normální formy se týká, jak je vidět, stejně jako ze 4NF pouze relací s vícenásobným primárním klíčem a to ještě klíčem složeným ze tří a více atributů. [1]

3.3 Dotazovací jazyk SQL

Jazyk SQL je především jazykem relačních databází. SQL stroj je součástí relačního SŘBD (Systém Řízení Báze Dat).

3.3.1 Historie SQL

Vznik jazyka SQL je datován rokem 1974. Pod původním názvem Sequel byl charakteristický hlavně svou dotazovací částí. Jeho prototyp byl součástí Systému R vyvíjeného v letech 1974-1977 v laboratoři IBM v kalifornském San Jose, kde byl zaměstnán i tvůrce RMD (Relační Model Dat) - dr. E. F. Codd. Akronym Sequel znamenal tehdy Structured English QUEry Language. Od poněkud nadnesené koncepce strukturované angličtiny se však ustoupilo a dnes se hovoří pouze o SQL (Structured Query Language).

Výchozí koncepce jazyka SQL je tedy stará již téměř 40 let. K prvním významnějším produktům obsahujících SQL patřily tři systémy IBM, jichž je SQL

součástí. Byly to následující systémy: DB/2, SQL/DS a QMF. SQL/DS a DB/2 vycházející v základní koncepci ze Systému R.

S nástupem osobních počítačů se v 80. letech začala na implementaci SQL orientovat řada renomovaných výrobců software. Od jednouživatelských SŘBD (Systém Řízení Báze Dat) s SQL (případně verzí těchto systémů v lokálních sítích) se přešlo k vytváření výkonných SŘBD v prostředí UNIX, resp. databázových serverů založených přímo na SQL. Z implementací této třídy jsou dnes nejznámější SŘBD Oracle, Informix, Sybase, DB/2, SQL Server od firmy Microsoft a další. SQL bývá použit i jako rozhraní jiných, obecně nerelačních systémů.

Mezníkem ve vývoji SQL se stala jeho standardizace organizací ANSI (American National Standards Institute) v roce 1986 a její přijetí ISO v roce 1987. Standard byl založen na dialektu SQL firmy IBM a byl charakterizován jako "průnik existujících implementací". Byl nazýván též SQL86. Rozšíření definičního jazyka týkající se možností definovat integritní omezení byla zaznamenána ve zprávě ISO z roku 1989. Takto rozšířený SQL byl nazýván SQL89. Další vývoj jazyka vyvrcholil standardem SQL92, který byl přijat ANSI a ISO v r. 1992. Dále se tedy hovořilo o SQL jako o SQL3. Tímto názvem byly označeny všechny vyvíjené koncepce. Od roku 1999 mají standardy SQL značení ve stylu SQL: rrrr, např. SQL: 2006, který je zatím poslední oficiální verzí standardu.

Též existuje verze X/OPEN SQL jako standard vycházející z ANSI, ovšem s některými rozdíly souvisejícími s použitím UNIXu jako implementačního prostředí. Firma IBM také publikovala svůj vlastní standard - Systems ApplicationArchitecture Database Interface (SAA-SQL). [5]

3.3.2 Struktura jazyka SQL

Jazyk SQL lze používat jako:

Přímé provedení SQL, který je implementován přímo do SŘBD jako příkazový interpret určený pro dialog uživatele s bází dat. Příkladem této alternativy

je program SQLPLUS, jako součást databázového systému ORACLE nebo SQL Shell, který je součástí systému MySQL.

Integrovaný jazyk SQL, je přidán do prostředí jiného programovacího jazyka (např. C, Pascal, Cobol, PHP). Protože SQL má od hostitelského jazyka rozdílnou syntaxi, musí existovat spojovací funkce, která je schopna zajistit přenos dat mezi oběma jazyky.

Dynamický SQL je variantou vnořeného SQL do hostitelského jazyka, který umožňuje napsat programy, dovolující za chodu (nikoliv pouze při psaní programu) vkládat a spouštět SQL dotazy. [6]

Dotazovací jazyk SQL nerozlišuje mezi velkými a malými písmeny při zadávání klíčových slov příkazů SQL. Toto však neplatí při zadávání znakových literálů, zde se důsledně rozlišuje mezi velkými a malými písmeny. [1] Při zadávání literálů je možno použít zástupné znaky tzv. wildcards, kterými je možno nahradit jeden znak či skupinu znaků. Dalším základním pravidlem je ukončení každého příkazu odesílaného ke zpracování SŘBD středníkem[;]. Tento znak se dá změnit, například pro účely zadávání procedur, kdy nechceme, aby byl středník rozhraním SQL pokládán za ukončení vkládacího příkazu, pomocí příkazu DELIMITER [nový znak].

% – Nahrazuje žádný nebo několik libovolných znaků.

_ – Nahrazuje právě jeden libovolný znak.

[seznam znaků] – Nahrazuje právě jeden znak ze seznamu.

[!seznam znaků] – Nahrazuje všechny znaky, které neobsahuje seznam. [6]

3.3.3 Rozdělení SQL příkazů

Standardní SQL příkazy jsou rozděleny do 4 skupin:

1. DDL – Data Definition Language

Touto skupinou SQL příkazů je definována struktura databáze. Vytváří se jimi tabulky, indexy a ostatní objekty relační databáze. Patří mezi ně tyto příkazy:

CREATE DATABASE CREATE TABLE ALTER TABLE

DROP TABLE CREATE INDEX DROP INDEX

CREATE VIEW ALTER VIEW DROP VIEW

CREATE SEQUENCE ALTER SEQENCE DROP SEQUENCE

CREATE PROCEDURE DROP PROCEDURE CREATE TRIGER

DROP TRIGER

2. DML – Data Manipulation Language

Příkazy pomocí nichž lze hledat, vkládat, odebírat a měnit data.

Skupina je reprezentována těmito příkazy:

SELECT INSERT DELETE

UPDATE

3. DCL – Data Control Language

DCL jsou příkazy pro řízení přístupu k datům. Umožňují nastavit oprávnění přístupu jednotlivým uživatelům tak, že oprávnění jsou rozdělena do logických celků neboli transakcí.

Příkazy, kterými se mohou uživatelům přidělovat, měnit, odnímat práva, vytvořit nebo odstranit uživatelský účet jsou:

GRANT ALTER USER REVOKE

CREATE USER DROP USER

4. TCC – Transaction Control Commands

Jde o příkazy, kterými je možno zajistit možnost vrátit změny provedené v databázi pomocí DML příkazů, případně přijmout provedení transakce. Transakce je sled SQL příkazů, se kterými interpret SQL zachází jako s jedním celkem.

Například příkazy pro vytvoření transakce, uložení určitého bodu v transakci, návrat k uloženému bodu nebo přijetí transakce jako celku.

SET TRANSACTION SAVEPOINT ROLLBACK

COMMIT [6]

3.3.4 Užití SQL příkazů DDL

3.3.4.1 Vytvoření databáze

Standard SQL nedefinuje pojem databáze ani syntaxi příkazu pro její vytvoření, avšak většina SŘBD s pojmem databáze pracuje. Umožňuje nám vytvořit samostatný logický objekt, základ hierarchické struktury dalších datových objektů, který obsahuje sadu informací datového (užití nestandardní znakové sady) či bezpečnostního charakteru (tabulky uživatelů a jejich práv). U SŘBD, které pojem databáze podporují, je nutno vždy před začátkem práce s daty příslušný objekt databáze zvolit. Některé SŘBD navíc podporují vytváření schématu databáze, které je možno použít při přenosu databáze do prostředí SŘBD, který vytváření databází jako samostatných objektů nepodporuje. [6] Nejjednodušší syntaxe příkazu pro vytvoření databáze je:

CREATE DATABASE *název_databáze*

3.3.4.2 Tvorba a modifikace tabulek (relací)

V SQL jsou podporovány tři druhy tabulek. Jsou to tabulky základní, odvozené a posledním typem jsou pohledy. Základní tabulky jsou většinou objekty, tvořící základní datovou strukturu databáze. Odvozené tabulky jsou vytvořeny dotazem na

data v databázi. Pohledy jsou pak určitý druh odvozených tabulek, jejichž definice je trvale uložena ve schématu databáze.

Základní tabulky mohou být definovány pomocí SQL jako trvalé nebo dočasné.

Trvalé tabulky jsou nejběžnějším typem objektů v databázi, uchovávají uložená data.

Dočasné tabulky mají stejnou strukturu jako tabulky trvalé, avšak jsou v databázi dostupné pouze po dobu trvání připojení, ve kterém byly vytvořeny. Po ukončení tohoto spojení přestávají existovat. Přístup k dočasným tabulkám má pouze uživatel, za jehož připojení byly vytvořeny, to je důvod, pro který může současně existovat více dočasných tabulek se stejným jménem, pokud byly vytvořeny různými uživateli v různých připojeních.

Příkazem určeným v SQL pro tvorbu nových tabulek je „CREATE [TEMPORARY] TABLE“. Příkaz musí obsahovat jedinečné jméno tabulky, definice sloupců a může obsahovat vlastnosti tabulky. Definici sloupce tvoří jedinečné jméno sloupce v tabulce, datový typ nebo doména, výchozí hodnota a omezení sloupce. [6]

Datové typy obsahuje následující oddíl této práce. Doména je množina přípustných hodnot určitého datového tytu pro daný atribut. Výchozí hodnota je předdefinovaná hodnota zadaná pro případ, že dané pole nebude vyplněno. Omezení sloupce jsou uvedena ve výpisu níže. [6] Vlastnosti tabulky mohou obsahovat atributy dané tabulky, jako jsou údaje o umístění a datové velikosti tabulky, heslo pro přístup k datům v tabulce umístěných a další.

V existující tabulce lze pomocí SQL provádět změny příkazem „ALTER TABLE“. Je možno přidat sloupec určitého jména a datového typu specifikací „ADD COLUMN“. Změnit vlastnosti existujícího sloupce lze specifikací „ALTER COLUMN“. Pokud však měněný sloupec obsahuje data, musí všechna vyhovovat novým vlastnostem sloupce. Lze také odebrat sloupec specifikací za pomoci příkazu „DROP COLUMN“.

Celou tabulku lze odebrat z databáze příkazem „DROP TABLE“. Použití tohoto příkazu však musí předcházet úprava všech vztahů s odebíranou tabulkou, jinak nebude možno odebrání tabulky provést. Vztahy je myšleno její obsažení v pohledech, umístění jejího primárního klíče v jiných tabulkách jako cizí klíč nebo použití jejího obsahu v rutinách. [6]

3.3.4.3 Datové typy SQL

SQL umožňuje definovat datové typy numerické, znakové řetězce, bitové řetězce, čas a časové intervaly.

1. **Numerické typy**

Tabulka 1 – Celočíselné datové typy

Celá čísla		
Datový typ	**bytů**	**rozsah**
TINYINT	1	SIGNED -128-127; UNSIGNED 0-255
SMALLINT	2	SIGNED -32768-32767; UNSIGNED 0-65535
MEDIUMINT	3	SIGNED -8388608-8388607; UNSIGNED 0-16777215
INTEGER (INT)	4	SIGNED -2147483648-2147483647; UNSIGNED 0-4294967295
BIGINT	8	SIGNED -9223372036854775808-9223372036854775807; UNSIGNED 0-18446744073709551615

Zdroj: [8]

Tabulka 2 – Aproximativní číselné typy

Aproximativní číselné typy		
Datový typ	**bytů**	**rozsah**
FLOAT(p)	4 nebo 8	p-počet číslic v rozsahu 0-24 => 4 byty 25-53 => 8 bytů
FLOAT	4	Číslo s plovoucí desetinnou čárkou. Základní přesnost do 24 číslic celkem.
REAL	8	Číslo s plovoucí desetinnou čárkou. S pevnou přesností danou implementací,
DOUBLE PRECISION	8	Číslo s plovoucí desetinnou čárkou. S pevnou přesností danou implementací větší než REAL
DECIMAL(M, D)	4 až 8	Číslo s M číslicemi a D desetinnými místy
NUMERIC(M, D)	4 až 8	Číslo s max. M číslicemi a max. D desetinnými místy

Zdroj: [8]

2. Znakové a bitové řetězce

Tabulka 3 – Znakové a bitové řetězce

Znakové řetězce		
Datový typ	**Délka**	**rozsah**
CHARACTER (CHAR)	Deklarovaný počet znaků	Znaky (musí být ze znakové sady) budou doplněny mezerami do deklarovaného počtu.
VARCHAR2(S [BYTE | CHAR])	SBYTES CHAR * délka písmene	řetězec znaku délky max. s bytů nebo s písmen
CHARACTER LARGE OBJECT (CLOB)	Počet znaků řetězce	Pro velké skupiny znaků. Řetězec není doplněn mezerami na deklarovanou délku.
NATIONAL CHARACTER (NCHAR)	Deklarovaný počet znaků	Jako datový typ CHARACTER založen na implementací definované znakové sadě.
NATIONAL CHARACTER VARYING (NCHAR VARYING)	Počet znaků řetězce	Jako VARCHAR2 je však založen na implementací definované znakové sadě.

NATIONAL CHARACTER LARGE OBJECT (NCLOB)	Počet znaků řetězce	Pro velké skupiny znaků. Řetězec není doplněn mezerami na deklarovanou délku. Založen na implementací definované znakové sadě.
BIT VARYING(M)	(M+7)/8	Bitový řetězec. Implicitně je délky 1 bit. M určuje požadovanou délku řetězce.

Zdroj: [6][8]

3. Časové a datové typy

Tabulka 4 – Časové a datové typy

Datový typ	**formát**
DATE	DATE ‚YYYY-MM-DD‘
TIME(zlomky sek.)	TIME ‚1997-01-31 09:26:50.12‘ Podporuje sekundy v rozsahu 00 až 59.999. Zlomky sekund neobsahuje, pokud nejsou zadány.
TIMESTAMP	TIMESTAMP [(přesnost_ve_zlomcích_sekund)] TIMESTAMP ‚YYYY-MM-DD HH24:MI:SS.FF‘
TIME WITH TIME ZONE	Stejné jako typ TIME. Navíc obsahuje informace o UTC a časových zónách.
TIMESTAMP WITH TIME ZONE	Stejné jako typ TIMESTAMP. Navíc obsahuje informace o UTC a časových zónách.

Zdroj: [6]

- **Datový typ INTERVAL**

Datový typ INTERVAL zachycuje rozdíl mezi dvěma hodnotami data a času. SQL podporuje dva základní druhy intervalů. Intervaly typu rok-měsíc a intervaly typu den-čas. Do závorky za deklaraci konstruktoru YEAR, MONTH nebo DAY v datovém typu INTERVAL lze zadat počet číslic přesnosti zobrazení hodnoty (výchozí hodnota je 2). Za konstruktor SECOND lze vložit počet desetinných míst zlomků sekund. [6]

Tabulka 5 – Datový typ INTERVAL

Datový typ	formát
Rok – měsíc	
INTERVAL YEAR	Interval počítaný v rocích.
INTERVAL MONTH	Interval počítaný v měsících.
INTERVAL YEAR TO MONTH	Interval počítaný v rocích a měsících. Číslice pro měsíc budou od číslic pro rok odděleny pomlčkou.
Den – čas	
INTERVAL DAY	Interval počítaný ve dnech
INTERVAL DAY TO HOUR	Interval počítaný ve dnech a hodinách. Oba údaje budou odděleny pomlčkou.
INTERVAL DAY TO MINUTE	Interval počítaný ve dnech a minutách. Oba údaje budou odděleny pomlčkou.
INTERVAL DAY TO SECOND	Interval počítaný ve dnech a sekundách. Oba údaje budou odděleny pomlčkou.

Zdroj: [6]

- **Datový typ BOOLEAN**

Velmi jednoduchý datový typ. Může dosahovat jen tří hodnot: TRUE (pravda), FALSE (nepravda) a UNKNOWN (neznámá). SQL hodnota NULL je v datovém typu BOOLEAN považována za hodnotu UNKNOWN.

Porovnávání datových typů BOOLEAN se řídí touto logikou:

- TRUE je větší než FALSE.
- Výsledkem porovnávání jakékoli hodnoty s hodnotou NULL je neznámá (UNKNOWN) hodnota.
- Neznámá hodnota může být sloupci přiřazena, pokud sloupec podporuje hodnoty NULL. [6]

Typy omezení

Omezení je nepovinný parametr, ale pro některé atributy zcela nezbytný, může nabývat těchto hodnot:

- NOT NULL — sloupec nesmí obsahovat hodnotu NULL
- UNIQUE — žádná hodnota se ve sloupci neopakuje
- PRIMARY KEY — sloupec je primárním klíčem tabulky
- FOREIGN KEY — sloupec je cizím klíčem (definuje referenci k jiné tabulce)
- CHECK — obsah sloupce bude zadán logickým výrazem
- AUTO INCREMENT — automaticky vloží do sloupce jedinečné celé číslo, které se při zadání každého nového řádku zvětšuje (MySQL)

3.3.4.4 Práce s definovanými pohledy v databázi

Dalším typem objektu ve schématu databáze je pohled. Pohled je virtuální tabulka, která ve schématu existuje pouze jako uložená definice a umožňuje sledovat data zapsaná do jedné nebo více tabulek. V pohledech nejsou tedy uložena žádná data, ale umožňují přístup k datům jako základní tabulka. Pohled je tedy jedinečně pojmenovaný objekt.

Výhodou užití pohledů je, že může být definován velmi složitý dotaz, ten se uloží do definice pohledu a později, při další potřebě použití tohoto dotazu, je možno volat pouze definovaný a pojmenovaný pohled místo opětovného vytváření dotazu. Pohledy mohou být způsobem prezentace informací uživatelsky vhodným způsobem, to znamená v logickém pořadí a bez zbytečných nebo bezpečnostně citlivých dat. Jednoduchým způsobem jak znepřístupnit citlivá data širokému okruhu uživatelů je vytvoření pohledu bez těchto dat, přestože jsou v daných tabulkách zaznamenány. Lze totiž povolit uživatelům přístup k těmto datům pouze přes pohled, ne přímo přes tabulky.

V definici pohledu je možno přejmenovat zobrazované sloupce tak, aby jejich názvy byly uživatelsky příjemnější než často velice zjednodušující názvy sloupců

tabulek. Pokud nový název sloupce nebude v definici pohledu uveden, zdědí sloupec pohledu název od příslušného sloupce tabulky. Datové typy sloupců pohledu jsou vždy zděděny od příslušných sloupců tabulky.

Pohledy lze také používat pro úpravu dat obsažených v příslušné tabulce po zobrazení v pohledu. Použitím matematických nebo řetězcových funkcí lze v definici pohledu nastavit obsah polí sloupců v pohledu na jinou hodnotu, než je uvedena v základní tabulce, ale vždy na hodnotu z dat v základní tabulce vycházející.

Pohledy mohou být použity i pro modifikaci dat v základních tabulkách. Modifikace dat přes pohledy však podléhá určitým omezením.

Modifikační omezení pohledů:

- Data v pohledu nemohou být seskupována, přepočítávána nebo automaticky eliminována.
- Alespoň jeden sloupec v zobrazované základní tabulce musí být aktualizovatelný.
- Každý sloupec v pohledu musí být logickým obrazem jediného sloupce v jediné zdrojové tabulce.
- Každý řádek pohledu musí být vysledovatelný k jedinému řádku ve zdrojové tabulce.

Jako příklad lze uvést, že pokud pohled zobrazuje jen část sloupců základní tabulky, je možno jej použít k přidání řádku pouze tehdy, pokud nezobrazené sloupce zdrojové tabulky povolují hodnotu NULL. Vložení nového řádku do tabulek přes pohled samozřejmě nemůže vložit hodnoty do sloupců, které nejsou v pohledu zobrazeny. SŘBD do nich pak bude chtít automaticky vložit NULL nebo předvolenou hodnotu, pokud je definována.

Definici pohledů lze měnit příkazem „ALTER VIEW“ nebo odstraňovat příkazem „DROP VIEW“ zcela bez omezení, neboť se jedná pouze o virtuální tabulky bez obsahu dat. [6]

3.3.5 SQL příkazy kategorie DML

3.3.5.1 Dotazy na data uložená v databázi

Základním příkazem standardu SQL je příkaz SELECT. Základní konstrukce příkazu je SELECT – FROM – WHERE. Tento příklad vybírá data z tabulek podle zadání proměnných v příkazu. K užití tohoto příkazu je třeba znát alespoň část struktury databáze.

Složka SELECT obsahuje v nejjednodušší variantě seznam názvů sloupců, které chceme zobrazit, tvořících strukturu tabulky ze které je vybíráno. Jednotlivé proměnné jsou odděleny čárkou. Pokud je požadováno zobrazení všech sloupců tabulky, je možno použít znak * (hvězdička). Pokud je třeba zobrazit data ze sloupců ve více tabulkách, je možno použít celé názvy sloupců tj. název tabulky a název sloupce oddělené tečkou. Složka se dále dá zpřesnit konstruktory DISTINCT ON () nebo UNIQUE – od každého jeden. Na jednu ze zobrazených položek může být použita funkce COUNT(), která zobrazí počet vyhledaných hodnot. [8]

Složka FROM specifikuje zdroj dotazu. V základním případě jde o jednu tabulku, případně dříve nebo aktuálně definovaný pohled (view). Ve složitějším případě více tabulek spojených pomocí klíčů a konstruktoru JOIN.

Složka WHERE obsahuje logickou podmínku zobrazení řádků. Podle parametrů této podmínky se zobrazí jen ty řádky, které podmínce odpovídají. Tato složka příkazu je nepovinná. [6]

3.3.5.2 Vkládání dat do základní tabulky

Dalším velmi potřebným SQL příkazem je INSERT. Vkládání dat do prázdných polí databáze.

Sekce INSERT INTO udává název tabulky, do které se budou data vkládat. Pokud je cílem vkládat jen do některých sloupců, jejich názvy se zapíšou do seznamu uzavřeného do kulatých závorek. Sloupce je možno do seznamu vložit v libovolném pořadí.

Sekce VALUES obsahuje seznam hodnot vkládaných do jednotlivých sloupců tabulky. Pořadí hodnot musí být stejné jako pořadí sloupců v seznamu sekce INSERT INTO a stejně tak i počet hodnot. U novějších verzí jazyka SQL je možno tímto příkazem vkládat hodnoty i do několika řádků. V takovém případě se seznamy hodnot pro jednotlivé sloupce, uzavřené do závorek, napíšou několikrát za sebou a pouze se oddělí čárkou.

Je rovněž možno vložit do tabulky výsledek hledání v databázi pomocí příkazu SELECT, který se připojí na konec příkazu INSERT INTO místo klauzule VALUES. Podmínkou je jen shoda počtu hodnot požadovaných příkazem INSERT a vyvolaných příkazem SELECT. [6]

3.3.5.3 Aktualizace dat v základní tabulce

Pro změnu dat v tabulkách se v SQL používá příkaz „UPDATE“. První údaj v příkazu „UPDATE“ musí být uveden název tabulky, která bude aktualizována. Za názvem tabulky následuje klauzule „SET“. Pak musí být uveden alespoň jeden výraz složený z názvu aktualizovaného sloupce, rovnítka a vkládané hodnoty. Název sloupce stejně jako rovnítko jsou povinné údaje, avšak vkládaná hodnota může být vyjádřena explicitně nebo jako matematická respektive řetězcová funkce nad vkládanou hodnotou. Jako operand použité funkce může být použita stávající hodnota aktualizovaného pole. Samozřejmě aktualizovaná hodnota musí odpovídat datovému typu a omezením sloupce, do kterého je vkládána. Není možné vkládat do jednoho sloupce různé hodnoty do různých řádků jedním příkazem. Místo vkládané hodnoty je taktéž možno vložit do aktualizovaného sloupce hodnotu vyhledanou pomocí příkazu SELECT. Tento příkaz bude vložen na pravou stranu rovnítka se syntaxí uvedenou výše a ohraničen závorkami. [6]

3.3.5.4 Odstraňování dat z tabulky

Řádky z tabulky je možno odstraňovat příkazem „DELETE“, jehož syntaxe je velice jednoduchá. Povinnou klauzulí je v něm pouze sekce „FROM“ zapsaná v první části příkazu a obsahující název tabulky, ze které je požadováno řádky odstranit. Pokud je třeba odstranit všechny řádky bez výběru, příkaz zde bude

ukončen. Pokud je však třeba odstranit řádky podle výběru, je nutno připojit na konec příkazu klauzuli „WHERE“, jež se používá stejně jako u příkazů „SELECT“ nebo „UPDATE“.

Pokud je třeba odstranit z tabulky jen hodnoty v některých sloupcích, je třeba použít příkaz UPDATE. [6]

3.3.6 Užití SQL příkazů DCL

3.3.6.1 Přidání uživatele do databáze

Příkaz „CREATE USER“ vytvoří dalšího uživatele pro přístup k databázi. Tímto příkazem bude vytvořen jeden nový záznam do seznamu uživatelů. K uživatelskému jménu bude přiděleno heslo. Uživatelský účet bude zatím bez specifikovaných práv. K přidělení práv je třeba použít příkaz „GRANT“. Syntaxe příkazu je:

CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'my_passw'; [8]

3.3.6.2 Přidělení práv vytvořenému uživateli

V SQL se provádí příkazem „GRANT“. Tento příkaz může uživatelskému účtu přidělit právo k přístupu k celé databázi nebo konkrétní části databáze (tabulce, pohledu, proceduře) a to jak na úrovni změny, tak na úrovni prohlížení. Taky je možno tímto příkazem omezit frekvenci přístupu k údajům v databázi. Správce databáze, který práva může přidělit příkazem „GRANT“, musí mít sám oprávnění „GRANTOPTION“. Uživateli je možno přidělit práva určité úrovně pro celý SŘBD, tj. pro všechny databáze SŘBD spravované, konkrétní databázi nebo konkrétnímu objektu v dané databázi. Práva se mohou různě kombinovat. Pro různé objekty může mít uživatel různá práva.

3.3.6.3 Změna nastavení uživatelského účtu

Změna nastavení uživatelského účtu je příkaz pro změnu parametrů uživatelského účtu, ne však nastavení práv. Tímto příkazem lze například nastavit pro daný

uživatelský účet nové heslo nebo zajistit vypršení hesla a tím vyžádat od uživatele zadání hesla nového. Příklad použití příkazu:

ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE; [8]

3.3.6.4 Změna uživatelských práv

Tímto příkazem lze odejmout určitá oprávnění konkrétnímu uživateli. Příklad odnětí práva „vkládat záznamy do všech tabulek“ je tento:

REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost'; [8]

3.3.6.5 Zrušení uživatelského účtu

Jedná se o vymazání jednoho nebo více uživatelů i jejich práva. Příkladem použití příkazu je:

DROP USER 'jeffrey'@'localhost';[8]

3.3.7 Syntaxe SQL příkazů TCC

3.3.7.1 Definice transakcí

Transakce jsou úseky kódu databáze, které mohou být během vytváření kódu potvrzeny nebo vráceny zpět. Příkazem „SET TRANSACTION“ lze vytvořit sadu SQL příkazů kategorie DML, které na sebe funkčně navazují a po dobu jejich provádění nemůže dojít ke změnám v daném úseku databáze ze strany jiných uživatelů. Transakce se provádí vždy jako celek, tedy může buď skončit celá úspěšně nebo neúspěšně. Zápis transakce by měl začínat atributem „COMMIT“, pro zajištění ukončení transakce, která byla spuštěna dříve a nebyla řádně ukončena. Stejným atributem by měl zápis transakce končit pro řádné ukončení.

„COMMIT“ -- Příkaz provede zápis proběhlé transakce do databáze a její ukončení.

Příkaz „SET TRANSACTION“ může obsahovat tyto klauzule:

READ WRITE (default) - transakce může obsahovat příkazy čtení i zápisu

READ ONLY - transakce může obsahovat jen příkazy čtení

NAME *název transakce* - transakce může být pojmenována [8]

- **ROLLBACK**

Vrátí změny provedené transakcí do původního stavu. V případě, že za příkazem následuje TO *nazev savepointu,* bude výsledek transakce vrácen jen po zadaný savepoint a předcházející část transakce bude uložena do databáze s provedenými změnami.

- **SAVEPOINT**

Příkaz následovaný názvem označí pojmenovaný bod v transakci, kam budou odvolány příkazy použitím ROLLBACK TO. [8]

3.3.8 Rutiny

Do standardu SQL:1999 byla zařazena organizacemi ANSI (American National Standards Institute) a ISO (International Organization for Standardization) možnost vytvářet procedury a funkce spouštěné z SQL. Abychom mohli zadat rutiny se správnou syntaxí, musíme před zápisem kódu rutiny změnit znak ukončení a odeslání dotazu pomocí příkazu DELIMITER. Pokud bychom tak neučinili, implementace SQL by považovala každé užití středníku za pokyn k odeslání předchozího textu jako dotaz. Po ukončení zadávání rutiny nesmíme zapomenout změnit znak ukončení dotazu zpět na středník, abychom mohli dále pracovat se standardní syntaxí dotazů. [6]

3.3.8.1 Pravidla pro vytváření rutin spouštěných z SQL

Příkazem „CREATE PROCEDURE“ se vytváří procedura. Základní syntaxe příkazu je:

CREATE PROCEDURE *název procedury*([*deklarace parametru*]{, *deklaraceparametru*}…)
BEGIN
[DECLARE *proměnná* [, *proměnná*] ... type [DEFAULT *hodnota*]]
Tělo rutiny
END;

Pro zajištění kompaktního provedení celé procedury, je výhodné zapsat celé tělo rutiny jako transakci. [6]

3.3.8.2 Funkce spouštěné z SQL

Příkaz „CREATE FUNCTION“ slouží k vytvoření funkce spouštěné z SQL. Syntaxe příkazu je poněkud odlišná od syntaxe vytváření procedury. Odlišnost spočívá ve vrácení vypočtené hodnoty do kódu, ze kterého byla funkce volána. K vrácení hodnoty je využita klauzule RETURN. [6]

3.3.8.3 Podmínkové příkazy větvení

Větvení IF – THEN – ELSE je základní příkaz větvení. Zápis příkazu začíná klauzulí „IF“ následovanou testovací podmínkou. Pokud je výsledek podmínky pravda, pokračuje běh programu přes následující klauzuli „THEN“ k bloku příkazů, které se mají provést nad určenou databází. Pokud je výsledek testu podmínky nepravda a za blokem příkazů následuje klauzule „ELSE“ nebo „ELSEIF“ pro test jiné podmínky, je proveden blok příkazů zapsaný za touto klauzulí. Příkaz musí být zakončen klauzulí „END IF“.[6]

Větvením CASE lze provádět vícenásobné větvení rutiny. Výběrem definované hodnoty testované proměnné bude zvolen příslušný blok příkazů. Tento příkaz může být použit ve dvou syntaxích.

Syntaxe první:

CASE *testovaná proměnná*
  WHEN *hodnota* THEN *blok příkazů;*
  [WHEN *hodnota* THEN *blok příkazů;*]…
  [ELSE *blok příkazů;*]
END CASE;

Syntaxe druhá:

CASE
  WHEN *podmínka* THEN *blok příkazů;*
  [WHEN *podmínka* THEN *blok příkazů;*]…
  [ELSE *blok příkazů;*]
END CASE;

V první syntaxi se testuje hodnota určité proměnné a podle výsledku testu je zvolen příslušný příkazový blok. V druhé syntaxi jsou testovány definované podmínky a podle výsledku je proveden příkazový blok. [6]

3.3.8.4 Příkazy cyklů

Cyklus LOOP: tento příkaz vytváří smyčku s odkazy na návěští.

Syntaxe příkazu je:

[*návěští*:] LOOP

blok příkazů;

podmínka ukončení ITERATE *návěští*|LEAVE *návěští*;

END LOOP [*návěští*];

Klauzule INTERATE provede skok na začátek smyčky.

Klauzule LEAVE ukončí běh smyčky. Pokud zadání tohoto příkazového slova bude vynecháno, dojde při běhu rutiny k nekonečné smyčce.

Cyklus REPEAT – UNTIL: příkaz, který vytváří cyklus probíhající, dokud platí podmínka zapsaná za programovým slovem UNTIL. Blok příkazů zapsaný v těle cyklu se provede minimálně jednou. Příkaz je ukončen klauzulí END REPEAT.

Cyklus WHILE – DO: příkaz vytvoří cyklus, který bude prováděn, pokud a dokud platí vstupní podmínka. Příkaz je ukončen klauzulí END WHILE. [6]

3.3.8.5 Spouštění procedur

Procedury se spouštějí příkazem CALL, který je následován názvem procedury se vstupními parametry uzavřenými v kulatých závorkách. Celý příkaz musí být ukončen středníkem. Funkce se spouští jejím prostým umístěním do kódu jako proměnnou, funkce pak vrátí výsledek a zařadí ho do procedury, ze které byla volána.

4 Databáze WIRING

4.1 Popis databázového systému WIRING

Databázový systém WIRING je lokální databáze založená na SŘBD MySQL, nad kterým je jako uživatelské rozhraní postaven PHP skriptovací jazyk. Účelem této databáze je evidovat datová spojení provedená metalickými spoji mezi jednotlivými spojovacími body. V praxi společnosti Telefónica Czech Republic a.s., kde je databáze WIRING využívána, jsou spojovací body realizovány koncovými bloky firmy Siemens nebo Krone. Ke kontaktům na koncových blocích jsou nastálo připojeny vstupně-výstupní porty elektronických zařízení a pomocí dočasných spojů mezi kontakty se provádí propojení jednotlivých zařízení. Tato dočasná propojení jsou evidována v databázi WIRING. Kromě záznamu spojení musí být v databázi WIRING možnost jejich modifikace.

4.2 Datová základna

Data, která systém WIRING používá, jsou uložena v databázi vytvořené v SŘBD MySQL.

4.2.1 SŘBD MySQL

Obrázek 4 – Logo MySQL

Zdroj: [9], vlastní zpracování

Databázový systém navržený švédskou firmou MySQL AB nyní vlastněnou společností Sun Microsystems, dceřinnou společností Oracle Corporation. Je k dispozici se dvěmi licencemi, bezplatnou GPL a placenou komerční licencí.

GPL (General Public License) poskytuje nabyvateli software produktu právo jej modifikovat, kopírovat a dál rozšiřovat i jakoukoli odvozenou verzi tohoto produktu

za předpokladu, že všechny díla vzniklá za využití tohoto produktu budou dále poskytována opět s touto licencí a nebudou využita pro účely prodeje.

SŘBD MySQL patří do rodiny relačních databázových systémů a jejím programovacím jazykem je SQL. Jde o verzi jazyka SQL mírně odlišnou od jazyka používaného v SŘBD Oracle, avšak odlišnosti nejsou nikterak významné, a proto lze považovat SŘBD MySQL za standardního zástupce relačních databázových modelů.

Samozřejmě MySQL umožňuje síťové připojení a víceuživatelskou modifikaci dat. [9]

4.3 Uživatelské rozhraní

Protože uživatelský komfort MySQL není velký, jde v podstatě o terminálový přístup přes uživatelský řádek, bylo nutno připojit SŘBD MySQL k hypertextovému preprocesoru pracujícímu na platformě webové stránky. K tomuto účelu se hodí PHP.

4.3.1 PHP Hypertext Preprocesor

Obrázek 5 - Logo PHP

Zdroj:[3], vlastní zpracování

PHP je serverový skriptovací jazyk, který umožňuje tvořit dynamické webové stránky s mnohými možnostmi. PHP původně znamená Personal Home Page a vzniklo v roce 1996, od té doby prošlo velkými změnami a nyní tato zkratka znamená Hypertext Preprocessor. Pomocí PHP lze měnit, ukládat a mazat data, což se děje přímo na webovém serveru, uživatelům jsou pak posílány pouze výsledky těchto operací. Z toho plyne výhoda, že uživatelům stačí použít pouze webový prohlížeč a není nutno, aby měli na svém počítači instalované jakékoli doplňky. Stejně tak je tím zaručeno, že všichni uživatelé dostanou stejná data a ve stejné formě, která byla k dispozici v okamžiku dotazu klienta na server. [3]

4.3.2 Aplikace webového serveru

Aby server mohl pracovat s PHP v zabezpečeném režimu, je nutno mít na něm nainstalovánu službu webový server. Protože u databázového systému WIRING bylo předpokládáno, že operačním systémem serveru budou MS Windows, byl jako webový server zvolen Apache. Apache je obecně uznáván jako světově nejoblíbenější webový server (HTTP server). Původně byl určen pro prostředí Unix, později byl webový server Apache portován do Windows a dalších operačních systémů. Jméno "Apache" pochází ze slova "patchy", protože vývojáři byl tento server používán k popisu dřívější verze jejich softwaru.

Web server Apache nabízí celou řadu funkcí webového serveru, včetně CGI, SSL, a virtuální domény. Apache také podporuje plug-in moduly pro rozšiřitelnost. Apache je svobodný software, distribuovaný společností Apache Software Foundation, která podporuje různé zdarma poskytované open source pokročilé webové technologie. [4]

4.4 Struktura databáze

4.4.1 Tabulky v databázi WIRING

4.4.1.1 Tabulka “ wiring“

Tabulka „wiring“ je základní tabulkou databáze. Obsahuje údaje o jednotlivých spojeních mezi spojovacími body. Spojovací body jsou pozice, mezi kterými je vytvořeno spojení. Mohou to být vstupně-výstupní porty jakéhokoliv zařízení. Databáze WIRING byla navržena speciálně pro telekomunikační firmu, aby zaznamenávala spojení na spojovacím rámu DDF (Digital Distribution Frame).

Struktura:

Tabulka 6 – Struktura tabulky WIRING

Poz.	Pole	Typ	Null	Klíč	Default	Extra
	Id	int(10) unsigned		PRI	NULL	auto_increment
Spoj. bod 1	Loc1	varchar(20)				
	Device1	varchar(20)				
	Port1	varchar(10)				
	Ddf1	varchar(10)	YES		NULL	
Spoj. bod 2	Loc2	varchar(20)				
	Device2	varchar(20)				
	Port2	varchar(10)				
	Ddf2	varchar(10)	YES		NULL	
Vzdal. zaříz.	Loc3	varchar(20)	YES		NULL	
	Device3	varchar(20)	YES		NULL	
	Port3	varchar(10)	YES		NULL	
	Special	varchar(40)	YES		NULL	
	Popis	varchar(200)	YES		NULL	
	Modified	varchar(20)	YES		NULL	

Zdroj: používaná aplikace WIRING, vlastní zpracování

Primárním klíčem této tabulky je automaticky vkládané číslo v poli „Id“. Pole Loc# obsahuje název lokality, kde se spojovací body nacházejí. PoleDevice# a pole Port# jsou označení zařízení a vstupně-výstupního portu. Pole Ddf# obsahuje souřadnice spojovacího bodu na rámu DDF.

4.4.1.2 Tabulka “spojovaky_wiring“

Tato tabulka obsahuje záznamy o propojení spojovacích bodů mezi jednotlivými lokalitami.

Struktura:

Tabulka 7 – Struktura tabulky spojovaky wiring

Pole	Typ	Null	Klíč	Default	Extra
Id	int(10)		PRI	NULL	auto_increment
leva	varchar(15)				
prava	varchar(15)				

Zdroj: používaná aplikace WIRING, vlastní zpracování

Pole Id je automaticky se zvyšující primární klíč této tabulky. Pole „leva“ a pole „prava“ obsahují údaj z pole „device1“ v tabulce wiring.

4.4.1.3 Tabulka “wiring_acces“

Tabulka „wiring_acces“ obsahuje záznam přístupových jmen, hesel a přístupových práv jednotlivých uživatelů databáze.

Struktura:

Tabulka 8 – struktura tabulky wiring acces

Pole	Typ	Null	Klíč	Default	Extra
User	varchar(15)		PRI		
Password	varchar(16) binary				
Priority	varchar(5)				

Zdroj: používaná aplikace WIRING, vlastní zpracování

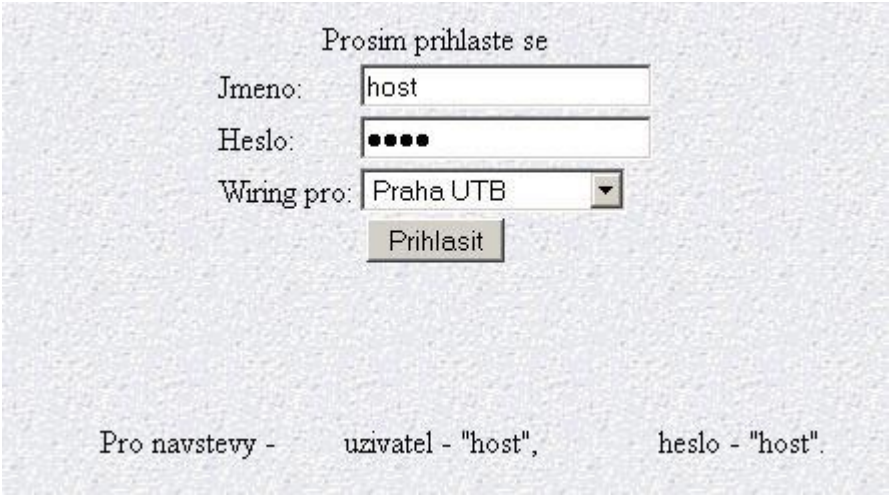
Primární klíč této tabulky je pole User, obsahuje přihlašovací jméno. Password je přihlašovací heslo uživatele databáze zakódované pomocí kódovacího algoritmu MySQL. Pole Priority určuje úroveň práv uživatele. Uživatel s prioritou 3 má právo k prohlížení databáze, uživatel s prioritou 2 může navíc provádět změny záznamů v tabulkách wiring a spojovaky_wiring. Prioritu 1 má pak pouze administrátor databáze.

4.5 Struktura uživatelského rozhraní

4.5.1 Vzhled přístupových formulářů

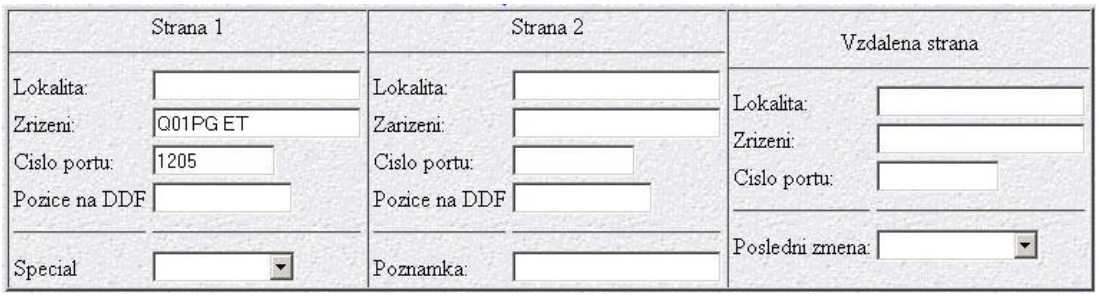
Vstupní stránka do systému WIRING žádá přihlášení. Po zadání přihlašovacích údajů, volbě databáze a kliknutí na tlačítko „Přihlásit“ přejde uživatel na stránku vyhledávání.

Obrázek 6 - Vstupní dotaz



Zdroj: používaná aplikace WIRING, vlastní zpracování

Obrázek 7 - Základní formulář



Zdroj: používaná aplikace WIRING, vlastní zpracování

Vyplněním příslušných políček hledanými údaji bude po kliknutí na tlačítko „Hledej“ zobrazen seznam všech řádků databáze vyhovující zadaným údajům. Pro vyhledání údajů pomocí částečného řetězce lze použít znak „%“ nahrazující libovolný počet libovolných znaků.

K zobrazeným řádkům jsou automaticky přidána tlačítka „editovat“a „rozpojit“.

Obrázek 8 - Vyhledaný řádek tabulky

Strana1				Strana2				Strana3			Poznámky		
Lokalita	Zaoizení	Port	Ddf	Lokalita	Zaoizení	Port	Ddf	Lokalita	Zaoizení	Port	Special	Poznámka	Modifikoval
UTB_DDF1	Q01PG ET	1205	R/10-14	UTB_DDF1	SDH 22/411	31	30/10-7						kropac_m

Zdroj: používaná aplikace WIRING, vlastní zpracování

Poklepáním na tlačítko editovat bude znovu zobrazen základní formulář s vyplněnými hodnotami, které je možno změnit výběrem hodnoty z rolovacího seznamu v poli „Zarizeni“ v části „Strana 2“. Zapsáním čísla do pole „Cislo portu“ je připojovaný spojový bod jednoznačně identifikován.

Po použití tlačítka rozpojit je zobrazen základní formulář, kde na místě „Cislo portu“ v části „Strana 2“ je předvyplněno NC (Not Connected). Tento údaj je indikátorem pro rozpojení spojových bodů. Vyhledaný řádek v tabulce „wiring“ je pomocí SQL příkazu „UPDATE“ změněn tak, aby v sekci „Strana2“ neobsahoval žádné údaje, kromě „NC“ v poli „Port2“. Stejně je upraven i řádek obsahující v sekci „Strana1“ stejné údaje, které zobrazuje i řádek v sekci „Strana2“.

4.5.2 SQL příkazy

Pro ověření uživatele je v PHP souboru použit SQL příkaz:

" SELECT * FROM wiring_acces WHERE User = '$User_entry' ", kde proměnná '$User_entry' je obsah pole „jmeno“ ve vstupním dotazu. Obsah pole „Heslo“ je pak porovnán s obsahem atributu Password a při shodě je volána obrazovka s vyhledávacím formulářem.

Vyhledávací formulář používá SQL příkaz:

"SELECT * FROM $tabulka WHERE Loc1 LIKE '$Loc1edit' AND Device1 LIKE '$Device1edit' AND Port1 LIKE '$Port1edit' AND Ddf1 LIKE '$Ddf1edit'

AND Loc2 LIKE '$Loc2edit' AND Device2 LIKE '$Device2edit' AND Port2 LIKE '$Port2edit' AND Ddf2 LIKE '$Ddf2edit' AND Loc3 LIKE '$Loc3edit' AND Device3 LIKE '$Device3edit' AND Port3 LIKE '$Port3edit' AND Special LIKE '$Specialedit'AND Popis LIKE '$Popisedit' AND Modified LIKE '$Modifiededit' ORDER BY Ddf1 "

Kde proměnné mají tento obsah:

Tabulka 9 – Obsah proměnných ve vyhledávacím SQL příkazu

Proměnná	Obsah
$tabulka	Tabulka Wiring jako zdroj záznamů
$Loc1edit $Loc2edit $Loc3edit	Obsah pole lokalita ve všech oddílech formuláře
$Device1edit$Device2edit $Device3edit	Obsah pole zarizeni ve všech oddílech formuláře
$Port1edit $Port2edit $Port3edit	Obsah pole Cislo portu ve všech oddílech formuláře
$Ddf1edit $Ddf2edit	Obsah pole Pozice na DDF
$Specialedit	Obsah rolovacího seznamu Special
$Popisedit	Obsah pole poznámka
$Modifiededit	Obsah rolovacího seznamu Poslednizmena

Zdroj: používaná aplikace WIRING, vlastní zpracování

Každá proměnná je předem vyplněna znakem „%“, který v SQL znamená libovolný počet libovolných znaků, proto není-li určité pole formuláře vyplněno, je považováno za libovolné.

5 Vlastní zpracování návrhu systému WIRING

5.1 Identifikace bariér v databázi

Databáze obsahuje jen jednu hlavní tabulku, kterou je tabulka wiring. Tato tabulka neodpovídá první normální formě, neboť obsahuje násobná data v atributech Loc1, Loc2, Loc3, Device1, Device2 a Device3. Dále, každé spojení musí být v této tabulce zapsáno dvakrát, pro zobrazení obou směrů spojení.

Struktura databáze je udržována pomocí algoritmů skriptovacího jazyka PHP. Není využíváno funkcí, které poskytuje SŘBD MySQL, ani cizích klíčů pro soustředění vícenásobných záznamů do zvláštní tabulky. To znamená, že databáze nemůže být funkční bez připojení na toto rozhraní. S tímto nedostatkem je spojena obtížnost začlenění databáze do vyššího databázového systému.

5.2 Oprava struktury databáze

5.2.1 Změna struktury tabulek

Aby bylo dosaženo stavu tabulky wiring podle první normální formy, je třeba tuto tabulku rozdělit do tří tabulek. Z tabulky wiring budou odstraněny atributy Device a Loc, budou pro ně zřízeny samostatné tabulky devices s locality a tabulka wiring bude pro větší srozumitelnost přejmenována na con_point.

Tabulka 10 – Struktura tabulky con_point

Poz.	Pole	Typ	Null	Klíč	Default	Extra
Prim. klíč	idCON_POINT	int(11)	NO	PRI	NULL	auto_increment
Cizí klíč	idLOCALITY	int(11)	YES	MUL	NULL	
Cizí klíč	idDEVICE	int(11)	YES	MUL	NULL	
Č. portu	PORT	varchar(15)	YES		NULL	
Označení pozice	DDF	varchar(15)	YES		NULL	
Poznámkake spoj. bodu	NOTE	text	YES		NULL	

Zdroj: vlastní zpracování

Tabulka 11 – Struktura tabulky locality

Poz.	Pole	Typ	Null	Klíč	Default	Extra
Prim. Klíč	idLOCALITY	int(11)	NO	PRI	NULL	auto_increment
Jedinečné označení	TITLE	varchar(30)	NO	UNI	NULL	
Popis umístění	LOCATION	varchar(255)	YES		NULL	

Zdroj: vlastní zpracování

Tabulka 12 – Struktura tabulky devices

Poz.	Pole	Typ	Null	Klíč	Default	Extra
Prim. Klíč	idDEVICE	int(11)	NO	PRI	NULL	auto_increment
Jedinečné označení	DEV_NAME	varchar(80)	NO	UNI	NULL	

Zdroj: vlastní zpracování

Jednoznačná identifikace každého spojovacího bodu je zajištěna existencí alespoň jednou z identifikačních dvojic údajů, kterými jsou označení lokality a označení pozice na DDF, označení zařízení a číslo portu. Níže je popsán způsob jak této jednoznačné identifikace dosáhnout.

Dvojnásobný zápis spojení bude nahrazen samostatnou tabulkou connection, která propojí dva zápisy z tabulky con_point. Tato nová tabulka stanoví základ pro zajištění, aby nemohl být propojen jeden spojovací bod se dvěma různými spojovacími body stejným typem spojení.

Tabulka 13 – Struktura tabulky connection

Poz.	Pole	Typ	Null	Klíč	Default	Extra
Prim. klíč	idCONNEC	int(11)	NO	PRI	NULL	auto_increment
Cizí klíč idCON_POINT	SIDE1	int(11)	NO	MUL	NULL	
Cizí klíč idCON_POINT	SIDE2	int(11)	NO	MUL	NULL	
Cizí klíč typ spojení	idTYPE	int(11)	NO	MUL	1	
Poznámka ke spojení	CONNEC_NOTE	text	YES		NULL	

Zdroj: vlastní zpracování

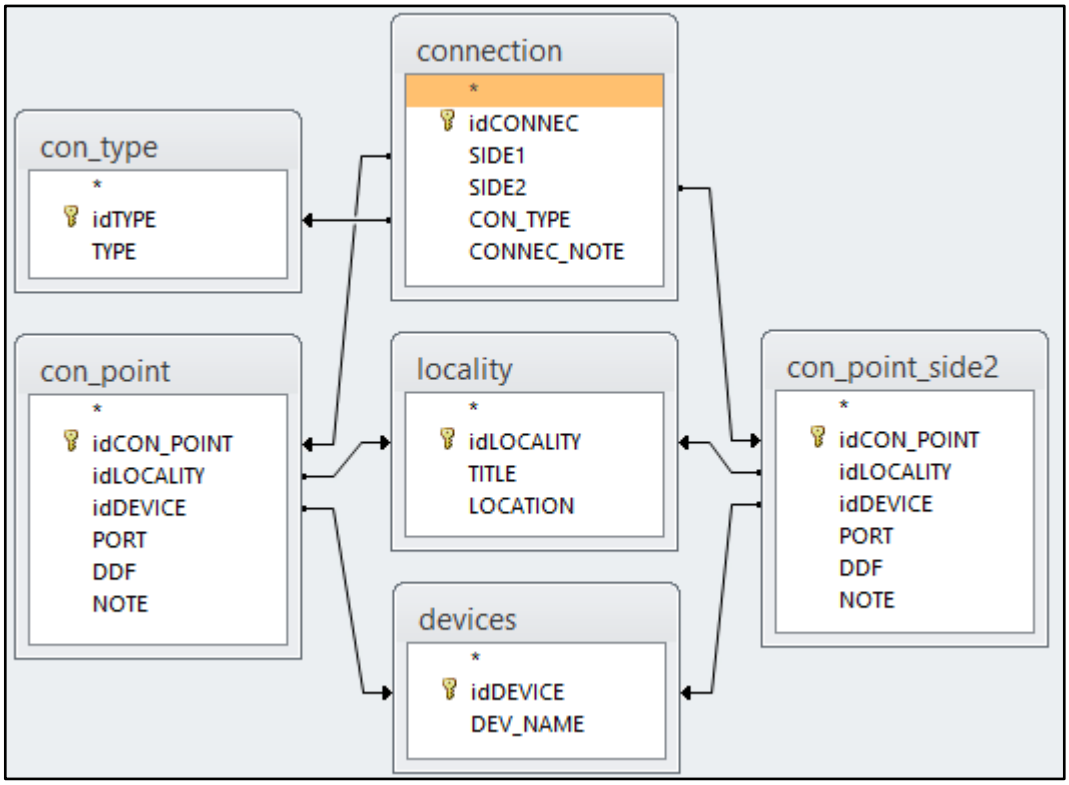
Pro zajištění jednotného značení typu spojení bude do databáze vložena pomocná tabulka konstant obsahující označení typů spojení. Její název bude con_type.

Tabulka 14 – Struktura tabulky con_type

Poz.	Pole	Typ	Null	Klíč	Default	Extra
Prim. Klíč	idTYPE	int(11)	NO	PRI	NULL	
Konstanta typu spojení	TYPE	varchar(45)	NO	UNI	NULL	

Zdroj: vlastní zpracování

Obrázek 9 – Schéma propojení tabulek pomocí klíčů



Zdroj: vlastní zpracování

5.2.2 Pohledy

Pro jednodušší organizaci dat je výhodné vložit do databáze Wiring pohledy spojující data z různých tabulek, která spolu logicky i funkčně souvisí. Další výhodou je i fakt, že pohledy bude možno použít pro tvorbu uživatelského rozhraní.

Pohled con_points a con_points_side2, který je přejmenovanou kopií pohledu con_points, umožní příkazu LEFT JOIN v pohledu connected_points spojení dvou řádků z tabulky con_point v jednom řádku pohledu. Pravidla standardního SQL neumožňují provést spojení tabulky se sebou samou, proto bude nutno použít pohledy, ke kterým SQL přistupuje jako k tabulkám.

- Příkaz vytvářející pohled con_points viz. Příloha 1
- Příkaz vytvářející pohled con_points_side2 viz. Příloha 2

Bude také vhodné vytvořit pomocný pohled connection_type, který propojí tabulky connection a con_type a zjednoduší předpis celkového pohledu na databázi Wiringconnected_points.

- Příkaz vytvářející pohled connection_type viz. Příloha 3

- Příkaz vytvářející pohled connected_points viz. Příloha 4

5.2.3 Rutiny spouštěné z SQL

Pro zajištění pravidel a postupů zadávání, modifikace a mazání dat v databázi je vhodné nahradit procedury PHP, použité ve verzi databáze Wiring zbavované bariér, procedurami SŘBD MySQL. Do SŘBD MySQL byla možnost vytvářet procedury implementována až od verze 5.0. [9]

5.2.3.1 Definice a volání procedury add_con_point

Tato procedura zajistí vložení nového, nebo úpravu existujícího připojovacího bodu. Procedura bude volána příkazem „calladd_con_point(*p_loc, p_dev, p_port, p_ddf, p_note*)“ Proměnné ve volacím příkazu jsou:

- p_loc název lokality (může být již zapsaná nebo nová)
- p_dev název zařízení (může být již zapsané nebo nové)
- p_port číslo portu (pro dané zařízení jedinečný)
- p_ddf označení pozice na rozvodu (pro danou lokalitu jedinečné)
- p_note poznámka týkající se přímo připojovacího bodu

Zda zápis existuje, bude určeno podle jedinečných dvojic atributů lokalita-pozice na rozvodu nebo zařízení-port.

Zápis procedury add_con_point viz. Příloha 5.

5.2.3.2 Definice a volání procedury del_con_point

Tato procedura vymaže nepotřebný zápis připojovacího bodu. Před smazáním zápisu připojovacího bodu odstraní zápis spojení daného bodu s jiným připojovacím bodem, pokud existuje. Pokud půjde o poslední zápis dané lokality nebo zařízení v pohledu con_points, budou z tabulek vymazány locality respektive devices. Mazaný zápis bude identifikován podle jedinečných dvojic popsaných výše. Procedura bude volána příkazem call del_con_point (*p_loc, p_dev, p_port, p_ddf*). Proměnné v tomto příkazu jsou:

- p_loc název lokality
- p_dev název zařízení
- p_port číslo portu
- p_ddf označení pozice na rozvodu

Zápis procedury del_con_point viz. Příloha 6.

5.2.3.3 Definice funkce id_conpoints

Tato funkce vyhledá v pohledu con_points daný zápis a vrátí jeho ID. Funkce id_conpoints bude použita v následujících procedurách. Jako vstupní parametry jsou použity tyto hodnoty, parametry budou seskupeny podle jedinečných dvojic:

- p_loc název lokality
- p_ddf označení pozice na rozvodu
- p_dev název zařízení
- p_port číslo portu

Zápis funkce id_conpoints viz. Příloha 7.

5.2.3.4 Definice a volání procedury mod_connection

Procedurou mod_connection bude možno vytvořit nebo změnit spojení mezi dvěma spojovacími body. Vstupní parametry musí obsahovat pro oba spojované body alespoň jednu úplnou jedinečnou dvojici. Spojení připojovacích bodů bude provedeno dvakrát, aby bylo zaznamenáno spojení bodů v obou směrech. Volání procedury bude CALL mod_connection (*p_loc1, p_dev1, p_port1, p_ddf1, p_loc2, p_dev2, p_port2, p_ddf2, p_comtype, p_note*).

Význam vstupních parametrů:

- p_loc1; p_loc2 název lokality 1. a 2. strany spojení
- p_dev1;p_dev2 název zařízení 1. a 2. strany spojení
- p_port1; p_port2 číslo portu 1. a 2. strany spojení
- p_ddf1; p_ddf2 označení pozice na rozvodu 1. a 2. strany spojení
- p_contype typ spojení (default BASIC);
- p_note poznámka ke spojení

Zápis procedury mod_connection viz. Příloha 8.

5.2.3.5 Definice a volání procedury del_connection

Tato procedura bude volána příkazem CALL del_connection(p_side1, p_side2) v případě potřeby zrušení spojení mezi dvěma spojovacími body, pokud budeme chtít oba body zachovat v databázi.

Význam Parametrů procedury je:

- p_side1; p_side2 indexy záznamů připojovacích bodů

Zápis procedury del_connection viz. Příloha 9.

5.2.3.6 Nastavení přístupových práv

V upravené databázi WIRING bude vhodné přesunout agendu přístupových práv přímo do SŘBD MySQL. Bude tím zabezpečen i přístup přímo do databázového systému bez užití uživatelského rozhraní. V současné verzi databáze WIRING existuje sada přístupových práv definovaná pro přístup přes uživatelské rozhraní PHP. Každý přístupový účet je definován v tabulce wiring_access s přiřazenou úrovní přístupu. Každý účet se pak hlásí s univerzálními přístupovými právy s MySQL podle své úrovně. Toto řešení může způsobit, že pokud administrátor vytvoří přístupová práva pro tuto databázi přímo v MySQL, může vzniknout duplicitní sada práv. Pokud budou přístupová práva přidělena uživatelům přímo v SŘBD, k tomuto bezpečnostnímu riziku nemůže dojít.

Práva pro přístup můžou být přidělována opět ve třech kategoriích:

- Kategorie administrátor s právy na kompletní přístup ke všem možnostem správy databáze. Nastavena příkazem:
  GRANT ALL ON wiring.* TO *už_jmeno*@'%';
- Kategorie editor s právy prohlížet obsah a spouštět procedury. Nastavena příkazem:
  GRANT SELECT,EXECUTE ON wiring.* TO *už_jmeno*@'%';
- Kategorie host právy na prohlížení obsahu databáze. Nastavena příkazem:
  GRANT SELECT ON wiring.* TO *už_jmeno*@'%' [9]

5.3 Oprava uživatelského rozhraní

Po zařazení pohledů a rutin do základního datového systému, je možno vypustit z uživatelského rozhraní veškeré modifikační procedury. Uživatelské rozhraní může být zaměřeno výhradně na aktivity, které vyplývají z účelu této části databázového systému WIRING, tedy zprostředkovat uživateli srozumitelný a přehledný přístup k informacím, které jsou v systému uloženy.

Základní formulář původního systému WIRING může zůstat beze změny, jen sekce vzdálené připojení bude odstraněna jako nepotřebná. Do sekcí Strana 1 a Strana 2 bude přidáno pole „Poznámka spojovacího bodu“ a pole „Poznámka“ bude přejmenováno na „Poznámka spojení“. Pole „Special“ bude využito pro obsah atributu Typ_spojení. Příslušný SQL dotaz SELECT se bude nově dotazovat pohledu connected_points. Na dalším pohledu pak budou zobrazeny jednotlivé řádky výsledku dotazu. Pro vyšší přehlednost nebudou zobrazeny atributy id_1 a id_2.

Zde budou využita původní tlačítka z tohoto pohledu. Tlačítkem „rozpojit“ bude zavolána procedura del_connection. Tlačítko „editovat“ zobrazí vyplněný základní formulář. Pro úpravu pole „Special“ bude pole možno rozkliknout na rolovací seznam obsahující hodnoty atributu TYPE z tabulky con_type. Po provedení požadovaných změn bude obsah polí odeslán do procedury mod_connection.

Uživatelské rozhraní bude dále obsahovat pomocné menu, které bude dávat možnost vložit nový spojovací bod nebo nepotřebný spojovací bod smazat. Obě funkce budou provedeny přes částečné zobrazení základního formuláře, kde bude možno vyplnit pouze pole v sekci strana 1. Data z tohoto formuláře budou použity v proceduře add_con_point respektive del_con_point.

Pro zabezpečení přístupu k datům bude možno použít standardních nástrojů, které jsou součástí SŘBD MySQL, tedy příkazů skupiny DCL. Výhodou tohoto přesunutí funkce řízení přístupu bude jednotnost řízení databáze přes uživatelské rozhraní s přímým vstupem do SŘBD.

6 Závěr

Předkládaná bakalářská práce popsala užití relační databázové technologie v praxi technické části telekomunikačního podniku na příkladu databázového nástroje WIRING.

Používání tohoto nástroje si vyžádala potřeba přesných a aktuálních informací o fyzických propojeních provedených na spojovacím rámu jednotlivých lokalit s umístěnou technologií. Pro snadné provedení prací je třeba mít tyto informace k dispozici přímo na technologické lokalitě, bez rizika nedostupnosti z důvodu výpadku datové sítě. Dále je třeba, aby pracovník, který provedl změnu přímo na rozvodu DDF, měl možnost změny neprodleně zaznamenat do databáze a tím zajistit, aby nedocházelo k významnému časovému rozdílu mezi skutečným stavem propojení a stavem zaznamenaným. Tomuto účelu nejlépe vyhovuje lokální databáze, jejíž paměťové jádro je uloženo na počítači nejběžněji používaném na těchto lokalitách, tedy na PC s operačním systémem WINDOWS. Pro tento účel byl logicky zvolen SŘBD MySQL pro možnost jej použít s volnou licencí GPL, neboť nejde o komerční užití tohoto SŘBD. Dalším důvodem pro tuto volbu byla možnost použít skriptovací jazyk PHP, užívaný pro tvorbu internetových prezentací, jako platformu pro vhodné uživatelské rozhraní, o níž je k dispozici velké množství informací, popsaných v odborné literatuře.

Sdílením dat po podnikové vnitřní síti se otevírá možnost předat uložené informace dalšímu, vyššímu informačnímu systému, kde mohou být použity pro účely celopodnikového plánování.

Z důvodu užití starší verze SŘBD MySQL, která ještě neobsahovala možnost použít procedury a funkce spouštěné z SQL pro zjednodušení a zautomatizování zápisu dat do databáze (verze před zavedením standardu SQL:1999 [6]), byl autor nástroje WIRING nucen využít k tomuto účelu programovacích prostředků jazyka PHP.

Jelikož PHP je univerzální programovací nástroj pro vytváření internetových prezentací a obsahuje širokou škálu nástrojů pro zápis dat, může při jeho použití docházet k chybám vzniklým integritními omezeními struktury databáze nebo neúplným zápisem propojení spojovacích bodů. Je velice obtížné všechny tyto možné chyby ošetřit tak, aby spolupráce uživatelů, PHP a MySQL pracovala naprosto bezvadně.

Zavedením standardu SQL:1999 organizacemi ANSI a ISO byla dána možnost tyto bariéry odstranit. Tato bakalářská práce se také zabývá návrhem možnosti využití rutin v databázovém nástroji WIRING. Tento přístup umožňuje normalizaci databáze WIRING až na třetí normální formu, která v původní verzi databáze nebyla dodržena z důvodu zjednodušení užití nástroje PHP jako hlavního prostředku pro zajištění možnosti zápisu dat.

Zjištěním datové normalizace WIRINGu bylo odstraněno nebezpečí redundance dat, snížily se nároky na datový prostor a hlavně bylo umožněno snazší sdílení dat s jinými informačními systémy.

Opravená verze databáze WIRING je natolik univerzální, že ji je možno používat pro evidenci propojení i jiných typů zařízení než je pouhé propojení na spojovacím rámu DDF. Jako příklad lze uvést záznamy o spojení prvků strukturované kabeláže, kde jsou spojovacími body jednotlivé zásuvky datových kabelů v příslušném rozvaděči. Další možností je využití databáze WIRING pro záznamy o fyzickém propojení počítačů v rozsáhlé datové síti. Při použití většího počtu portů na mnoha síťových prvcích by mohl být problém určit, bez přesné evidence, do kterého portu jsou jednotlivá zařízení připojena.

Z uvedeného je zřejmé, že databáze WIRING je velice praktický nástroj umožňující zaznamenat a následně vyhledat propojení vstupně-výstupních rozhraní mezi sebou bez nutnosti znalosti způsobu jak najít tyto údaje přímo v propojovaných zařízeních. Je vhodný pro pracovníky bez hlubší odborné znalosti propojovaných systémů. Jeho databázový charakter a užití datové sítě umožňuje sdílení informací a případné modifikace z míst vzdálených od samotného úložiště informací.

7 Seznam literatury

[1] RIORDAN, Rebecca M. Vytváříme relační databázové aplikace. Praha: ComputerPress, 2000. ISBN 8-07-226360-9

[2] CONOLLY, Thomas, BEGG, Carolyn, HOLOWCZAK, Richard.Mistrovství - databáze Profesionální průvodce tvorbou efektivních databází.vyd. Brno: ComputerPress, 2009. ISBN 978-80-251-2328

[3] LACKO, Ľuboslav. PHP a MySQL, hotová řešení; CP Books. a. s. Brno, 2005. 299 s.ISBN 80-251-0397-8

[4] PONKRÁC, Miroslav; PHP a MySQL bez předchozích znalostí. Brno: ComputerPress a. s. 2011. 221 s. IBSN 978-80-251-1758-3

[5] POKORNÝ, Jaroslav; Halaška, Ivan; DATABÁZOVÉ SYSTÉMY; Vydavatelství ČVUT 2004. 148 s. IBSN 80-01-02789-9

[6] SHELDON, Robert; SQL začínáme programovat; GradaPublishing, Praha; 500 s. IBSN 80-247-0999-6

[7] SCHWARTZ, Baron, ZAITSEV Peter, TKACHENKO, Vadim, ZAWODNY, Jeremy D, LENTZ, Arjen, BALLING, Derek J; MySQL profesionálně - optimalizace pro vysoký výkon; ZONER software a. s. Brno. 712 s. IBSN 978-80-7413-035-9

7.1 Internetové informační zdroje

[8] Webová knihovna společnosti ORACLE dokumentace databáze ORACLE 10g release 2 dostupná na http://www.oracle.com/pls/db102/homepage

[9] Webová knihovna společnosti ORACLE věnovaná databázi MySQL dostupná na http://dev.mysql.com/doc/refman/5.7

8 Seznam tabulek

9 Seznam obrázků

10 Seznam příloh

Přílohy

Příloha 1 - Příkaz definice pohledu con_points

```
CREATE VIEW `con_points` AS
select
    `con_point`.`idCON_POINT` AS `ID`,
    `locality`.`TITLE` AS `LOKALITA`,
    `devices`.`DEV_NAME` AS `ZARIZENI`,
    `con_point`.`PORT` AS `PORT`,
    `con_point`.`DDF` AS `POZICE_DDF`,
    `con_point`.`NOTE` AS `POZNAMKA_BODU`
from
    (`con_point`
leftjoin (`devices`
join `locality`) ON (((`devices`.`idDEV` = `con_point`.`idDEVICE`)
        and (`locality`.`idLOCALITY` = `con_point`.`idLOC`))));
```

Příloha 2 – Příkaz definice pohledu a con_points_side2

```
CREATE VIEW `con_points_side2` AS
select
    `con_points`.`ID` AS `id_2`,
    `con_points`.`LOKALITA` AS `Lokalita_2`,
    `con_points`.`ZARIZENI` AS `Zarizeni_2`,
    `con_points`.`PORT` AS `Port_2`,
    `con_points`.`POZICE_DDF` AS `Pozice_DDF_2`,
    `con_points`.`POZNAMKA_BODU` AS `Poznamka_bodu_2`
from
    `con_points`
```

Příloha 3 – Příkaz definice pohledu connection_type

```
CREATE VIEW `connection_type` AS
select
      `connection`.`idCONNEC` AS `idCONNEC`,
      `connection`.`SIDE1` AS `SIDE1`,
      `connection`.`SIDE2` AS `SIDE2`,
      `con_type`.`TYPE` AS `TYPE`,
      `connection`.`CONNEC_NOTE` AS `CONNEC_NOTE`
from
      (`connection`
       leftjoin `con_type` ON ((`connection`.`CON_TYPE` =
`con_type`.`idTYPE`)))
```

Příloha 4 – Příkaz definice pohledu connected_points

```
CREATE VIEW `connected_points` AS
select
    `con_points`.`ID` AS `id_1`,
    `con_points`.`LOKALITA` AS `Lokalita_1`,
    `con_points`.`ZARIZENI` AS `Zarizeni_1`,
    `con_points`.`PORT` AS `Port_1`,
    `con_points`.`POZICE_DDF` AS `Pozice_DDF_1`,
    `con_points`.`POZNAMKA_BODU` AS `Poznamka_bodu_1`,
    `connection_type`.`TYPE` AS `Typ_spojeni`,
    `connection_type`.`CONNEC_NOTE` AS `Poznamka_spojeni`,
    `con_points_side2`.`id_2` AS `id_2`,
    `con_points_side2`.`Lokalita_2` AS `Lokalita_2`,
    `con_points_side2`.`Zarizeni_2` AS `Zarizeni_2`,
    `con_points_side2`.`Port_2` AS `Port_2`,
    `con_points_side2`.`Pozice_DDF_2` AS `Pozice_DDF_2`,
    `con_points_side2`.`Poznamka_bodu_2` AS `Poznamka_bodu_2`
from
    (`con_points`
leftjoin (`connection_type`
                join `con_points_side2`) ON (((`con_points`.`ID` =
                `connection_type`.`SIDE1`)
        and (`connection_type`.`SIDE2` = `con_points_side2`.`id_2`))))
```

Příloha 5 – Zápis procedury add_con_point

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `add_con_point`
(
		IN p_loc VARCHAR(30),
		IN p_dev VARCHAR(80),
		IN p_port VARCHAR(15),
		IN p_ddf VARCHAR(15),
		IN p_note VARCHAR(255)
)
BEGIN
#Definice dočasných promennych
DECLARE t_idCon INT(11);
DECLARE t_idLoc INT(11);
DECLARE t_idDvc INT(11);
DECLARE t_port VARCHAR(15);
DECLARE t_ddf VARCHAR(15);
DECLARE t_note VARCHAR(255);
#Zajistit kontinuitu zmeny databáze
COMMIT; -- ukoncenipredchozi transakce
SET TRANSACTION READ WRITE;
#Najit id lokality
SET t_idLoc = (SELECT idLOCALITY FROM locality
	WHERE TITLE = p_loc);
SET t_idDvc = (SELECT idDEV FROM devices
	WHERE DEV_NAME = p_dev);
IF t_idDvc IS NULL
#Pridatzarizeni
THEN
	INSERT INTO devices(DEV_NAME)
		VALUES (p_dev);
#Zaznamenat id novehozarizeni
	SET t_idDvc = (SELECT idDEV FROM devices
		WHERE DEV_NAME = p_dev);
END IF;
SET t_idCon = (SELECT idCON_POINT FROM con_point
	WHERE (idLOC = t_idLoc AND DDF = p_ddf)
	OR (idDEVICE = t_idDvc AND PORT = p_port));
IF t_idCon IS NULL
#Pridat spojovací bod
THEN
	INSERT INTO con_point(idLOC,idDEVICE,PORT,DDF,NOTE)
		VALUES (t_idLoc,t_idDvc,p_port,p_ddf,p_note);
ELSE
#Modifikovat spojovaci bod
IF p_loc IS NULL
```

```
        THEN
        SET t_idLoc = (SELECT idLOC FROM con_point WHERE idCON_POINT =
t_idCon);
        END IF;
        IF t_idDvc IS NULL
        THEN
        SET t_idDvc = (SELECT idDEVICE FROM con_point WHERE
idCON_POINT = t_idCon);
        END IF;
        IF p_port IS NULL
        THEN
        SET p_port = (SELECT PORT FROM con_point WHERE idCON_POINT =
t_idCon);
        END IF;
        IF p_ddf IS NULL
        THEN
        SET p_ddf = (SELECT DDF FROM con_point WHERE idCON_POINT =
t_idCon);
        END IF;
        IF p_note IS NULL
        THEN
        SET p_note = (SELECT NOTE FROM con_point WHERE idCON_POINT =
t_idCon);
        END IF;
                UPDATE con_point
           SET idLOC = t_idLoc,
                idDEVICE = t_idDvc,
                PORT = p_port,
                DDF = p_ddf,
                NOTE = p_note
           WHERE idCon_POINT = t_idCon;
        END IF;
        COMMIT; -- ukonceni teto transakce
        END$$
                        DELIMITER ;
```

Význam vstupních parametrů

p_loc	název lokality (může být již zapsaná nebo nová)
p_dev	název zařízení (může být již zapsané nebo nové)
p_port	číslo portu (pro dané zařízení jedinečný)
p_ddf	označení pozice na rozvodu (pro danou lokalitu jedinečné)
p_note	poznámka týkající se přímo připojovacího bodu

Příloha 6 – Zápis procedury del_con_point

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE
`del_con_point`(p_locvarchar(30),
        p_devvarchar(80),
        p_portvarchar(15),
        p_ddfvarchar (15))
BEGIN
DECLARE t_idint(11);/*Docasny index*/
DECLARE t_loc VARCHAR(30);/*Docasnynazev lokality*/
DECLARE t_dev VARCHAR(80);/*Docasnynazevzarizeni*/
#Zajistit kontinuitu zmeny databáze
COMMIT; -- ukoncenipredchozi transakce
SET TRANSACTION READ WRITE;

SET t_id = id_conpoints(p_loc,p_ddf,p_dev,p_port);
IF (t_id IS NOT NULL)/*Provest pokud existuje*/
THEN
/*Ulozitnazvy lokality a zarizeni*/
SET t_loc = (SELECT LOKALITA FROM con_points WHERE ID = t_id);
SET t_dev = (SELECT ZARIZENI FROM con_points WHERE ID = t_id);
/*Vymazat spojeni na dany spojovaci bod*/
DELETE FROM connection WHERE SIDE1 = t_id OR SIDE2 = t_id;
/*Vymazat dany spojovaci bod*/
DELETE FROM con_point WHERE idCON_POINT = t_id;
/*kontrola zda existuje jestedalsipouziti dane lokality*/
IF ((SELECT ID FROM con_points WHERE LOKALITA = t_loc)IS NULL)
THEN
/*Vymazat nepotrebnou lokalitu*/
DELETE FROM locality WHERE TITLE = t_loc;
/*kontrola zda existuje jestedalsipouzitidanehozarizeni*/
END IF;
IF ((SELECT ID FROM con_points WHERE ZARIZENI = t_dev)IS NULL)
THEN
/*Vymazat nepotrebnezarizeni*/
DELETE FROM devices WHERE DEV_NAME = t_dev;
END IF;
END IF;
END$$
DELIMITER ;
```

Význam vstupních parametrů

p_loc	název lokality
p_dev	název zařízení
p_port	číslo portu
p_ddf	označení pozice na rozvodu

Příloha 7 – Zápis funkce id_conpoints

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` FUNCTION `id_conpoints`
(       p_locvarchar(30),
        p_ddfvarchar(15),
        p_devvarchar(80),
        p_portvarchar (15)
) RETURNS int(11)
BEGIN
DECLARE t_id INT(11);
/*Pokud existuje jedinyzapisspojovaciho bodu*/
IF (SELECT COUNT(*) FROM con_points WHERE
        (Lokalita = p_loc AND Pozice_DDF = p_ddf)
        OR
        (Zarizeni = p_dev AND Port = p_port) = 1)
THEN
/*Funkce vratiidentifikacnicislozapisudaneho bodu*/
        SET t_id = (SELECT id FROM con_points WHERE
        (Lokalita = p_loc AND Pozice_DDF = p_ddf)
        OR
        (Zarizeni = p_dev AND Port = p_port)LIMIT 1);
ELSE
        /*V jinychpripadechvratinull*/
SET t_id = NULL;
END IF;
RETURN t_id;
END$$
DELIMITER ;
```

Význam vstupních parametrů

p_loc	název lokality
p_ddf	označení pozice na rozvodu
p_dev	název zařízení
p_port	číslo portu

Příloha 8 – Zápis procedury mod_connection

```
DELIMITER $$
USE `wiring_new`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `mod_connection`
        (p_loc1 varchar(30),            /*lokalita 1. con_point*/
        p_dev1 varchar(80),             /*zarizeni 1. con_point*/
        p_port1 varchar (15),           /*port 1. con_point */
        p_ddf1 varchar(15),             /*DDF pozice 1. con_point */
        p_loc2 varchar(30),             /*lokalita 2. con_point*/
        p_dev2 varchar(80),             /*zarizeni 2. con_point*/
        p_port2 varchar (15),           /*port 2. con_point */
        p_ddf2 varchar(15),             /*DDF pozice 1. con_point */
        p_contypevarchar(45),                   /*typ spojeni*/
        p_note text)                    /*poznámka ke spojeni*/
BEGIN
DECLARE t_id1 INT(11); /*ID 1. pripojovanehocon_point*/
DECLARE t_id2 INT(11); /*ID 2. pripojovanehocon_point*/
DECLARE t_id3 INT(11); /*ID 1. puv. pripojenehocon_point*/
DECLARE t_id4 INT(11); /*ID 2. puv. pripojenehocon_point*/
DECLARE t_id_typ INT(11);/*ID typu spojeni*/
#Zajistit kontinuitu zmeny databáze
COMMIT; -- ukoncenipredchozi transakce
SET TRANSACTION READ WRITE;
#zjistit ID typu spojeni
SET t_id_typ = (SELECT idTYPE FROM con_type
        WHERE TYPE = p_contype);
/*zjistit ID 1. pripojovanehocon_point*/
SET t_id1 = id_conpoints(p_loc1,p_ddf1,p_dev1,p_port1);
/*zjistit ID 1. puv. pripojenehocon_point*/
SET t_id3 = (SELECT SIDE2 FROM connection
        WHERE SIDE1 = t_id1 AND CON_TYPE = t_id_typ);
/*smazat 1. puv. spojeni */
CALL`del_connection`(t_id1,t_id3);
/*zjistit ID 2. pripojovanehocon_point*/
SET t_id2 = id_conpoints(p_loc2,p_ddf2,p_dev2,p_port2);
/*zjistit ID 2. puv. pripojenehocon_point*/
SET t_id4 = (SELECT SIDE2 FROM connection
        WHERE SIDE1 = t_id2 AND CON_TYPE = t_id_typ);
/*smazat 2. puv. spojeni */
CALL`del_connection`(t_id2,t_id4);
/*zapsat propojeni 1. smerem*/
INSERT INTO `wiring_new`.`connection`
                (`SIDE1`, `SIDE2`, `CON_TYPE`, `CONNEC_NOTE`)
VALUES      (t_id1, t_id2, t_id_typ, p_note);
/*zapsat propojeni 2. smerem*/
INSERT INTO `wiring_new`.`connection`
```

```
                (`SIDE1`, `SIDE2`, `CON_TYPE`, `CONNEC_NOTE`)
    VALUES  (t_id2, t_id1, t_id_typ, p_note);
    COMMIT; -- ukonceni teto transakce
    END$$
    DELIMITER ;
```

Význam vstupních parametrů:

p_loc1; p_loc2	název lokality 1. a 2. strany spojení
p_dev1;p_dev2	název zařízení 1. a 2. strany spojení
p_port1; p_port2	číslo portu 1. a 2. strany spojení
p_ddf1; p_ddf2	označení pozice na rozvodu 1. a 2. strany spojení
p_contype	typ spojení (default BASIC);
p_note	poznámka ke spojení

Příloha 9 – Zápis procedury del_connection

```
DELIMITER $$
CREATE DEFINER=`root`@`localhost` PROCEDURE `del_connection`
(       p_side1 int(11),
        p_side2 int(11)
)
BEGIN
#Zajistit kontinuitu zmeny databáze
COMMIT; -- ukoncenipredchozi transakce
SET TRANSACTION READ WRITE;
DELETE IGNORE FROM connection
WHERE SIDE1 = p_side1 AND SIDE2 = p_side2 LIMIT 1;
DELETE IGNORE FROM connection
WHERE SIDE1 = p_side2 AND SIDE2 = p_side1 LIMIT 1;
COMMIT; -- ukonceni teto transakce
END$$
DELIMITER ;
```

Význam vstupních parametrů:

p_side1; p_side2	indexy záznamů připojovacích bodů