ZÁPADOČESKÁ UNIVERZITA V PLZNI

FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

DIPLOMOVÁ PRÁCE

Návrh modulu GTN pro editaci dat jízdního řádu vlaků

Vedoucí práce: Ing. Petr Hloušek, PhD.

Autor: Bc. Petr Benedikt

2012

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Petr BENEDIKT**
Osobní číslo:	**E09N0179P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Dopravní elektroinženýrství a autoelektronika**
Název tématu:	**Návrh modulu GTN pro editaci dat jízdního řádu vlaků**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Prostudujte a popište funkce systému graficko-technologické nadstavby.
2. Navrhněte modul GTN pro vkládání a editaci dat jízdního řádu vlaků dle zadaných požadavků.
3. Otestujte navržené řešení a zhodnoťte dosažené výsledky.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí diplomové práce: **Ing. Petr Hloušek, Ph.D.**
Katedra aplikované elektroniky a telekomunikací

Konzultant diplomové práce: **Ing. Vlastimil Polach, Ph.D.**
AŽD Praha

Datum zadání diplomové práce: **18. října 2010**

Termín odevzdání diplomové práce: **11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Západočeská univerzita v Plzni
Fakulta elektrotechnická

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

Anotace

Benedikt, P. Návrh modulu GTN pro editaci jízdního řádu. Katedra aplikované elektroniky a telekomunikací, Západočeská univerzita v Plzni – Fakulta elektrotechnická, 2012, 37 s., vedoucí: Ing. Petr Hloušek, PhD.

V rámci této diplomové práce je popsána funkce Graficko-technologické nadstavby zabezpečovacího zařízení. Jsou zde prozkoumány vlastnosti relační databáze a možnosti jejího propojení s programy v různých programovacích jazycích. Následně je prozkoumána struktura zadané databáze a je navržen uživatelský scénář, který by měl zajistit správnou posloupnost zadávání dat a zamezit jejich chybné identifikaci. Na základě popsaného uživatelského scénáře je realizován návrh aplikačního prostředí editoru jízdních řádů s použitím jedné z metod propojení databáze s programovacím prostředím.

Klíčová slova

Graficko-technologická nadstavba zabezpečovacího zařízení, GTN, editor, jízdní řády, databáze MySQL, Java

Annotation

Benedikt, P. Design of GTN module for train timetable data editation. Department of applied electronics and telecommunication, University of West Bohemia in Pilsen – Faculty of electrical engineering, 2012, 37 p., head: Ing. Petr Hloušek, PhD.

A function of Graphical and Technological Layer of signalling system is described in this thesis. A relational database functions and connection options of the database and programs are explored in number of programming languages. The structure of particular database is further explored and the user scenario is suggested. This scenario should provide secure sequence of data input and prevent from false identification. A draft of the timetable editor's graphical user interface(GUI) is designed on the basis of described user scenario. This GUI is using one of the chosen methods to interconnect database and programming interface.

Key words

Graphical and Technological Layer of signalling system, GTN, editor, timetable of transport, MySQL database, Java

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne…… ……

Podpis

Obsah

Seznam použitých zkratek

GTN	*Graficko-Technologická Nadstavba zabezpečovacího zařízení*
AŽD	*Automatizace Železniční Dopravy*
DOZ	*Dálkové Ovládané Zabezpečovací zařízení*
ISOŘ	*Informační Systém Operativního Řízení*
CEVIS	*CEntrální Vozový Informační Systém*
CDS	*Centrální Dispečerský Systém*
ZZ	*Zabezpečovací Zařízení*
JOP	*Jednotné Obslužné Pracoviště*
GVD	*Grafikon Vlakové Dopravy*
INISS	*INtegrovaný Informační Systém Stanice*
PIK	*Personální Identifikační Karta*
ELDODO	*ELektronická DOpravní DOkumentace*
IŘS ŽD	*Informační a Řídicí Systém Železniční Dopravy*
TSI TAF	*Technical Specification for Interoperability for Telematic Applications for Freight (Technická specifikace interoperability telematických aplikací pro dopravu)*
TSI TAP	*Technical Specification for Interoperability for Telematic Applications for Passenger system (Technická specifikace interoperability telematických aplikací pro cestující)*
SQL	*Structured Query Language*
GPL	*GNU General Public License*
CRUD	*Create – Retrieve – Update – Delete*
ER	*Entity Relationship*
API	*Application Programming Interface*
ODBC	*Open DataBase Connectivity*
DSN	*Data Source Name*
DBMS	*DataBase Management System*
ADO.NET	*ActiveX Data Object for .NET*
JDBC	*Java DataBase Connectivity*
DAO	*Data Access Object*
DTO	*Data Transfer Object*
ORM	*Object Relational Mapping*
JPA	*Java Persistence API*
PHP	*Hypertext Preprocessor*

Úvod

GTN se již dostatečně rozšířila do praxe a společností AŽD Praha je požadován další vývoj aplikace, umožňující pracovníkům dráhy v řízené oblasti samostatné přidávání či editování jízdních řádů používaných v GTN. Zadáním tedy je vytvoření aplikace, která umožní tabelární zadávání dat do databáze. Správná funkce editoru však nespočívá jen v tabelárním zadávání, ale zároveň také v možnosti data zobrazovat a editovat v případě, že byl zadán špatný údaj nebo jen v průběhu užívání dojde k jejich změně.

V zadání není specifikováno, jak by aplikace měla vypadat pro uživatele, ani jaké programové prostředky by měly být zvoleny, zde se tedy otevírá prostor pro tvůrčí možnosti při návrhu aplikace. Nejprve však bylo nutné prozkoumání vlastností GTN, dodané databáze, a možností pro spojení databáze s aplikační vrstvou.

V následujících kapitolách jsou tedy popsány funkce stávající aplikace GTN, dále prozkoumány možnosti práce s databází a její možnosti propojení s aplikační vrstvou.

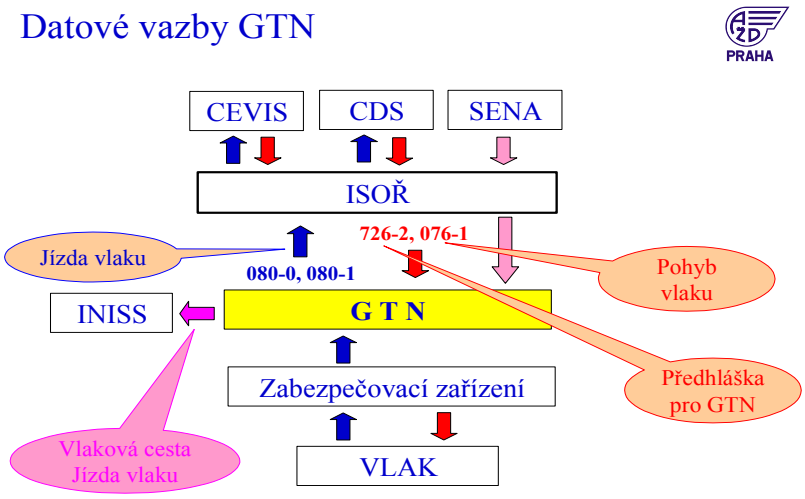
Je zde proveden rozbor zadané databáze a na základě tohoto rozboru je navržen uživatelský scénář a potřebné databázové příkazy, které by měly sloužit při samotné implementaci do programovacího prostředí.

Výstupem práce by měl být návrh aplikace pro potřeby editace a vkládání dat jízdních řádů. Jako vstupní a výstupní data programu by měla sloužit zadaná databáze a program by měl obsahovat vhodné funkce pro kontrolu správnosti dat.

1. GTN – Graficko-technologická nadstavba

Počítačová aplikace Graficko-technologická nadstavba zabezpečovacího zařízení je telematická aplikace vyvinutá firmou AŽD Praha jako nástroj pro podporu efektivního řízení dopravních procesů. Aplikace slouží ke sběru informací o reálném pohybu vlaků. Tím jsou v reálném čase poskytovány nezkreslené informace o uskutečněné a výhledové dopravní situaci provozním zaměstnancům železnice. Je určená k podpoře řízení dopravních procesů na vymezeném úseku železniční sítě, především na tratích s dálkově ovládaným zařízením (DOZ), ale může být využita i v izolovaných stanicích.

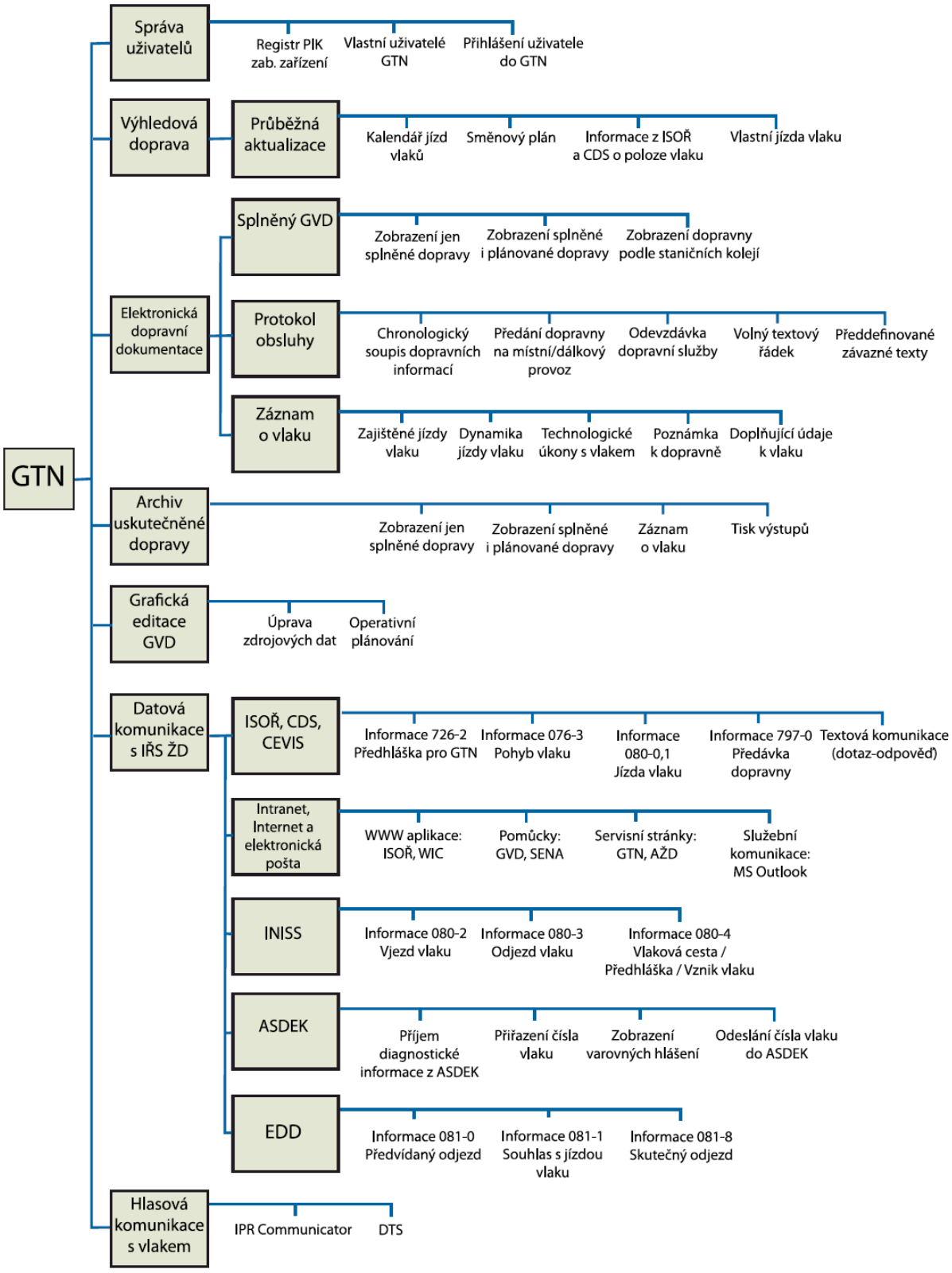
Zdrojem dat pro GTN jsou čísla vlaků, která jsou přenášena od elektronických zabezpečovacích zařízení s přenosem čísel vlaků (elektronická stavědla[1]). GTN dále zajišťuje vazbu mezi zabezpečovacím zařízením a informačními a řídicími systémy (ISOŘ, CEVIS, CDS, atd.). Vazby na tyto systémy je zobrazena na obr. 1.1. Propojení techniky tratě s těmito informačními a řídicími systémy tak umožňuje sběr informací o pohybu vlaku v reálném čase a bez účasti lidského činitele.



Obr. 1.1 Vazby GTN na ZZ a informační a řídicí systémy

1.1. Funkce GTN

[1] Elektronické stavědlo je zabezpečovací zařízení, jehož volící a logickou bezpečnostní část



Obr. 1.2 Blokové schéma integrace GTN [2]

Struktura GTN je zobrazena na obr. 1.2. Základní funkce GTN lze shrnout do následujících bodů:

- formou GVD (grafikon vlakové dopravy) je zobrazována a dokumentována realizace dopravy na traťovém úseku a v jednotlivých dopravnách – záznam o vlaku, splněný grafikon vlakové dopravy, protokol obsluhy, je zaznamenáván dopravní provoz po každé použité staniční a traťové koleji a dalších 23 provozně-technologických událostí vlaku,
- bezprostředně jsou využívány informace o aktuálním stavu vlakové dopravy pro tvorbu prognostického modelu - průběžnou aktualizací polohy trasy vlaku je umožněno okamžité vyhodnocení průběhu dopravního procesu a je upozorněno na konfliktní situace v aktuálním grafikonu,
- výhledově je možné měnit organizaci dopravy – plánování dopravy, zavedení/odřeknutí vlaku, grafická editace vlaku,
- ve spojení s elektronickými stavědly firmy AŽD Praha je možné automatické stavění vlakových cest podle výhledového grafikonu,
- na základě PIK (personální identifikační karta provozního zaměstnance v zabezpečovacím zařízení) je dokumentována odevzdávka dopravní služby a předávka stanice na místní/dálkový provoz (při DOZ),
- je možná komunikace s informačním systémem operativního řízení (ISOŘ) a centrálním dispečerským systémem (CDS), pomocí GTN je poskytována v reálném čase informace o vjezdu a odjezdu vlaku do/z stanice, je umožněno i vložení informace o narušení jízdního řádu. Od ISOŘ jsou získávány předhlášky na nákladní vlaky (délka, hmotnost, lokomotivy a jejich funkce na vlaku, trasa, aktuální poloha a další), od CDS jsou získávány předhlášky na osobní vlaky (aktuální poloha).
- je umožněn přístup a vkládání parametrických dotazů na intranetových portálech systémů provozního řízení, je obsažena elektronická pošta,
- je umožněna komunikace s hlasovým a informačním systémem pro cestující (INISS), kterému jsou poskytovány v reálném čase informace o postavených vlakových cestách a o vjezdu a odjezdu vlaku do/z stanice sloužící k automatickému hlášení ovládaného jízdou vlaku,
- je možná komunikace s diagnostickým systémem kolejových vozidel ASDEK, provozním zaměstnancům je zobrazováno hlášení o překročení limitních hodnot

teploty ložisek, obručí kol, disků kotoučových brzd a nepravidelností na obvodu kola (plochá kola),

- je možné zprostředkování fonického spojení se strojvedoucím vlaku přímo z prostředí GTN integrovanou DTS. [2]

1.2. Použití GTN

Z provozního hlediska je možné využití aplikace GTN:

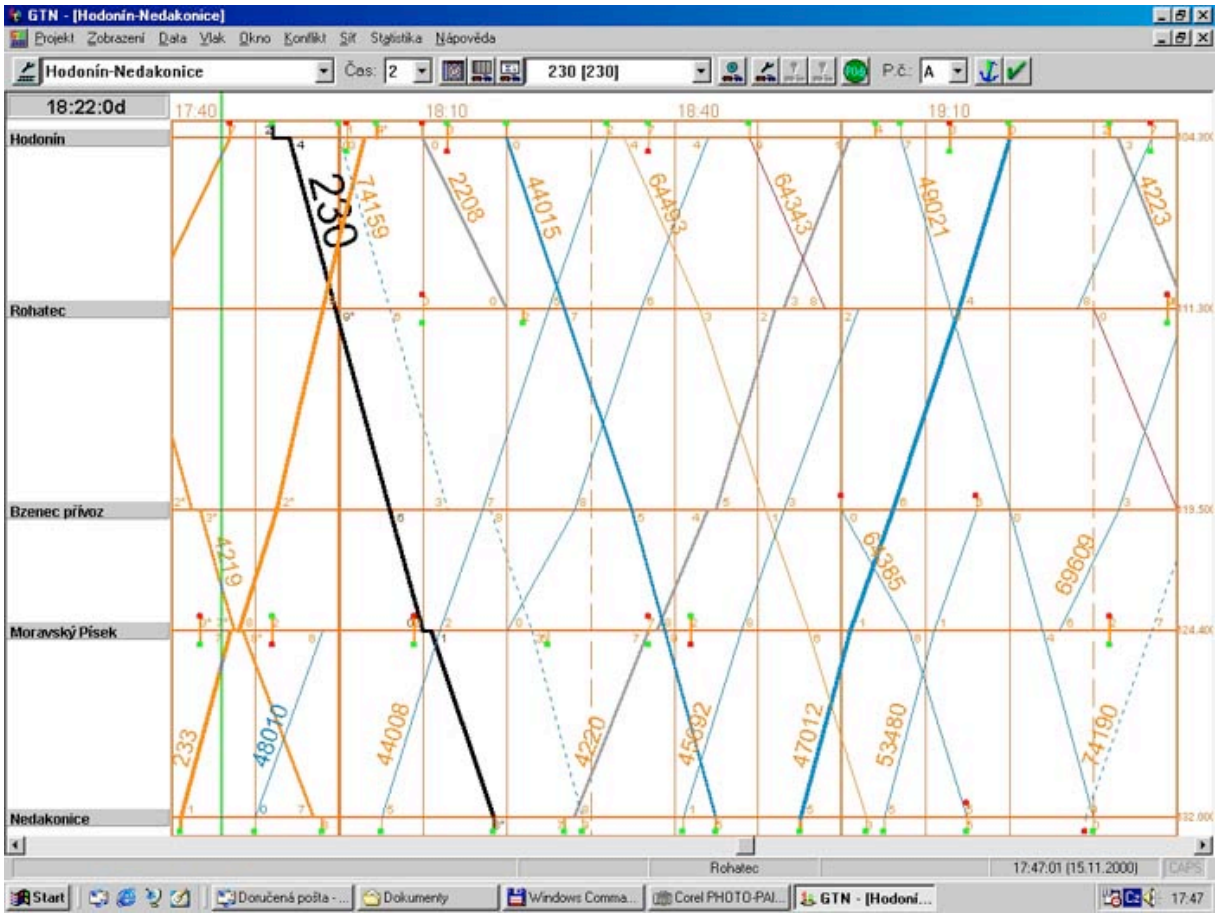
- na pracovišti dispečera dálkového ovládání, výpravčího nebo staničního dispečera pro podporu řízení liniových a místních dopravních procesů, s automatickým vedením dopravní dokumentace,
- na pracovišti výpravčího nebo staničního dispečera k manuálnímu vedení elektronické dopravní dokumentace ve stanici nevybavené zabezpečovacím zařízením s přenosem čísla vlaku,
- jako pracoviště pro administrátora GTN, kontrolní a vyšetřovací činnost, úpravu vstupních dat GVD, prohlížení archivních souborů GTN a analýzu dopravní situace,
- aplikace GTN může být instalována na běžném PC v kterékoliv kanceláři s připojením na Intranet provozovatele dráhy. [2]

1.3. Elektronická dopravní dokumentace

Pomocí aplikace GTN je vedena elektronická dopravní dokumentace (ELDODO). Pomocí ELDODO se zpracovávají a uchovávají informace o uskutečněné vlakové dopravě. Dokumentování automaticky pořízených dat ze zabezpečovacího zařízení nahrazuje stávající ručně vyplňované dokumentace. ELDODO dokumentuje splněný GVD, záznam o pohybu vlaku a protokol obsluhy. Díky automaticky sbíraným údajům ze zařízení sledující pohyb vlaku jsou v ELDODO prokazatelně přesné a nezpochybnitelné záznamy.

1.4. Prostředí aplikace GTN

V hlavním okně aplikace GTN je zobrazen list GVD řízené oblasti. List GVD je aktivní a zobrazuje aktuální stav na trati, splněný GVD (uskutečněná doprava) a výhledovou dopravu (organizace dopravy pro nadcházející časové období). GVD se aktualizuje na základě kalendáře jízd vlaků, komunikací s ISOŘ, manuálními zásahy přímo v GTN (plánování dopravy) a vlastní jízdou vlaku.[1]



Obr. 1.3 Hlavní okno GTN

1.5. Historie GTN

Vývoj graficko-technologické nadstavby je datován od roku 1999, kdy byly uskutečněny první studie o možnostech vedení dopravní dokumentace na počítači. Původně bylo GTN určeno k dopravní dokumentaci pro malé úsekově řízené oblasti s DOZ. Proto bylo potřeba vyvinout 26 vysoce sofistikovaných dopravně-technologických funkcí pro přenos čísel vlaků. Z důvodu mnohotvárnosti železničního provozu, staničních kolejišť, vleček,

nákladišť a dálkově ovládaných tratí to znamenalo navržení téměř 100 různých typů dopravních procesů, které musely být v GTN splněny.

První zkušební provoz přenosu čísel vlaků do GTN byl zahájen na železniční stanici Lysá nad Labem na podzim roku 2000.

Postupem času však bylo na základě nových uživatelských požadavků rozšířeno o další funkce, především o výhledovou dopravu a vazby na informační systém operativního řízení ISOŘ.

Nové uživatelské požadavky a rozvoj centrálních dispečerských stanovišť s DOZ vedly k tomu, že v roce 2006 došlo k nutnému generačnímu zlomu a začalo se s vývojem již 4. generace GTN, kdy došlo k úplné změně architektury.

Tato zatím poslední generace GTN již umožňuje realizovat telematické vazby a přidává funkce pro podporu řízení dopravy v souladu s předpisy TSI TAF a TSI TAP.

Poslední verze GTNv4.1 získala v červnu 2010 souhlas od SŽDC (Správa železniční dopravní cesty) k provozování na železniční dopravní cestě ve vlastnictví státu. Po schválení dochází k postupnému nasazení této verze v oblastech, kde doposud byla využívána verze GTNv4.0. Na tratích, kde se využívá ještě GTNv3.3, se nasazení nové verze odkládá do doby výměny jejich HW.

V současné době je GTN provozována v 55 řízených oblastech s celkem 180 dopravními body v Čechách i na Slovensku. [3]

2. Návrhové metody

Aplikace GTN využívá pro uchování dat relační databáze MySQL. Tato databáze byla dodána společností AŽD Praha jako vstupní datová struktura pro účely vývoje editoru jízdních řádů. V následující kapitole bude probrána práce s tímto typem databáze, a dále pak možnosti propojení s různými programovacími jazyky.

2.1. MySQL databáze

Aplikace GTN využívá MySQL databázi pro uchovávání dat. Jsou v ní obsažena veškerá data, popisující reliéf trati, stanice, traťové a staniční koleje, grafikon atd. Navrhovaný program pro editaci jízdních řádů tedy vyžaduje spojení s touto databází, při tvorbě programu bylo nutné se seznámit s dodanou databází a vytvořit potřebné příkazy, které se využijí pro implementaci v programu, proto následuje popis základních příkazů v databázi MySQL.

MySQL je relační databázový systém založený na databázovém jazyce SQL[2] (Structured Query Language). Byl vyvinut švédskou společností MySQL AB, v současné době vyvíjený a distribuovaný společností Oracle Corporation. Systém je dostupný v open-source verzi, tzn., že jsou k dispozici veškeré zdrojové kódy. Distribuovaný je pod licencí GPL (GNU General Public License). Pro komerční účely je možné zakoupit licenci.

Relační databázový systém ukládá data do separátních tabulek namísto, aby všechna data byla v jedné rozsáhlé tabulce. Mezi tabulkami se pak vytvářejí tzv. relace, které určují vztahy mezi jednotlivými tabulkami. Díky tomuto řešení je dosahováno větší rychlosti a flexibility v přístupu k databázi.

2.1.1. Relace mezi tabulkami

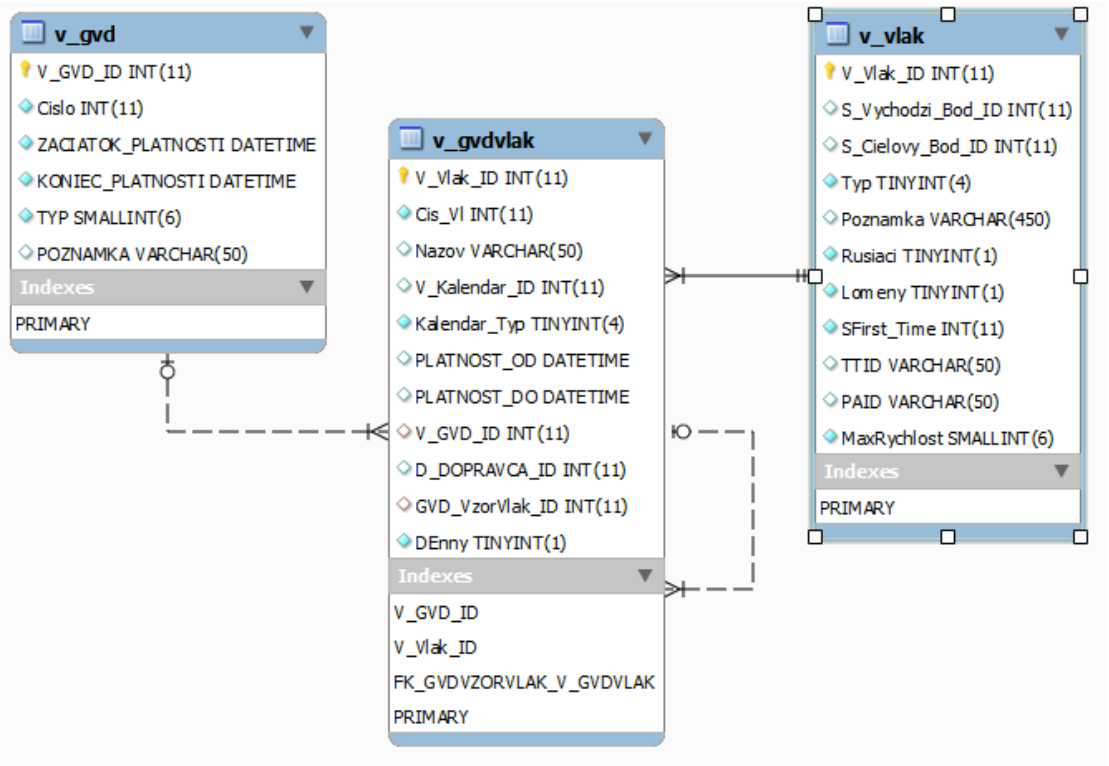
Relace se vytváří na základě primárního klíče v tabulce, který se předá do druhé tabulky, kde je poté označený jako cizí klíč a slouží jako odkaz na první tabulku. Lze je rozdělit na dva základní typy – identifikovatelné a neidentifikovatelné relace[3].

[2] SQL je jeden z nejrozšířenějších databázových jazyků a je definován normou ANSI ISO/IEC 9075-1.
[3] Z anglického překladu identifying and non-identifying relationship.

Identifikovatelná vazba je taková, kdy záznam v sekundární tabulce nemůže být jednoznačně určen bez záznamu z tabulky primární. Ve schématu se taková vazba značí plnou, nepřerušovanou čarou.

U neidentifikovatelné vazby je možné záznamy v obou tabulkách jednoznačně určit bez znalosti dat z navázané tabulky. Tato relace se ve schématech značí přerušovanou čarou.

Relace také mohou být vytvářeny v různém poměru. Poměry relace mohou být 1:1, tj. primární klíč jedné tabulky se může vyskytovat pouze jednou v tabulce druhé, dále pak relace 1:n, tj. primární klíč z jedné tabulky se může vyskytovat vícekrát v tabulce druhé, a nakonec relace v poměru n:m, tj. primární klíč z první tabulky se může vyskytovat vícekrát v druhé tabulce, ale zároveň primární klíč z druhé tabulky se může vyskytovat vícekrát v tabulce první. Relace n:m se příliš nevyužívá a spíše se nahrazuje několika vazbami typu 1:n.



Obr. 2.1 Relace mezi tabulkami databáze

Relace mezi tabulkami jsou naznačeny na obr. 2.1. Zde můžeme vidět, že v tabulkách **v_gvd** a **v_vlak** jsou umístěny primární klíče, na které je odkazováno z tabulky **v_gvdvlak**. Dle typu čar je také vidět, že mezi tabulkami **v_gvd** a **v_gvdvlak** je neidentifikovatelná vazba, zatímco mezi tabulkami **v_vlak** a **v_gvdvlak** je vazba identifikovatelná, tudíž tabulka **v_gvdvlak** nelze jednoznačně identifikovat, aniž by tabulka **v_vlak** existovala – to již také

naznačuje primární klíč *v_vlak_ID* (první pole v obou tabulkách), který je pro obě tabulky totožný, ale vytváří se v tabulce **v_vlak**.

2.1.2. Databázové dotazy

MySQL funguje v tzv. režimu klient – server. Na serveru běží instance s databází, příp. na serveru může běžet i více instancí. Klient zasílá požadavky ve formě SQL dotazů, tzv. *query*, a server na tyto dotazy odpovídá zasláním příslušných dat z databáze. Příkazy klient posílá v textové formě v předem dohodnutém formátu, z toho plyne, že základní verze klienta je pouze terminálová aplikace, ale je možné využít i grafickou nadstavbu (MySQL Workbench), která umožňuje rozšířený pohled na databáze. Spojení mezi klientem a serverem navážeme voláním příkazu:

```
$ mysql [-h host] -u user -p

Enter password:*****
mysql>
```

Volání tohoto příkazu je podmíněno přítomností cesty v proměnné $PATH[4] k souboru *mysql*. Parametr *–h* uvozuje cestu k serveru (např. localhost nebo IP adresa) a je nepovinný, pokud se uživatel přihlašuje přímo ze stroje, na kterém běží instance serveru. Voláním parametru *–u* předáváme uživatelské jméno, pod kterým se chceme připojit na server a parametr *–p* znamená, že pro přihlášení má být vyžadováno zadání hesla, po spuštění příkazu je tedy vyžádáno heslo a po jeho zadání dojde k přihlášení na server. Pokud server umožňuje anonymní přihlášení, pak je dokonce možné vypustit z příkazu i parametry *–u* a *–p*.

Po přihlášení na server je možné začít pracovat s databází. Pro databázové dotazy je použita tzv. koncepce CRUD (Create – Retrieve – Update(Modify) - Delete[5]), která charakterizuje základní sadu operací, které lze nad databází provádět. Z pohledu uživatele je možné tyto příkazy volat z příkazové řádky, nebo lze využít také grafického klienta, který usnadňuje práci s databází, ale pro potřeby této práce není potřeba a nebude zde popisován. Omezíme se pouze na přístup z příkazové řádky. Nyní se tedy podíváme blíže na koncepci CRUD a popíšeme, jak takové dotazy vypadají a z čeho se skládají.

[4] V systémech typu UNIX/Linux je toto možné nastavit v souboru .profile v domovském adresáři a v systému Windows lze toto nalézt v konfiguraci systému pod položkou proměnné prostředí.
[5] V překladu tyto slova znamenají vytvořit – získat – aktualizovat(upravit) – smazat.

CRUD - **C**reate

Začneme tedy od příkazů pro vytváření databáze, tabulky a záznamu v tabulce. Pro vytvoření nové databáze a nové tabulky je užit příkaz CREATE, pro vytvoření záznamu v tabulce pak příkaz INSERT. Následující příklad ukazuje posloupnost příkazů, kterými je vytvořena nová databáze s názvem *database*, databáze je zvolena jako výchozí pro práci, vytvořena tabulka s názvem *table*, která obsahuje dvě proměnné typu int, z čehož proměnná s názvem *prom_id* je nastavena jako primární klíč s automatickým navyšováním hodnoty, a na závěr je vložen jeden záznam do této tabulky:

```
mysql> CREATE DATABASE database;           # vytvoření nové db
mysql> USE database;                       # zvolení db
mysql> CREATE TABLE table (prom_id INT AUTO_INCREMENT NOT NULL PRIMARY
       KEY, cislo INT NOT NULL);
mysql> INSERT INTO table VALUES (NULL, 201204)
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6),(7,8,9);
```

Při dalším řešení však bude potřeba pouze příkazu INSERT pro vkládání záznamů do již existujících tabulek. Zde je proto uvedena plná syntaxe tohoto příkazu:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
{VALUES | VALUE} ({expr | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE
col_name=expr
[, col_name=expr] ... ]

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name
SET col_name={expr | DEFAULT}, ...
[ ON DUPLICATE KEY UPDATE
col_name=expr
SQL Statement Syntax
[, col_name=expr] ... ]

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] tbl_name [(col_name,...)]
SELECT ...
[ ON DUPLICATE KEY UPDATE
col_name=expr
[, col_name=expr] ... ]
```

Příkaz INSERT existuje v několika modifikacích. První dvě možnosti syntaxe – INSERT ... VALUES a INSERT ... SET – slouží pro zadávání přímo specifikovaných hodnot. Poslední syntaxe – INSERT ... SELECT pro vložení dat zvolených z jiných tabulek. [4]

C**R**UD - **R**etrieve

Příkaz pro čtení z databáze, v koncepci označovaný slovem retrieve, je v MySQL charakterizován příkazy SHOW TABLES, DESCRIBE *table* a SELECT * FROM *table*,

případně příkaz SELECT lze doplnit o další upřesňující příkazy, které můžete vidět v plné syntaxi příkazu:

```
SELECT
[ALL | DISTINCT | DISTINCTROW ]
[HIGH_PRIORITY]
[STRAIGHT_JOIN]
[SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
[SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
select_expr [, select_expr ...]
[FROM table_references
[WHERE where_condition]
[GROUP BY {col_name | expr | position}
[ASC | DESC], ... [WITH ROLLUP]]
[HAVING where_condition]
[ORDER BY {col_name | expr | position}
[ASC | DESC], ...]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
[PROCEDURE procedure_name(argument_list)]
[INTO OUTFILE 'file_name'
[CHARACTER SET charset_name]
export_options
| INTO DUMPFILE 'file_name'
| INTO var_name [, var_name]]
      [FOR UPDATE | LOCK IN SHARE MODE]]
```

Parametry uvedené v hranatých závorkách jsou nepovinné, a tak základní struktura příkazu se zredukuje na pouhý SELECT *select_expr* FROM *table_references*.

V následujících příkladech bude popsána funkce několika základních parametrů příkazu, více se pak můžete dočíst v [4]. V příkladech již budou zobrazována i data ze zadané databáze.

Nejprve si ještě ukážeme příkaz SHOW TABLES, ten zobrazí všechny tabulky obsažené v databázi:

```
mysql> SHOW TABLES; #zobrazeni obsahu db
+---------------------+
| Tables_in_gtn_modul |
+---------------------+
| c_druh_vlaku        |
# výpis omezen
| v_bod_trasa_prechod |
| v_depo              |
| v_gvd               |
| v_gvdvlak           |
| v_kalendar          |
| v_vlak              |
+---------------------+
25 rows in set (0.00 sec)
```

Pro zobrazení podrobností o tabulce slouží příkaz DESCRIBE. Jsou zde vidět typy proměnných, jakou mají výchozí hodnotu, jakého typu je proměnná nebo zda se jedná o primární, či cizí klíč, atd.:

```
      mysql> DESCRIBE v_gvd;
+--------------------+-------------+------+-----+---------+--------+
| Field              | Type        | Null | Key | Default | Extra  |
+--------------------+-------------+------+-----+---------+--------+
| V_GVD_ID           | int(11)     | NO   | PRI | NULL    | autoInc|
| Cislo              | int(11)     | NO   |     | NULL    |        |
| ZACIATOK_PLATNOSTI | datetime    | NO   |     | NULL    |        |
| KONIEC_PLATNOSTI   | datetime    | NO   |     | NULL    |        |
| TYP                | smallint(6) | NO   |     | 0       |        |
| POZNAMKA           | varchar(50) | YES  |     | NULL    |        |
+--------------------+-------------+------+-----+---------+--------+
6 rows in set (0.02 sec)
```

Příkaz SELECT již slouží k výběru samotných záznamů z tabulky. Základní formule příkazu je SELECT * FROM *table*, kde * slouží jako zástupný znak označující všechny proměnné v tabulce, tedy uvedený zápis příkazu znamená vyber všechno z tabulky table. Z důvodu velkého rozsahu dat v tabulkách zde nebude uveden výpis tohoto příkazu. Samozřejmostí je, že zástupný znak lze nahradit proměnnými, které nás zajímají. Jako např.:

```
mysql> SELECT v_gvd_id, Cislo, zaciatok_platnosti, koniec_platnosti FROM
v_gvd;
+----------+--------+---------------------+---------------------+
| v_gvd_id | Cislo  | zaciatok_platnosti  | koniec_platnosti    |
+----------+--------+---------------------+---------------------+
|        2 |      0 | 2010-12-12 00:00:00 | 2011-06-11 23:59:00 |
|        3 | 201104 | 2011-06-12 00:00:00 | 2011-09-04 23:59:59 |
|        4 | 201105 | 2011-09-05 00:00:00 | 2011-12-10 23:59:59 |
|        5 | 201200 | 2011-12-11 00:00:00 | 2012-02-12 23:59:59 |
|        6 | 201201 | 2012-02-13 00:00:00 | 2012-12-08 23:59:59 |
|        7 | 201220 | 2012-03-24 21:18:23 | 2012-03-24 21:18:23 |
|        8 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
|        9 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
|       10 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
|       11 | 205001 | 2050-01-01 00:00:00 | 2050-12-31 23:59:59 |
+----------+--------+---------------------+---------------------+
10 rows in set (0.00 sec)
```

Jak je vidět v uvedené tabulce, tak v proměnné *Cislo* se několikrát opakují záznamy s číslem 201221. Pokud bychom chtěli zobrazit pouze hodnoty jedenkrát, pak lze využít příkaz SELECT DISTINCT *prom* FROM *table*, který vybere z tabulky všechny záznamy z tabulky podle proměnné *prom*, ale pokud se vyskytují záznamy se stejnou hodnotou v proměnné *prom*, tak ve výpisu tuto položku zobrazí pouze jednou):

```
mysql> SELECT DISTINCT Cislo FROM v_gvd;
+--------+
| Cislo  |
+--------+
| 201201 |
| 201220 |
| 201221 |
| 205001 |
+--------+
8 rows in set (0.00 sec)
```

To je vhodné zejména v případech, kdy nás především zajímá, které hodnoty se v tomto poli vyskytují, ale nepotřebujeme vědět, že se stejná položka vyskytuje mnohonásobně.

Nevýhoda SELECT DISTINCT je však ve výpisu pouze jediného sloupce. Pokud je však potřeba ve výpisu vidět všechny sloupce tabulky, nebo alespoň více sloupců, než jen jediný, podle kterého je potřeba omezit výběr, pak lze použít příkazu SELECT * FROM *table* GROUP BY *prom*. Tímto příkazem docílím stejného výsledku jako při příkazu SELECT DISTINCT, s tím rozdílem, že ve výpisu budou zobrazeny všechny sloupce tabulky (příp. sloupce, které vypíšeme namísto *).

Dále lze využít upřesňujícího příkazu ORDER BY *prom* ASC/DESC, který umožňuje seřazení záznamů podle zvolené proměnné vzestupně či sestupně. Zde je ukázka příkazu ORDER DESC:

```
mysql> SELECT v_gvd_id,Cislo,zaciatok_platnosti,koniec_platnosti FROM
v_gvd ORDER BY Cislo DESC;
   +----------+--------+---------------------+---------------------+
   | v_gvd_id | Cislo  | zaciatok_platnosti  | koniec_platnosti    |
   +----------+--------+---------------------+---------------------+
   |       11 | 205001 | 2050-01-01 00:00:00 | 2050-12-31 23:59:59 |
   |       10 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
   |        9 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
   |        8 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
   |        7 | 201220 | 2012-03-24 21:18:23 | 2012-03-24 21:18:23 |
   |        6 | 201201 | 2012-02-13 00:00:00 | 2012-12-08 23:59:59 |
   |        5 | 201200 | 2011-12-11 00:00:00 | 2012-02-12 23:59:59 |
   |        4 | 201105 | 2011-09-05 00:00:00 | 2011-12-10 23:59:59 |
   |        3 | 201104 | 2011-06-12 00:00:00 | 2011-09-04 23:59:59 |
   |        2 |      0 | 2010-12-12 00:00:00 | 2011-06-11 23:59:00 |
   +----------+--------+---------------------+---------------------+
                     10 rows in set (0.00 sec)
```

Pro vyhledávání podle dané proměnné slouží příkaz WHERE *prom*=?, kde za *prom* se doplní daná proměnná a za otazník požadovaná hodnota. Samozřejmě je možné za WHERE umístit i více podmínek. Podmínky se pak oddělují logickými operátory AND a OR a složené podmínky se rozdělují pomocí závorek. Nejprve příklad jednoduché podmínky:

```
mysql> SELECT v_gvd_id, Cislo, Zaciatok_platnosti, koniec_platnosti FROM
v_gvd WHERE Cislo=201221;
+----------+--------+---------------------+---------------------+
| v_gvd_id | Cislo  | Zaciatok_platnosti  | koniec_platnosti    |
+----------+--------+---------------------+---------------------+
|        8 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
|        9 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
|       10 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
+----------+--------+---------------------+---------------------+
3 rows in set (0.00 sec)
```

A dále příklad složené podmínky:

```
mysql>  SELECT v_gvd_id, Cislo, Zaciatok_platnosti, koniec_platnosti FROM
v_gvd WHERE (Cislo=201221 AND v_gvd_id=8) OR Cislo=0;
+----------+--------+---------------------+---------------------+
| v_gvd_id | Cislo  | Zaciatok_platnosti  | koniec_platnosti    |
+----------+--------+---------------------+---------------------+
|        2 |      0 | 2010-12-12 00:00:00 | 2011-06-11 23:59:00 |
|        8 | 201221 | 2012-02-12 10:00:00 | 2012-03-12 10:00:00 |
+----------+--------+---------------------+---------------------+
2 rows in set (0.00 sec)
```

Nutno dodat, že v podmínce ve výše uvedeném příkladu je podmínka *Cislo*=201221 redundantní, protože *v_gvd_id* je primární klíč s funkcí AUTO_INCREMENT (to je zřejmé z výpisu příkazu DESCRIBE **v_gvd**), a proto nemůže existovat jiný záznam s tímto číslem, podmínka by tedy mohla být pouze *v_gvd_id*=8 OR *Cislo*=0 a výstupem by byla stejná tabulka, ale pro názornost použití složených podmínek je takto uvedeno.

Na závěr lze ještě uvést složitější příkaz SELECT nad rozsáhlejší tabulkou, ve kterém je využito dříve zmíněného příkazu WHERE se složenou podmínkou a ORDER BY. Na tomto příkladu budou také ukázány možnosti pro spojování tabulek pomocí příkazu LEFT/RIGHT JOIN ON.

```
mysql> SELECT gvdvlak.v_gvd_id GVD, gvdvlak.cis_vl VlakCis,
    -> bod1.nazov VychoziBod, bod2.nazov CilovyBod
    -> FROM v_gvdvlak gvdvlak
    -> LEFT JOIN v_vlak vlak ON gvdvlak.v_vlak_id=vlak.v_vlak_id
    -> LEFT JOIN s_bod bod1 ON vlak.s_vychodzi_bod_id=bod1.s_bod_id
    -> RIGHT JOIN s_bod bod2 ON vlak.s_cielovy_bod_id=bod2.s_bod_id
    -> WHERE ((bod1.s_bod_id=8652 AND bod2.s_bod_id=8260)
    -> AND (cis_vl=751 OR cis_vl=7313 OR cis_vl=67701 )
    -> AND (gvdvlak.v_gvd_id=4))
    -> ORDER BY VlakCis ASC;
+------+---------+------------+-------------------+
| GVD  | VlakCis | VychoziBod | CilovyBod         |
+------+---------+------------+-------------------+
|    4 |    7313 | Cheb       | Plzeň hl.n.os.n.  |
|    4 |   67701 | Cheb       | Plzeň hl.n.os.n.  |
+------+---------+------------+-------------------+
2 rows in set (0.00 sec)
```

Z výše uvedeného příkladu můžeme vidět mnoho možností, jak s výběrem tabulek zacházet. Ve zkratce tento SELECT vybírá data z tabulek **v_gvdvlak**, **v_vlak** a **s_bod**, zde si můžeme například všimnout toho, že pro volená data byly nastaveny aliasy, takže hlavičky sloupců zobrazují námi zvolené názvy namísto skutečných názvů proměnných, dále je zde použit levý i pravý JOIN, jejich rozdíl však není úplně patrný, protože jsme omezili výběr na málo proměnných, aby bylo možné zobrazit výstupní data.

To byl tedy jen krátký výčet možností příkazu SELECT, který při řešení bohatě postačí.

CR**U**D - **U**pdate

Další z koncepce CRUD je příkaz UPDATE. Ten je využíván pro aktualizování stávajících záznamů. Opět se nejprve podívejme na úplnou syntaxi příkazu UPDATE, popíšeme si jednotlivé části, a pak si uvedeme několik příkladů, jak tento příkaz funguje. Syntaxe příkazu je tedy následující:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
[WHERE where_condition]
[ORDER BY ...]
[LIMIT row_count]
```

První slovo příkazu vždy uvozuje typ operace, proto zde začíná slovem UPDATE. V hranatých závorkách jsou uvedeny volitelné parametry. Za slovem UPDATE tedy může bezprostředně následovat odkaz do tabulky (*table_reference*). Následuje příkaz SET, který uvozuje ty položky, které se v záznamu mají modifikovat. Při aktualizaci více položek se oddělují čárkou. Základní příkaz se tedy opět zredukuje na pouhý UPDATE *table* SET *col=expr*. Z volitelných parametrů je zde LOW_PRIORITY, který umožňuje zápis se zpožděním, tj. pokud aktuálně jiný uživatel přistupuje k tabulce, pak v případě použití tohoto parametru se vyčká, až uživatel ukončí práci s tabulkou. Dalším volitelným parametrem je IGNORE. Ten způsobí to, že pokud při zápisu dojde k chybě, tak je příkaz i nadále vykonáván. Položky u kterých dojde k chybě, jsou tak přeskočeny. Zbytek parametrů tj. WHERE, ORDER BY a LIMIT se již funkčně shodují s parametry použitými v příkazu SELECT. Nyní se tedy podívejme, jak příkaz funguje v praxi:

```
mysql> SELECT v_vlak_id ID, s_vychodzi_bod_id Vychozi, s_cielovy_bod_id
      FROM v_vlak LIMIT 1;
+-------+---------+------------------+
| ID    | Vychozi | s_cielovy_bod_id |
+-------+---------+------------------+
| 15261 |    7944 |             8652 |
+-------+---------+------------------+
1 row in set (0.00 sec)
```

Zde máme výpis z tabulky, nyní budeme chtít změnit např. hodnotu Vychozi, použijeme tedy příkaz:

```
mysql> UPDATE v_vlak SET s_vychodzi_bod_id=7945 WHERE v_vlak_id=15261;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Po vykonání příkazu je vypsáno, co vše bylo změněno a zda došlo k nějakým chybám. Kontrolu bychom provedli zavoláním stejného příkazu jako před provedením aktualizace, tentokrát však bude hodnota *Vychozi* změněna.

CRU**D** - **D**elete

Na závěr této podkapitoly se podíváme na poslední článek koncepce CRUD. Tím je příkaz DELETE. Syntaxe příkazu DELETE je následující:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
[WHERE where_condition]
[ORDER BY ...]
[LIMIT row_count]
```

Tento příkaz se ve svém zápise shoduje s příkazem SELECT, v tomto případě tedy základní příkaz pro smazání jednoho řádku bude vypadat následovně:

```
mysql> DELETE FROM v_vlak WHERE v_vlak_id=15261;
```

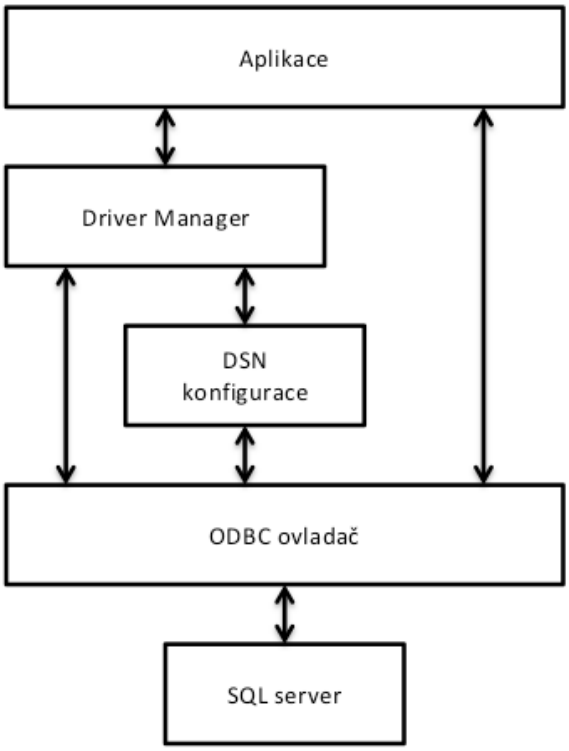
Příkazy uvedené v této kapitole jsou jen krátkým výčtem z mnoha příkazů, které databáze MySQL umožňuje, ale pro potřeby této práce to postačuje. Jsou uvedené pouze příkazy, které se budou hodit při implementaci v programu. Více o příkazech a funkcích používaných v MySQL databázi je možné vyčíst z [4].

2.2. Programové spojení s databází

Společnost Oracle nabízí kromě MySQL serveru a klienta také tzv. konektory pro spojení s databází z různých programovacích prostředí. Konektor je aplikační programovací rozhraní (API), které nabízí nízko-úrovňový přístup k MySQL databázi. Na stránkách jsou k dispozici ke stažení zdrojové kódy v různých programovacích jazycích pro použití ve vyvíjené aplikaci. Konektory od společnosti Oracle jsou k dispozici např. pro vývojová prostředí .NET Framework (Connector/NET), Java (Connector/J), C++ (Connector/C++) a další. Dále pak jsou k dispozici komunitní verze konektoru např. pro PHP, Perl, Python atd. Všechny verze, včetně komunitních, jsou podporovány ze strany Oracle a na stránkách MySQL je k dispozici dokumentace. Nyní se tedy podívejme na některé konektory, které mohou mít uplatnění při návrhu modulu.

2.2.1. MySQL Connector/ODBC

ODBC (Open Database Connectivity) je průmyslový standard, jehož cílem je zachování nezávislosti databázového systému a operačního prostředí. ODBC konektor je navržen v programovacím jazyce C a poskytuje nativní rozhraní s SQL databází. Architektura ODBC je založena na pěti komponentách, jak je zřejmé ze schématu na obr. 2.2:



Obr. 2.2 Architektura MySQL Connector/ODBC

Aplikace používá ODBC konektor pro přístup k datům na serveru. ODBC zase používá spojení s Driver Managerem. Aplikace komunikuje s Driver Managerem s pomocí standardních ODBC příkazů. Aplikace potřebuje znát pouze DSN (Data Source Name) konfiguraci, nepotřebuje však znát, kde nebo jak jsou data uložena. Běžně aplikace vykonává několik úkolů, bez ohledu jak používá ODBC:

- volí SQL server a připojuje se k němu,
- vytváří SQL požadavky k vykonání,
- předává nebo vrací zpět transakci uzavřenou v SQL příkazu,
- získává výsledky (pokud jsou nějaké),
- zpracovává chyby,
- ukončuje spojení se serverem.

Hlavní úkolem aplikace, která používá ODBC, je předávání SQL příkazů a získávání jakýchkoliv výsledků získaných z těchto příkazů.

Driver Manager (Správce spojení) je knihovna implementovaná v operačním systému, která spravuje komunikaci mezi aplikací a ovladačem či ovladači. Provádí následující úkoly:

- zajišťuje zpracování DSN konfigurace, která identifikuje použitý ovladač, adresu serveru, jméno databáze a přihlašovací údaje k databázi. Protože je konfigurace uchována v DSN, jakákoliv aplikace využívající ODBC může tuto konfiguraci využít.
- Inicializuje a ukončuje spojení s ovladačem použité databáze. Správný ovladač je zvolen na základě informace z DSN. Například, pokud je využita databáze MySQL, bude načten Connector/ODBC, a pokud bude využit např. Microsoft SQL server, pak bude načten ovladač specifický pro tuto databázi.
- Zpracovává funkční volání nebo je předává ovladači ke zpracování.

DSN konfigurace je soubor, ve kterém jsou uchovány informace potřebné pro připojení. Je použita správcem spojení k inicializaci ovladače a spojení s databází.

ODBC driver (ovladač) je knihovna, která implementuje funkce podporované v API. Zpracovává funkční volání, předává SQL dotazy serveru a vrací výsledky dotazů zpět aplikaci. Pokud je potřeba, tak ovladač upravuje dotazy zaslané z aplikace do podoby vhodné pro použitý SQL server.

SQL server je zodpovědný za uchování dat v databázi. Slouží jako zdroj dat při volání dotazů pro výběr dat, a jako cílová destinace při volání funkcí pro ukládání dat.

Driver Manager je součástí většiny dostupných operačních systémů (Windows, Linux/Unix, Mac OS). Pro spojení s databází je nutná pouze instalace ovladače pro požadovaný typ databáze. Instalace ODBC ovladače pro MySQL je popsána v [4].

ODBC je využíván, pokud je vyžadována nezávislost na databázovém systému nebo simultánní přístup k rozdílným datovým zdrojům. Vhodný je při použití s nástroji pro administraci relačních databází, jako jsou Microsoft Access nebo Crystal Reports. Ale také lze využít v programovacích jazycích podporovaných společností Microsoft, jako např. Visual Basic, C# a Perl.

Výhody ODBC

- Je navržen pro vysokou interoperabilitu mezi různými databázovými systémy (DBMS),

- při změně DBMS není nutné měnit strukturu aplikace,
- snadné použití.

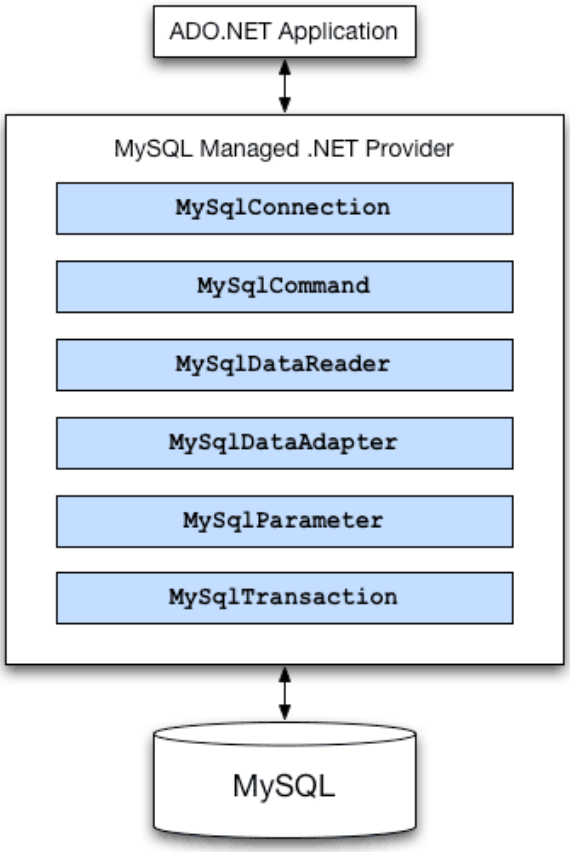
Nevýhody ODBC

- Výkonnost tohoto spojení není příliš vysoká, protože obsahuje příliš vrstev.

2.2.2. MySQL Connector/NET

Ovladač Connector/NET umožňuje vývoj aplikací ve vývojovém prostředí .NET Framework. Connector/NET je implementován v rozhraní ADO.NET (ActiveX Data Objects for .NET) a instalací se integruje do vývojového prostředí Visual Studio. Ovladač je kompletně napsán v programovacím jazyce C#.

Struktura ADO.NET rozhraní je znázorněna na následujícím obr. 2.3:



Obr. 2.3 Architektura MySQL Connector/NET

ADO.NET Application je aplikace využívající ADO.NET rozhraní.

MySQL Managed .NET Provider je samotné ADO.NET rozhraní, které zajišťuje všechny funkce jako v ODBC Driver Manager a ODBC driver, včetně správy spojení (DSN konfigurace).

Instalace a konfigurace ADO.NET je popsána v [4], včetně potřebných zdrojových kódů pro vytvoření aplikace.

Výhody ADO.NET (Connector/NET)

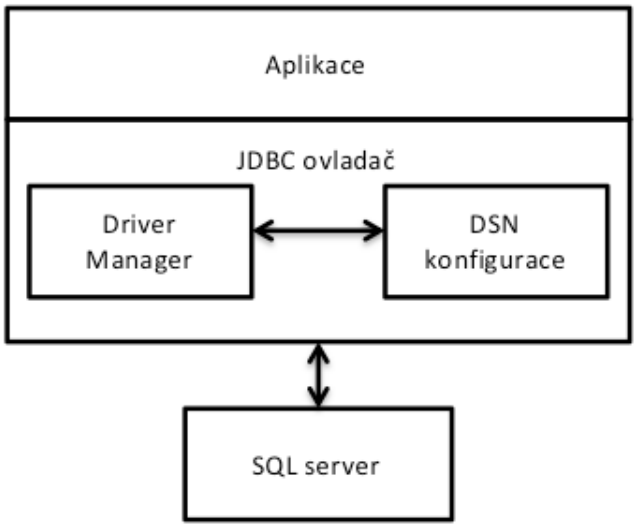
- Vysoká rychlost
- Nabízí zabezpečení a tzv. garbage collection
- Dobrá správa databázových příkazů

Nevýhody ADO.NET (Connector/NET)

- Přístup k DBMS není standardizován

2.2.3. MySQL Connector/J

Connector/J, jinak také zvaný JDBC driver, poskytuje konektivitu pro aplikace vyvinuté v programovacím jazyce Java. JDBC driver typ 4 je plně implementován v jazyce Java a je nezávislý na klientských knihovnách MySQL.



Obr. 2.4 Architektura MySQL Connector/J

Architektura JDBC(Java Database Connectivity) driveru sestává ze stejných komponent jako ODBC architektura. V případě JDBC ovladače je však Driver Manager součástí ovladače a ovladač je přímo implementován v aplikaci.

Výhody JDBC (Connector/J)

- K dispozici zdrojové kódy
- Nezávislost na operačním systému

JDBC a programovací jazyk Java byly zvoleny při návrhu samotné implementace, proto zde budou popsány metody implementace.

Pro spojení Java aplikace s databází MySQL lze využít tří možností. První možností je přímý přístup k databázi s pomocí JDBC ovladače, druhou možností je implementování rozhraní a objektů DAO (Data Access Object), které umožňuje mapování tabulek databáze do objektů DTO (Data Transfer Object) a vytvoření pomocného rozhraní JDBC driveru pro přístup k databázi, a poslední možností je doplnění DAO objektů o rozhraní typu Persistence, tzv. ORM (Object Relational Mapping). Samotné JDBC umožňuje přímé volání databázových příkazů, zatímco při použití DAO a ORM již vzniká jistá míra abstrakce, usnadňují tak práci s rozsáhlejší databází a díky tomu je snazší upravit aplikaci v případech, kdy se změní struktura databáze. Každá z těchto možností má svá pro a proti, jejich možnosti budou probrány v následujících podkapitolách a na jejich základě bude zvolena metoda pro navrhovanou aplikaci.

Přímý přístup s pomocí JDBC

JDBC je Java™ API (Application Programming Interface), distribuované pod licencí GPLv2, které umožňuje provádění SQL příkazů. Je tedy důležitou součástí při spojení Java aplikace s SQL databází. Jedná se o sadu objektů a rozhraní napsané v programovacím jazyce Java, která poskytuje nástroj pro vývoj databázově orientovaných programů. Umožňuje správu spojení s databází, vytváření databázových dotazů a jednoduchou práci s daty v databázi.

Metoda přímého přístupu je z dostupných metod nejjednodušší na implementaci, je však zapotřebí znát databázové příkazy. Metoda používá přímou implementaci databázových dotazů v kódu aplikace, dotazy se vykonávají pomocí metody executeQuery(String sql) objektu java.sql.Statement a pro práci s daty se využívají objekty typu java.sql.ResultSet. V následujícím výpisu kódu je jednoduchý příklad, jak vytvořit spojení s databází, zavolat SQL dotaz a zpracovat výsledek:

```
Connection con = DriverManager.getConnection (
      "jdbc:mysql://IP/scheme", "user", "String password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (rs.next()) {
      int x = getInt("a");
      String s = getString("b");
      float f = getFloat("c");
      }
```

Tato metoda se hodí v případech, kdy databáze v MySQL je malého rozsahu. Pokud je však databázový model rozsáhlý, tak už není příliš výhodná, protože objekty typu ResultSet neumožňují pokročilejší práci s daty a je doporučeno využít DAO nebo ORM. Tento přístup je obecně méně vhodný, protože implementace je náročná při práci s rozsáhlejší databází a při změně schématu databáze není možné zaručit její funkčnost.

Mapování dat do DAO objektů

Metoda s použitím návrhového vzoru DAO objektů již nabízí určitou míru abstrakce při práci s databází. Pro implementaci této metody je potřeba vytvořit objekty typu POJO (Plain Old Java Object), v architektuře DAO označované DTO (Data Transfer Object), dále rozhraní spojující DTO a JDBC a v neposlední řadě pak objekty s přímou implementací databázových dotazů.

DTO objekty typu POJO jsou jednoduché objekty, které jsou vytvářeny pro jednotlivé tabulky z databáze a jejich funkce je charakterizovat tabulku pomocí objektu v Javě s atributy, které korespondují s proměnnými v databázi. Objekt DTO tedy obsahuje proměnné odpovídající sloupcům mapované tabulky a dále obsahuje metody get a set pro získávání a nastavování hodnot proměnných.

Rozhraní DAO obsahuje abstraktní metody, které jsou pak přímo implementovány v objektech, které toto rozhraní využívají pro přístup k DTO.

Objekty DAO pak přímo obsahují implementaci databázových dotazů. Umožňují dynamickou tvorbu dotazů, takže stačí vytvořit metody, kterým se předávají parametry typu databázových atributů.

Mapování pomocí ORM

Jak už bylo zmíněno, tak ORM (Object Relational Mapping) využívá také objektů DAO, ale ORM však vytváří ještě více abstraktní rozhraní mezi daty v databázi a objekty v Javě. Pro použití ORM existuje mnoho nástrojů jako např. ORMLite, Hibernate, JPA (Java Persistence API), Ebean atd. Při použití ORM již uživatel nepracuje vůbec s databázovými příkazy, pro volání dotazů nad databází se využívá metod implementovaných v rozhraní

ORM, takže uživatel již ani nemusí mít znalosti databázových dotazů, i když znalost dotazů je spíše doporučována. Pro mapování objektů s databází využívá konfigurační soubory ve formátu xml. Bohužel vysoká míra abstrakce může způsobit snížení výkonu při vykonávání složitějších dotazů, např. obsahující spojování tabulek (JOIN).

2.2.4. MySQL PHP API

PHP (Hypertext Preprocessor) je skriptovací jazyk pro vývoj dynamických webových stránek. Nabízí dvě různá rozšíření pro spojení s MySQL databází.

První možností je rozšíření `mysql`, které je k dispozici pro PHP verze 4 i 5 a pracuje pouze s MySQL serverem do verze 4.1. Toto rozšíření neumožňuje vylepšený autentizační protokol, ani nepodporuje volání předpřipravených příkazů.

Druhou možností je vylepšené rozšíření `mysqli`, které je k dispozici jen pro PHP verze 5 a podporuje MySQL server verze 4.1.1 a vyšší. Toto rozšíření již umožňuje vylepšený autentizační protokol použitý v MySQL serveru 5.0, stejně jako předpřipravené příkazy. Navíc toto rozšíření nabízí pokročilé rozhraní pro objektově orientované programování.

3. Návrh řešení a implementace

Po prozkoumání databázových dotazů v obecném směru, je nyní zapotřebí prozkoumat zadanou databázi a připravit vhodné databázové dotazy pro výběr a vkládání dat do tabulek.

3.1. Zadaná databáze

Na obr. 3.1 je zobrazena část ER (Entity-Relationship) modelu databáze. Zde jsou uvedeny jen některé tabulky databáze, které budou využity při vývoji navrhovaného řešení. Každá tabulka má prefix, který značí příslušnost tabulky. Prefix *s* značí data sítě, prefix *v* značí data vlaků, prefix *g* značí data pro potřeby GTN. V tomto případě tabulky g_trat, g_tratbodrz a g_bodrz obsahují data o stanicích v řízené oblasti. Tabulka s_bod obsahuje seznam všech bodů v železniční síti. A tabulky s prefixem v obsahují data týkající se jízdního řádu vlaků.

3.2. Uživatelský scénář

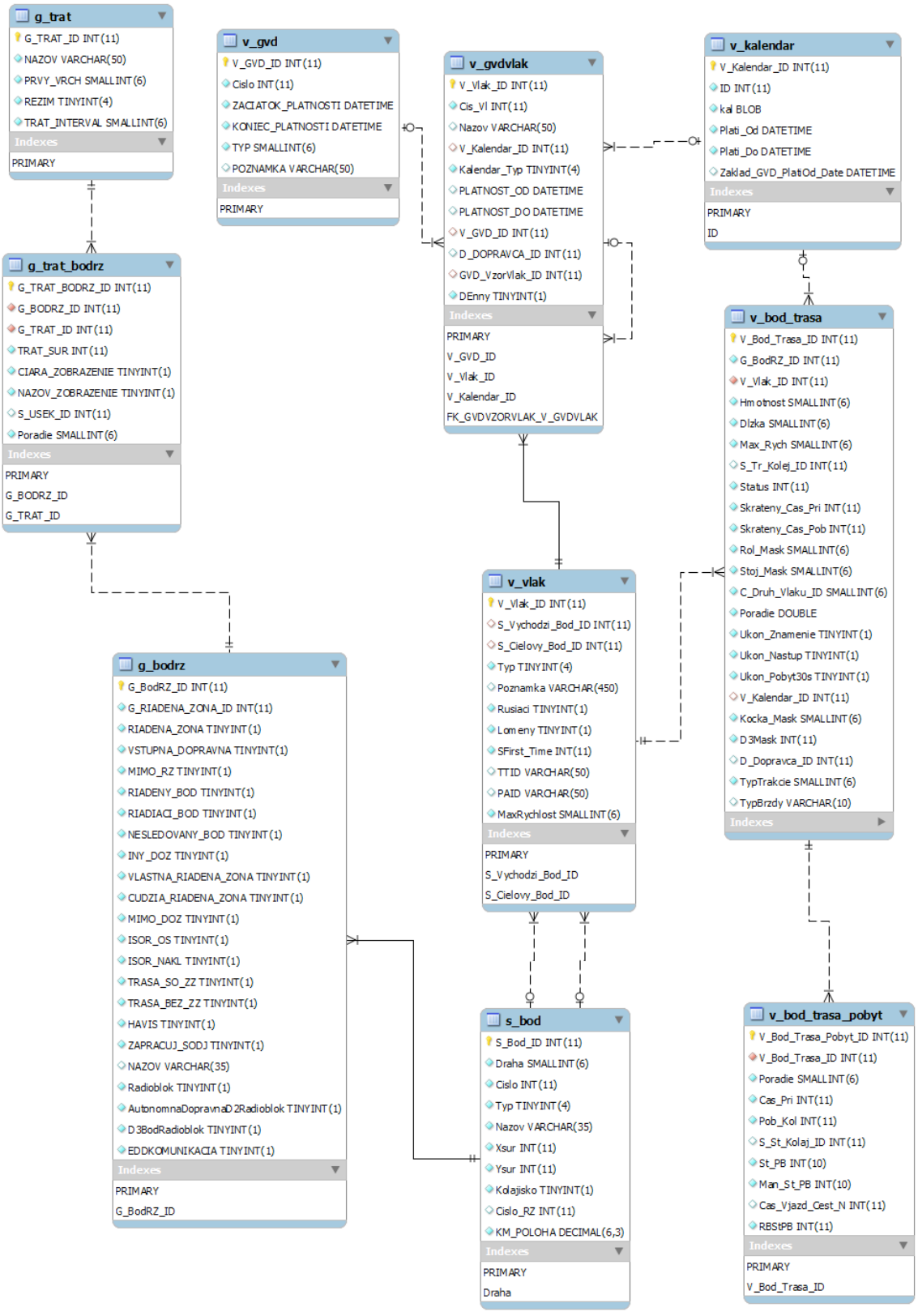
Na základě dat obsažených v tabulkách, především na relacích mezi nimi a existenci jejich primárních klíčů, byla zvolena následující posloupnost, kterou musí aplikace splňovat, aby nedošlo k chybné reprezentaci dat:

Prvním bodem je vytvoření záznamu v tabulce **v_gvd**, která obsahuje rozsah platnosti GVD a její primární klíč se dále předává do tabulky **v_gvdvlak** a je tím identifikován vlak v příslušném GVD.

Následuje vytvoření záznamu v tabulce **v_vlak**, která obsahuje informace o typu vlaku a jeho výchozí a cílovou stanici (ty se mohou lišit od samotných stanic v jízdním řádu, protože jízdní řád je tvořen pro řízenou oblast, ale vlak může mít výchozí a cílovou stanici mimo tuto oblast, např. rychlík, expres, atd.). Samotným záznamem v tabulce **v_vlak** však není vlak jednoznačně identifikován, a proto zároveň s tím se musí vytvořit také tabulka **v_gvdvlak**, která obsahuje další doplňující informace o vlaku, jako např. číslo vlaku, název, typ kalendáře a příslušný GVD.

Vlaku může být na základě typu kalendáře také přiřazen kalendář. Typ kalendáře je vyjádřen číselnou hodnotou. Hodnota může nabývat „*-1*“, „*0*“ a „*1*“. Hodnota „*-1*“ reprezentuje kalendář typu „*Jede denně*“, v tom případě není potřeba přiřazovat žádný

kalendář, dále hodnota „*0*“ reprezentuje „*Vlastní kalendář*“ a v tom případě je nutné přiřadit vlaku také kalendář, reprezentovaný primárním klíčem z tabulky **v_kalendar**. A poslední



Obr. 3.1 ER model zadané databáze

hodnota „*1*“ reprezentuje kalendář typu „*Kalendář v dopravně*“, což znamená, že kalendář není přiřazen vlaku, ale je přidáván až samotným stanicím, kterými vlak na dané trase projíždí.

Existence záznamu v tabulce **v_kalendar** tedy není nezbytnou podmínkou při vytváření vlaku, pokud vlak jede denně, nebo má kalendář v dopravně, ale pokud bude mít vlak vlastní kalendář, pak je nutné předat primární klíč z této tabulky do tabulky **v_gvdvlak**. To může být řešeno dialogem, který při vytváření vlaku umožní také vytvoření kalendáře.

Po vytvoření tabulek **v_vlak**, **v_gvdvlak**, příp. **v_kalendar** je potřeba identifikovat trať. Trať je reprezentována tabulkami **g_trat**, **g_tratbodrz** a **g_bodrz**. Tyto tabulky nejsou zadávány z editoru, ale měly by již být předem nadefinované z prostředí GTN. Obsah těchto tabulek bude vysvětlen dále v textu, kde bude ukázáno, jak z těchto tabulek získat posloupnost stanic na dané trati.

Po zvolení stanic je již tedy možné vytvářet záznamy v tabulkách určených pro data týkající se jízdy a pobytu vlaku. Jsou to tabulky **v_bod_trasa** a **v_bod_trasa_pobyt**.

Tabulka **v_bod_trasa** obsahuje informace o stanici (primární klíč z tabulky **g_bodrz**), o vlaku (primární klíč z tabulky **v_vlak**) a traťové koleji (primární klíč z tabulky **s_usekkolaj**). Tabulka dále obsahuje informace o hmotnosti, délce, maximální rychlosti, druhu vlaku, příp. nastavit kalendář. Vytvořením záznamu v této tabulce je vygenerován primární klíč **v_bod_trasa_id**, kterým je dále stanice identifikována v tabulce **v_bod_trasa_pobyt**.

Tabulka **v_bod_trasa_pobyt** pak obsahuje údaje o staniční koleji (*s_st_kolaj_id* – odkaz na tabulku **s_bodkolaj**) a časové údaje – čas příjezdu (*cas_pri*) a délka pobytu (*pob_kol*). Nutno podotknout, že v této tabulce může existovat více záznamů pro jeden klíč z tabulky **v_bod_trasa**. To je dáno tím, že každý záznam v této tabulce reprezentuje pobyt na jedné staniční koleji, pokud tedy vlak přejíždí ve stanici přes více staničních kolejí, pak musí existovat pro každou tuto kolej příslušný záznam v této tabulce.

Tímto jsou již pokryty všechny tabulky, které je nutné pomocí editoru vyplňovat či editovat. Nyní se tedy podívejme na posloupnosti SQL příkazů, které bude potřeba volat z prostředí aplikace.

3.2.1. Databázové příkazy při editaci dat

- Volba existujícího GVD nebo vytvoření nového GVD

Vyhledání všech záznamů v tabulce **v_gvd**:

- čísla GVD mohou být zobrazena např. v rozevíracím seznamu (comboboxu), z kterého může uživatel zvolit požadovaný GVD. Po zvolení GVD by měly být uživateli zobrazeny doplňující informace o GVD. To může být realizováno pomocí textových polí (textfield).

```
mysql> SELECT * FROM v_gvd;
+----------+--------+---------------------+---------------------+
| v_gvd_id | cislo  | zaciatok_platnosti  | koniec_platnosti    |
+----------+--------+---------------------+---------------------+
|        2 |   2000 | 2010-12-12 00:00:00 | 2011-06-11 23:59:00 |
|        3 | 201104 | 2011-06-12 00:00:00 | 2011-09-04 23:59:59 |
+----------+--------+---------------------+---------------------+
1 row in set (0.00 sec)
```

Vytvoření nového záznamu:

Na základě příkazu z prostředí aplikace by měl být vyvolán dialog, který bude obsahovat textová pole, do kterých uživatel vyplní potřebná data (číslo GVD, platnost od/do, příp. poznámku). U dat platnosti může být funkce, která bude generovat rozsah platnosti (např. platnost OD nastaví na aktuální datum s časem 00:00 a platnost DO nastaví na datum o rok později s časem 23:59). Po doplnění potřebných údajů by mělo proběhnout uložení do databáze, tzn. mělo by být zkontrolováno, zda stejné číslo GVD již náhodou neexistuje a případně tyto údaje uložit.

Kontrola existence čísla GVD:

```
mysql> SELECT * FROM v_gvd WHERE cis_vl=?;
```

Pokud číslo GVD existuje, měl by program uživatele upozornit a vrátit zpět do formuláře, aby číslo změnil. Když číslo neexistuje, pak se provede následující příkaz pro vložení a zároveň vrátí číslo primárního klíče.

```
mysql> INSERT INTO v_gvd
    -> (Cislo,
    -> ZACIATOK_PLATNOSTI,
    -> KONIEC_PLATNOSTI,
    -> TYP,
    -> POZNAMKA)
    -> VALUES
    -> ('1','2012-01-01 00:00:00','2012-01-01 23:59:59','1',
    -> 'poznamka');
Query OK, 1 row affected (0.02 sec)
```

```
mysql> SELECT last_insert_id();
+------------------+
| last_insert_id() |
+------------------+
|               38 |
+------------------+
1 row in set (0.00 sec)
```

Případně pokud bude uživatel data editovat, pak na základě zvoleného čísla GVD by měl být vyvolán dialog, který bude předvyplněný stávajícími daty, uživateli bude umožněno tato data upravovat a při ukončení dialogu bude vyvolán příkaz UPDATE:

```
mysql> UPDATE v_gvd SET Cislo=2012001 WHERE v_gvd_id=38;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Opět by měly existovat algoritmy, které v případě změny budou kontrolovat, zda změněná data jsou v pořádku (existující číslo GVD, správně zadaný formát data). Pokud bude změněno datum, tak by mělo dojít k aktualizaci dat v tabulkách **v_gvdvlak** a **v_kalendar**, protože dle informací získaných z AŽD by měla být v těchto tabulkách tyto data kopírována.

Pro aktualizaci dat v tabulce v_gvdvlak by mohl být zavolán následující příkaz:

```
mysql> UPDATE v_gvdvlak SET platnost_od='2012-01-01 00:00:00',
    -> platnost_do='2012-12-31 23:59:59'
    -> WHERE v_gvd_id=4;
Query OK, 257 rows affected (0.00 sec)
Rows matched: 257  Changed: 257  Warnings: 0
```

Při aktualizaci tabulky **v_kalendar** již nastává drobný problém, protože v této tabulce není využito primárního klíče tabulky **v_gvd**.

Teprve až je vytvořen nebo zvolen GVD by mělo být uživateli umožněno výběru vlaku. To je dáno tím, že v tabulce **v_gvdvlak** může existovat stejné číslo vlaku vícekrát, ale vždy jen jednou pro jediné **v_gvd_id**.

- Volba existujícího vlaku nebo vytvoření nového vlaku

Po zvolení GVD by tedy mělo dojít k vyhledání vlaků, které existují pro dané GVD. To lze provést pomocí následujícího příkazu:

```
mysql> SELECT * FROM v_gvdvlak
    -> LEFT JOIN v_vlak ON v_gvdvlak.v_vlak_id=v_vlak.v_vlak_id
    -> WHERE v_gvd_id='3';
#
# výstup je příliš rozsáhlý pro zobrazení
#
257 rows in set (0.00 sec)
```

Tímto došlo k nalezení všech vlaků, které náleží zvolenému GVD (v tomto příkladu *v_gvd_id*=3).

Může také nastat případ, kdy bude vytvořen úplně nový GVD, to by znamenalo, že pro něj nebude existovat žádný vlak. Tento případ by bylo vhodné také ošetřit tak, aby bylo možné v případě vytváření nového GVD zvolit z existujících vlaků a ty překlopit do nového GVD. Pro výběr existujících vlaků můžeme zvolit mezi těmito příkazy:

```
mysql> SELECT DISTINCT cis_vl FROM v_gvdvlak;
280 rows in set (0.00 sec)
```

Pomocí SELECT DISTINCT zobrazit uživateli pouze čísla vlaků a na základě jeho volby, pak algoritmicky dohledat zbývající informace s pomocí SELECT a podmínky WHERE *cis_vl*=<*zvolené číslo*>.

Případně druhá možnost je použití následujícího příkazu:

```
mysql> SELECT * FROM v_gvdvlak GROUP BY cis_vl;
280 rows in set (0.00 sec)
```

Z počtu vypsaných řádek (280 řádek) v obou případech je zřejmé, že oba příkazy zvolí stejné hodnoty, rozdíl je v tom, že při použití SELECT DISTINCT je vybrán pouze sloupec *cis_vl*, zatímco v případě GROUP BY je načtena celá tabulka.

Ještě pro lepší optimalizaci by bylo vhodné doplnit GROUP BY o JOIN, aby došlo jedním příkazem k výběru z obou tabulek.

Pomocí výše uvedených příkazů bylo docíleno vypsání všech dostupných vlaků (buď přímo k příslušnému GVD nebo obecně všech dostupných vlaků nezávisle na GVD). Tato data by opět bylo vhodné zobrazit v rozvinovacím/rozevíracím seznamu (combobox), z kterého by uživatel mohl zvolit požadovaný vlak. Zvolením by došlo opět k načtení dostupných informací do textových boxů, čímž by uživatel viděl ihned obsažené hodnoty a případně by měl možnost tyto boxy editovat po zvolení editačního módu (např. dialogové okno, které by obsahovalo vyplněné údaje o vlaku).

Nyní je tedy potřeba vyřešit, jakým způsobem duplikovat záznam vlaku pro použití s novým GVD. Data vlaku jsou rozdělena do dvou tabulek – **v_vlak** a **v_gvdvlak**. Tabulka **v_vlak** obsahuje primární klíč s funkcí AUTO_INCREMENT (automatické generované ID záznamu) a informace o typu vlaku a o výchozí a cílové stanici, zatímco **v_gvdvlak** používá jako referenci klíč z tabulky **v_vlak**, a pak obsahuje informace upřesňující příslušnost k určitému GVD. Pokud tedy bude zvolen určitý vlak z dříve zmíněného příkazu SELECT ... GROUP BY, pak známe číslo vlaku a známe také *v_vlak_id*. Pokud je tedy známo *v_vlak_id* je možné vytvořit nejprve kopii v tabulce **v_vlak** pomocí příkazu:

```
mysql> INSERT INTO v_vlak (S_Vychodzi_Bod_ID,S_Cielovy_Bod_ID, Typ,
    -> Poznamka,Rusiaci,Lomeny,SFirst_Time,TTID,PAID,MaxRychlost)
    -> SELECT S_Vychodzi_Bod_ID,S_Cielovy_Bod_ID,Typ,Poznamka,
    -> Rusiaci,Lomeny,SFirst_Time,TTID,PAID,MaxRychlost
    -> FROM v_vlak
    -> WHERE v_vlak_id='58692';
Query OK, 1 row affected (0.00 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

Tímto příkazem byl vytvořen nový záznam s primárním klíčem, který lze získat zavoláním funkce SELECT last_insert_id().

Pak na základě znalosti nově vzniknutého ID, čísla vlaku, a znalosti **v_gvd_id**, které bylo zvoleno v předchozích krocích, tak by mělo proběhnout vložení tohoto vlaku do tabulky **v_gvdvlak**.

- Volba trati a zastávek na trati

Data trati jsou pevně stanovena, tj. tato data nebudou editována, pouze slouží pro identifikaci trati a potřebné klíče budou později předány do tabulky, která identifikuje stanice. Následujícím příkazem jsou vypsány všechny existující trati v databázi.

```
mysql> SELECT g_trat_id, nazov FROM g_trat;
+-----------+------------------------+
| g_trat_id | nazov                  |
+-----------+------------------------+
|         4 | Planá u M.L. - Tachov  |
+-----------+------------------------+
```

Výpis nalezených hodnot je zobrazen uživateli (v aplikaci se stačí omezit na výpis názvů tratí), jehož úkolem je nyní zvolit trať, pro kterou bude editovat hodnoty. Zvolením je získáno *g_trat_id*, které je předáno do následujícího příkazu:

```
mysql> SELECT g_bodrz_id, poradie  FROM g_trat_bodrz WHERE g_trat_id=4
    -> ORDER BY Poradie ASC;
    +------------+---------+
    | g_bodrz_id | poradie |
    +------------+---------+
    |       8632 |       1 |
    |       8793 |       2 |
    +------------+---------+
    2 rows in set (0.00 sec)
```

Tímto příkazem jsou nalezeny všechny stanice, které se na zvolené trati nacházejí. Tabulka neobsahuje názvy stanic, proto bude nutné zavolat další příkazy, které dohledají názvy všech stanic na zvolené trati:

```
mysql> SELECT g_bodrz_id, nazov FROM g_bodrz WHERE g_bodrz_id=8632 OR
g_bodrz_id=8793;
+------------+---------------+
| g_bodrz_id | nazov         |
+------------+---------------+
|       8632 | Planá u M.L.  |
|       8793 | Tachov        |
+------------+---------------+
2 rows in set (0.00 sec)
```

Pro optimalizaci lze využít příkazu LEFT JOIN, kdy dohledáme v rámci jediného příkazu také názvy z druhé tabulky.

```
mysql> SELECT g_trat_id IDTrat, TratBody.g_bodrz_id g_bodrzID,
    -> nazov Nazev, poradie Poradi from g_trat_bodrz TratBody
    -> LEFT JOIN g_bodrz B1 ON TratBody.g_bodrz_id=B1.g_bodrz_id
    -> WHERE g_trat_id=4;
+--------+-----------+--------------+--------+
| IDTrat | g_bodrzID | Nazev        | Poradi |
+--------+-----------+--------------+--------+
|      4 |      8632 | Planá u M.L. |      1 |
|      4 |      8793 | Tachov       |      2 |
+--------+-----------+--------------+--------+
2 rows in set (0.00 sec)
```

Tímto byly vyhledány stanice, které budou sloužit pro zadávání nových dat do jízdního řádu. Pokud bude potřeba editovat stávající data, je potřeba vyhledat tato data z tabulky **v_bod_trasa** a **v_bod_trasa_pobyt**. Z tabulky **v_bod_trasa** lze dohledat data pomocí klíčů *g_bodrz_id* a *v_vlak_id*, z tabulky **v_bod_trasa_pobyt** pak pomocí *v_bod_trasa_id*. Data jsou tedy identifikována pomocí stanice a čísla vlaku.

3.3. Návrh aplikace

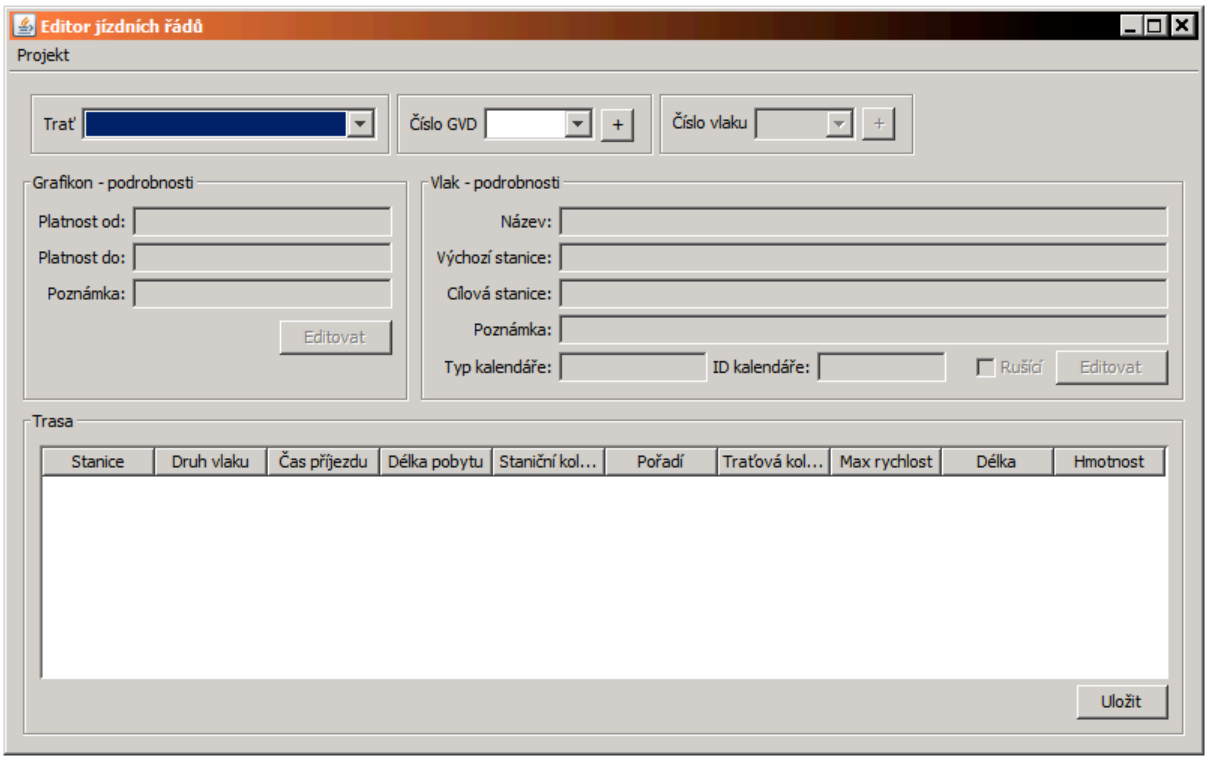
Při návrhu možného řešení bylo použito programovacího jazyku Java, vývojové prostředí Eclipse, pro spojení s databázovým serverem je použito JDBC ovladače a pro mapování do objektů bylo použitou DAO objektů.

Pro vytvoření DAO objektů byla použita aplikace Firestorm/DAO od společnosti CodeFutures. Tato aplikace je licencovaná, ale pro studijní účely společnost nabízí vzdělávací program, v rámci kterého lze použít zdarma generátor DAO objektů a pracovat s vygenerovanými zdrojovými kódy.

Program byl vyvíjen v operačním systému Windows, ale jelikož programovací jazyk Java je nezávislý na platformě, tak by mělo být možné aplikaci spustit i v jiných OS. Funkce v jiných OS však nebyla testována.

Základní okno aplikace je zobrazeno na obr. 3.2. Po spuštění je uživateli umožněna pouze volba trati a GVD z rozevíracích seznamů.

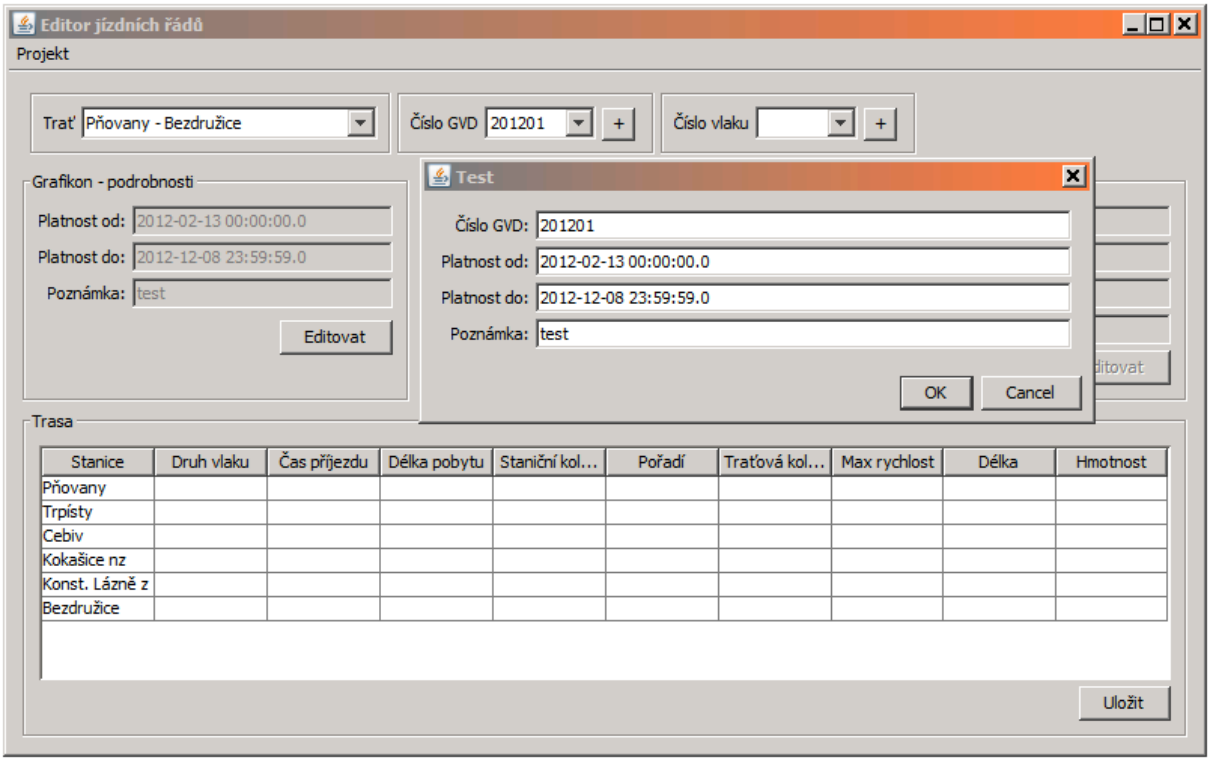
Po zvolení trati jsou dohledány stanice na této trati a je vygenerována tabulka pro zadávání příslušných hodnot týkajících se stanice, zobrazená v panelu Trasa.



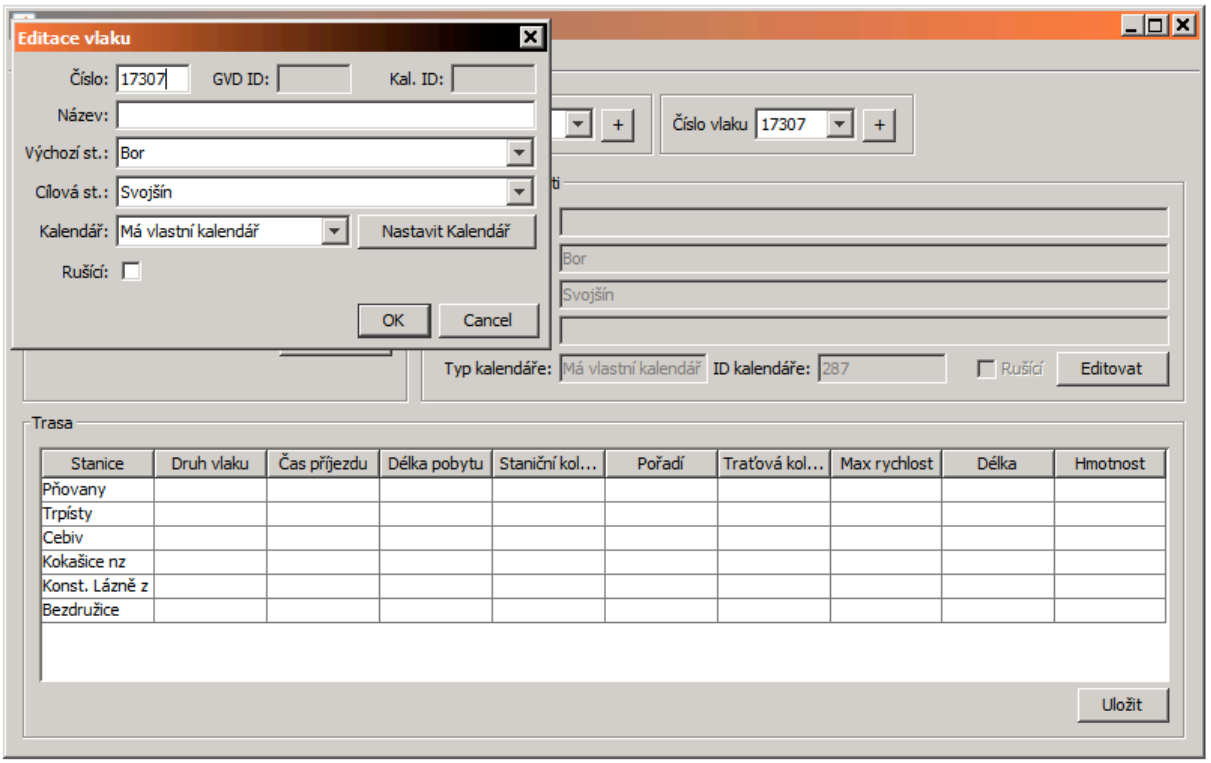
Obr. 3.2 Základní okno aplikace po spuštění

Po zvolení čísla GVD jsou zobrazeny podrobnosti v panelu Grafikon – podrobnosti a pomocí tlačítka Editovat je možné vyvolat Dialog pro editaci zvoleného GVD. Případně je možné vytvořit nový GVD stisknutím na tlačítko „+“, které vyvolá stejný dialog jako při editaci pouze s tím rozdílem, že v polích je předvyplněný jen rozsah platnosti (generovaný na základě aktuálního data s ročním rozsahem). Dialogové okno pro editaci vlaku je ukázáno na obr. 3.3. Na uvedeném obrázku lze také vidět doplněnou tabulku se stanicemi zvolené trasy. Po vybrání příslušného GVD je dále povolena volba vlaku.

Vlak je možné opět vybrat z rozevíracího seznamu a po zvolení jsou zobrazeny informace o vlaku v panelu Vlak – podrobnosti. Možnost editace a přidání nového vlaku je stejná jako v případě GVD, tedy stisknutím tlačítka Editovat je vyvolán dialog s předvyplněnými údaji o vlaku a stisknutím tlačítka „+“ je vyvolán prázdný dialog. Dialogové okno je zobrazeno na obr. 3.4.



Obr. 3.3 Dialog pro editaci údajů GVD



Obr. 3.4 Dialog pro editaci údajů vlaku

Po výběru trati, GVD a vlaku jsou dohledány existující záznamy z tabulky **v_bod_trasa** a **v_bod_trasa_pobyt** a zobrazeny v tabulce. Následně po editaci dat trasy by

mělo dojít k uložení stisknutím tlačítka Uložit. Tlačítko vyvolává sadu kontrolních funkcí, které kontrolují splnění sounáležitostí.

Doposud jsou implementovány funkce pro zobrazování a výběr dat trati, dále pak lze ukládat a editovat GVD a vlak, a částečně jsou implementovány funkce pro editaci a vytváření kalendáře.

3.3.1. Instalace a konfigurace aplikace

Aplikace je na přiloženém CD ve formě zdrojových kódů projektu aplikace Eclipse a dále je obsažen spustitelný soubor *editor.jar*. Aplikaci není potřeba instalovat, ale je potřeba ji spouštět s předáním argumentu obsahující přihlašovací údaje k databázi. Pro tyto účely slouží dávkovací soubor *editor.bat*, který spustí aplikaci s potřebným argumentem, který načte konfiguraci ze souboru. Konfigurace je obsažena v souboru *settings*. V souboru je potřeba před spuštěním aplikace nastavit adresu MySQL serveru, jméno databáze a přihlašovací údaje.

Pokud by byla vyžadována míra integrace do OS, pak by bylo možné vytvořit instalační program, který by vytvořil zástupce na ploše a v nabídce start a umístil spouštěcí soubor aplikace do složky programů.

4. Závěr

Cílem práce bylo seznámení se s funkcemi Graficko-technologické nadstavby zabezpečovacího zařízení a navržení modulu pro vkládání a editaci dat jízdního řádu vlaků.

Jsou zde tedy popsány klíčové funkce GTN, možnosti použití a krátce popsána historie vývoje.

Pro samotný návrh řešení byla společností AŽD zadána relační databáze MySQL, obsahující potřebná data. Pro potřeby editoru byla však potřeba jen její část, která je v práci detailně popsána.

Pro volání databázových dotazů bylo nutné použití jazyku SQL. Proto byla také podrobně popsána struktura těchto příkazů a na příkladech předvedeno jejich použití. Při popisu databáze byly shrnuty možnosti programového spojení s databází. Uvedeny byly především typy ovladačů, které připadaly v úvahu při vývoji aplikace.

Na základě znalosti struktury zadané databáze a databázových dotazů byl navržen uživatelský scénář, který bude uživatele provázet při práci s aplikací, a byly vytvořeny optimalizované databázové dotazy, které budou volány v pozadí aplikace.

Po navržení uživatelského scénáře následovala samotná implementace. Při návrhu aplikace byl zvolen programovací jazyk Java pro jeho přenositelnost mezi různými platformami, a proto ke spojení s databází byl využit ovladač JDBC. Pro optimalizovanější přístup k datům v databázi bylo zvoleno mapování do DAO objektů.

Aplikace umožňuje načtení a zobrazení dat trati, grafikonu a vlaků, a dále umožňuje vytváření a editaci grafikonu.

Navržené databázové dotazy, návrh grafického rozhraní aplikace a kontrolní funkce bude potřeba podrobit testování na straně zadavatele, zda-li nedochází k chybám při plnění databáze. A v případě, že by docházelo k chybnému plnění, pak alespoň návrh aplikace může posloužit společnosti AŽD jako stavební kámen pro další vývoj programu.

Literatura

1. POLACH, Vlastimil; HOUDA, Pavel. *Graficko-technologická nadstavba zabezpečovacího zařízení* [online]. Praha, 2001[cit. 2012-04-14]. Dostupné z: http://www.cdrail.cz/vts/CLANKY/1109.pdf

2. POLACH, Vlastimil. AŽD PRAHA, s.r.o. *Sběr informací o pohybu vlaku*. Praha, 2006.

3. POLACH, Vlastimil. *Aktuální stav vývoje GTN po 10letém fungování.* Reportér AŽD, Praha, 2009, č. 3. s. 10–11. Dostupný také z WWW: http://www.azd.cz/pro-media/casopis-reporter/

4. POLACH, Vlastimil. *Co nového v GTNv4.1?*. Reportér AŽD, Praha, 2010, č. 3. s. 21. Dostupný také z WWW: http://www.azd.cz/pro-media/casopis-reporter/

5. *MySQL 5.5 Reference Manual* [online]. 2011, 2011-10-20 [cit. 2012-04-24]. Dostupné z: http://dev.mysql.com/doc/refman/5.5/en/

6. *JDBC™ Guide: Getting Started.* DCA-FEEC-UNICAMP [online]. Mountain View, California: Sun Microsystems, ©1997 [cit. 2012-05-04]. Dostupné z: ftp://ftp.dca.fee.unicamp.br/pub/docs/gudwin/java/jdbc.pdf

PŘÍLOHY

Příloha A – Obsah CD

CESTA:	- **OBSAHUJE:**
./aplikace/bin/editor.jar	- Zkompilovaná aplikace editoru
./aplikace/bin/editor.bat	- Spouštěcí soubor
./aplikace/bin/settings	- Konfigurační soubor pro nastavení databáze
./aplikace/projekt/Editor-GUI/	- Zdrojové kódy, javadoc a další soubory projektu
./databaze/plzen_cheb.sql	- Soubor s obsahem zadané databáze
./Technicka_zprava/*.pdf	- Technická zpráva diplomové práce
	-
	-
	-
	-
	-
	-
	-
	-
	-
	-
	-

Evidenční list

Souhlasím s tím, aby moje diplomová / bakalářská práce byla půjčována k prezenčnímu studiu v Univerzitní knihovně ZČU v Plzni.

V Plzni dne…… ……

Podpis

Uživatel stvrzuje svým čitelným podpisem, že tuto diplomovou / bakalářskou práci použil ke studijním účelům a prohlašuje, že ji uvede mezi použitými prameny.

Jméno	Fakulta/katedra	Datum	Podpis