ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická
Katedra počítačů

Stromové procházení v Admineru
Tree-like data access in Adminer

Diplomová práce

Studijní program: Otevřená informatika
Studijní obor: Softwarové inženýrství

Vedoucí práce: RNDr. Ondřej Žára

Petro Kostyuk
Praha 2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kostyuk** Jméno: **Petro** Osobní číslo: **440937**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Stromové procházení v Admineru

Název diplomové práce anglicky:

Tree-like data access in Adminer

Pokyny pro vypracování:

Seznamte se s webovým nástrojem Adminer, určeným k pohodlné správě relačních databází ve webovém prohlížeči. Nastudujte možnosti jeho rozšíření a relevantní PHP API určené pro tvoru pluginů.
Na základě zkušeností z bakalářské práce navrhněte uživatelsky přívětivý způsob stromového procházení relačně provázaných dat. Uvažte jak dopředné procházení (směrem k primárnímu klíči), tak i reverzní (výčet hodnot primárního klíče v závislých tabulkách). Naimplementujte tuto funkcionalitu jako plugin do nástroje Adminer. Výsledný produkt zdokumentujte, proveďte kvalitativní uživatelské testování a software vystavte na místě, kde bude pohodlně k dispozici ostatním uživatelům Admineru.
Uvažte, je-li publikované rozšíření nezávislé na použité databázi (a jejích vlastnostech, např. InnoDB či MyISAM), či vázané na konkrétní RDBMS. Zhodnoťte, zda-li bylo rozhraní Admineru vhodné a použitelné pro tento úkol. Pokud během implementace došlo ke střetu s limity těchto rozhraní, navrhněte teoreticky, jaké úpravy API by byly pro tyto konkrétní případy vhodné.

Seznam doporučené literatury:

Adminer API, https://www.adminer.org/cs/extension/#api
Beighley, Morrison: Head First PHP & MySQL, O'Reilly 2011, ISBN 9788184046588
https://developer.mozilla.org/en-US/docs/Web

Jméno a pracoviště vedoucí(ho) diplomové práce:

RNDr. Ondřej Žára, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **10.02.2020** Termín odevzdání diplomové práce: **05.01.2021**

Platnost zadání diplomové práce: **30.09.2021**

RNDr. Ondřej Žára podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Praha, 28.12.2020

……...
Petro Kostyuk

Poděkování

Chtěl bych poděkovat vedoucímu této práce, RNDr. Ondřeji Žárovi za jeho vstřícnost během vedení a za všechny jeho rady a připomínky. Dále děkuji své rodině a přátelům za veškerou podporu, kterou mi pomohli k dovedení této práce do konce.

Abstrakt

Tato práce si dává za cíl navrhnout a vytvořit řešení, které umožní správcům relačních databází zobrazit související záznamy napříč různými tabulkami databáze. Řešení bude formou pluginu do existující aplikace pro správu databází, konkrétně aplikace Adminer. Druhotným cílem této práce je prozkoumat aplikační rozhraní, které Adminer nabízí pro vytváření pluginů a zhodnotit, zda bylo pro daný plugin vhodné.

Klíčová slova

Relační databáze, zobrazení souvisejících dat, cizí klíče, Adminer, Adminer plugin, Adminer API

Abstract

This work aims to design and create a solution that allows relational database administrators to view related records across different database tables. The solution will take the form of a plugin to an existing database management application, specifically the Adminer application. The secondary goal of this work is to examine the application interface that Adminer offers for creating plugins and to evaluate whether it was suitable for this plugin.

Keywords

Relational database, related data visualisation, foreign keys, Adminer, Adminer plugin, Adminer API

Obsah

1 Úvod do problematiky

1.1 Adminer

Adminer je webový nástroj pro správu databází napsaný v jazyce PHP. Nabízí napojení na různé databázové konektory jako MySQL, PostgreSQL, Oracle a další. Mezi základní nástroje které nabízí a které souvisí s touto prací patří výpis všech tabulek v databázi, zobrazování dat v jednotlivých tabulkách a pro jednotlivé tabulky zobrazení struktury tabulky a cizích klíčů, které jsou pro tabulku definované. Díky tomu umožňuje vyhledat data která potřebujeme a pomocí cizích klíčů zobrazit další související výběry dat.

1.2 Stromové procházení dat

Zatímco Adminer umožňuje zobrazit data odkazovaná pomocí cizího klíče v nové záložce, při velkém množství otevřených záložek přestává být tento způsob přehledný. A to je jeden z problémů, které by měl plugin pro stromové procházení vyřešit. Tedy zobrazit související data společně na jednom místě a s jasně zobrazenými vazby mezi jednotlivými pohledy.

Další problém který Adminer neumí řešit je zobrazení dat odkazovaných přes ‘zpětný cizí klíč’. Tím je myšleno, že pokud tabulka **Kniha** má cizí klíč odkazující na tabulku **Knihovna**, tak pokud máme vybraná data z tabulky **Kniha**, je jednoduché zobrazit v Admineru pro každý záznam relevantní záznam z tabulky **Knihovna**, ale pokud máme naopak vybraná data z tabulky **Knihovna**, tak není pohodlný způsob jak zobrazit související záznamy z tabulky **Kniha**, které na příslušný záznam z tabulky **Knihovna** odkazují. A jednoduché zobrazení dat přes ‘zpětný cizí klíč’ je druhý problém, který by měl plugin pro stromové zobrazení vyřešit.

Stromové zobrazení by tedy mělo být nástrojem, který umožní uživateli zobrazit přehledně a na jednom místě související data a to jak ty propojené přímo přes cizí klíč, tak ty, které jsou propojené zpětným cizím klíčem.

1.3 Možnosti rozšíření pro Adminer

Adminer nabízí rozhraní, které lze použít pro vytváření doplňků. Toto rozhraní funguje na principu toho, že se admineru dodá plugin, který má implementované některé z Adminerem definovaných metod. Adminer pak při spuštění vykonává svůj kód a na některých místech v kódu kontroluje, zda je v dodaném pluginu metoda s takovým názvem, který zrovna potřebuje. Pokud ano, tak použije implementaci v pluginu. Pokud ne, použije vlastní implementaci která odpovídá výchozí funkcionalitě Admineru. A právě tímto způsobem je možné rozšířit Adminer o nové funkcionality.

1.4 Návaznost na bakalářskou práci

Tato diplomová práce navazuje na bakalářskou práci “Administrační nástroj pro databázové systémy (stromové procházení, rozdíly)” kterou jsem odevzdal v květnu 2017.

Má bakalářská práce měla tři cíle.

1. Vytvořit nástroj pro správu databáze.
2. Vytvořit v nástroji z bodu 1 modul, který bude schopen porovnat strukturu dvou databází, zobrazí rozdíly a nabídnout nástroje pro úpravu rozdílných částí.
3. Vytvořit v nástroji z bodu 1 modul, který nabídne uživateli možnost stromového procházení dat.

Výstupem bakalářské práce byla aplikace pro správu databází. Byla napsaná bez návaznosti na žádný dosavadní nástroj pro správu databází a to v jazyce PHP za pomoci českého frameworku Nette. Hlavním cílem dané bakalářské práce bylo vyzkoušet navrhnou a vytvořit moduly popsané v bodech 2 a 3, které nejsou v běžných online nástrojích pro správu databází obvyklé. Proto jsem zvolil cestu zcela nové aplikace, která by mě neomezovala již existující strukturou, ale poskytla mi svobodu zkoušet různé přístupy a návrhy.

Zatímco přístup vytvoření zcela nové aplikace opravdu nabídnul svobodu, která byla zapotřebí pro vytvoření modulů 2 a 3, nevýhodou tohoto přístupu bylo to, že přestože aplikace nabízela nově vzniklé moduly které konkurenční aplikace neměly, sama o sobě neměla dostatečně propracované standardní funkcionality, které jsou pro aplikace pro správu databází zcela běžné. Příkladem může být jednoduché prohlížení a editace záznamů v tabulkách, stránkování tabulek, filtrace a tomu podobné funkcionality. Dále jsem se v rámci bakalářské práce nezaměřoval na bezpečnost a rychlost vytvářené aplikace a tudíž se dá označit spíše za ‘proof of concept’ než za konkurenčně schopnou aplikaci.

A právě na tyto problémy se zaměří tato diplomová práce. Za použití zkušeností získaných během bakalářské práce teď bude výstupem diplomové práce plugin, který bude řešit stejný problém jako ten, který jsem v bakalářské práci řešil v modulu z bodu 3. Nicméně na rozdíl od bakalářské práce kde bylo výstupem konceptové řešení, bude řešení z diplomové práce vytvořené se záměrem možnosti použití v reálném provozu a to jako plugin do aplikace Adminer, která je jednou z rozšířených aplikací pro správu databází. Díky tomu, že se jedná o plugin, tak se o standardní funkcionality postará Adminer a tento plugin se může plně věnovat jen stromovému procházení dat.

2 Plugin pro stromové procházení dat

Nejprve bych chtěl projít vzniklý plugin z uživatelského pohledu. Ukázal bych jaké nabízí možnosti a jak se používá a až poté bych ukázal jak funguje uvnitř.

V ukázce budu používat databázi **library2**, která je k dispozici v přiložených souborech jako mysql skript. Databáze obsahuje tabulky **address**, **book**, **employee**, **library** a propojovací tabulku **library_book** s následujícími vztahy:

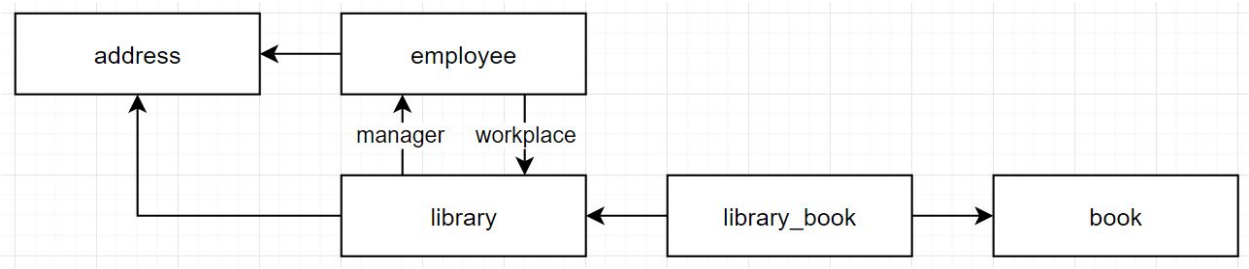


Diagram 1: Vztahy mezi tabulkami v ukázkové databázi.

2.1 Uživatelský pohled

Veškerá aktivita pluginu probíhá v samostatném modálním okně a nezasahuje do původního UI aplikace Adminer. Jediná změna v původním UI je sloupec **Tree**, který slouží jako vstupní bod pro otevření modálního okna se stromovým procházením. Dostaneme se k němu tak, že kliknutím na odkaz **select** nejprve zvolíme tabulku s daty kterou chceme začít naše procházení (v ukázkovém screenshotu se jedná o tabulku **library**) a u zobrazených záznamů v tabulce uvidíme nový sloupec **Tree**, přidaný pluginem (Obrázek 2). Ten obsahuje pro každý záznam v tabulce odkaz **view**, pomocí kterého otevřeme modální okno se stromovým procházením pro daný záznam v tabulce.

Obrázek 2: Vstupní bod pro stromové procházení.

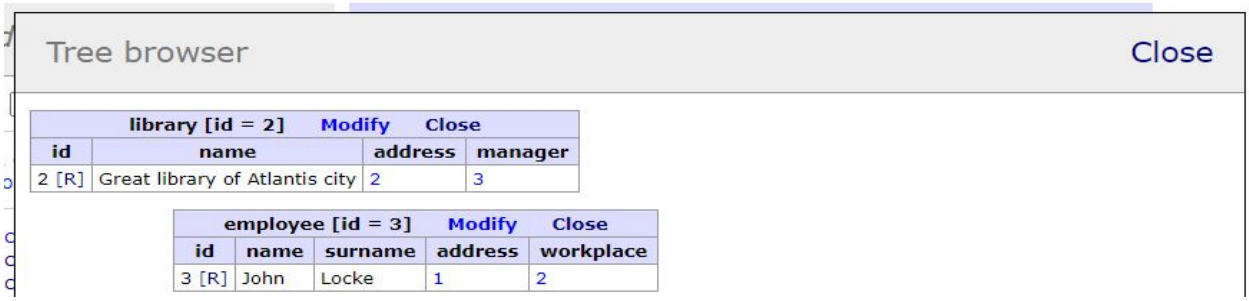
Kliknutím na odkaz **view** dojde k otevření modálního okna s vybraným záznamem (Obrázek 3).

Obrázek 3: Modální okno s prvotním výběrem dat.

V modálním okně vidíme tabulku odpovídající záznamu, pro který jsme otevřeli stromové procházení. Jelikož pojem **tabulka** má v relačních databázích svůj význam, tuto tabulku která se skládá z hlavičky a těla a obsahující záznam (nebo záznamy, viz dále) který jsme si zobrazili v rámci procházení dat nazvěme **výběr dat**. Celé modální okno je možné zavřít pomocí odkazu **Close** (Obrázek 3, položka E). U modálního okna a jeho obsahu jsem se pokusil dodržet výchozí styl zavedený aplikací Adminer.

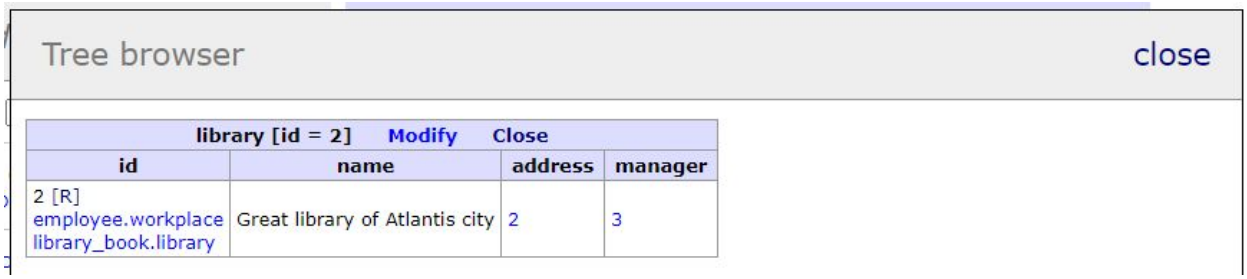
Výběr dat má hlavičku která se skládá ze dvou řádků. V horním řádku je identifikátor výběru. Skládá se z názvu tabulky, ze které jsou daná data a z podminky (nebo sady podmínek), pomocí kterých jsme záznamy z tabulky vybrali. Dále obsahuje odkazy na dva příkazy a to **Modify** (Obrázek 3, položka A) a **Close** (Obrázek 3, položka B). Příkaz **Modify** otevře novou záložku, ve které bude standardní Adminerový formulář pro editaci záznamu. Příkaz **Close** zavře konkrétní výběr dat a jeho podvýběry (viz dále). V případě, že byl zavřený prvotní výběr dat a v modálním okně žádné výběry dat nezůstaly, dojde i k zavření celého modálního okna. Druhý řádek hlavičky obsahuje názvy sloupců.

Tělo výběru dat obsahuje záznam nebo záznamy, které byly pro daný výběr dat nalezeny. U každého záznamu jsou zvýrazněny sloupce s cizími klíči (Obrázek 3, položka C) a lze je prokliknout. Prokliknutím cizího klíče se v daném výběru dat objeví podvýběr dat, který odpovídá záznamu odkazovaném cizím klíčem. Prokliknutím cizího klíče **manager 3** z ukázky na Obrázku 3 dostaneme záznam z tabulky **employee** u kterého je **id = 3** (Obrázek 4).



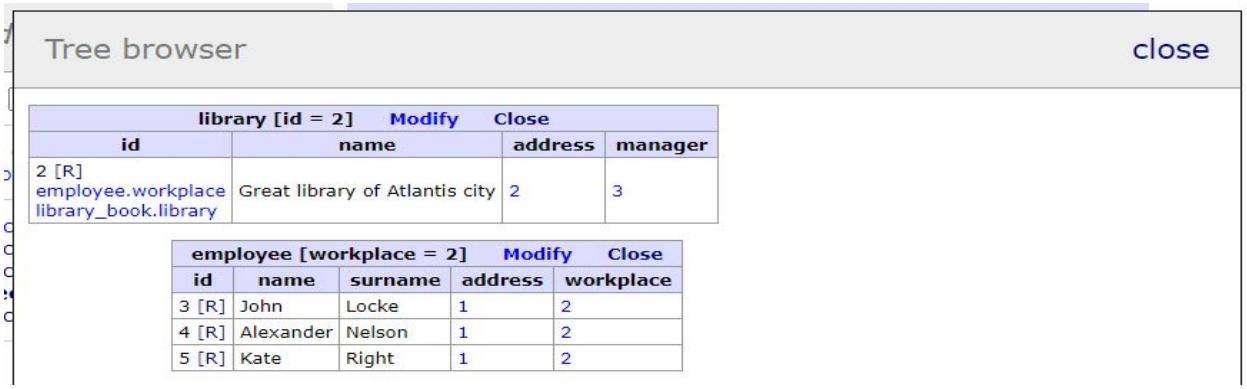
Obrázek 4: Stav po prokliknutí cizího klíče manager 3.

Pokud pro výběr dat existují zpětné cizí klíče, tedy cizí klíče z jiných tabulek, které odkazují na některé sloupce tabulky ve výběru dat, bude u příslušných sloupců ve výběru dat odkaz **[R]** (Obrázek *3*, položka D). Po kliknutí na odkaz dojde k rozbalení seznamu, ve kterém jsou vypsány všechny sloupce z jiných tabulek, které na daný sloupec ve výběru dat odkazují. Například na sloupec **library.id** odkazují sloupce **employee.workplace** a **library_book.library**, což můžeme vidět na Obrázku 5.

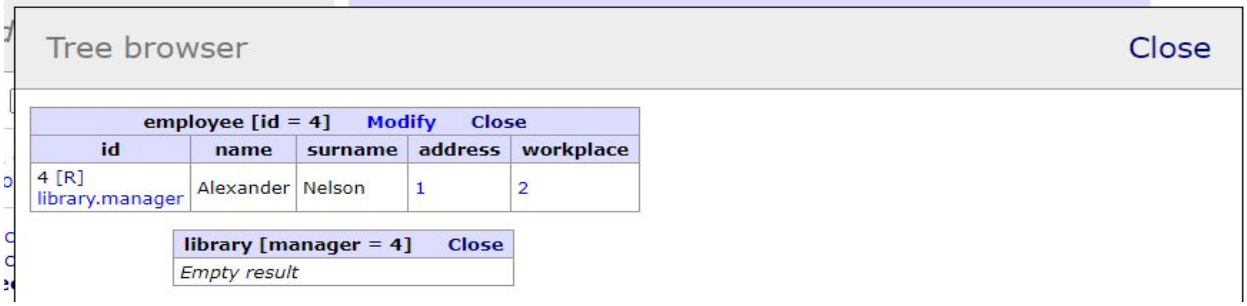


Obrázek 5: Přehled zpětných cizích klíčů po kliknutí na odkaz [R].

Prokliknutím zpětného cizího klíče dostaneme opět odpovídající záznamy, kterých ale na rozdíl od přímého cizího klíče může být i několik (zaměstnanci z knihovny id = 2, Obrázek 6), nebo i žádné (knihovny, ve kterých je zaměstnanec id = 4 manažerem, Obrázek 7)



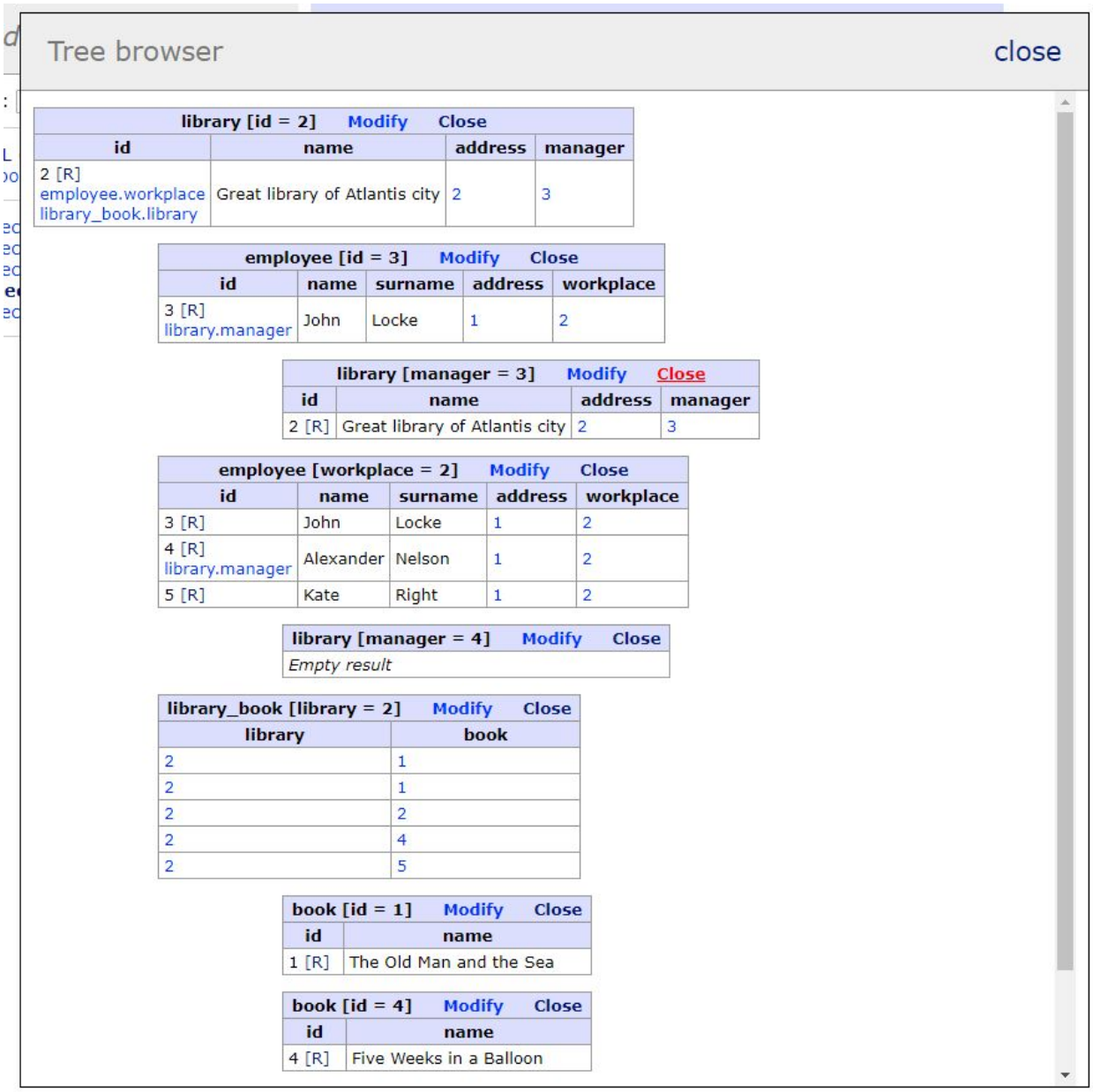
Obrázek 6: Výběr dat přes zpětný cizí klíč.



Obrázek 7: Výběr dat neobsahuje žádné záznamy.

Název **[R]** je od prvního písmene výrazu Reverse foreign key (zpětný cizí klíč), a zvolil jsem pouze první písmeno z důvodu ušetření místa v tabulce. Jelikož celý název je dlouhý a sloupců s cizími klíči může být několik a mohu mít libovolnou pozici, ne jen zleva, tak by bylo nepřehledné vypisovat celý název. Ze stejného důvodu jsem se rozhodl nezobrazovat všechny zpětné klíče už při načtení výběru dat, ale zobrazit jejich seznam až po kliknutí na odkaz **[R]**.

Pro ukázku: pokud budeme otevírat další a další výběry dat, můžeme dostat stromový přehled souvisejících dat jako je na obrázku 8.



Obrázek 8: Ukázka stromového přehledu dat.

2.2 Vývojářský pohled

Adminer je napsaný v jazyce PHP. Jednotlivé funkcionality nejsou napsané objektově, ale pomocí sady příkazů a funkcí. Nicméně ve zdrojových kódech je několik objektů, které se používají napříč všemi funkcionalitami.

Jednou z těchto tříd je třída **Adminer**. Obsahuje velké množství metod volaných na různých místech aplikace a právě tato třída je základem pro rozšiřování funkcionality Admineru pomocí uživatelských pluginů. Pluginy fungují tak, že uživatel vytvoří třídu, která obsahuje některé z metod implementovaných ve třídě **Adminer**. Uživatel který chce některé pluginy použít musí vytvořit PHP soubor, ve kterém každý z pluginů inicializuje a při vytváření objektu **Adminer** je Admineru předá jako aktivní pluginy.

Následně pokračuje standardní běh programu a ten má k dispozici objekt **Adminer** rozšířený o uživatelské pluginy. Program volá metody na objektu **Adminer** a ten kontroluje, zda některý z uživatelských pluginů nemá metodu se stejným názvem jako ta, která se zrovna volá. Pokud ne, tak použije vlastní výchozí implementaci. Pokud ano, tak použije implementaci z daného pluginu a vlastní implementaci volat nebude.

```
3   class AllowEmptyPasswordPlugin {
4       function login($login, $password) {
5           return true;
6       }
7   }
8
9   function adminer_object() {
10      // Adminer customization allowing usage of plugins
11      require_once "./plugin.php";
12      require_once "./AdminerTreeViewer/AdminerTreeViewer.php";
13
14      return new AdminerPlugin([
15          new AllowEmptyPasswordPlugin(),
16          new AdminerTreeViewer("AdminerTreeViewer/script.js")
17      ]);
18  }
19
20  // include original Adminer or Adminer Editor
21  include './adminer-4.7.4.php';
```

Obrázek 9: Ukázka souboru index.php, ve které je vidět jednak zapojení pluginů AdminerTreeViewer a AllowEmptyPasswordPlugin do Admineru, jednak ukázka velmi jednoduchého pluginu AllowEmptyPasswordPlugin který povoluje použití prázdného hesla (což je v současné verzi Admineru pro zvýšenou bezpečnost zakázané).

Adminer má každou funkcionalitu napsanou jako sadu samostatných příkazů a má jen pevně dané metody které jsou ve třídě Adminer které jsou určené k rozšiřování. Proto není pohodlný způsob jak přímo v PHP pluginu přidat celou novou funkcionalitu (viz uživatelský pohled na plugin, kde je vidět že se jedná o modální okno uvnitř kterého je samostatná nová funkcionalita). Proto jsem se rozhodl tento plugin napsat v jazyce JavaScript a nabízené PHP rozhraní pro rozšíření použít jen pro jeho načtení tohoto JavaScript pluginu do aplikace Adminer.

2.2.1 PHP část

Jedná se o soubor **src/AdminerTreeView.php**. Tato část má pouze dva jednoduché úkoly a to za prvé načíst na stránku JS skript, který se postará o veškerou funkcionalitu na stránce a za druhé ke každé tabulce vypsat meta tag s informací o cizích klíčích.

O načtení JS se stará metoda **AdminerTreeView.navigation**, kterou Adminer volá při vykreslování levého menu. Díky tomu, že se menu vykresluje na každé stránce právě jednou, je tato metoda ideální pro rozhodnutí, zda se jedná o stránku kterou chceme rozšířit o nové možnosti z pluginu a pokud ano tak se postarat o spuštění JS. Je pravda, že tato metoda není původně určena k načítání JS, nicméně jak už jsem uvedl, Adminer nabízí předem definované metody které je možné v rámci pluginu použít a tato metoda splňovala to, že se volá na každé stránce právě jednou.

O načtení cizích klíčů se stará metoda **AdminerTreeView.rowDescriptions**, kterou Adminer volá při vykreslování konkrétní tabulky s daty. Tato metoda se postará o získání všech cizích klíčů pro celou databázi, serializuje je do formátu JSON, ten vloží jako hodnotu do tagu <meta name="foreign-keys" /> a vypíše jej na stránku vedle tabulky. Tato metadata jsou pak k dispozici JS části pluginu.

```
2  [
3    {
4      "sourceTable": "employee",
5      "sourceColumns": [
6        "address"
7      ],
8      "targetTable": "address",
9      "targetColumns": [
10       "id"
11     ]
12   },
13   {
14     "sourceTable": "employee",
15     "sourceColumns": [
16       "workplace"
17     ],
```

Obrázek 10: Ukázkový úryvek de-serializovaných dat uložených do meta tagu foreign-keys.

2.2.2 JS část

Jde o soubor **src/script.ts**, který je napsaný v jazyce TypeScript a následně kompilovaný do jazyka JavaScript. Jedná se o hlavní část pluginu. Obsahuje třídu **AdminerAjaxConnector** která se stará o komunikaci se serverovou částí Admineru pomocí technologie AJAX. Dále třídu **AdminerTreeView**, ta se stará o rozhraní s uživatelem. Vykresluje modální okno a výběry dat v modálním okně a zpracovává uživatelovy příkazy. Třída **HtmlGenerator** je pomocná třída která se stará se o konstrukci komplexnějších HTML struktur.

2.2.3 Struktury pro výběry dat

V JS části jsou definované dvě struktury pro popis a přenos výběru dat. Struktury bez vlastní logiky určené pouze pro definici a přenos dat a někdy bývají označované pojmem DTO (Data Transfer Object). Jedná se o třídy **SelectionQuery** a **SelectionData**. Uvnitř kódu jsem používal výraz Selection pro označení toho, čemu jsem v uživatelském pohledu nazýval výběrem dat. Tedy něco co reprezentuje výběr několika záznamů z konkrétní tabulky databáze.

```
1
2   /** DTO containing information about selection we want to get from server **/
3   class SelectionQuery {
4
5       /** name of table */
6       tableName: string = '';
7
8       /** associative array of columnName: columnValue of conditions to be used on selection */
9       whereConditions: object = {};
10  }
11
12  /** DTO with information about table content and foreign keys */
13  class SelectionData {
14
15      /** array of names of columns */
16      headers: string[] = [];
17
18      /** array of rows, each containing associative array of columnName: columnValue */
19      body: object[] = [];
20
21      /** associative array of columnName: foreignKey. For each column up to one foreign key */
22      directForeignKeys: object = {};
23
24      /** associative array of columnName: arrayOfReverseForeignKeys. For each column can be multiple foreign keys */
25      reverseForeignKeys: object = {};
26  }
```

Obrázek 11: Struktura tříd SelectionQuery a SelectionData.

Obě struktury reprezentují stejné záznamy v databázi ale v jinou formou. **SelectionQuery** je reprezentuje pomocí dotazu kterým záznamy můžeme získat (tabulka ve které se data nacházejí a sada podmínek kterými je můžeme v tabulce najít). Naproti tomu **SelectionData** je reprezentuje výpisem konkrétních záznamů, názvů jednotlivých sloupců, hodnot jednotlivých polí každého záznamu a dále informacemi o přímých a zpětných cizích klíčů, které se týkají sloupců zvoleného výběru dat.

2.2.4 Komunikace se serverem

Zmíněné struktury používá třída **AdminerAjaxConnector** pro komunikaci s ostatními třídami. Jedná se o třídu, která má na starosti komunikaci JS aplikace se serverovou částí Admineru. O získávání dat se stará metoda **getSelectionData(selectionQuery, callback)** která má dva parametry a zařizuje asynchronní získání dat pro konkrétní výběr dat. Prvním parametrem je struktura **SelectionQuery** ve které je definováno jaká data chceme získat a druhým je callback, tedy metoda, která se zavolá po získání dat a jejich převedení na strukturu **SelectionData**.

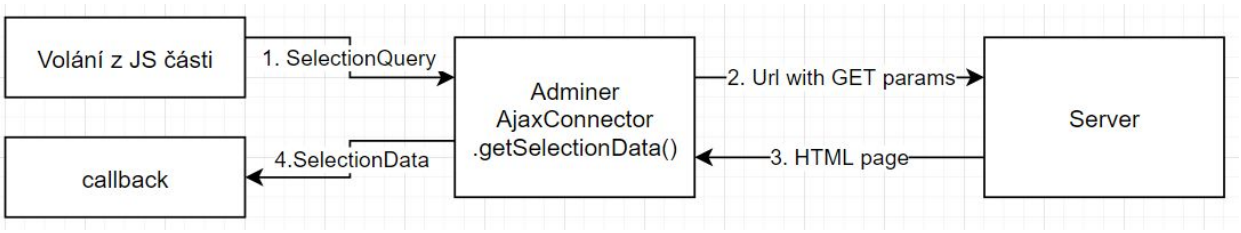


Diagram 12: Průběh volání metody AdminerAjaxConnector.getSelectionData.

Jak je vidět z diagramu 12, ke komunikaci mezi **AdminerAjaxConnector** a serverem dochází pomocí standardního HTTP dotazu který získá HTML stránku. Bohužel, vrácená HTML stránka obsahuje spoustu nepotřebných dat, které nejsou pro vytvoření objektu **SelectionData** nezbytné (menu, záhlaví stránky, zápatí, všechno HTML okolo tabulky atd.). Pokoušel jsem se najít způsob, jak udělat, aby Server vracel pouze nezbytná data, ale kvůli omezením rozhraní, které Adminer nabízí pro vývoj uživatelských pluginů se mi nepodařilo najít způsob, jak toho dosáhnout.

Jedna cesta k předávání dat bez nepotřebného balastu která vypadala velmi slibně byla pomocí úpravy kódu, který donačítal záznamy do tabulky v případě, že bylo záznamů více než kolik se vešlo na stránku (viz obrázek 13). Po stisknutí odkazu uživatelem došlo získání dat pomocí varianty technologie AJAX, bez znovunačtení stránky, a přestože server vrací HTML a ne JSON (což by bylo optimální), tak dojde jen k přenosu záznamů v tabulce a ne celé HTML stránky.

Obrázek 13: Načítání dalších záznamů ve vícestránkovém výpisu.

Nicméně problémem v použití tohoto přístupu bylo, že toto donačítávání dat počítá s tím, že na stránce již existuje hlavička tabulky s názvy sloupců a kód který tato data generuje nemá přístup k názvům sloupců. Pouze vygeneruje příslušné řádky tabulky, pošle je zpět JS stránce Admineru která již sloupce má a ta je jednoduše vloží do existující tabulky na konec.

A jelikož je pro stromové procházení pro každé načtení nového výběru dat potřebuje i informace o názvu tabulky a názvu sloupců aby mohlo dojít k vygenerování hlavičky tabulky a zpracování cizích klíčů a v kódu se mi nepovedlo najít způsob jak zmíněné volání obohatit o názvy sloupců, tak jsem nemohl tento způsob získávání výběru dat použít.

Z tohoto důvodu jsem nakonec pro načítání výběru dat musel použít takový postup, kdy pomocí JS požádám server o celou HTML stránku, která v sobě obsahuje i tabulku s potřebnými daty a na této stránce za pomoci regulárního výrazu najdu očekávanou tabulku. Tu dále pomocí regulárního výrazu pročistím od všech pomocných dat které do ní adminer vkládá a dále pracuji se vzniklým HTML kódem tabulky. Z tohoto kódu vytáhnu meta tag s cizími klíči, který do tabulky vkládá PHP částu pluginu a ten se dále používá pro zpracování přímých a zpětných cizích klíčů. Zbývající část HTML kódu tabulky je naparsována a převedena na DOM elementy a pomocí JS jsou z nich vytažena všechna data potřebná k sestrojení **SelectionData** objektu.

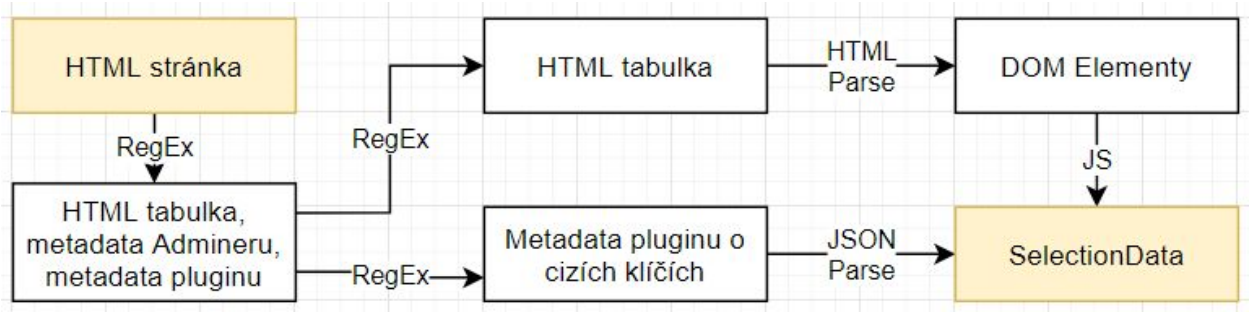


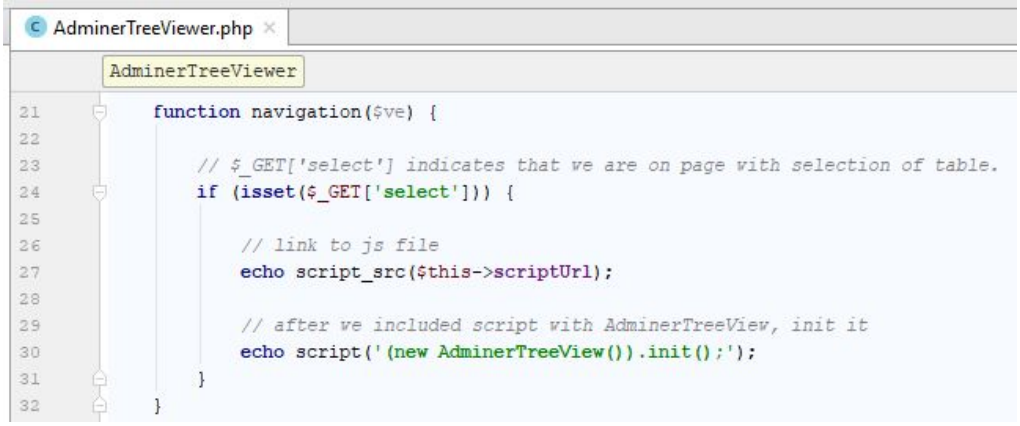
Diagram 14: Schéma konverze HTML stránky do objektu SelectionData.

Bohužel, tento způsob komunikace se serverem není zdaleka optimální. Kvůli načítání celé stránky dochází k mnohonásobně většímu přenosu dat než je nezbytně nutné a kvůli hledání tabulky pomocí regulárního výrazu může dojít v budoucnu k rozbití kompatibility pluginu a Admineru v případě že se Adminer rozhodne na stránku umístit více tabulek nebo změnit strukturu tabulky. Nicméně se mi nepodařilo najít lepší způsob, jak obejít omezení rozhraní Admineru nabízeného pro uživatelské pluginy a dosáhnout původního záměru, kdy jsou všechna potřebná data připravená na serveru pomocí tohoto pluginu a zaslána ve formátu JSON, čímž by se předešlo jednak velkému přenosu dat, jednak závislosti na formě kterou si určuje Adminer.

Na druhou stranu díky tomu, že se o práci s HTML stránkou stará pouze jedna třída **AdminerAjaxConnector**, je zbytek kódu od těchto problémů odstíněn a pro jakékoliv změny kompatibility bude v budoucnu stačit úprava tohoto kousku pluginu a ostatní kód pracující se strukturami **SelectionQuery** a **SelectionData** nebude vyžadovat žádnou změnu.

2.2.5 Uživatelské rozhraní

O uživatelské rozhraní se starají dvě třídy, **AdminerTreeView** a **HtmlGenerator**. **AdminerTreeView** se stará o přípravu uživatelského rozhraní, vykreslování HTML, zobrazování tabulek a správu událostí. Vstupním bodem je metoda init(), která se volá po vykreslení stránky.



Obrázek 15: Snippet z PHP části který se stará o spuštění AdminerTreeView.init() v JS části.

Metoda Init je spuštěna pouze na stránkách, které obsahují datové tabulky, tedy tabulky se záznamy, pro které je možné otevřít stromové procházení. Ty poznáme pomocí GET parametru **select** v url. Po spuštění si metoda najde datovou tabulku a na konec každého řádku přidá odkaz **view** pro zobrazení modálního okna pro stromové procházení.

Modify	id	name	address	manager	Tree
edit	2	Great library of Atlantis city	2	3	view
edit	3	Quicksilver town little library	3	6	view

*Obrázek 16: Sloupec **Tree** je generovaný pomocí JS dynamicky až po načtení stránky.*

Po stisknutí na odkaz **view** dojde ke spuštění události zobrazení modálního okna. Nejprve dojde ke kontrole, zda už modální okno na stránce existuje z předchozího volání a pokud ne, tak se vytvoří zcela nové prázdné modální okno a vloží se na konec těla dokumentu. Pokud již existuje, tak se smaže jeho minulý obsah. V této fázi je zajištěno, že modální okno obsahuje css vlastnost **display: none;** aby uživatel neviděl data z minulého volání (což by se mohlo stát kdyby měl například pomalé připojení k internetu a modální okno zatím nenačetlo nová data).

Po zajištění prázdného a zatím pro uživatele neviditelného modálního okna si aplikace najde odkaz **edit** na začátku zvoleného řádku. Z odkazu **edit** si vytáhne url pro editaci, a z parametrů tohoto url zjistí název tabulky a podmínky potřebné pro získání zvoleného záznamu.

Například z dané url je možné zjistit, že jde o editovanou tabulku **library** a záznam s **id = 2**: http://localhost/adminer-plugin/demo/?username=root&db=library2&edit=library&where[id]=2. Tyto informace zjistíme z parametrů **edit** a **where** a použijí se k sestavení DTO struktury **SelectionQuery**. Po přípravě DTO pro získání dat dojde k zobrazení prázdného modálního okna a DTO se předá metodě **AdminerAjaxConnector.getSelectionData** pro získání dat z Admineru.

Jako callback se používá metoda **AdminerTreeView**.**openSelectionIntoContainer** která umí vykreslit obsah struktury **SelectionData** do HTML a umístit je do předem určeného HTML elementu, v tomto případě do elementu pro obsah modálního okna. Vygenerovaný HTML element se skládá z wrapperu a dvou samostatných částí: vlastní tabulky se záznamy výběru dat a prostorem pro další podvýběry dat.

Do tabulky se záznamy výběru dat se přidají odkazy s různými akcemi jak bylo uvedeno v kapitole o uživatelském pohledu na plugin. Každá akce, která pomocí přímých nebo zpětných cizích klíčů vytváří nové podvýběry dat použije pro načtení těchto dat stejný postup jako prvotní výběr dat, tedy zavolání metody **AdminerAjaxConnector.getSelectionData** a předání callback metody **AdminerTreeView**.**openSelectionIntoContainer**, akorát na rozdíl od prvotního volání není cílem umístění nového výběru dat kořen modálního okna, ale cílem pro umístění nově vzniklé tabulky je prostor ve volajícím výběru dat, který je určený pro podvýběry dat. Díky tomu se podvýběry zanořují jeden do druhého podle hierarchie volání a díky odsazení podvýběrů je vizuálně vidět který podvýběr dat patří kterému výběru dat.

Pomocnou třídou pro generování větších kusů kódu je třída **HtmlGenerator**. Má dvě hlavní metody, jednu pro vygenerování modálního okna a druhou pro vygenerování výběru dat. Obě metody vrací DOM model. Kromě správných elementů v DOM modelu se tato třída stará i o nahrazení dynamických hodnot, například naplnění tabulky výběru dat konkrétními daty nebo uvedení názvu tabulky a podmínek do záhlaví výběru dat.

Kromě zmíněných úkonů také tato třída nastaví na některé elementy obecnější event handlery, které jsou platné jen pro generovaný prvek, například pro modální okno naváže event handler který po kliknutí na akci **Close** modální okno zavře. Tato událost nemá vliv nikde mimo modální okno (tedy vytvářený element) a tudíž můžeme událost přehledně vytvořit uvnitř třídy **HtmlGenerator**. O navázání složitějších událostí které mají vliv i mimo vytvořený objekt se postará třída **AdminerTreeView**, která pomocnou třídu **HtmlGenerator** používá. Příkladem je akce **Close** u výběru dat, kde platí že zavřením posledního výběru dat v modálním okně dojde i k zavření celého modálního okna. Tedy akce **Close** u výběru dat může mít vliv i na modální okno, které je vně generovaného elementu výběru dat.

2.2.6 Kompilace zdrojových kódů

V kořenovém adresáři je podadresář ./src ve kterém se nachází PHP část pluginu a TypeScript část. PHP část je možné nasadit na server ve stavu v jakém je, jelikož Adminer je napsaný v PHP a pracuje se zde přímo se zdrojovými kódy, ale TypeScript je potřeba zkompilovat do JavaScript souboru.

TypeScript (TS) je jazyk velmi podobný JavaScriptu (JS), ale na rozdíl od JS dává vývojářům možnost definovat si u jednotlivých proměnných o jaký typ proměnné se jedná. Díky tomu je možná strojová kontrola datových typů. Tím lze předejít chybám, které může vývojář zanést do kódu při práci s proměnnými. Další předností TS je možnost jasné definice tříd a jejich vlastností a zajištění typové kontroly těchto vlastností. To je výhodou oproti oproti JS, ve kterém jsou vlastnosti definované za běhu v konstruktoru nebo i později v jednotlivých metodách.

Kompilace TS je možná několika způsoby, například vložením do online formuláře na domovském webu typescriptu, ale v rámci této práce jsem využíval kompilaci pomocí typescript nástroje v NPM.

NPM (NodeJs Package Manager) je správcem balíčků pro Node.js, který umožňuje stahovat balíčky a přidávat je do projektů, nebo stahovat nástroje a spouštět je, jako právě v tomto případě. Použití nástroje pro kompilaci TS souboru je vidět v kompilačním skriptu (Obrázek 17) na řádku 6. Příkaz **npx tsc** je spuštění NPM nástroje **typescript compiler**.

compile.bat

```
1  REM # prepare dist folder
2  @RD /S /Q "dist"
3  mkdir dist
4
5  REM # compile dist folder content
6  CMD /C npx tsc --outFile dist\script.js src\script.ts
7  copy src\AdminerTreeViewer.php dist\AdminerTreeViewer.php
8
9  REM # copy to demo
10 RD /S /Q "demo\AdminerTreeViewer"
11 mkdir "demo\AdminerTreeViewer"
12 copy dist\AdminerTreeViewer.php demo\AdminerTreeViewer\AdminerTreeViewer.php
13 copy dist\script.js demo\AdminerTreeViewer\script.js
```

Obrázek 17: Bat soubor, který se stará o kompilaci TS a přípravu složek dist a demo.

Zkompilovaný JS soubor je už možné použít v rámci pluginu do Admineru.

Ukázkový skript (Obrázek 17) pro vyšší pohodlí připravuje složku ./dist která obsahuje všechny soubory potřebné pro nasazení pluginu a zároveň tyto výsledné soubory rovnou nasadí do složky ./demo, která obsahuje všechny potřebné soubory a konfiguraci pro spuštění admineru společně se zkompilovaným pluginem.

2.2.7 Deploy zkompilovaných souborů

Pro zapojení pluginu do admineru potřebujeme nejprve připravit prostředí, které načte plugin a předá jej Admineru. Takto vypadá soubor index.php s minimální potřebnou konfigurací:

```
1   <?php
2   function adminer_object() {
3       // Adminer customization allowing usage of plugins
4       require_once "./plugin.php";
5       require_once "./AdminerTreeViewer/AdminerTreeViewer.php";
6
7       return new AdminerPlugin([
8           new AdminerTreeViewer("AdminerTreeViewer/script.js")
9       ]);
10  }
11
12  // include original Adminer or Adminer Editor
13  include './adminer-4.7.4.php';
```

Obrázek 18: Ukázka souboru index.php, ve které je vidět minimální konfigurace potřebná pro zapojení pluginu AdminerTreeViewer.

“Konfigurací” je spustitelný PHP soubor, který spustí Adminer společně s pluginy.

Tento soubor musí mít nadefinovanou funkci **adminer_object**, která vrací instanci objektu **AdminerPlugin** s polem instancí jednotlivých pluginů, které chceme v Admineru mít.

Začneme tedy vytvořením souboru index.php ve kterém vytvoříme funkci **adminer_object** (řádek 2). Uvnitř této metody načteme soubor **plugin.php** (řádek 3), který obsahuje implementaci třídy **AdminerPlugin**. Jedná se o pomocnou třídu pro zapojení pluginů do Admineru a soubor s touto třídou je k dispozici ke stažení na oficiálních stránkách Admineru s sekci o pluginech.

Dále načteme soubor **AdminerTreeView.php** (řádek 5), který je součástí této práce (viz předchozí kapitola Kompilace zdrojových souborů).

Po načtení zdrojových kódů můžeme vytvořit instanci třídy **AdminerPlugin** která bude zároveň návratovou hodnotou funkce **adminer_object** (řádek 7).

Jako parametr konstruktoru objektu **AdminerPlugin** předáme pole všech pluginů, které chceme do aplikace Adminer zapojit, v této ukázce se jedná pouze o instanci třídy **AdminerTreeView** (řádek 8). Tato třída potřebuje znát url zkompilovaného JS souboru, který načte. URL může být buď relativní vzhledem k url admineru, nebo absolutní. V této ukázce je předané relativní url.

A nakonec, stačí spustit standardní soubor s aplikací Adminer (řádek 13).

3 Vytváření pluginů do aplikace Adminer

3.1 Princip pluginů

Adminer (obyčejným písmem budu označovat aplikaci) má nadefinovanou a implementovanou třídu **Adminer** (tučným písmem budu označovat třídu nebo její instanci), která má celou řadu metod používaných napříč aplikací Adminer. Tento objekt **Adminer** je možné uvnitř Admineru nahradit jiným objektem s jinou funkcionalitou. To se stane tak, že bude za běhu Admineru existovat funkce **adminer_object** (viz ukázka v předchozí kapitole, Deploy zkompilovaných souborů), která vrací alternativní objekt k objektu **Adminer**. A právě předáním tohoto objektu pomocí funkce **adminer_object** a tím, že nahradí původní objekt **Adminer**, začne aplikace volat jeho metody místo standardních metod a tím dochází k zapojení pluginu do aplikace Adminer.

```
1054
1055    $adminer = (function_exists('adminer_object') ? adminer_object() : new Adminer);
1056    if ($adminer->operators === null) {
1057        $adminer->operators = $operators;
1058    }
1059
```

Obrázek 19: Úryvek kódu z Admineru, který se stará o nahrazení objektu Adminer.

Tímto způsobem kdy lze objekt **Adminer** nahradit jiným objektem je možné zapojit pouze jeden plugin. Dále, aby objekt který funkce **adminer_object** vrací, mohl nahradit objekt **Adminer** se standardní implementací, musí mít nadefinované všechny veřejné metody jako objekt **Adminer**. Pokud bude funkce **adminer_object** vracet objekt u kterého některé metody chybí tak se aplikace spustí, ale jakmile dojde k volání některé nedefinované metody, dojde k pádu aplikace.

A pro vyřešení těchto problému nabízí Adminer na svých domovských stránkách ke stažení třídu **AdminerPlugin**, která slouží ke zjednodušení psaní pluginů. Je určena k tomu, aby byla návratovou hodnotou funkce **adminer_object** a tím pádem právě tím jedním pluginem, který nahradí standardní chování.

Sama o sobě tato třída nepřidává žádnou funkcionalitu, ale působí jako prostředník mezi uživatelskými pluginy a aplikací Adminer a je schopna zprostředkovat funkcionalitu více pluginům najednou. Především, je potomkem třídy **Adminer** a tím pádem dědí všechny metody které má samotná třída **Adminer**. Díky tomu nemůže dojít k pádu aplikace z důvodu chybějící metody. V konstruktoru vyžaduje pole pluginů, jejichž funkcionalitu má Admineru zprostředkovávat. Po předání pole pluginů se postará o to, že pokud pluginy definují nějaké metody se stejným názvem jako některé z metod **Admineru**, použijí se implementace z pluginů místo použití standardní implementace z **Admineru**.

```php
function adminer_object() {
    // Adminer customization allowing usage of plugins
    require_once "./plugin.php";
    require_once "./AdminerTreeViewer/AdminerTreeViewer.php";

    return new AdminerPlugin([
        new AllowEmptyPasswordPlugin(),
        new AdminerTreeViewer("AdminerTreeViewer/script.js")
    ]);
}
```

Obrázek 20: Ukázka použití instance třídy AdminerPlugin ve funkci adminer_bject.

Implementace třídy **AdminerPlugin** má několik částí. Nejprve se jedná o konstruktor, který převezme pole pluginů a jediné co udělá je, že je uloží do proměnné pro pozdější použití. Dále jsou v této třídě implementované dvě pomocné metody, **_applyPlugin** a **_appendPlugin**. Obě metody očekávají jako argument název některé z původních metod třídy Adminer a pole parametrů. Poslední částí třídy **AdminerPlugin** je přetížení všech metod z původní třídy Adminer novou implementací.

Metoda **_applyPlugin** prochází postupně pluginy, které má třída AdminerPlugin uložené a pro každý z nich zkontroluje, zda má implementovanou metodu uvedenou jako argument. Pokud ano, dojde k jejímu zavolání a návratová hodnota z pluginu se pošle jako návratová hodnota metody **_applyPlugin**. Pokud má několik pluginů implementovanou hledanou metodu, tak dojde ke spuštění pouze první z nich. Pořadí procházení pluginů je dáno pořadím pluginů v poli, které se předává v konstruktoru třídy **AdminerPlugin**. Pokud by v ukázce na Obrázku 20 měly obě třídy **AllowEmptyPasswordPlugin** a **AdminerTreeViewer** implementovanou stejnou funkci, došlo by pouze ke spuštění implementace ze třídy **AllowEmptyPasswordPlugin**. V případě, že ani jeden plugin požadovanou metodu nemá implementovanou, zavolá se implementace z rodiče třídy **AdminerPlugin**, kterou je třída **Adminer**.

```php
function _applyPlugin($function, $args) {
    foreach ($this->plugins as $plugin) {
        if (method_exists($plugin, $function)) {
            switch (count($args)) { // call_user_func_array() doesn't work well with references
                case 0: $return = $plugin->$function(); break;
                case 1: $return = $plugin->$function($args[0]); break;
                case 2: $return = $plugin->$function($args[0], $args[1]); break;
                case 3: $return = $plugin->$function($args[0], $args[1], $args[2]); break;
                case 4: $return = $plugin->$function($args[0], $args[1], $args[2], $args[3]); break;
                case 5: $return = $plugin->$function($args[0], $args[1], $args[2], $args[3], $args[4]); break;
                case 6: $return = $plugin->$function($args[0], $args[1], $args[2], $args[3], $args[4], $args[5]); break;
                default: trigger_error('Too many parameters.', E_USER_WARNING);
            }
            if ($return !== null) {
                return $return;
            }
        }
    }
    return $this->_callParent($function, $args);
}
```

Obrázek 21: Implementace metody _applyPlugin.

```
93      function name() {
94          $args = func_get_args();
95          return $this->_applyPlugin(__FUNCTION__, $args);
96      }
97
98      function credentials() {
99          $args = func_get_args();
100         return $this->_applyPlugin(__FUNCTION__, $args);
101     }
102
103     function connectSsl() {
104         $args = func_get_args();
105         return $this->_applyPlugin(__FUNCTION__, $args);
106     }
107
108     function permanentLogin($create = false) {...}
112
113     function bruteForceKey() {...}
117
118     function serverName($server) {...}
122
123     function database() {...}
127
128     function schemas() {...}
```

~~~~~~

```
378     function importServerPath() {...}
382
383     function homepage() {...}
387
388     function navigation($missing) {
389         $args = func_get_args();
390         return $this->_applyPlugin(__FUNCTION__, $args);
391     }
392
393     function databasesPrint($missing) {
394         $args = func_get_args();
395         return $this->_applyPlugin(__FUNCTION__, $args);
396     }
397
398     function tablesPrint($tables) {
399         $args = func_get_args();
400         return $this->_applyPlugin(__FUNCTION__, $args);
401     }
402
403 }
```

Obrázek 22: Implementace metod zděděných ze třídy Adminer ve třídě AdminerPlugin. Až na výjimky, všechny tyto metody používají totožné volání metody _applyPlugin.

Metoda **_appendPlugin** má podobnou funkcionalitu jako metoda **_applyPlugin**, ale na rozdíl od ní nenahrazuje původní implementaci ze třídy Adminer, ale doplňuje ji. Nejprve dojde k zavolání původní implementace metody z rodičovské třídy **Adminer** a uložení výsledku do proměnné. Dále, podobně jako u metody **_applyPlugin**, dojde k postupnému procházení všech pluginů a kontrole, zda mají vlastní implementaci hledané metody. U každého pluginu u kterého je implementace nalezená se tato implementace zavolá a její výsledek se připojí na konec výstupu uloženého v proměnné s výstupem. Nalezením pluginu s implemnetovanou metodou nedojde k ukončení hledání, ale po uložení výstupu dojde k hledání v dalších pluginech. Výstupem metody **_appendPlugin** je výstup ze třídy Adminer obohacený o výstup ze všech
~~~~~~

pluginů. Její volání se používá jen v několika metodách u kterých nemá smysl nahrazovat původní funkcionalitu třídu **Adminer**, ale má smysl ji rozšířit.

```php
function _appendPlugin($function, $args) {
    $return = $this->_callParent($function, $args);
    foreach ($this->plugins as $plugin) {
        if (method_exists($plugin, $function)) {
            $value = call_user_func_array(array($plugin, $function), $args);
            if ($value) {
                $return += $value;
            }
        }
    }
    return $return;
}

// appendPlugin

function dumpFormat() {
    $args = func_get_args();
    return $this->_appendPlugin(__FUNCTION__, $args);
}

function dumpOutput() {
    $args = func_get_args();
    return $this->_appendPlugin(__FUNCTION__, $args);
}

function editFunctions($field) {
    $args = func_get_args();
    return $this->_appendPlugin(__FUNCTION__, $args);
}
```

Obrázek 23: Implementace metody _appendPlugin a její použití.

Pro příklad, metoda **dumpFormat** v původní implementaci vrací pole formátů, do kterého umí Adminer exportovat data (SQL, CSV a TSV). Plugin, který by přidával další formát pro export dat nepotřebuje odebírat existující formáty, pouze přidat další. Proto u metody **dumpFormat** dává větší smysl obohacovat původní implementaci o nové prvky než ji nahradit za zcela novou implementací a proto je v tomto případě použita metoda **_appendPlugin**, nikoliv **_applyPlugin**.

3.2 Životní cyklus HTTP dotazu Admineru

Adminer je v produkčním prostředí pouze jeden soubor, ale pro ukázku životního cyklu zde budu používat zdrojové kódy před kompilací, ve kterých je celý adminer pro přehlednost rozdělený do mnoha samostatných souboru.

Vstupním bodem pro volání Admineru je soubor index.php sloužící jako rozcestník, který podle HTTP parametrů určí jakou funkcionalitu HTTP požadavek vyžaduje.

```
16  if (isset($_GET["select"]) && ($_POST["edit"] || $_POST["clone"]) && !$_POST["save"]) {
17      $_GET["edit"] = $_GET["select"];
18  }
19  if (isset($_GET["callf"])) {
20      $_GET["call"] = $_GET["callf"];
21  }
22  if (isset($_GET["function"])) {
23      $_GET["procedure"] = $_GET["function"];
24  }
25
26  if (isset($_GET["download"])) {
27      include "./download.inc.php";
28  } elseif (isset($_GET["table"])) {
29      include "./table.inc.php";
30  } elseif (isset($_GET["schema"])) {
31      include "./schema.inc.php";
32  } elseif (isset($_GET["dump"])) {
33      include "./dump.inc.php";
```

Obrázek 24: Ukázka rozcestníku funkcionalit

Jak je vidět v ukázce na Obrázku 24, rozcestník nejprve předpřipraví některá data z GET parametrů a nasledně pomocí jednoduché sady if/else rozhodne o tom, jaká funkcionalita je požadována. Každá funkcionalita je implementována v samostatném souboru. Např. soubor **dump.inc.php** se stará o export databáze, soubor **table.inc.php** se stará o vypsání a práci se strukturou tabulky a soubor **select.inc.php** se stará o zobrazení dat z tabulky a práci s daty (a právě funkcionalita z tohoto souboru je obohacena pluginem vyvinutým v rámci této diplomové práce).

Použití této implementace rozcestníku je velmi jednoduché a rozhodování tímto způsobem je rychlé. Nicméně tato implementace má i své nevýhody a jednou z nich je neschopnost rozšíření pomocí pluginů. Právě z tohoto důvodu jsem nemohl plugin navrhnout jako samostatnou, novou funkcionalitu, návrhově podobnou ostatním funkcionalitám Admineru (tedy napsanou pouze v jazyce PHP, funkcionalita odlišena vlastním GET parametrem s doplňujícími GET parametry určujícími akce v rámci funkcionality), ale musel jsem použít velmi odlišný přístup ve kterém jsem napsal novou funkcionalitu v JS a pomocí PHP části ji vložil do existující stránky.

Po rozhodnutí toho, o jakou funkcionalitu se jedná, pokračuje běh programu do odpovídajícího souboru. Jednotlivé funkcionality jsou implementované převážně jako sada příkazů, které jsou větvené pomocí if/else podmínek. Často jsou v těchto souborech implementované pomocné funkce, které se v rámci běhu programu volají. Pokud je v rámci funkcionality potřeba vygenerovat HTML stránku nebo HTML kus kódu, výpis probíhá přímo v běhu programu a není od hlavního běhu programu nijak oddělen.

```
307     if (!$rows) {
308         echo "<p class='message'>" . lang('No rows.') . "\n";
309     } else {
310         $backward_keys = $adminer->backwardKeys($TABLE, $table_name);
311
312         echo "<div class='scrollable'>";
313         echo "<table id='table' cellspacing='0' class='nowrap checkable'>";
314         echo script("mixin(qs('#table'), {onclick: tableClick, ondblclick: partialArg(tableClick, true), onkeydown: e
315         echo "<thead><tr>" . (!$group && $select
316             ? ""
317             : "<td><input type='checkbox' id='all-page' class='jsonly'>" . script("qs('#all-page').onclick = partial(
318                 . " <a href='" . h($_GET["modify"] ? remove_from_uri("modify") : $_SERVER["REQUEST_URI"] . "&modify=1
319         $names = array();
320         $functions = array();
321         reset($select);
322         $rank = 1;
323         foreach ($rows[0] as $key => $val) {
324             if (!isset($unselected[$key])) {
325                 $val = $_GET["columns"][key($select)];
326                 $field = $fields[$select ? ($val ? $val["col"] : current($select)) : $key];
327                 $name = ($field ? $adminer->fieldName($field, $rank) : ($val["fun"] ? "*" : $key));
328                 if ($name != "") {
329                     $rank++;
330                     $names[$key] = $name;
331                     $column = idf_escape($key);
332                     $href = remove_from_uri('(order|desc)[^=]*|page') . '&order%5B0%5D=' . urlencode($key);
333                     $desc = "&desc%5B0%5D=1";
334                     echo "<th>" . script("mixin(qsl('th'), {onmouseover: partial(columnMouse), onmouseout: partial(co
335                     echo '<a href="' . h($href . ($order[0] == $column || $order[0] == $key || (!$order && $is_group
336                     echo apply_sql_function($val["fun"], $name) . "</a>"; //! columns looking like functions
337                     echo "<span class='column hidden'>";
338                     echo "<a href='" . h($href . $desc) . "' title='" . lang('descending') . "' class='text'> ↓</a>";
339                     if (!$val["fun"]) {
340                         echo '<a href="#fieldset-search" title="' . lang('Search') . '" class="text jsonly"> =</a>';
341                         echo script("qsl('a').onclick = partial(selectSearch, '" . js_escape($key) . "');");
342                     }
343                     echo "</span>";
344                 }
345                 $functions[$key] = $val["fun"];
346                 next($select);
347             }
348         }
349
```

Obrázek 25: Ukázkový úryvek ze souboru select.inc.php.

Objekty používá vnitřní část Admineru, která slouží jako knihovna funkcí pro jednotlivé soubory se samostatnými funkcionalitami. Už jsem zmiňoval třídu Adminer. Další jsou například třídy které obstarávají funkcionalitu databázových konektorů.

```
1     <?php
2     $drivers["pgsql"] = "PostgreSQL";
3
4     if (isset($_GET["pgsql"])) {
5         $possible_drivers = array("PgSQL", "PDO_PgSQL");
6         define("DRIVER", "pgsql");
7         if (extension_loaded("pgsql")) {
8             class Min_DB {...}
111
112           class Min_Result {...}
145
146       } elseif (extension_loaded("pdo_pgsql")) {
147           class Min_DB extends Min_PDO {...}
184
185       }
186
187
188
189       class Min_Driver extends Min_SQL {
190
191           function insertUpdate($table, $rows, $primary) {...}
210
211           function slowQuery($query, $timeout) {...}
216
217           function convertSearch($idf, $val, $field) {...}
225
226           function quoteBinary($s) {...}
229
230           function warnings() {
231               return $this->_conn->warnings();
232           }
233
```

Obrázek 26: Ukázka souboru pgsql.inc.php který se stará o konektor na PostgreSQL.

Podle mého názoru je výhodou tohoto způsobu implementace rychlost. Při každém HTTP dotazu je běh programu vedený do samostatné větve if/else ve které je přesně definováno, co daná funkcionalita potřebuje pro běh a dochází k minimální potřebné definici funkcí nebo inicializaci objektů.

Nicméně cenou za tuto rychlost je snížená přehlednost kódu (např. množství vnořených if/else, generování HTML přímo v běhu PHP, místo použití šablon, atd.) a dále také omezená možnost rozšiřování aplikace pomocí pluginů.

Závěr

V rámci této diplomové práce jsem vytvořil plugin do aplikace Adminer. Plugin nabízí uživatelům nový způsob zobrazení dat, tzv. stromové procházení dat. Výhodou tohoto způsobu zobrazení je přehlednost souvisejících dat napříč více tabulkami.

Předností této práce mělo být vytvoření nástrojem, který bude možné používat v reálném provozu. Toho bylo dosaženo rozšířením již existující a v provozu běžně používané aplikace Adminer a dodržením UX a UI stylu používaného v Admineru.

Druhotným cílem který tato práce splňuje je prozkoumání zdrojového kódu aplikace Adminer a způsobu, kterým pracuje s doplňky. V této práci jsem sepsal poznatky které ke kterým jsem během procházení zdrojového kódu došel. Zdrojový kód Admineru je určený především pro rychlost běhu aplikace, ale cenou za to je mimo jiné omezení toho, jaké části Admineru je možné rozšířit o pomocí doplňků.

Zdroje

Adminer homepage [online]. [cit. 2021-01-04].
Dostupné z: https://www.adminer.org/cs/

Oficiální stránky aplikace Adminer. Jsou zde k dostání informace o Admineru a jeho vlastnostech a odkazy pro stažení.

Adminer - Rozšíření [online]. [cit. 2021-01-04].
Dostupné z: https://www.adminer.org/cs/plugins/
Informace pro vývojáře Adminer pluginů. Obsahuje postupy pro vytváření a zapojení uživatelsky vytvořených pluginů, které byly zapotřebí pro vytvoření pluginu v této práci.

Github - vrana/adminer [online]. [cit. 2021-01-04].
Dostupné z: https://github.com/vrana/adminer/
Repozitář se zdrojovými kódy aplikace Adminer. V rámci této práce byly nezbytné pro prozkoumání způsobu, jakým Adminer pracuje s pluginy.

MDN Web Docs, JavaScript [online]. [cit. 2021-01-04].
Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript
Dokumentace k jazyku JavaScript. Používal jsem tento zdroj jako příručku implementaci JS části pluginu.

W3schools, ECMAScript 2015 - ES6 [online]. [cit. 2021-01-04].
Dostupné z: https://www.w3schools.com/js/js_es6.asp
Návody a ukázky použití některých komponent v jazyku JavaScript. Používal jsem tento zdroj jako příručku implementaci JS části pluginu.

TypeScript homepage [online]. [cit. 2021-01-04].
Dostupné z: https://www.typescriptlang.org/
Domovská stránka jazyku TypeScript. Nabízí ke stažení kompilátor TypeScript souborů, dokumentaci jazyka a ukázky zdrojových kódů.

Přílohy

Seznam přiložených souborů

Složky jsou označené tučným písmem, soubory obyčejným písmem

- **demo** — **Ukázka zapojení pluginu do admineru**
  - **AdminerTreeViewer** — **Výstupní soubory tohoto pluginu**
    - AdminerTreeViewer.php — PHP část pluginu
    - script.js — JS část pluginu
  - index.php — Vstupní PHP soubor s konfigurací pluginu
  - plugin.php — Třída pro zapojení pluginů Admineru
  - adminer-4.7.4.php — Soubor s nezměněnou verzí Admineru
  - library2.sql — SQL skript obsahující ukázkovou databázi
- **dist** — **Zkompilované soubory pluginu**
  - AdminerTreeViewer.php — PHP část pluginu
  - script.js — JS část pluginu
- **src** — **Zdrojové kódy pluginu**
  - AdminerTreeViewer.php — PHP část pluginu
  - script.ts — TypeScript část pluginu
- compile.bat — Kompilační script, generuje dist ze src
- .gitignore — Konfigurace ignorovaných souborů pro GIT

Seznam obrázků a diagramů