F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Vyhodnocování agregačních funkcí v SQL dotazech

Olesia Cheremnykh

Květen 2023
Obor: Softwarové inženýrství a technologie
Vedoucí práce: RNDr. Ingrid Nagyová, Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Cheremnykh** Jméno: **Olesia** Osobní číslo: **498952**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Vyhodnocování agregačních funkcí v SQL dotazech

Název bakalářské práce anglicky:

Evaluation of aggregate functions in SQL queries

Pokyny pro vypracování:

Cílem projektu je analyzovat proces vyhodnocování agregačních funkcí v SQL dotazech a navrhnout a implementovat simulátor, který dokáže prezentovat tento proces na množinové reprezentaci dat. Simulátor umožní jednoduché zadání dotazu a vizualizuje proces jeho zpracování. Systém bude prakticky testován ve výuce.
Požadavky projektu:
1. Seznamte se s procesem vyhodnocování SQL dotazů.
2. Analyzujte praktické úkoly, které se v předmětu Databázové systémy probírají. Pro dotazy, které využívají agregační funkce, specifikujte proces zpracování těchto dotazů na množinové reprezentaci dat.
3. Vyhledejte dostupné nástroje pro podporu výuky databázových systémů. Zaměřte se na způsob, jakým jednotlivé nástroje prezentují proces zpracování agregačních funkcí v SQL dotazech.
4. Navrhněte systém pro názornou simulaci vyhodnocování agregačních funkcí v SQL dotazech.
5. Systém implementujte, otestujte a prakticky ověřte ve výuce předmětu Databázové systémy.

Seznam doporučené literatury:

Svoboda, Martin. 2021. „Přednášky z předmětu Databázové systémy.“ Dostupné 31. led. 2023 https://www.ksi.mff.cuni.cz/~svoboda/courses/182-B0B36DBS/
Horcasitas, Jeanelle. 2022. „How To Use Mathematical Expressions and Aggregate Functions in SQL.“ Dostupné 31. led. 2023 https://www.digitalocean.com/community/tutorials/how-to-use-mathematical-expressions-in-sql#analyzing-data-with-aggregate-function
Done, Paul. 2022. „Practical MongoDB Aggregations.“ Dostupné 31. led. 2023 https://www.practical-mongodb-aggregations.com/credits.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ingrid Nagyová, Ph.D. kabinet výuky informatiky FEL

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

Datum zadání bakalářské práce: **12.02.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

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

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studentky

Poděkování / Prohlášení

S hlubokým uznáním a vděčností bych ráda vzdala hold vedoucí mého projektu, paní RNDr. Ingrid Nagyové. Její nespočetné hodiny podpory, významných rad a odborného vedení byly klíčové pro dokončení tohoto projektu. Nesmírně si vážím také času, poznatků a rad všech jednotlivců, kteří se podíleli na této práci. Jejich cenné příspěvky a investovaný čas zásadně ovlivnily výsledek tohoto projektu. Na závěr bych ráda vyjádřila své upřímné poděkování svému partnerovi za jeho morální podporu během těchto náročných časů. Jeho soucit a podpora mi poskytly potřebnou sílu a odhodlání pokračovat, i když jsem se setkala s překážkami.

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

V Praze dne 26. 05. 2023

.....

Abstrakt / Abstract

Tato bakalářská práce se zabývá analýzou metod praktického učení jazyka SQL s využitím vizualizace dat. Zaměřuje se na návrh a implementaci simulátoru s cílem usnadnit studentům porozumění principům SQL a postupům vyhodnocování SQL operací. V teoretické části studie jsou zkoumány základy databází, s důrazem na relační databáze a jazyk SQL. Práce zkoumá problematiku učení SQL pro začátečníky, založenou na kurzu „Databázové systémy“ vyučovaného na ČVUT. Analýza pokrývá základní operace, které jsou v kurzu vyučovány, a identifikuje klíčové operace pro vizualizaci v aplikaci. V rámci výzkumu jsou studovány existující aplikace pro vizualizaci práce s databázemi, jako jsou Relax, Tableau a SQL Academy. Na základě těchto nástrojů jsou vybrány efektivní metody interaktivního učení pro implementaci. Implementace zahrnuje vývoj React aplikace na základě architektury na straně klienta, kde uživatel interaguje s databází pomocí SQL.js. Aplikace umožňuje vytvářet SQL dotazy pomocí interaktivního pole, vizualizovat výsledky dotazů pomocí tabulek a grafických elementů, umožňuje vizuálně zobrazit databázové tabulky importované uživatelem. Testování aplikace zahrnuje jednotkové a integrační testy pomocí Jest a React Testing Library, stejně jako uživatelské testování. Výsledky ukazují, že aplikace je účinná jako nástroj pro výuku SQL, s potenciálem pro další zlepšení na základě uživatelské zpětné vazby.

Klíčová slova: SQL, databáze, vzdělávání, vizualizace dat, webová aplikace

This bachelor's thesis examines the analysis of methods for practical learning of the SQL language through data visualization. It focuses on the design and implementation of a query simulator to facilitate students' understanding of SQL principles and procedures for evaluating SQL operations. The theoretical part of the study examines the fundamentals of databases, with an emphasis on relational databases and the SQL language. The work explores the issues of learning SQL for beginners, based on the „Database Systems“ course taught at CTU. The analysis covers the basic operations taught in the course and identifies key operations for visualization in the application. In the context of research, existing applications for visualizing database work, such as Relax, Tableau, and SQL Academy, are studied. Based on these tools, effective methods of interactive learning for implementation are selected. The implementation includes the development of a React application based on client-side architecture, where the user interacts with the database using SQL.js. The application allows the creation of SQL queries using an interactive field, visualizing query results using tables and graphic elements, and visually displaying database tables imported by the user. Testing the application includes unit and integration tests using Jest and React Testing Library, as well as user testing. The results show that the application is effective as a tool for teaching SQL, with the potential for further improvements based on user feedback.

Keywords: SQL, database, education, data visualization, web application

Title translation: Evaluation of aggregate functions in SQL queries

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

V dnešní době jsou databáze a jazyk SQL nedílnou součástí mnoha odvětví a činností. Znalost a schopnost pracovat s jazykem SQL je důležitou dovedností pro mnoho profesí, protože většina moderních organizací používá k ukládání a správě informací databáze.

SQL je velmi výkonný nástroj, který umožňuje provádět různé operace s daty, včetně vytváření a mazání tabulek, přidávání, změn a mazání dat, načítání dat, propojování dat z různých tabulek, vytváření indexů a nastavování přístupových práv. Proto je nezbytný pro práci s relačními databázemi, které jsou v dnešním světě nejběžnější a nejpoužívanější.

Existují různé metodiky a techniky pro učení tohoto jazyka, jedním z nejúčinnějších přístupů je aplikace získaných znalostí v praxi. Praktická zkušenost je klíčovým faktorem při učení SQL, protože umožňuje prohloubit porozumění a zpevnit znalosti na reálných příkladech. Další efektivní metodou v praxi je vizualizace prováděných operací, která pomáhá studentům lépe pochopit a analyzovat své činnosti a také může pozitivně ovlivnit osvojení nových poznatků.

1.1 Cíle projektu

Vzhledem k důležitosti a složitosti SQL pro současnou práci s databázemi je hlavním cílem této bakalářské práce analyzovat existující metody praktického učení jazyka SQL, se specifickým zaměřením na techniky vizualizace dat. Na základě získaných poznatků bude navržen a implementován simulátor dotazů SQL, který je zaměřený na poskytnutí názorných příkladů a vizualizace dat pro podporu efektivního osvojení jazyka. Tento simulátor má za cíl usnadnit studentům porozumění principům jazyka SQL a postupů vyhodnocování SQL operací, což přispěje k lepší přípravě budoucích odborníků v oblasti databázových technologií.

Kromě toho, tento projekt zahrnuje také detailní testování aplikace, jak na úrovni jednotkových a integračních testů, tak na úrovni uživatelského testování. Testování je klíčovým prvkem, který zajišťuje, že simulátor je nejen funkční, ale také účinný jako nástroj pro výuku SQL. Výsledky testování poskytnou cennou zpětnou vazbu, která bude v budoucnu využita pro další vylepšení aplikace.

Kapitola 2
Teoretická část

2.1 Databáze a systémy jejich řízení

Databáze jsou důležitým nástrojem pro ukládání a zpracování velkého množství informací. Jsou úložištěm informací, které lze považovat za druh elektronické kartotéky, tj. úložiště nebo kontejner pro sbírku počítačových datových souborů, poskytující ukládání informací pohodlným a organizovaným způsobem [1]. Z tohoto důvodu se databáze často používají k ukládání informací v různých organizacích, aplikacích, podnicích a dalších strukturách, které vyžadují neustálou interakci s velkým objemem nejrůznějších dat. Používání databází přináší také následující výhody [2] :

- Integrace aplikací a sdílení databází.
- Výrazné snížení času a nákladů na vývoj aplikací.
- Lepší zabezpečení a soukromí dat.
- Snížení redundance dat.
- Poskytnutí výkonnější metody správy souborů.
- Vylepšení integrity dat.

Pro uživatelsky přívětivou interakci s fyzickou datovou základnou je k dispozici softwarová vrstva - DBMS (Database Management System). DBMS je softwarový systém, který umožňuje uživatelům vytvářet a spravovat databáze. Tato vrstva poskytuje uživatelům vnímání databáze, které je poněkud vyšší než hardwarová úroveň, a podporuje uživatelské operace, které jsou vyjádřeny v termínech tohoto vnímání na vyšší úrovni [1]. Tato softwarová vrstva implementuje interakce s pevným diskem a operační pamětí na základě použitého datového modelu (výkres, jak jsou data organizována a jak jsou mezi sebou propojena) a také poskytuje potřebné nástroje pro realizaci základních funkcí interakce s daty, jako jsou vytváření, ukládání a zpracování informací. V současné době nejpopulárnějšími DBMS jsou Oracle, MySQL a Microsoft SQL Server [3].

2.2 Relační databáze

Na základě nejpopulárnějších systémů pro správu databází uvedených v sekci 2.1 lze konstatovat, že relační databáze jsou jedním z nejrozšířenějších a nejvýkonnějších prostředků pro ukládání a správu velkého množství informací.

Relační databáze jsou založeny na relačním datovém modelu. Relační typ modelu popisuje vztahy mezi tabulkami, kde jsou informace reprezentovány jako řádky a sloupce. Klíčovým prvkem těchto vztahů jsou cizí klíče, které jedinečně identifikují záznamy v tabulkách. Použití cizích klíčů umožňuje efektivní propojování dat mezi tabulkami, což

zvyšuje efektivitu a flexibilitu při manipulaci s daty [4]. Na rozdíl od nerelačních databází, které ukládají informace v jiném formátu, například ve formě stromů nebo grafů, relační databáze umožňují používat k interakci s daty standardizovaný dotazovací jazyk SQL. Výhodou učení jazyka SQL je, že nutí uživatele konfrontovat se s datovými strukturami používanými k ukládání informací a porozumět jim [5].

Na obrazku č. 2.1 je uveden příklad tabulky popisující lety, která by mohla existovat v relační databázi. Data jsou zde rovněž rozdělena do sloupců (Id, CompanyId, Plane) a řádků, v nichž jsou uloženy příslušné informace.

Trip

Id	CompanyId	Plane
1	145	Star East Airline
2	342	Aero Services Mali
3	223	Adygea Airlines
4	22	Flight Alaska

Obrázek 2.1. Příklad vyplněné tabulky letů v relační databázi.

2.3 SQL

2.3.1 Definice a příkazy

Základním požadavkem pro práci s relačními DBMS je výkonný jazyk, který umožňuje provádět všechny operace, které uživatelé potřebují. SQL je standardní jazyk pro přistupování k relačním databázím, kde se příkazy dělí na čtyři základní skupiny [5].

- **Příkazy pro definici dat (DDL)** (CREATE, ALTER, DROP, …). Tyto příkazy se používají k vytváření, modifikaci a mazání struktur databází a tabulek.
- **Příkazy pro manipulaci s daty (DML)** (SELECT, INSERT, UPDATE, DELETE, …). Tato skupina příkazů umožňuje uživatelům získávat, vkládat, měnit a mazat data v tabulkách.
- **Příkazy pro řízení transakcí (TCL)** (START TRANSACTION, COMMIT, ROLLBACK). Tyto příkazy řídí, jak jsou operace provedeny jako jednotlivé transakce, což pomáhá zajišťovat konzistenci dat.
- **Příkazy pro řízení přístupových práv (DCL)** (GRANT, REVOKE). Tyto příkazy řídí, kdo má přístup k určitým datům a jaké operace mohou provádět.

2.3.2 Agregační funkce

Agregační funkce v SQL jsou velmi užitečným nástrojem pro analýzu dat. Tyto funkce umožňují uživateli provádět různé operace na skupině hodnot, aby bylo dosaženo jediného souhrnného výsledku. Patří sem například výpočet průměru (AVG), součtu

(SUM), minima (MIN), maxima (MAX) a počtu prvků (COUNT). Na obrázku 2.2 vidíte příklad použití agregačních funkcí v SELECT dotazu. Tento dotaz vrátí celkový počet (použití operatoru COUNT) objednávek a průměrnou cenu (použití operatoru AVG) všech objednávek.

```
37. SELECT AVG(capacity) AS average, COUNT(number) AS count
    FROM Room;
```

Obrázek 2.2. Příklad použití agregačních funkcí v semináři č. 5 „SQL: Data Querying“ předmětu „Databázové systémy“.

Je důležité si uvědomit, že výsledkem agregační funkce je vždy jedna hodnota na skupinu řádků. Pokud není specifikována skupina řádků pomocí klauzule GROUP BY, výsledkem je jedna hodnota pro celou tabulku.

2.3.3 Dotazování

Při sestavování různých dotazů do databáze se často používá příkaz SELECT z části pro manipulaci s daty. Jeho celková struktura se skládá jak ze samotného příkazu SELECT, tak i z jeho parametrů. Abychom mohli vybrat nějakou informaci, musíme určit, odkud ji chceme vzít. K tomu se používá parametr FROM, za kterým mohou následovat různé příkazy JOIN. Pokud je vyžadována nějaká podmínka, použije se parametr WHERE v kombinaci s různými operátory porovnávání. Obecná struktura dotazu vypadá takto:

SELECT [DISTINCT | ALL] - pole tabulky, DISTINCT pro unikátní řádky.
FROM - seznam tabulek s možnostmi jejích propojení.
[JOIN - propojení tabulek na základě souvisejícího sloupce] (nepovinné).
[WHERE - podmínky výběru] (nepovinné).
[GROUP BY - seskupení řádků do agregovaných dat] (nepovinné).
[HAVING - podmínky pro seskupení] (nepovinné).
[ORDER BY - řazení výsledeku [ASC|DESC] (nepovinné).
[LIMIT - limit záznamů výsledků] (nepovinné).

2.3.4 Postup provádění operací SQL

Proces provádění každého SQL dotazu začíná dotazem do databáze na konkrétní tabulky, z nichž lze poté provádět různé operace filtrování a třídění. Proto se části dotazu SQL provádějí postupně a v následujícím pořadí [6–7]:

- FROM a JOIN. Operátor FROM určuje tabulku, která bude použita pro výběr. Více tabulek lze kombinovat pomocí operátoru JOIN.
- WHERE. Tento operátor určuje podmínku, která musí být splněna, aby byl řádek zahrnut do výběru.
- GROUP BY. Operátor seskupí řádky tabulky do jedné nebo více podmnožin řádků, přičemž každá podmnožina obsahuje všechny řádky původní tabulky, které mají shodné hodnoty jednoho nebo více cílových atributů vybraných na vstupu. Často se používá s agregačními funkcemi (jako jsou COUNT, MAX, MIN, SUM a AVG) pro seskupení výsledných dat podle jednoho nebo více sloupců.

- HAVING. Určuje podmínku, která musí být splněna, aby byl řádek tabulky, na který se vztahuje operátor GROUP BY, zahrnut do výběru. HAVING se používá k filtrování výsledků agregačních funkcí.
- SELECT. Vyběr sloupců, které budou zahrnuty do výstupu výběru.
- ORDER BY. Seřazení sady výsledků vzestupně nebo sestupně.
- LIMIT/OFFSET. Vyloučení řádků, které nespadají do rozsahu zadaného pomocí LIMIT a OFFSET.

Správné pořadí provádění operací SQL je důležité pro získání požadovaných výsledků a efektivní práci s daty v databázi. Porozumění těmto krokům umožňuje uživatelům sestavit komplexní a přesné dotazy pro manipulaci a analýzu dat.

Kapitola 3
Analýza problému

3.1 Definice problému

Porozumění principům relačních databází a efektivní interakce s jejich tabulkami může být na začátku studia jazyka SQL výzvou. Pro dosažení plného ovládnutí tohoto jazyka je nezbytné se naučit formulovat a implementovat různé dotazy a porozumět jejich praktickému fungování.

SQL se od imperativních programovacích jazyků, jako je Java nebo C++, výrazně odlišuje tím, že představuje deklarativní programovací jazyk. Zatímco imperativní jazyky vyžadují posloupnost příkazů pro provedení úkolů, SQL se zaměřuje na specifikaci požadovaných dat z databáze bez podrobného popisu konkrétních postupů pro jejich získání. Samotný algoritmus provádění dotazů v SQL je závislý na konkrétní implementaci systému pro správu databází (DBMS).

Lidi, kteří se tento jazyk učí po imperativním programovacím jazyku, musí přehodnotit své obvyklé programovací paradigma, aby mohli úspěšně vytvářet dotazy. Rovněž je důležité spojit své dovednosti s teorií množin při spojování tabulek s cílem dosáhnout optimálních výsledků.

Zvládnutí těchto klíčových aspektů je nezbytné pro tvorbu přesných a efektivních dotazů, které plně odpovídají požadavkům na získávání a analýzu dat z relačních databází. V tomto kontextu může vývoj aplikace s vizualizací provádění SQL dotazů hrát klíčovou roli při podpoře učebního procesu. Vizualizace SQL dotazů umožní uživatelům názorně zkoumat a analyzovat získaná data, což zlepší jejich chápání základů a souvislostí v databázovém prostředí. Vizuální prezentace dat zlepšuje učební proces tím, že umožňuje uživatelům lépe vnímat a interpretovat výsledky svých dotazů.

3.2 Výuka databázových systémů

V rámci bakalářského studia na Fakultě elektrotechnické ČVUT je vyučován předmět Databázové systémy (DBS), který pro mnoho vysokoškolských studentů je prvním seznámením se základy databázových systémů. Předmět je koncipován jako základní databázový kurz, v němž je důraz kladen zejména na schopnost samostatného návrhu datového modelu, zvládnutí jazyka SQL a schopnosti zvolit vhodný stupeň izolovanosti transakcí. Studenti se seznámí s nejběžněji používanými technikami indexace, architekturou databázových systémů a jejich správou [8]. Také se zde studenti naučí používat všechny možné datové typy, základní operace s databázovými tabulkami, vyzkouší si různé podmínky a omezení, agregační funkce, spojování tabulek a operace CRUD (create, read, update, delete).

3.3 Použití databázových operací

V obsahu cvičení 4 až 7 z předmětu DBS popsaného v sekci 3.2 bylo prezentováno 79 příkladů obsahujících nejrůznější konstrukce SQL [9]. Provedená analýza ukázala, že

mezi ně patří 18 DDL (Data Definition Language) příkladů pro definici dat, 54 DML (Data Manipulation Language) příkladů pro manipulaci s daty a 4 TCL (Transaction Control Language) příkladů pro řízení transakcí (viz sekce 2.3.1).

Vezměme si příklad 11 popisující DML dotaz (viz obrazek č. 3.1), který vybírá všechny studijní výsledky studenta s ID = „4301“ za předchozí semestr s ID = „201“ a proanalyzujeme. Kritériem tohoto dotazu je seřadit řádky podle výsledků a názvů kurzů sestupně. Analýza ukázuje, že dotaz obsahuje operace SELECT, FROM a WHERE, které patří mezi základní operace dotazování. Také u operace WHERE se k vytvoření podmínky při porovnávání ID studenta a čísla semestru používají porovnávací predikáty „=“ a operátor AND. Můžeme si také všimnout spojení tabulek Enrollment a Course pomocí operatoru NATURAL JOIN, abychom získali požadované parametry v rámci SELECT. Dotaz končí sestupným seřazením podle výsledku a názvu, které je realizováno pomocí základní operace ORDER BY a operace modifikace dat DESC.

```
11. SELECT code, title, result
    FROM Enrollment NATURAL JOIN Course
    WHERE (student = 4301) AND (semester = 201)
    ORDER BY result, title DESC;
```

Obrázek 3.1. Příklad SQL dotazu uvedený v semináři č. 5 „SQL: Data Querying“.

Podobným způsobem byl zanalyzován každý ze 79 předložených příkladů a na základě operací SQL použitých na cvičeních a operací SQL popsaných na přednáškách tohoto kurzu byla sestavena tabulka 3.1, která porovnává a vypočítává, jak často byly tyto operace v rámci cvičení použity v praxi. Pro větší přehlednost byly operace rozděleny do příslušných skupin, jako jsou: základní operace, podmínky, modifikace výstupu, agregační funkce, spojování tabulek, množinové operace, úprava dat, datové typy a omezení integrity. Počet operací pro každý ze 79 příkladů je uveden v souboru SQL-DBSoperations.xlsx.

Výsledná tabulka 3.1 ukazuje, že nejčastěji používané konstrukce jsou SELECT, porovnováci predikáty, FROM a WHERE. Přestože se SELECT bez příkazu FROM nejčastěji nepoužívá, jeho samostatné použití je také možné a bylo implementováno např. v příkladu 59 (obrázek 3.2) se správou transakcí pro volání implementované transakční funkce.

```
59. BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    SELECT transfer(10000040, 501, 502, 1000.00);
    COMMIT TRANSACTION;
```

Obrázek 3.2. Použití funkce SELECT k volání dříve definované transakční funkce transfer().

Základní dotazování:	Použití:
SELECT	51
FROM	49
WHERE	48
GROUP BY	2
HAVING	1
ORDER BY	8
Podmínky:	
AND, OR, NOT	39
ANY, SOME, ALL	3
Porovnávací predikáty	50
LIKE	1
IN	5
EXISTS	8
Modifikace výstupu:	
ALL	13
DISTINCT	11
DSC	7
Agregační funkce:	
COUNT	5
SUM	1
AVG	5
MIN	0
MAX	0
Spojování tabulek:	
NATURAL JOIN	10
JOIN ON	15
LEFT JOIN	1
RIGHT JOIN	0
FULL OUTER JOIN	0

Množinové operace:	Použití:
UNION	2
INTERSECT	0
EXCEPT	2
Úprava dat:	
CREATE	18
INSERT	8
DELETE	4
UPDATE	5
Datové typy:	
INTEGER	10
DECIMAL	3
FLOAT	0
BOOLEAN	1
CHAR()	4
VARCHAR()	8
DATE	3
Omezení integrity:	
NOT NULL	12
UNIQUE	4
PRIMARY KEY	13
FOREIGN KEY	3
CHECK	5

Tabulka 3.1. Počet použití jednotlivých operací SQL v rámci předmětu „Databázové systémy“.

3.4 Shrnutí

Díky výše uvedené tabulce 3.1 můžeme konstatovat, že SELECT se svými parametry je jedním z nejčastěji používaných operátorů při vytváření SQL dotazů, a to jak v rámci předmětu DBS, tak v praxi v jiných oblastech. Proto cílem této práce je vizualizovat DML funkce pro manipulaci s daty na základě dotazů SELECT pro výběr dat.

Pro úplné pochopení SQL dotazů s použitím SELECT je nezbytné zkoumat nejen další operátory v jeho struktuře (viz 2.3.3), ale také se seznámit s různými agregačními

funkcemi, způsoby spojování tabulek, seřazováním dat a tvorbou různých omezení. V rámci kurzu DBS jsou některé z těchto parametrů probírány málo nebo vůbec, což může u studentů způsobit nejasnosti, pokud se s nimi setkají mimo kurz databázových systémů. Proto z tabulky 3.1 byly vybrány nejdůležitější parametry ke studiu, které se vztahují na DML SELECT dotazy (viz tabulka 3.2).

SQL operace	Počet příkladů
Základní dotazování:	
WHERE	39
SELECT	37
FROM	37
GROUP BY	2
HAVING	1
Modifikace výstupu:	
DISTINCT	9
DSC	2
Agregační funkce:	
AVG()	5
COUNT()	4
SUM()	1
Spojování tabulek:	
JOIN ON	13
LEFT JOIN	1
RIGHT JOIN	0
FULL OUTER JOIN	0
Podminky:	
LIKE	1

Tabulka 3.2. Vybrané operace k provedení a počet použití.

Pro lepší pochopení interakce s databázemi bude v této práci implementována vizuální simulace výše uvedených operací, která studenta postupně provede procesem vytváření a volání SQL dotazu.

Kapitola 4
Existující aplikace

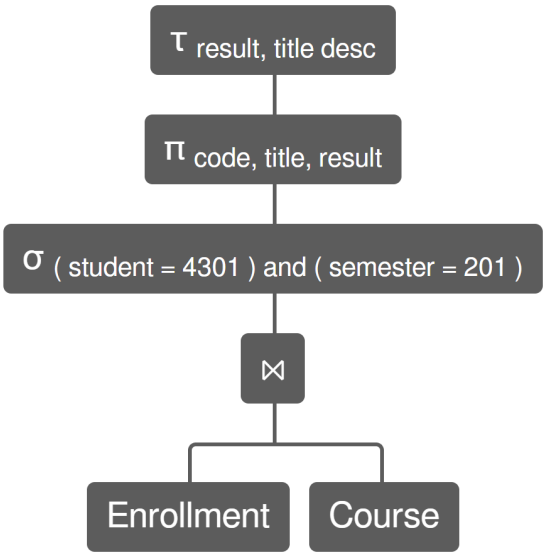
V současné době existuje mnoho různých aplikací pro výuku jazyka SQL, ale jen některé z nich jsou zaměřeny na jiný druh vizualizace dat než zobrazení tabulek. Tato sekce se zabývá aplikacemi, které se svou vizualizací liší od všech ostatních, z jejichž analýzy jsou formulovány závěry pro danou vyvíjenou aplikaci.

4.1 Relax

Relax [10] je simulátorem relační algebry. Ačkoli se tento nástroj zaměřuje především na interakci s relační algebrou, dokáže také vytvářet tabulky v databázi a simulovat dotazy SQL, které jsou pak pomocí relační logiky transformovány do stromu operací. Služba neposkytuje možnost vizualizace stejného stromu operací pomocí syntaxe SQL, ale pokud rozumíte souboru pravidel relační algebry, pomůže vám pochopit logiku provádění dotazů v databázi.

4.1.1 Vizualizace

Relax má osobitou vizualizaci dotazů SQL, které převádí na strom operací popsaných relační algebrou. Na obrázku 4.1 je zobrazen strom po provedení dotazu, realizovaného na základě výše uvedeného dotazu z obrázku 3.1. Na tomto příkladu vidíme, jak je dotaz rozdělen na všechny části jeho provádění. Po najetí na libovolnou část se zobrazí podrobnosti o provedené operaci, sloupcích, se kterými se pracovalo, a průběžný stav dotazu.



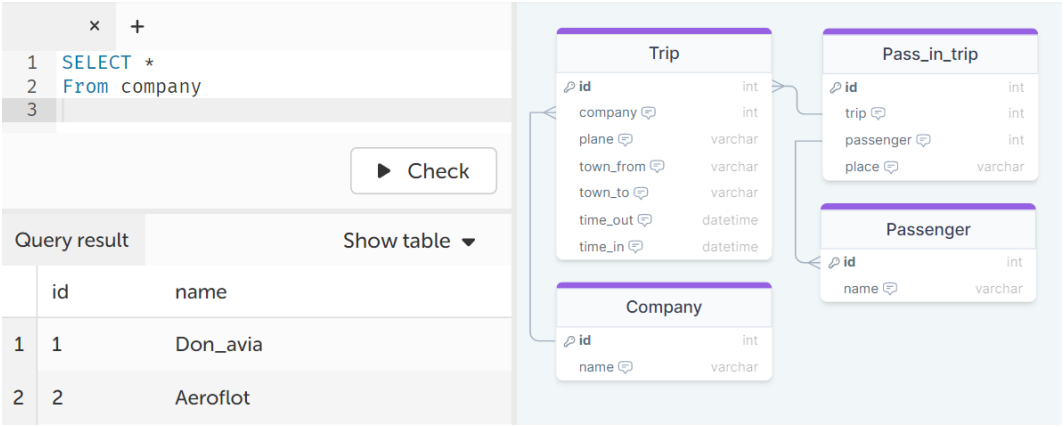
Obrázek 4.1. Strom operací sestavený pomocí Relax.

4.2 SQL Academy

SQL Academy [11] je služba pro výuku SQL pomocí řešení různých úloh, které jsou uvedeny v její vlastní databázi. Služba nepředstavuje vizuální simulaci dotazování, ale umožňuje vám po vyřešení úloh zkontrolovat správnost provedených akcí pomocí nejrůznějších testů. Velmi výhodné je také to, že se na obrazovce zobrazuje pole pro psaní dotazů, měnící se tabulka a ER diagram. Po zapsání jakéhokoli dotazu do databáze pro konkrétní úlohu můžete výsledek dotazu zobrazit v konečné tabulce, což také pomáhá porozumět dotazům SQL a analyzovat je.

4.2.1 Vizualizace

SQL Academy poskytuje velmi pohodlný způsob ovládání zápisu dotazu díky kombinované přítomnosti prvků, jako jsou pole dotazu, tabulka výsledků a ER (entity relationship) diagram na obrazovce (viz obrazek 4.2). Uživatel se může v tomto diagramu pohodlně pohybovat a díky tomu může přehledně analyzovat vztahy mezi tabulkami, což je důležité zejména v relační databázi. Navíc to může pomoci efektivně napsat požadovaný dotaz.



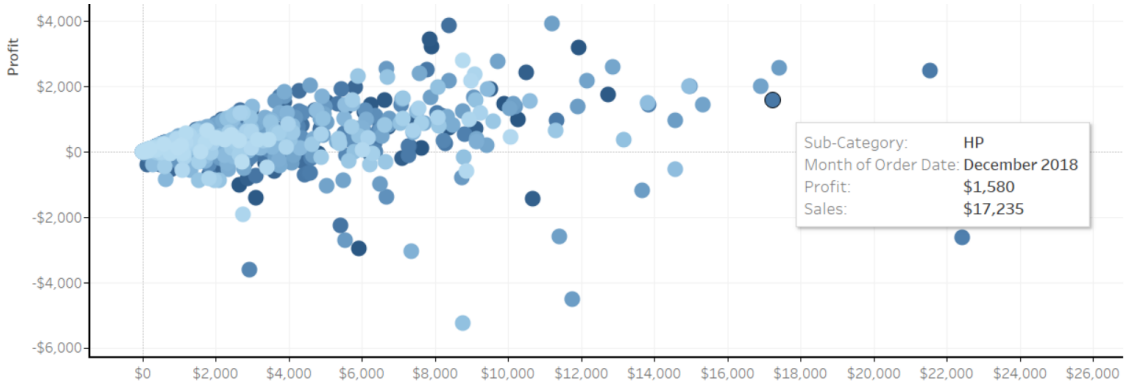
Obrázek 4.2. Vstupní pole dotazu, tabulka výsledků a ER diagram SQL Academy.

4.3 Tableau

Tableau [12] je nástroj pro vizualizaci dat a business analýzu. Pomáhá uživatelům vytvářet různé grafy, diagramy, mapy, informační panely a příběhy pro vizualizaci a analýzu dat. Vizualizaci dat mohou provádět i uživatelé bez programátorských dovedností. Tato služba však nenabízí vizualizaci procesu dotazování a nevizualizuje prováděné operace SQL, takže pro přímou interakci s vytvářením a prováděním dotazu SQL je již nutná určitá znalost jazyka.

4.3.1 Vizualizace

Na obrázku 4.3 je zobrazen jeden z možných diagramů, korelační diagram, který vizualizuje v kartézských souřadnicích vztah mezi číselnými proměnnými. Zde se zobrazí všechna data z vybrané tabulky. Po najetí na některý z objektů se zobrazí jeho podrobné informace, které si uživatel zpočátku sám definuje.



Obrázek 4.3. Korelační diagram realizovaný v Tableau.

4.4 Shrnutí

Pro výuku jazyka SQL a používání databází je dnes k dispozici mnoho různých nástrojů, ale každý z nich se zaměřuje na specifické způsoby práce s informacemi obsaženými v tabulkách. Některé aplikace mají například za svou hlavní funkci analýzu dat a statistiku, nebo se zaměřují na učení při řešení reálných problémů, případně na transformaci různých dotazů do stromu operací, jako je tomu u výše uvedených aplikací. Pro realizaci optimálních řešení pro tento projekt byly analyzovány jejich silné stránky, mezi které patří:

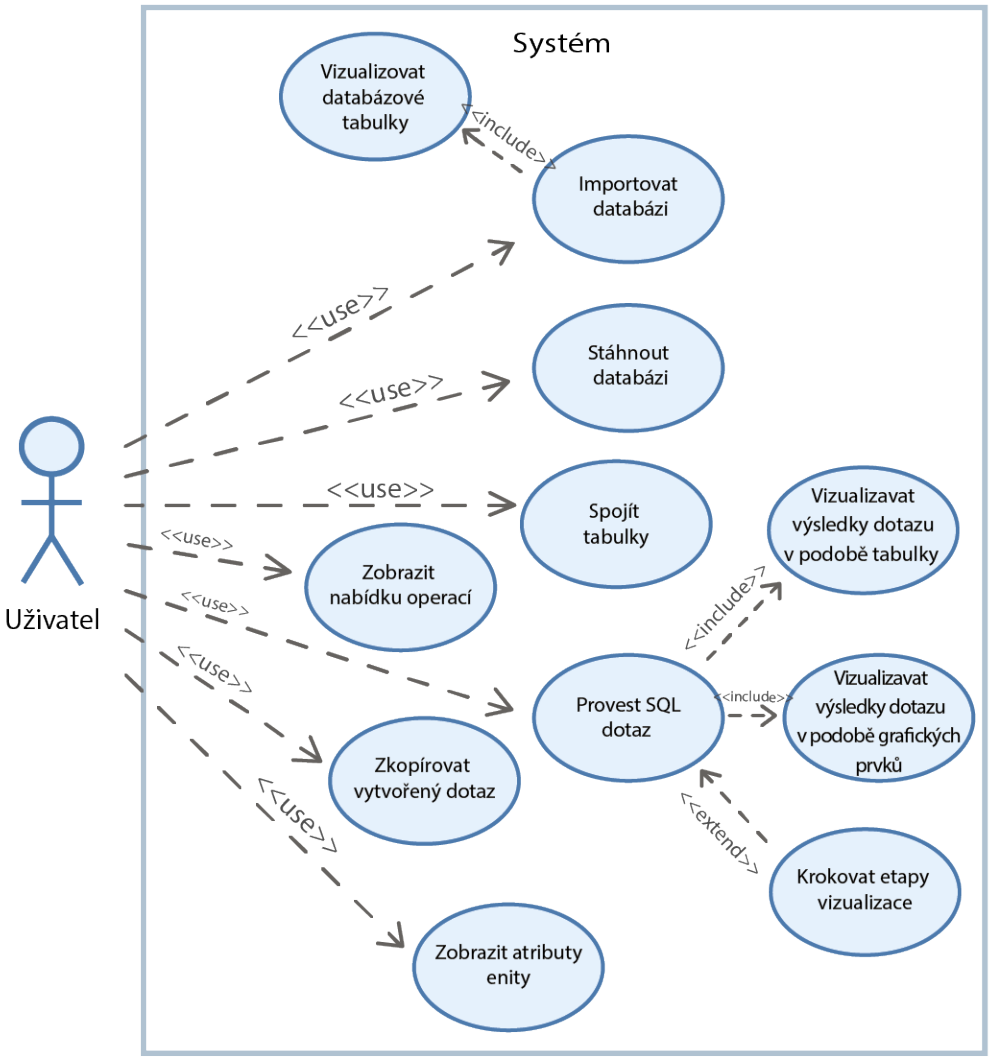
- **Postupné zobrazení procesu dotazování**. Pro ty, kteří se teprve začínají učit jazyk SQL, je důležité mít představu o tom, co se děje s jejich dotazem po jeho zadání a jak se data mění, aby vznikl konečný výsledek (Relax).
- **Snadný přístup k diagramu vztahů a tabulce výsledků**. Při zadávání dotazů je obzvláště důležité si uvědomit, které entity spolu vzájemně komunikují. To uživateli pomůže správně vybrat a propojit tabulky pro následné dotazy. Výsledná tabulka ukáže, zda uživatel na základě svého dotazu obdržel správná data (SQLAcademy).
- **Zobrazení dat ve formě grafických prvků**. Grafické prvky díky své jednoduchosti pomáhají uživateli rychleji pochopit, jaké akce na nich byly provedeny a jak se změnily (Relax, Tableau).
- **Zobrazení podrobných informací při najetí na objekt**. Tato funkce přispívá k optimalizaci rozhraní tím, že minimalizuje faktory potenciálně rušící uživatele, a přesto zachovává funkci pro zobrazení detailních informací (Relax, Tableau).

Všechny tyto metody jsou efektivní a mohou se vzájemně doplňovat. Řešení, navržené v rámci této práce, se stane významným doplňkem k existujícímu výukovému materiálu pro SQL, neboť nabízí vizualizaci procesu provádění dotazů a odchyluje se od tradiční tabulkové prezentace dat ve prospěch grafických elementů, které umožňují simulaci dotazů.

Kapitola 5
Navrh aplikace

5.1 Případy užití

Diagram 5.1 popisuje všechny hlavní způsoby interakce s systémem možné pro uživatele, na základě kterých se tvoří funkční (viz 5.3) a nefunkční (viz 5.4) požadavky. Podrobněji o uživateli je popsáno v sekci 5.2.



Obrázek 5.1. Diagram případů užití systému.

5.2 Aktéři

V rámci implementovaného systému bude existovat pouze jeden typ uživatele, který bude mít přístup ke všem dostupným funkcím. V kontextu aplikace mohou roli uživatele plnit jak studenti, tak učitelé. Možnosti nahrávání a stahování databází, stejně jako funkce kopírování vytvořených dotazů, poskytují uživatelům pohodlný mechanismus pro výměnu integrovaných dat, se kterými navázali interakci. Takový přístup zjednodušuje proces učení a komunikace pro všechny účastníky vzdělávacího procesu.

5.3 Funkční požadavky

Funkční požadavky se zde řídí následujícím vzorem: systém umožní [role] [funkce]. U nefunkčních požadavků (viz 5.4) struktura odpovídá vzoru: systém musí mít [parametr].

- **F1**. Systém umožní uživateli pracovat se všemi svými funkcemi bez nutnosti prokázání osobních údajů nebo registrace.
- **F2**. Systém umožní uživateli zobrazit vizualizaci tabulek a jejich vztahů v databázi pro následnou analýzu a další dotazy na její základě.
- **F3**. Systém umožní uživateli exportovat a importovat vlastní databáze.
- **F4**. Systém umožní uživateli vyplnit existující tabulky daty.
- **F5**. Systém umožní uživateli sdílet vytvořené dotazy jejich kopírováním.
- **F6**. Systém umožní uživateli vybrat, se kterými konkrétními tabulkami z vizualizovaného diagramu databáze chce při zadávání dotazů pracovat, a spojit je pomocí nabízených typů spojení tabulek: NATURAL JOIN, INNER JOIN, LEFT JOIN, RIGHT JOIN nebo OUTER JOIN.
- **F7**. Systém umožní uživateli vytvořit SQL dotaz pomocí nabízených operací a spustit ho na předem vybraných tabulkách.
- **F8**. Systém umožní uživateli graficky zobrazit vizualizaci provádění dotazů SQL krok za krokem na základě vybraných operací SELECT dotazu.
- **F9**. Systém umožní uživateli zobrazit tabulku výsledků po provedení SQL dotazu na základě vybraných operací.
- **F10**. Systém umožní uživateli vrátit se ke spojení tabulek a vytvářet nové typy spojení tabulek pro další dotazy.
- **F11**. Systém umožní uživateli po provedení dotazu SQL zobrazit podrobné informace o všech vyplněných atributech entity.
- **F12**. Systém umožní uživateli smazat vytvořený dotaz a vytvořit nový.

5.4 Nefunkční požadavky

- **N1**. Systém musí mít intuitivní a srozumitelný design pro každého uživatele.

- **N2**. Systém musí být bez omezeni použitelný na všech běžných operačích systémech a prohližčích.
- **N3**. Systém musí mít přehlednou vizualizaci databázových tabulek, jejichž polohu na obrazovce lze měnit podle potřeby.
- **N4**. Systém musí dodržovat ACID principy pro SQL dotazy.
- **N5**. Systém musí podporovat export a import dat z databáze pomocí formatu .db.
- **N6**. Systém by měl být schopen se efektivně škálovat a zachovávat výkonnost při vysokém počtu uživatelů nebo velkém objemu dat.
- **N7**. Systém by měl dodržovat bezpečnostní standardy, aby ochránil data uživatelů a zabraňoval neautorizovanému přístupu.
- **N8**. Systém by měl zobrazovat případné chyby v dotazech generovaných uživatelem.

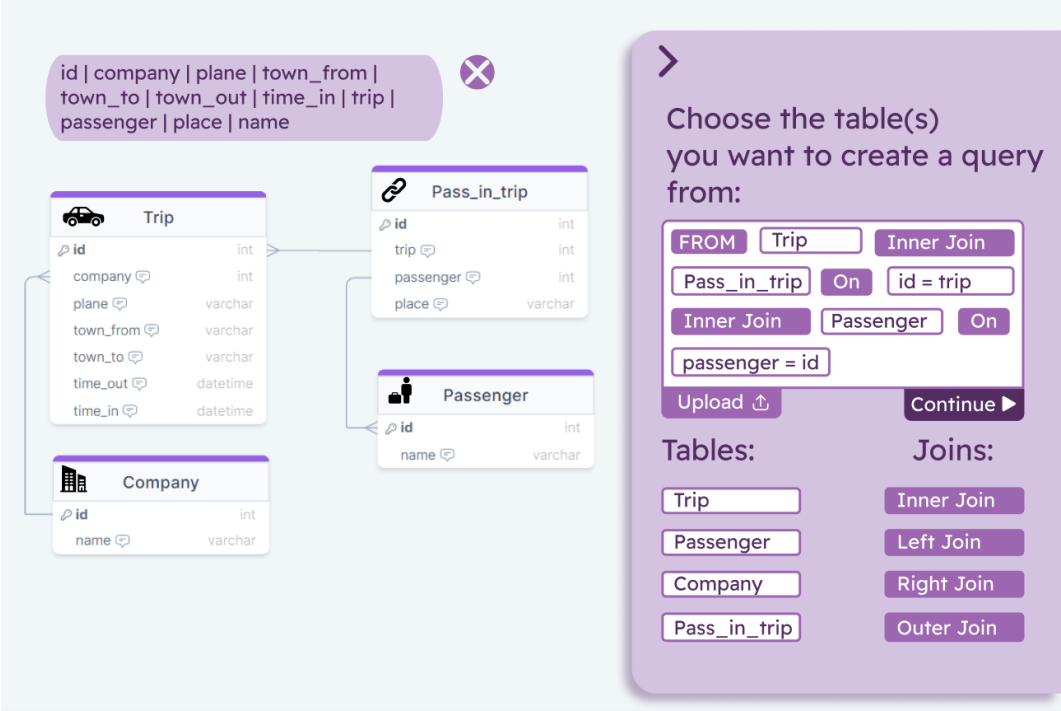
5.5 Návrh rozhraní

Na základě analýzy požadavků definovaných v sekcích 5.3 a 5.4 byl vytvořen návrh rozhraní se všemi funkcemi, které by aplikace měla obsahovat. Návrh rozhraní byl vytvořen v souladu s hlavními charakteristikami, které by mělo uživatelské rozhraní mít: použitelnost, přehlednost a atraktivita. Všechny prvky jsou strategicky rozmístěny na logických místech, díky čemuž je navigace v rozhraní pro uživatele intuitivní.

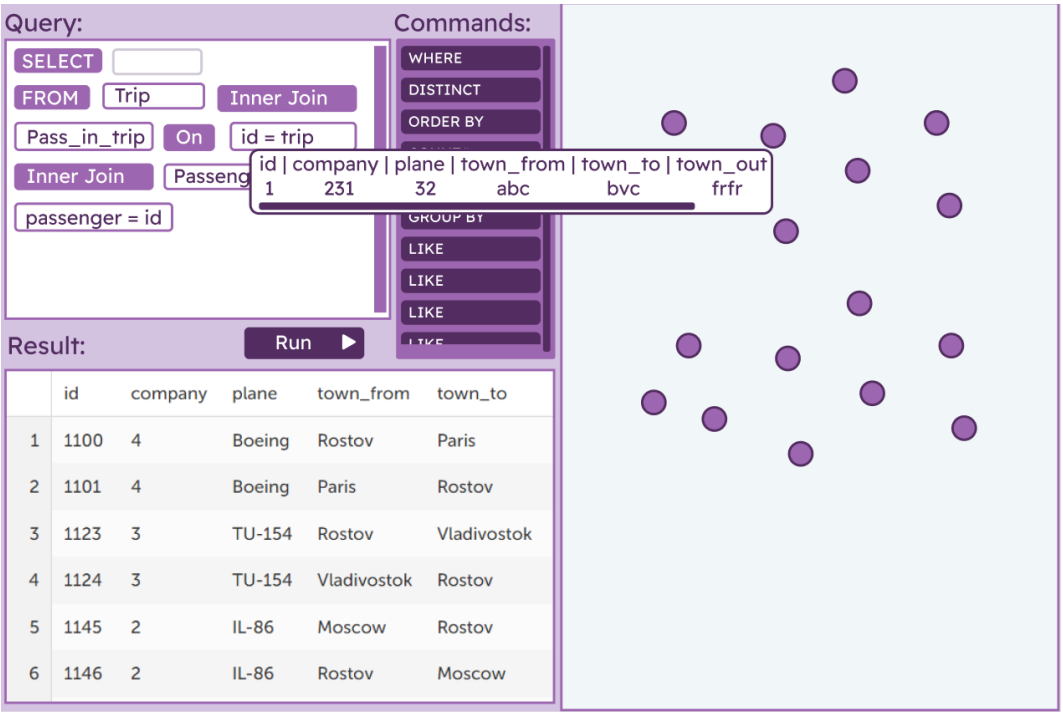
Aplikace je reprezentována třemi obrazovkami, a to úvodní obrazovkou popisující hlavní funkce aplikace, obrazovkou 5.2 pro slučování tabulek a obrazovkou 5.3 pro vytváření a vizualizaci dotazů SQL (viz plná verze v souboru „Prototype.pdf“).

Obrázek 5.2 ukazuje obrazovku pro vytvaření různých kombinací tabulek pomocí příkazů JOIN, na jejichž základě lze později psát dotazy SQL. Uživatel zde může zobrazit ER diagram k importované relační databázi, vybrat různé typy spojení pro entity v daném relačním modelu a analyzovat diagram stejně jako výsledky kombinace tabulek.

Po výběru entit, se kterými chce uživatel pracovat, se zobrazí obrazovka pro sestavení dotazů SQL. Zde se uživateli zobrazí všechny operace uvedené v kapitole 3.4. Výběrem a kombinací různých operací si uživatel může procvičovat sestavování dotazů a současně sledovat vizualizaci probíhajících operací na pravé straně obrazovky. Všechna data jsou prezentována v podobě kruhů, což zjednodušuje vizualizaci tím, že data abstrahuje do grafických prvků. Po odeslání dotazu bude program schopen postupně zobrazit vizualizaci provádění dotazu podle vybraných operací SELECT. Po najetí na prvek se o něm zobrazí podrobné informace a v levé dolní části obrazovky se zobrazí výsledky dotazu ve formě tabulky. To umožní uživateli rychle a s jistotou se orientovat v procesu dotazování a kontrolovat jeho konečný výsledek.

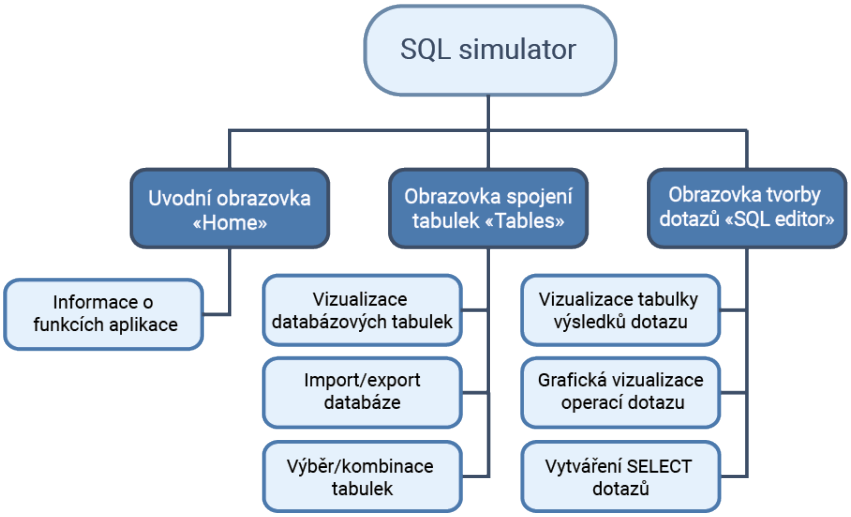


Obrázek 5.2. Navrh obrazovky pro slučování entit relačního modelu před vytvořením SELECT SQL dotazu.



Obrázek 5.3. Návrh obrazovky pro vytváření SELECT dotazů.

Obrázek 5.4 ukazuje navrhovanou strukturu webové stránky na základě výše popsaných navrhovaných obrazovek. Slova označená uvozovkami představují anglické názvy stránek v navigačním menu.



Obrázek 5.4. Návrh celkové strukruty webové stránky.

5.6 Architektura

Jako návrh byla zvolena architektura zcela realizovaná na straně klienta aplikace. Tato aplikace by poskytovala možnost práce se SQL dotazy a databázemi bez nutnosti použití serverové strany, což usnadňuje její nasazení a zjednodušuje podporu.

Hlavní výhody takové architektury spočívají v tom, že veškeré výpočty a manipulace s daty probíhají v prohlížeči uživatele, což zajišťuje rychlé a plynulé rozhraní. Takový přístup eliminuje zpoždění způsobené přenosem dat mezi klientem a serverem a snižuje zátěž na serverovou infrastrukturu [13].

Je třeba zdůraznit, že pro tento druh aplikace není efektivní řešení vytváření databáze, protože je nezbytné poskytnout uživateli možnost importovat svou osobní databázi do aplikace. Využití DBMS v takovém případě by vedlo k nežádoucímu zabrání pamětí, jelikož by bylo nutné pro každého uživatele vytvořit novou databázi a současně regulovat její odstranění za účelem efektivního využití paměťových zdrojů.

Implementace vypočtu SQL dotazů prostřednictvím webového prohlížeče uživatele umožňuje uchovávat informace o importovaných relačních tabulkách až do následného načtení stránky, což zbavuje vývojáře nutnosti řešení problému s využitím paměti aplikace [14]. Tento přístup představuje efektivní řešení pro správu databází v rámci klientem řízených aplikací a otevírá nové možnosti pro další vývoj a optimalizaci takových architektur.

Implementaci provádění SQL dotazů bez vytváření databáze na straně serveru aplikace můžeme dosáhnout převedením původního kódu SQL přímo do kódu JavaScript. Konvertovaný kód se pak může spouštět na straně klienta pro zpracování dotazů a správu dat, což umožňuje úplně obejít serverovou část aplikace.

5.7 Technologie

5.7.1 Jazyk implementace

Aplikace byla navržena tak, aby byla kompletně zpracovávána na straně klienta bez potřeby backendové části. JavaScript nebo TypeScript jsou ideální volbou pro takový

scénář, protože jsou široce podporovány webovými prohlížeči a nevyžadujou žádné další nástroje nebo knihovny pro základní funkce.

Hlavním rozdílem mezi těmito jazyky je statické a dynamické typování. JavaScript je dynamicky typovaný jazyk, kde typy proměnných nejsou určeny předem a mohou se měnit za běhu programu. Z druhé strany TypeScript je staticky typovaný jazyk, který rozšiřuje JavaScript o možnost definovat typy proměnných. Staticke typovani zlepšuje čitelnost kódu, zvyšuje jeho bezpečnost a usnadňuje nalezení a opravu chyb během vývoje, ale využiti TypeScriptu vyžaduje zvládnutí dodatečných konceptů a syntaxe, což může zpočátku zpomalit vývoj [15].

Na základě zkušeností získaných během studia kurzu „Tvorba klientských aplikací v JavaScriptu“ na ČVUT, stejně jako zkušeností nabytých při práci na různých osobních projektech, se zdá být výběr JavaScriptu jako ideální volba pro rychlý a flexibilní vývoj webové aplikace v mém případě.

5.7.2 Frameworky pro vývoj webových aplikací

Použití frameworků místo čistého JavaScriptu pro implementaci webových aplikací nabízí řadu výhod. Frameworky zjednodušují strukturování a organizaci kódu, podporují opakované využití komponent a zkracují dobu vývoje díky poskytování hotových řešení pro běžné úkoly. Navíc nabízejí podporu ze strany komunity vývojářů, vestavěné mechanismy optimalizace výkonu a zajišťují kompatibilitu s různými prohlížeči a platformami. Zkoumejme tři nejpopulárnější JavaScriptové frameworky pro vývoj uživatelských rozhraní, jak bylo zjištěno z analýzy statistik stažení balíčků NPM [16]:

- **React.js [17]** je lehký a flexibilní framework, který umožňuje rychlou tvorbu a údržbu aplikací. Využívá komponentově orientovaný přístup, který usnadňuje organizaci kódu a opakované použití komponent. Poskytuje také širokou škálu knihoven a nástrojů třetích stran, které mohou urychlit vývoj a zlepšit kvalitu kódu. Pro některé úlohy, jako je správa stavu aplikace nebo směrování, však React vyžaduje další nástroje.

- **Angular [18]** nabízí kompletní framework pro vytváření komplexních webových aplikací pomocí architektury založené na komponentách. Obsahuje mnoho nástrojů a mechanismů pro řešení typických úloh, jako je směrování, formuláře a validace, a má bohatý ekosystém a velkou komunitu vývojářů. Angular je však těžší a složitější než React, což může zpomalovat vývoj a komplikovat podporu aplikací. Angular navíc často používá jazyk TypeScript, který může vyžadovat další školení a přizpůsobení pro vývojáře, kteří používají pouze JavaScript.

- **Vue.js [19]** také nabízí přístup založený na komponentách a snadné použití podobně jako React.js, navíc s dalšími funkcemi, jako je obousměrné propojení dat a integrovaná podpora animací. Má aktivní komunitu vývojářů, ale ekosystém Vue.js je ve srovnání s React.js stále menší.

Každý z posuzovaných frameworků poskytuje všechny nezbytné prostředky pro vytváření logiky a rozhraní této aplikace. Nabízejí silné nástroje a možnosti pro vývoj složitých webových aplikací s komponentní architekturou a pružným řízením stavu. Avšak důležitým kritériem pro výběr je dostupnost knihoven, nástrojů a velikost vývojářské komunity. React.js je jedním z nejpopulárnějších JavaScriptových frameworků podle analýzy statistik stažení [16] a jeho vývojářská komunita nadále

roste, což činí tyto výhody nejvhodnější volbou pro tuto aplikaci a zajišťuje efektivní vývoj s důrazem na jedinečné funkce a uživatelské rozhraní.

Kapitola 6
Implementace

6.1 Zpracování SQL dotazů na straně klienta

Před implementací aplikace je nutné určit, jakým způsobem lze zpracovávat SQL dotazy na straně klienta. Tento krok je důležitý pro správné fungování a interakci komponent systému. Jedním z možných řešení této situace je použití JavaScriptové knihovny SQL.js, která umožňuje provádět SQL dotazy na straně klienta pomocí WebAssembly. Obrazovky implementované aplikace založené na dříve popsaném návrhu naleznete v příloze C.

6.1.1 SQL.js

Knihovna SQL.js [14] je JavaScriptová adaptace databáze SQLite, která umožňuje provádět SQL dotazy na klientské straně aplikace s využitím WebAssembly. WebAssembly je nízkoúrovňový binární formát kódu navržený pro bezpečné a vysokovýkonné provádění na webových stránkách.

SQL.js poskytuje lehkou a platformě nezávislou implementaci databáze, která umožňuje vývojářům pracovat s daty v prohlížeči bez nutnosti serverové komponenty. Hlavní vlastnosti a schopnosti knihovny SQL.js, jak bylo zjištěno z analýzy oficiální dokumentace [14], zahrnují:

- **Vytváření virtuální databáze**. SQL.js vytváří databázi v operační paměti prohlížeče, což zajišťuje rychlé a efektivní ukládání a načítání dat.
- **SQL dotazy**. Knihovna podporuje většinu SQL operátorů, což umožňuje provádět dotazy na vytváření, aktualizaci, mazání a výběr dat, stejně jako manipulaci s indexy, omezeními a transakcemi.
- **Konverze datových typů**. SQL.js automaticky převádí datové typy SQLite na odpovídající JavaScriptové typy a naopak, což poskytuje pohodlné rozhraní pro práci s daty.
- **Import a export dat**. Knihovna umožňuje importovat existující databáze SQLite ve formátu souborů, stejně jako exportovat virtuální databáze do souborů pro další použití a uložení.
- **Bezpečnost a důvěrnost**. Jelikož jsou veškerá data zpracovávána a ukládána lokálně v prohlížeči uživatele, sql.js poskytuje vyšší úroveň bezpečnosti a důvěrnosti dat.
- **Omezení**. Jedním z nedostatků SQL.js je omezení velikosti databáze kvůli omezené operační paměti prohlížeče. Je také třeba vzít v úvahu, že data v operační paměti jsou dočasná a po zavření záložky nebo prohlížeče budou ztracena.

Celkově lze říci, že knihovna SQL.js spolu s dalšími technologiemi pro ukládání dat na straně klienta poskytuje optimální řešení pro vyvíjenou aplikaci, které umožňuje dosáhnout cílů projektu s minimálními náklady na vývoj a údržbu.

Pro inicializaci knihovny SQL.js v jazyce JavaScript slouží funkce „initSqlJs“. Důležitým je načtení souboru sql-wasm.wasm, který je potřebný pro správnou funkčnost SQL.js:

```
const initSqlJs = require('sql.js');

const SQL = await initSqlJs({
  // Načtení wasm souboru a inicializace sql.js.
  // Při spuštění v nodu můžete locateFile zcela vynechat.
  locateFile: file => "https://sql.js.org/dist/${file}"
});

// Vytvoření databáze
const db = new SQL.Database();

//Provádění dotazů
const res = db.exec("SELECT * FROM hello");
//Vrací [{columns:['a','b'], values:[[0,'hello'],[1,'world']]}]
```

6.1.2 SQLite

Použití syntaxe SQLite je způsobeno použitím knihovny SQL.js, která je adaptací databáze SQLite pro použití na straně klienta. Protože SQL.js poskytuje virtuální databázi SQLite v prohlížeči, je nutné dodržovat pravidla a syntaxi jazyka SQL pro správné provádění SQL dotazů a manipulaci s daty.

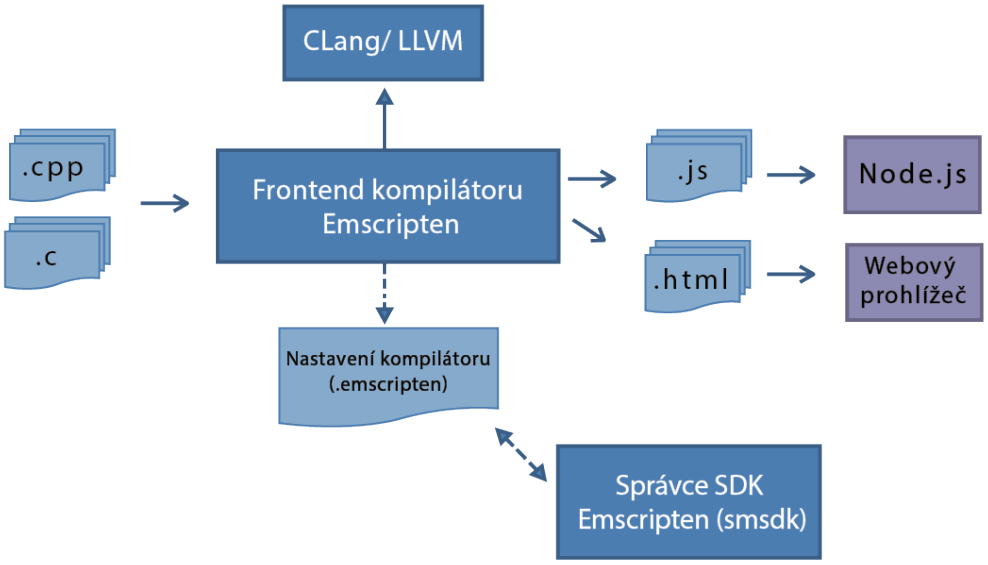
Syntaxe SQLite zahrnuje standardní SQL operátory, jako je SELECT, INSERT, UPDATE a DELETE, a podporuje vytváření a modifikaci tabulek, používání omezení, indexů, triggerů a dalších prvků databáze. Je důležité dodržovat syntaktická pravidla SQLite při psaní SQL dotazů, aby bylo zajištěno jejich správné provedení v rámci knihovny SQL.js a emulované databáze SQLite.

6.1.3 Emscripten a Webassembly

WebAssembly (Wasm) a Emscripten hrají důležitou roli v kompilaci SQLite do formátu WebAssembly a jeho spuštění v prohlížeči. WebAssembly je nízkoúrovňový binární formát, který zajišťuje vysoký výkon v webovém prohlížeči, umožňuje kompilaci kódu z různých programovacích jazyků, včetně C a C++ [20]. Emscripten je nástroj, který převádí kód v jazyce C a C++ na WebAssembly a JavaScript s využitím kompilátoru LLVM (Low Level Virtual Machine). Emscripten rovněž poskytuje knihovny a nástroje pro interakci mezi kódem WebAssembly a JavaScriptem [21].

V rámci převodu SQLite na WebAssembly kompiluje Emscripten kód SQLite napsaný v jazyce C do WebAssembly, čímž zajišťuje vysoký výkon a efektivitu provádění kódu SQLite v prohlížeči. Emscripten také vytváří JavaScriptové obaly pro interakci s modulem SQLite na WebAssembly, poskytující pohodlná API a funkce pro práci s databází SQLite pomocí JavaScriptu [20].

Takovým způsobem díky WebAssembly a Emscriptenu lze využít potenciál SQLite v webovém prohlížeči, což umožňuje dosáhnout vysokého výkonu a efektivity při provádění dotazů, které dříve vyžadovaly nasazení na serverové straně nebo použití omezených možností JavaScriptu.



Obrázek 6.1. Vysokoúrovňový pohled na sadu nástrojů Emscripten [21].

6.2 Vlastnosti práce s React

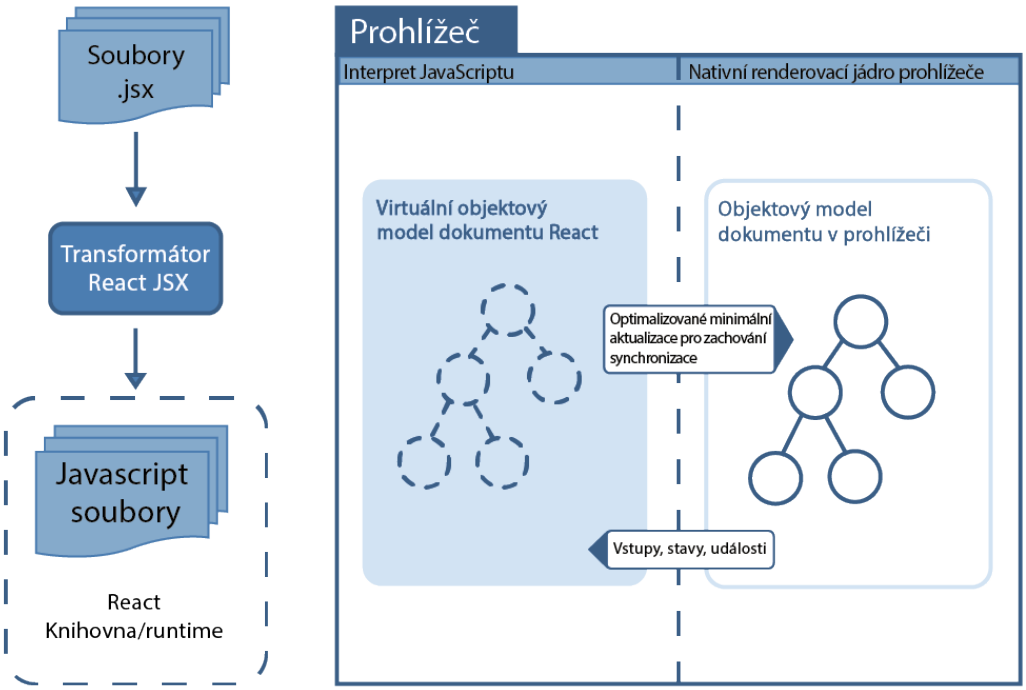
6.2.1 Virtuální DOM

Objektový model dokumentu (DOM, Document Object Model) je složitá struktura, reprezentováná XML nebo HTML, která poskytuje vnitřní reprezentaci webové stránky v kontextu operační paměti prohlížeče. Ve struktuře DOM se odrážejí všechny prvky stránky, od prostých textových bloků po komplexní grafické komponenty. Je důležité zdůraznit, že jakékoliv manipulace s prvky DOM mohou vyvolat proces překreslování stránky prohlížečem, který vyžaduje značné výpočetní zdroje. Pokud frekvence takových změn dosáhne příliš vysoké hodnoty, může to vést k výraznému poklesu výkonnosti a zhoršení uživatelského zážitku.

Jako reakci na tento problém JavaScriptová knihovna React používá koncept virtuálního DOM. Místo přímé interakce s reálným DOM při každé potřebě změny, React předem vytváří lehkou „virtuální“ repliku DOM (VDOM) v paměti (viz obrázek 6.2). Tento virtuální DOM má mnohem vyšší rychlost a efektivitu změn oproti skutečnému DOM.

Po provedení změn ve virtuálním DOM, React provádí proces „rekoncilace“, porovnávající nový virtuální DOM s jeho předchozí verzí. Během tohoto procesu React určuje minimální sadu operací, které je třeba provést na skutečném DOM, aby byla dosažena shoda s aktualizovaným virtuálním DOM. Tyto změny jsou poté aplikovány v prohlížeči současně, což z hlediska výkonnosti představuje mnohem optimálnější přístup než postupné aplikování každé změny zvlášť [22].

Virtuální DOM je klíčovou komponentou, která dělá z Reactu silný nástroj pro tvorbu interaktivních uživatelských rozhraní. Hraje centrální roli v zajištění vysoké výkonnosti React, dokonce i v kontextu rozsáhlých a složitých aplikací.



Obrázek 6.2. Vizualizace procesu změny DOM v React [23].

6.2.2 Komponentní přístup

V kontextu knihovny Reac komponent představuje samostatný, nezávislý a mnohokrát použitelný blok kódu, který řídí konkrétní segment uživatelského rozhraní webové aplikace. Komponenty interagují s libovolnými vstupními daty nazývanými „propsy“ a vrací React prvky, které popisují, jak by mělo být vizuální zobrazení na obrazovce. React poskytuje dva typy komponent: funkční komponenty a komponenty tříd. Ve verzích React od 16.8 byly přidány nástroje zvané „hooks“, které umožňují funkčním komponentám mít stav a flexibilněji řízený životní cyklus, takže se funkční metoda stala preferovaným způsobem vytváření komponent v Reactu díky své jednoduchosti a pohodlí [24].

V celkové struktuře aplikace jsou komponenty obvykle uspořádány do hierarchické struktury. To znamená, že některé komponenty mohou obsahovat jiné a vytvářet složité stromové struktury komponent (viz obrázek 6.2). Tato hierarchická struktura usnadňuje správu kódu a aktualizaci uživatelského rozhraní, protože změny provedené v jednom komponentu neovlivňují ostatní komponenty.

Významnou charakteristikou komponentů v Reactu je jejich vlastní „stav“ a schopnost ho řídit. Stav komponentu je objekt obsahující data specifická pro daný komponent, která se mohou měnit během jeho životního cyklu. Když se změní stav komponentu, automaticky se provede opětovné vykreslení komponentu, což umožňuje zobrazit všechny provedené aktualizace na obrazovce [24].

React také uplatňuje princip jednosměrného toku dat, který říká, že data proudí jedním směrem, obvykle od rodičovské komponenty k jejím potomkům. To znamená, že změny dat v rodičovské komponentě se přenesou do jejích potomků, ale ne naopak. Komponenty potomků nemohou přímo měnit data, která jim byla předána, mohou pouze přijímat a zobrazovat data nebo spouštět akce, které o změně informují rodičovskou komponentu [24]. Vzor jednosměrného toku dat podporuje jasný a předvídatelný tok

dat v celé aplikaci. To pomáhá zjednodušit pochopení vzájemné interakce komponent a usnadňuje analýzu stavu aplikace v daném okamžiku.

6.2.3 React Hook

React Hooks představují sadu funkcionalit, které umožňují rozšířené využití možností Reactu, včetně správy stavu a životního cyklu, v rámci funkčních komponentů, přičemž eliminují potřebu jejich konverze na třídní komponenty. Hooky, které byly poprvé uvedeny ve verzi React 16.8 v roce 2019 [25], nové možnosti pro správu stavu a vedlejších efektů v React aplikacích. Kromě základní sady hooků React nabízí možnost vytvoření vlastních, uživatelských hooků, které se implementují kombinací již existujících. To umožňuje inkapsulovat složitou logiku funkčních komponentů do samostatných, opakovaně použitelných funkcí. V důsledku toho aplikace této technologie přispívá ke zvýšení čistoty a čitelnosti kódu a zjednodušuje procesy opakovaného použití a testování komponentů. V rámci implementace tohoto projektu byly použity následující hooky:

- **useState**. Tento hook umožňuje integraci stavu do funkční komponenty. Výsledkem jeho volání je pár hodnot: aktuální stav a funkce pro jeho aktualizaci.
- **useEffect**. „useEffect“ plní funkci provádění vedlejších efektů v funkčních komponentách. Může být použit pro různé úkoly, včetně provádění požadavků na API, správy časovačů a poslechu událostí.
- **useContext**. Tento hook umožňuje použití kontextu, aniž by bylo nutné obalit komponentu do Consumer. useContext poskytuje přístup k aktuální hodnotě kontextu pro konkrétní komponentu.
- **useRef**. „useRef“ se používá pro vytváření odkazů na prvky. Výsledkem volání tohoto hooku je měnitelný ref-objekt, jehož vlastnost .current je inicializována předaným argumentem a udržuje svou hodnotu po celý životní cyklus komponenty.
- **useForm**. Hook z knihovny React Hook Form [26], který poskytuje funkce pro správu stavu formuláře. Umožňuje spravovat hodnoty vstupních polí, validaci, zpracování odeslání formuláře a další aspekty práce s formuláři. Tento hook výrazně zjednodušuje vytváření a správu formulářů v React aplikaci, čímž kód činí čistějším a srozumitelnějším.
- **useNavigate**. Hook, který je součástí knihovny React Router v6 a vyšší [27], poskytuje navigační funkce pro přechod mezi trasami v aplikaci. „useNavigate“ vrací funkci, kterou lze zavolat s cestou pro přesměrování. To činí navigaci více deklarativní a snadno použitelnou uvnitř funkčních komponentů.

6.2.4 Založení projektu

Pro vytváření projektů React a práci s nimi je potřeba nainstalovat Node.js a jeho správce balíčků npm. Pro zahájení práce jsem použila šablonu create-react-app, poskytnutou knihovnou SQL.js, která nabízí základní možnosti práce s knihovnou.

Použití SQL.js v projektu React může způsobit problémy s výchozími nastaveními sestavení, které poskytuje nástroj Create React App (CRA). Je to způsobeno tím, že CRA má předdefinované konfigurace Webpack (module bundler), které nemusí podporovat některé funkce potřebné pro práci se SQL.js, jako je načítání WebAssembly modulů, a proto je nutné použít CRACO.

6.2.5 Konfigurace CRACO

CRACO (Create React App Configuration Override) je nástroj, který umožňuje přizpůsobit konfiguraci CRA bez nutnosti provádět příkaz „eject“. Eject odstraňuje abstrakci CRA, čímž poskytne úplnou kontrolu nad konfigurací projektu, ale může vést ke složitostem v řízení a údržbě.

Použití CRACO umožňuje změnit konfiguraci Webpacku v CRA pro podporu SQL.js bez přímého zásahu do konfiguračních souborů. V souboru craco.config.js v kořenu projektu se definují nezbytné změny konfigurace Webpaku pro podporu provozu Wasm v SQL.js:

```
module.exports = {
  webpack: {
    configure: {
      // See https://github.com/webpack/webpack/issues/6725
      module: {
        rules: [
          {
            test: /\.wasm$/,
            type: "javascript/auto",
          },
        ],
      },
      ...
    }
  }
}
```

6.3 Knihovny použité při implementaci

Kromě knihovny SQL.js popsané dříve v sekci 6.1.1, která slouží k reprodukci databáze založené na SQLite v uživatelském prohlížeči, byly při realizaci projektu použity také níže popsané knihovny.

6.3.1 ReactFlow

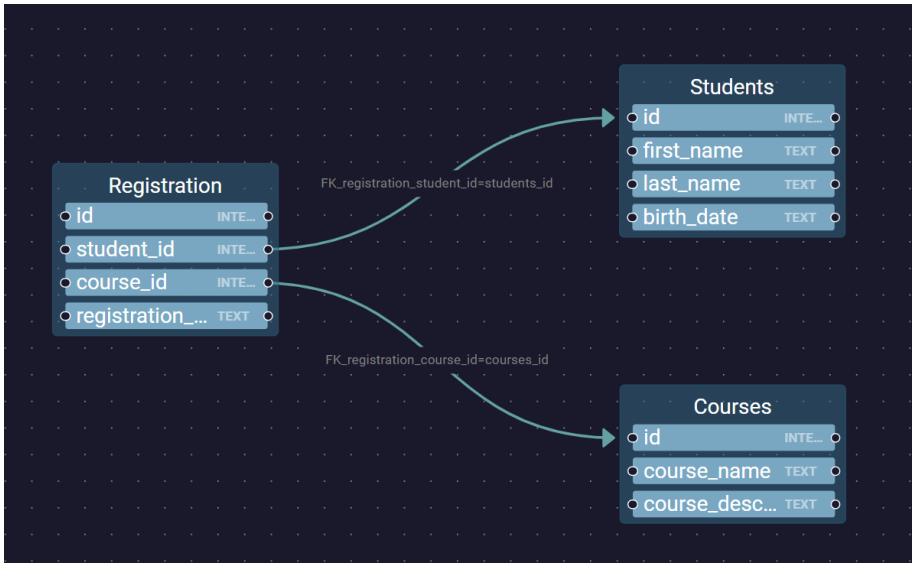
React Flow [28] je vysoce výkonná knihovna pro tvorbu interaktivních diagramů a schémat založená na React. Umožňuje vytváření složitého uživatelského rozhraní související s vizualizací a správou grafických struktur, jako jsou blokové diagramy, rozhodovací stromy, procesy atd.

Z oficiální dokumentace vyplývá, že React Flow je obzvláště efektivní při implementaci funkce přetahování prvků a také poskytuje flexibilitu přizpůsobení a stylizace. Podporuje různé typy uzlů a hran, což umožňuje přidání vlastních uzlů a hran pro vytvoření nejvhodnějších rozhraní pro konkrétní potřeby [28].

S využitím této knihovny byla v rámci projektu zajištěna funkčnost reprezentace nahrané uživatelem databáze prostřednictvím schematického zobrazení tabulek a vztahů mezi nimi (viz obrázek 6.3). V tomto kontextu se každá tabulka vytvořená uživatelem vizualizuje s detaily atributů a odpovídajících datových typů, které obsahuje. Navíc, vztahy mezi tabulkami jsou demonstrovány pomocí orientovaných vazeb založených na jejich cizích klíčích v databázi.

Výhodou této knihovny je možnost uživatelské manipulace s prvky, včetně jejich přetahování, změny měřítka zobrazených diagramů, a také navigace po vizualizované mapě pomocí mini-mapy.

Tento přístup usnadňuje proces analýzy dat o tabulkách z pohledu uživatele a přispívá k formulaci efektivních SQL dotazů na jejich základě.



Obrázek 6.3. Vizualizace databáze pomocí React Flow v aplikaci.

6.3.2 React Router

React Router Dom je klíčovou knihovnou v ekosystému React, určenou pro implementaci navigace a směrování v webových aplikacích založených na React [29]. Obsahuje sadu komponent a funkcí, které usnadňují vytváření složité navigační struktury v deklarativním stylu, což zjednodušuje správu a údržbu kódu. Tato knihovna podporuje dynamické směrování, což umožňuje realizovat různé navigační scénáře, včetně vnořených tras, chráněných tras, přesměrování a mnoho dalšího.

V rámci vývoje aplikace byly použity komponenty této knihovny, jako jsou Routes, Route a HashRouter. Komponenta Routes je používána pro seskupení a koordinaci komponent Route. Zajišťuje zobrazení pouze jedné komponenty Route, která odpovídá aktuálnímu URL, čímž zaručuje jedinečnost každé zobrazené trasy. Samotná komponenta Route vystupuje jako hlavní prvek směrování, definuje trasy a s nimi spojené komponenty. Při shodě URL adresy s danou trasou, Route zajišťuje zobrazení příslušné komponenty.

I když v React Router Dom existuje BrowserRouter, který využívá HTML5 History API pro synchronizaci uživatelského rozhraní s URL v adresním řádku prohlížeče a tvoří URL bez symbolu „#“ [29], pro nasazení aplikace na službě GitHub pages bylo zvoleno využití komponenty HashRouter. To je dáno specifiky GitHub Pages, který podporuje pouze statické webové stránky a neposkytuje funkčnost serverového směrování [30]. Místo toho HashRouter používá hašovací část URL (tj. window.location.hash) pro sledování historie prohlížeče. Takto všechny trasy ve skutečnosti představují kotvy na téže stránce index.html, což je ideální pro statické hostingy, jako je GitHub pages.

6.3.3 React Hook Form

React Hook Form je knihovna pro správu formulářů v React, založená na mechanismu práce s „useForm“. Je navržena s důrazem na optimalizaci výkonu a snadnost použití, také nabízí efektivní funkce pro validaci formulářů [31].

React Hook Form zjednodušuje validaci formulářů minimalizací opakovaného vykreslování a poskytuje správu stavu formulářů pomocí hooků. Knihovna poskytuje API (Application programming interface) založené na React Hook, včetně useForm pro vytváření a správu stavu formulářů, a také useController pro integraci s uživatelskými komponentami formulářů. React Hook Form zjednodušuje zpracování chyb pomocí řízeného objektu chyb, včetně automatické vizualizace zpětné vazby o chybách. To umožňuje snadno poskytovat uživatelům informativní zprávy o chybách validace.

V rámci realizace projektu byl použit hook „useForm“ pro správu stavu složitých dynamických formulářů, například při vytváření prvků FORM nebo SELECT dotazů v aplikaci. Tyto formuláře představují neklasické konstrukce, které realizují velké množství různorodé logiky při přidávání prvků, což potenciálně mohlo způsobit obtíže při vývoji bez použití této knihovny. Nástroje, které tato knihovna poskytuje, umožňují snadno sledovat přidávání, mazání a změnu prvků formuláře, stejně jako zpracovávat složitou validaci a zpracování chyb. To výrazně zjednodušuje vývoj a správu stavu dynamických formulářů, což umožňuje soustředit se na důležitější aspekty logiky aplikace.

6.3.4 React Awesome Reveal

React Awesome Reveal je sada vyso výkonných animačních komponent pro React, které využívají React Reveal pro poskytování omezené sady animací, které lze aplikovat na různé komponenty v aplikaci [32]. Tyto animace vytvářejí různé vizuální efekty, které pomáhají zlepšit uživatelský zážitek z aplikace, čímž ji činí interaktivnější a atraktivnější.

React Awesome Reveal obsahuje mnoho přednastavených efektů, včetně Fade, Zoom, Flip, Rotate a mnoha dalších. Také nabízí možnost přizpůsobení efektů pomocí konfigurace vlastností, což dává více kontroly nad tím, jak a kdy se animace zobrazí. Všechny efekty jsou založeny na CSS, což zajišťuje vysoký výkon a kompatibilitu s většinou prohlížečů.

Použití této knihovny lze vidět na hlavní stránce aplikace „Home“, kde se při skrolování stránky postupně objevují prvky s popisem funkčnosti aplikace. Tento efekt byl dosažen díky komponentu Fade z této knihovny, který implementuje animaci objevení se prvků, čímž zlepšuje vizuální složku webu.

6.3.5 React Icons

React Icons je užitečná knihovna pro React, která zahrnuje kolekce ikon z populárních knihoven, jako jsou FontAwesome, Ionicons, Material Design a další [33]. Tato knihovna umožňuje přidávat různé ikony do aplikace bez zvláštního úsilí a také optimalizuje finální balíček aplikace tím, že do něj zahrnuje pouze ty komponenty, které byly skutečně použity v aplikaci. Tyto ikony mohou být upraveny podle barvy, velikosti a dalších vlastností, což umožňuje rychle a flexibilně nastavit potřebné rozhraní.

6.4 Stylování pomocí CSS

CSS (Cascading Style Sheets) se používá k stylování HTML prvků webových stránek. V kontextu React CSS hraje důležitou roli ve formátování komponent. V projektu byl

použit modulární přístup k CSS, protože standardní způsob stylování bez CSS modulů může vést ke konfliktům jmen, problémům s oblastí viditelnosti a obtížím při údržbě kódu.

V modulárním CSS má každá React komponenta svůj vlastní CSS soubor, který je zodpovědný pouze za stylování této konkrétní komponenty. Toto izoluje styly každé komponenty, čímž se předchází nežádoucí interakci a konfliktům stylů. Tento přístup zvyšuje opakovatelnost a udržitelnost kódu, protože změny v stylech jedné komponenty neovlivňují ostatní. K použití modulárního CSS v Reactu je nutné vytvořit .module.css soubor pro každou komponentu a importovat jej přímo do souboru komponenty, přičemž se nastaví určitý classname pro stylování objektů [34].

Příklad modulárního přístupu na příkladě komponenty Home.jsx:

```
import style from "./Home.module.css";
...
return (
        ⟨div className={style.homePage}⟩
            ...
        ⟨/div⟩
);
```

V tuto chvíli vypadá soubor Home.module.css jako obvyklý CSS:

```
.homePage {
    width: 100%;
    background-image: url("../images/back.png");
    ...
};
```

6.5 Vizualizace dat

V této práci je hlavní pozornost věnována otázce vizualizace dat po vytvoření SQL dotazů uživatelem. Myšlenka spočívá v tom, aby uživatel mohl nejen vytvářet a provádět dotazy, ale také sledovat názorné zobrazení výsledků těchto dotazů.

Pro dosažení stanoveného cíle se nejoptimálnějším způsobem jeví parsování uživatelských dotazů, následné zpracování pomocí logických funkcí aplikace a konečná etapa - vizualizace získaných výsledků. Je důležité zdůraznit, že proces vizualizace není monolitický, ale je rozdělen na několik fází. To je dáno specifikou SQL a různými operacemi, které uživatel může vybrat při vytváření dotazu SELECT z příslušné nabídky operací, definovaných v tabulce 3.2.

Při přechodu mezi fázemi vizualizace má uživatel možnost postupně vidět vliv každé z vybraných operací na konečná data. Takový přístup nejen zpřehledňuje vizualizaci pro uživatele, ale také mu umožňuje lépe pochopit, jak fungují SQL dotazy a jaké výsledky mohou dát.

Jako metoda vizualizace byla zvolena vizualizace dat ve formě kruhů. Každý řádek dat z výsledkové tabulky dotazu je zobrazen jako samostatný kruhový grafický prvek v kontejneru vizualizace. Počet těchto grafických prvků tedy odpovídá počtu řádků ve výsledkové tabulce dotazu. Kromě toho je vizualizace umístěna přímo vedle uživatelského rozhraní pro vytváření dotazů. To umožňuje uživateli rychle a názorně hodnotit výsledky svých dotazů, čímž proces interakce se SQL stává intuitivnějším a vzdělávacím.

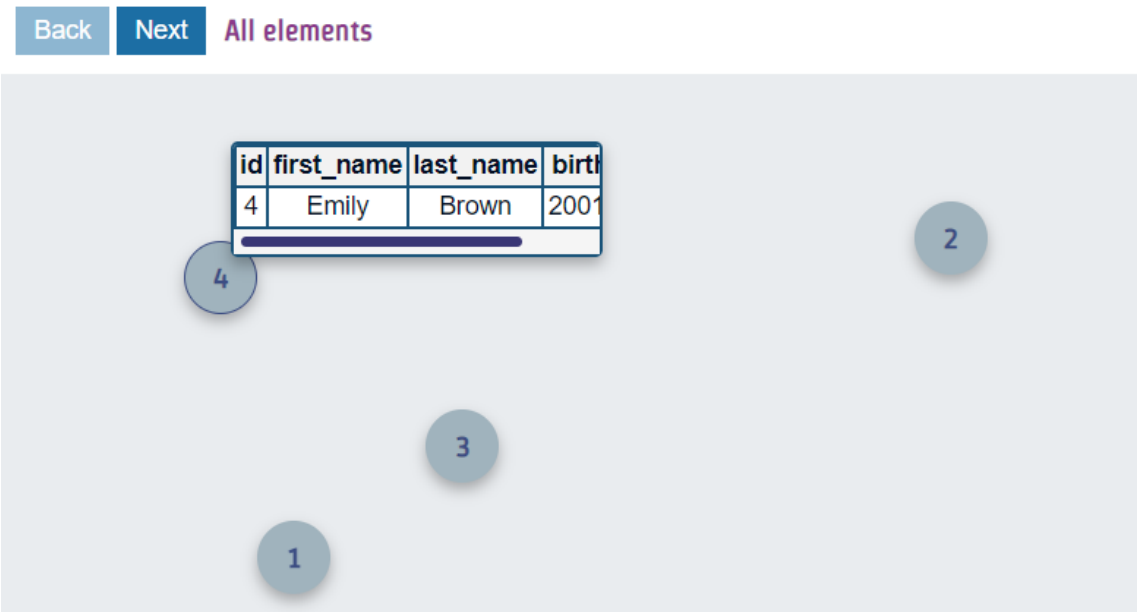
6.5.1 Umístění grafických prvků

Během implementace zobrazení grafických prvků se objevila otázka jejich efektivního umístění v kontejneru vizualizace. Původní myšlenkou bylo generovat pozici každého prvku náhodně, s cílem dodat každé vizualizaci jedinečný charakter, odlišný od předchozích.

Nicméně při zavádění tohoto přístupu se objevil problém překryvu grafických prvků. Takové překrytí bránilo správnému vnímání vizualizace, protože ji činilo méně čitelnou. S rostoucím počtem grafických prvků rostlo riziko jejich navzájem se překrývající, což problém ještě zhoršovalo.

K vyřešení tohoto problému bylo rozhodnuto použít klasický algoritmus pro hledání a odstraňování kolizí. Tento algoritmus umožnil efektivně detekovat překrývající se grafické prvky a korigovat jejich pozici tak, aby byla zajištěna maximální čitelnost vizualizace.

Je důležité zdůraznit, že tento přístup nejen zvyšuje čitelnost vizualizace, ale také zachovává její jedinečnost, protože počáteční umístění prvků je stále generováno náhodně. Nakonec, díky tomuto vylepšení se získaná vizualizace stala jasnější a informativnější, což uživateli usnadňuje proces analýzy výsledků SQL dotazů.



Obrázek 6.4. Příklad vizualizace SQL dotazu v aplikaci.

6.5.2 AABB algoritmus

Pro řešení otázky zpracování kolizí mezi kruhovými 2D prvky v této práci bylo rozhodnuto použít v komponentě „DataCircleContainer“ variantu algoritmu Axis-Aligned Bounding Box (AABB). AABB je metoda pro detekci kolizí, která využívá omezující obdélníky zarovnané na souřadnicové osy [35].

Tradičně se AABB používá pro zpracování kolizí mezi obdélníky, ale je možné ho přizpůsobit i pro kruhy. V tomto případě je každý kruh popsán omezujícím obdélníkem, který je zarovnán podle os. Hranice tohoto obdélníka odpovídají maximálním a minimálním hodnotám X a Y kruhu.

Kontrola kolizí probíhá ve dvou fázích. V první fázi se provádí rychlá kontrola kolizí AABB. Pokud se omezující obdélníky dvou kruhů nepřekrývají, lze s jistotou prohlásit,

že se ani samotné kruhy nepřekrývají. Tato fáze umožňuje snížit množství složitějších a výpočetně náročnějších kontrol kolizí kruhů. Pokud se však AABB překrývají, provádí se druhá fáze kontroly - přesná kontrola kolizí kruhů. K tomu se vypočítá vzdálenost mezi středy kruhů. Pokud je tato vzdálenost menší nebo rovna součtu poloměrů kruhů, kruhy se překrývají.

Tato dvoustupňová kontrola kolizí umožňuje významně urychlit zpracování kolizí. Rychlá kontrola překryvu AABB filtruje většinu případů, kdy je kolize nemožná, čímž snižuje množství složitějších a nákladnějších výpočtů ve druhé fázi.

Proto algoritmus AABB představuje efektivní a relativně jednoduchý přístup k detekci kolizí, zejména při práci s malým množstvím kruhových 2D prvků, protože pro jeho použití není třeba složitých matematických operací, jako je trigonometrie nebo maticové transformace (například v algoritmu SAT). Díky své jednoduchosti a efektivnosti může sloužit jako spolehlivé řešení pro mnoho grafických a herních aplikací.

6.5.3 Druhy vizualizace

V aplikaci lze rozlišit čtyři klíčové fáze vizualizace dat. Specifika zobrazení každé fáze značně závisí na SQL dotazu, který sestavil uživatel. Je důležité poznamenat, že některé operace, jako jsou WHERE nebo GROUP BY, mohou iniciovat specifické zpracování dotazu a následnou vizualizaci.

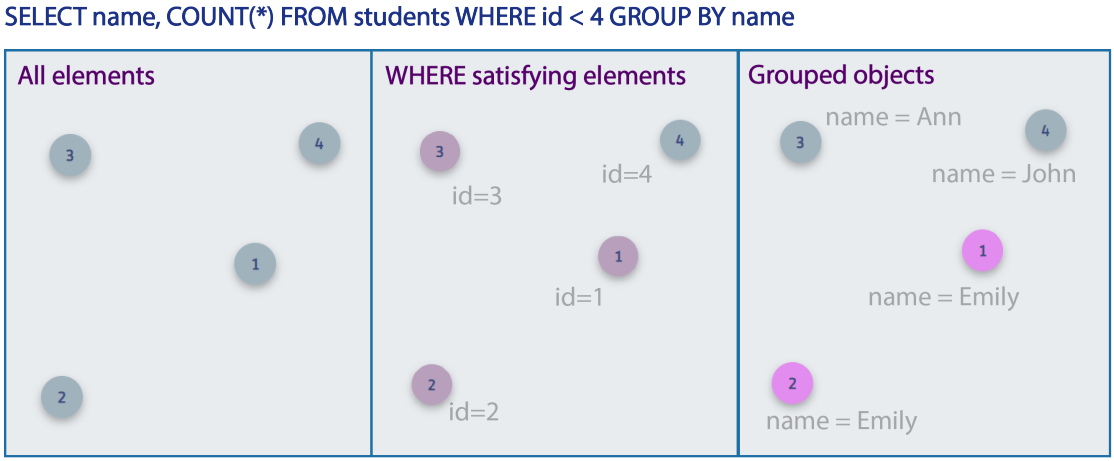
První fáze vizualizace začíná hned po tom, co je dotaz sestaven uživatelem. Na této fázi uživatel vidí vizualizaci všech prvků, které odpovídají dotazu SELECT * FROM selectedTable. Uživatel tak získává obecnou představu o datech prezentovaných ve vybrané tabulce.

Druhá fáze vizualizace je spojena s použitím podmínek WHERE v SQL dotazu. Pokud dotaz obsahuje výraz WHERE, zpracování dotazu se provádí tak, aby výsledek zahrnoval pouze prvky z vybraných tabulek, které odpovídají stanoveným kritériím WHERE. To se děje bez ohledu na přítomnost dalších konfigurací dotazu. Takto uživatel může vidět výsledky dotazu SELECT * FROM selectedTable WHERE condition, což mu umožňuje detailněji prozkoumat data odpovídající stanoveným podmínkám.

Ve třetí fázi vizualizace se zpracovávají dotazy obsahující kritérium GROUP BY a jednu nebo více agregačních funkcí z nabízených možností. Na tomto stupni dochází k seskupování prvků podle uvedených kritérií. Pro každou skupinu prvků se generuje náhodná barva, což usnadňuje vizuální oddělení prvků do skupin. Tato fáze umožňuje uživateli vidět, jak jsou data seskupena podle stanovených kritérií.

Poslední, čtvrtá fáze vizualizace ukazuje data s ohledem na všechna kritéria uvedená v uživatelském SQL dotazu. Vizualizovaná data plně odpovídají výsledné tabulce dotazu. To umožňuje uživateli vidět konečný výsledek zpracování dotazu a porovnat ho s vizualizací.

Celkově, díky možnosti přepínání mezi fázemi vizualizace, může uživatel hlouběji pochopit, jak jednotlivé operace ovlivňovaly data v tabulkách a jakým způsobem byl získán konečný výsledek dotazu.



Obrázek 6.5. Etapy vizualizace v aplikaci.

6.6 Struktura projektu

Struktura projektu založeného na Reactu hraje klíčovou roli v zajišťování přehledné navigace, modularity a snadné údržby kódu. Při implementaci tohoto projektu jsem usilovala o jeho rozdělení do kompaktních a přehledných částí, což značně usnadňuje orientaci v kódu a jeho následnou správu. Struktura projektu je podrobně popsaná v priloze A.

Každý konečný balíček obsahuje potřebné React komponenty, modulární styly aplikovatelné na ně, stejně jako soubor pro testování. Příklad je uveden v příloze A, kde je popsána podrobná struktura jednotlivých balíčků.

6.7 Nasazení aplikace

Nasazení aplikací vytvořených na základě React na GitHub Pages je běžnou praxí, která umožňuje pohodlné prezentování a demonstraci vyvinutých projektů. Tento proces je poměrně jednoduchý, ale vyžaduje určité porozumění základů práce se systémem kontroly verzí Git a hostingovou službou GitHub.

Prvním krokem je vytvoření repozitáře na GitHub a nahrání kódu do tohoto repozitáře. Před nasazením aplikace je nutné nainstalovat balíček gh-pages pomocí npm (Node Package Manager) nebo Yarn. Tento balíček poskytuje nezbytné nástroje pro nasazení aplikací na GitHub Pages.

Poté následuje modifikace souboru package.json. V souboru package.json aplikace je třeba přidat pole homepage, které ukazuje na adresu stránky, na které bude aplikace umístěna. Navíc jsou v sekci „scripts“ přidány příkazy pro sestavení a nasazení aplikace (například „predeploy“ a „deploy“). Pouze poté je možná další publikace aplikace na této službě [30].

Tento přístup k nasazování React aplikací na GitHub Pages nabízí pohodlný způsob prezentace vyvinutých projektů a umožňuje sledovat všechny změny díky systému kontroly verzí Git.

Produkční verze aplikace nasazená na GitHub Pages je k dispozici na adrese `https://noamorii.github.io/SQLsim-frontend/`.

Kapitola 7
Testování

Testování v oblasti vývoje softwaru hraje klíčovou roli a React není výjimkou. Testování pomáhá zajistit, že aplikace funguje tak, jak se očekává, a zároveň zajišťuje její stabilitu a spolehlivost v případě změny kódu nebo přidání nových funkcí. Navíc přispívá k udržení vysoké kvality kódu a snižuje čas a úsilí, které by musely být vynaloženy na opravu chyb v pozdějších fázích vývoje.

7.1 Testování komponent

Testování komponent React umožňuje ověřovat jejich funkčnost tím, že se zkoumá, jak reagují na různé vstupní data (props) a stavy, a zároveň se ujistit, že správně interagují s dalšími částmi aplikace. To může zahrnovat ověřování, že komponenta zobrazuje správný výstup na základě vstupních dat, správně zpracovává interakce uživatele, nebo správně aktualizuje svůj stav v reakci na změny vstupních dat nebo vnějších vlivů.

7.1.1 Typy testování

Existují různé přístupy k testování komponent React, které zahrnují:

- **Jednotkové testování (unit testing)**. Jednotkové testování je přístup, při kterém jsou jednotlivé komponenty testovány nezávisle na sobě. Jednotkové testy jsou obvykle používány k ověření malých, izolovaných částí kódu a jejich chování.
- **Integrační testování**. Tento přístup testuje interakci mezi různými komponentami a jak spolu pracují. To může zahrnovat ověřování, jak komponenty reagují na změny stavu ostatních a jak zpracovávají přenos dat.
- **E2E (End-to-End) testování**. Toto je přístup, při kterém se testuje celý tok práce aplikace v podmínkách co nejvíce přibližujících se skutečnosti. E2E testování se používá k ověření kompletního toku funkčnosti aplikace od začátku do konce.

V rámci některých projektů může být vhodné zvolit uživatelské testování jako alternativu k E2E testování. Přestože E2E testování poskytuje komplexní prohlídku funkcionality aplikace, uživatelské testování nabízí řadu jedinečných výhod, které mohou být v kontextu dané aplikace ještě hodnotnější (viz sekce 7.3).

7.2 Knihovny použíté při testování

Testovací knihovny v Reactu poskytují sadu nástrojů a funkcí, které výrazně zjednodušují proces psaní a provádění testů. Nejsou striktně povinné, ale poskytují řadu výhod, díky kterým jsou v procesu vývoje velmi cenné.

7.2.1 Jest

Jest je jednou z nejčastěji používaných knihoven pro testování JavaScriptu a má řadu klíčových vlastností, které ho činí obzvláště vhodným pro testování aplikací v React. Je otevřený a je široce používán pro jednotkové a integrační testování.

Jednou z hlavních funkcí Jest je schopnost napodobit chování funkcí a modulů. To umožňuje kontrolovat a izolovat vnější závislosti v testech, což zjednodušuje proces testování. V implementaci komponent tohoto projektu mnoho z nich využívá sdílený kontext pro interakci s databází, takže testování by se neobešlo bez této knihovny, protože umožňuje snadno simulovat chování volání kontextu nebo jakýchkoli dalších funkcí a modulů. Příklad použití Jest pro jednotkové testování simuluící databázi v aplikaci:

```
describe('FromQueryForm Component', () => {
    let mockDb, mockSql;

    beforeEach(() => {
        // Simulace databázového objektu.
        mockDb = { exec: jest.fn() };
        mockSql = { Database: function() { return mockDb; } };
    });

    it('should render form correctly', () => {
        render(
            ⟨SqlContext.Provider value={{ sql: mockSql, setSql: jest.fn() }}⟩
            ⟨DbContext.Provider value={{ db: mockDb}}⟩
            ...
        );
        ...
    });
```

Navíc Jest nabízí vestavěnou podporu pro testování asynchronního kódu. Vestavěné mechanismy umožňují testovat sliby (promises), asynchronní funkce a zpětné volání (callbacks), a ujistit se tak o jejich správnosti. Jest také poskytuje nástroje pro analýzu pokrytí kódu. To umožňuje určit, které části kódu byly otestovány a které ne, čímž pomáhá zlepšit kvalitu testů a zjednodušuje sledování úseků kódu, které vyžadují další testování.

7.2.2 RTL

React Testing Library (RTL) je populární knihovnou pro testování komponent React. Tato knihovna je zaměřená na chování namísto implementace, což umožňuje psát testy blíže k tomu, jak uživatelé interagují s kódem, v důsledku čehož jsou testy spolehlivější a méně ohrožené rozbitím při změně kódu. React Testing Library neposkytuje přístup k vnitřnostem komponenty a proto umožňuje přímou interakci se stavem nebo životním cyklem komponenty. To pomáhá vyhnout se překomplikování testů a psát je s větší mírou jistoty o jejich správnosti.

Jedním z klíčových principů React Testing Library je idea testování rozhraní tak, jak ho vidí uživatel. To znamená, že testy jsou více zaměřené na to, co komponenta ukazuje nebo jak reaguje na uživatelské akce, než na zaměření se na vnitřní strukturu komponenty. To přispívá k vytváření testů, které realističtěji odrážejí skutečné použití aplikace.

Hlavními funkcemi tohoto frameworku jsou render pro vykreslení komponent, fireEvent pro simulaci uživatelských akcí a množství funkcí dotazů, které pomáhají najít prvky na stránce pro kontrolu jejich stavu nebo chování. Příklad použití RTL při testování aplikace (FromQuery.text.jsx):

```
it('should display an error when "+" button is clicked', async () => {
        render(
            ...
        );

        const plusButton = screen.getByText('+');
        await userEvent.click(plusButton);

        const errorMessage = screen.findByText(/please set the tables/);
        expect(errorMessage).toBeInTheDocument();
    });
```

7.3 Uživatelské testování

Uživatelské testování je jedním z nejdůležitějších prvků v procesu vývoje, protože umožňuje získat cennou zpětnou vazbu o tom, jak cílová skupina interaguje s aplikací a jak ji vnímá. Tato metoda testuje předpokládané chování a interakci uživatelů s produktem v kontrolovaném prostředí. Použití uživatelského testování jako součásti procesu vývoje může přispět k vytvoření uživatelsky přívětivějších a efektivnějších aplikací, které splňují potřeby a očekávání koncového uživatele [36].

Existuje několik důvodů, proč je uživatelské testování důležitou součástí procesu vývoje [36]:

- **Porozumění uživatelům**. Uživatelské testování poskytuje podrobné porozumění tomu, jak cílová skupina interaguje s produktem. To umožňuje vývojovým týmům identifikovat implicitní a explicitní potřeby a očekávání uživatelů.
- **Identifikace problémů s použitelností**. Testování použitelnosti umožňuje identifikovat potenciální problémy s rozhraním nebo funkcionalitou produktu, které mohou bránit jeho efektivnímu využití.
- **Úspora času a zdrojů**. Prevence problémů s použitelností v rané fázi vývoje může ušetřit značné zdroje, které by jinak musely být vynaloženy na nákladné a pracné opravy po uvedení produktu na trh.
- **Zlepšení spokojenosti uživatelů**. Když je produkt snadno a příjemně použitelný, zvyšuje to celkovou spokojenost uživatele, což nakonec vede ke zvýšení loajality a udržení uživatelů.

Význam uživatelského testování nelze podceňovat. Bez tohoto kroku by byl vývoj založen pouze na vlastních předpokladech o tom, jak budou uživatelé s aplikací interagovat, což by mohlo vést k vytvoření produktu, který nesplňuje skutečné potřeby uživatelů.

7.3.1 Metodika testování

Byla provedena série dotazů mezi skupinou pěti uživatelů, kteří představují klíčovou cílovou skupinu. Každý z těchto uživatelů prošel připraveným dotazníkem „Hodnocení uživatelské zkušenosti s aplikací SQL simulator“, který zahrnoval řadu úkolů spojených s interakcí se základními funkcemi aplikace. Každý uživatel odpověděl na otázky týkající se jeho zkušeností s používáním aplikace. Všechny otázky obsažené v dotazníku lze najít v příloze B.

Během testování měli uživatelé možnost interagovat s hlavními funkcemi aplikace, včetně importu dat, vizualizace databáze, tvorby a provádění SQL dotazů a interpretace výsledků. Po každém kroku uživatelé sdíleli své zkušenosti a problémy, se kterými se během interakce setkali.

Cílem tohoto testování nebylo pouze posoudit schopnost uživatelů úspěšně využívat jednotlivé funkce, ale také získat cenné poznatky o jejich interakčních vzorcích a preferencích. Tyto informace jsou klíčové pro následné zlepšení a optimalizaci aplikace.

7.3.2 Výsledky uživatelského testování

Výsledky průzkumu byly zaznamenány a analyzovány. Všechny recenze, které byly získány od uživatelů, byly posouzeny z hlediska potřeb a preferencí uživatelů, stejně jako možných problémů a nedostatků, které by mohly ovlivnit celkový uživatelský zážitek.

Výsledky testování ukázaly, že 60% uživatelů úspěšně používalo aplikaci bez naražení na jakékoliv problémy. Nicméně zbývajících 40% uživatelů se setkalo s některými obtížemi a předložilo doporučení pro zlepšení. Je důležité poznamenat, že žádný z uživatelů nenarazil na kritické chyby, což svědčí o tom, že všichni uživatelé byli schopni úspěšně splnit úkoly (viz obrázek 7.1).

Mezi uživateli, kteří měli problémy, byla nejčastější obtížnost pochopení rozhraní. Z údajů vyplynulo, že uživatelé naráželi na problémy při pochopení procesu importu databáze, manipulace s prvky během tvorby dotazů z operačního menu a při orientaci v přepínacích režimech vizualizačního rozhraní. Níže jsou uvedeny několik recenzí od uživatelů týkajících se rozhraní.

- „Vložení pod sekcí import bylo trochu neintuitivní.“
- „Chtělo by se tam přidat výběr z existujících sloupců zadané databáze.“
- „Nejdříve jsem si nevšiml, že vizualizace má několik fází.“

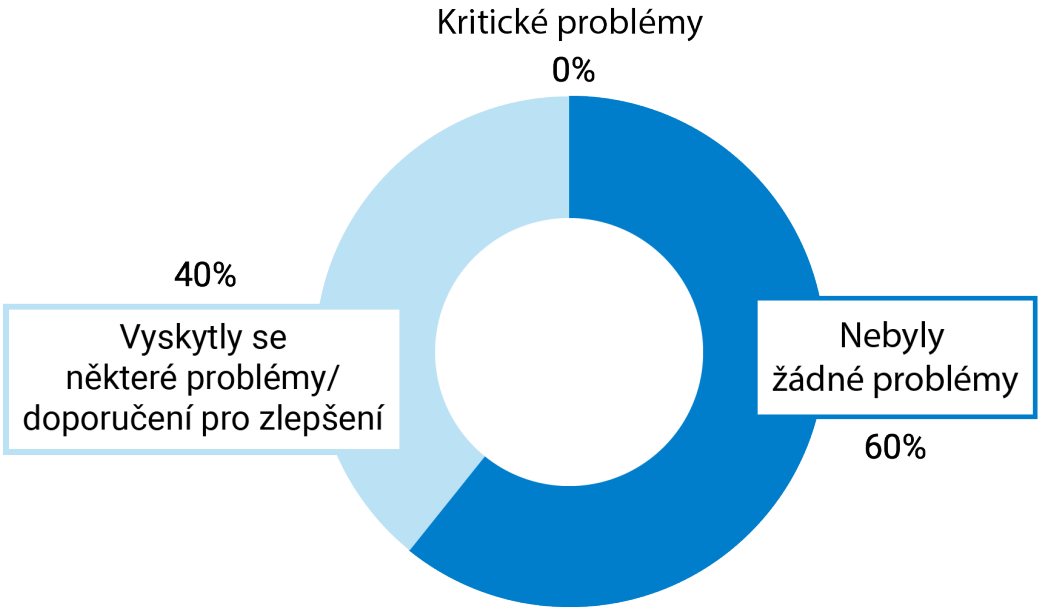
Skutečně, na základě komentářů lze navrhnout zlepšení aplikace. To zahrnuje přidání funkcionality, která poskytuje uživateli nápovědu o atributech tabulek při sestavování dotazů, a také optimalizaci uspořádání designových prvků aplikace pro zvýšení její uživatelské přívětivosti a zlepšení uživatelského zážitku.

Navzdory těmto obtížím považují všichni uživatelé aplikaci za užitečnou. Bylo však uvedeno, že podrobnější příručka nebo pokyny by mohly být užitečné pro uživatele, kteří se s aplikací setkávají poprvé. Zpětná vazba od jednoho z účastníků průzkumu na otázku „Považujete aplikaci za užitečný a srozumitelný nástroj pro ty, kdo se teprve začínají učit SQL?“:

- „Při doplnění o instruktážní manuál nebo někoho, kdo by to na něm ukazoval, ano“.

Každý účastník úspěšně zvládl řadu úkolů, které byly stanoveny v dotazníku, včetně vytváření konkrétních dotazů a získávání odpovídajících vizualizací dat. Shodně bylo

konstatováno, že aplikace představuje efektivní nástroj pro výuku SQL. Funkce vizualizace byla ze strany uživatelů vysoko hodnocena, jelikož tato funkce jim umožnila lepší pochopení dat získaných z dotazů. Kromě toho uživatelé velmi oceňovali možnost interaktivní vizualizace tabulek importovaných z databáze, což představuje další významnou funkci aplikace.



Obrázek 7.1. Diagram výsledků uživatelského testování.

7.3.3 Závěry na základě výsledků testování

Výsledky uživatelského testování se staly cenným zdrojem zpětné vazby, která může být využita pro další zdokonalování aplikace. Také výsledky uživatelského testování aplikace „SQL simulator“ potvrzují její účinnost jako nástroje pro výuku SQL. Přes některé zmíněné problémy s rozhraním se všem účastníkům podařilo zvládnout navrhované úkoly a aplikaci hodnotili jako užitečnou.

Je důležité poznamenat, že 60% uživatelů se nesetkalo s žádnými problémy, zatímco zbývajících 40% identifikovalo některé oblasti pro potenciální zlepšení. To ukazuje na potřebu další optimalizace rozhraní a poskytnutí podrobnějších pokynů pro usnadnění prvního zážitku uživatelů.

Nicméně absence kritických chyb potvrzuje, že aplikace „SQL simulator“ je plně funkční a schopná poskytnout vzdělávací hodnotu pro uživatele na všech úrovních ovládání SQL. Vizualizace dat byla uznána jako obzvláště užitečná funkce, která zdůrazňuje vzdělávací potenciál aplikace.

Celkově tyto výsledky zdůrazňují úspěch aplikace v dosažení její původní cíle - být efektivním nástrojem pro výuku SQL, se zaměřením na praktickou vizualizaci a interaktivní výuku. Další úsilí by mělo být zaměřeno na zlepšení uživatelského rozhraní a poskytování podrobných instrukcí pro zlepšení celkového uživatelského zážitku.

Kapitola 8
Závěr

V závěru své bakalářské práce chtěla bych se zaměřit na hlavní dosažené výsledky a jejich možný dopad na výuku SQL. Vzhledem k důležitosti a složitosti SQL v současných databázových technologiích hlavním cílem této práce bylo analyzovat existující metody praktického učení jazyka SQL, zejména s ohledem na vizualizaci dat (viz kapitola 1.1).

Byl navržen a implementován simulátor dotazů SQL, který poskytuje názorné příklady a vizualizaci dat pro podporu efektivního osvojení jazyka (viz kapitola 5). Zkoumání stávajících nástrojů, jako jsou Relax, Tableau a SQL Academy (viz kapitola 4), umožnilo identifikovat a přijmout nejúčinnější metody interaktivního učení pro implementaci do daného projektu. Tento simulátor, navržený s cílem usnadnit porozumění principům SQL a postupů vyhodnocování SQL operací (viz kapitola 2.3), byl podroben důkladnému testování, včetně jednotkových a integračních testů, jakož i uživatelského testování (viz kapitola 7). Tato fáze byla klíčová pro zajištění nejen funkčnosti simulátoru, ale také jeho účinnosti jako výukového nástroje. Testování potvrdilo, že simulátor je účinným nástrojem pro výuku SQL, poskytuje uživatelům možnost interaktivního vytváření SQL dotazů a vizualizace výsledků (viz kapitola 6). Kromě toho simulator umožňuje vizualizovat databázové tabulky importované do simulátoru, což se uživatelům líbilo nejvíce (viz obrázek 6.3). I když simulátor splnil stanovené cíle, analýza výsledků testování a zpětné vazby uživatelů (viz kapitola 7.3) odhalila několik aspektů, kde existuje prostor pro další optimalizaci. Tyto oblasti nabízejí příležitosti pro budoucí výzkum a vývojové úsilí.

Celkově lze konstatovat, že tento projekt přispěl k lepšímu pochopení, jak efektivně využít vizualizaci dat pro výuku SQL, a poskytl užitečný nástroj pro podporu osvojení tohoto klíčového jazyka v oblasti databázových technologií.

Literatura

[1] Chris J. Date. *An introduction to database systems*. Addison-Wesley, 1995. ISBN 0-321-18956-6.

[2] Varun Grover a James T.C. Teng. An examination of DBMS adoption and success in American organizations. *Information & Management*. 1992, 23 (5), 239-248. DOI https://doi.org/10.1016/0378-7206(92)90055-K.

[3] Statista Research Department. *Most popular database management systems 2022*. 2022. `https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/`. Online, cit. 2023-04-12.

[4] Terence Aidan Halpin a Tony Morgan. *Information modeling and relational databases*. Morgan Kaufmann, 2008. ISBN 978-0123735683.

[5] Alan Beaulieu. *Learning SQL*. O'Reilly, 2009. ISBN 978-0596520830.

[6] R. Kearns, Stephen Shead a Alan Fekete. *A teaching system for SQL*. In: *Proceedings of the 2nd Australasian conference on Computer science education*. 1997. 224–231. ISBN 0897919580.

[7] SQLBolt. *Learn SQL. SQL Lesson 12: Order of execution of a Query*. 2021. `https://sqlbolt.com/lesson/select_queries_order_of_execution`. Online, cit. 2022-12-15.

[8] Martin Řimnáč. *Popis předmětu - B0B36DBS*. 2022. `https://fel.cvut.cz/cz/education/bk/predmety/50/10/p5010606`. Online, cit. 2022-12-15.

[9] Martin Svoboda. *B0B36DBS - Databázové systémy*. 2022. `https://www.ksi.mff.cuni.cz/~svoboda/courses/202-B0B36DBS/`. Online, cit. 2022-12-15.

[10] J. Kessler. *RelaX - relational algebra calculator*. 2020. `http://clotho.uom.gr/relax/calc.htm`. Online, cit. 2022-12-15.

[11] A. Nepein. *SQL Academy*. 2020. `https://sql-academy.org/en/trainer?sort=byId`. Online, cit. 2022-12-16.

[12] Tableau. *Business Intelligence and Analytics Software*. 2003. `https://www.tableau.com/node/62770`. Online, cit. 2022-12-16.

[13] Jeffrey D. Walker a Steven C. Chapra. A client-side web application for interactive environmental simulation modeling. *Environmental Modelling & Software*. 2014, 55 49-60. DOI https://doi.org/10.1016/j.envsoft.2014.01.023.

[14] SQL.js team. *SQLite compiled to JavaScript through Emscripten*. 2023. `https://sql.js.org/`. Online, cit. 2023-05-18.

[15] Zheng Gao, Christian Bird a Earl T. Barr. *To Type or Not to Type: Quantifying Detectable Bugs in JavaScript*. In: *Proceedings of the 39th International Conference*

on Software Engineering. IEEE Press, 2017. 758–769. ISBN 9781538638682. https://doi.org/10.1109/ICSE.2017.75.

[16] NPM Trends. *Compare NPM package download statistics over time*. https://npmtrends.com/angular-vs-next-vs-react-vs-vue. 2023. Online, cit. 2023-05-18.

[17] React Documentation Contributors. *Learn React*. 2023. https://react.dev/learn. Online, cit. 2023-05-18.

[18] Angular Contributors. *Angular Documentation*. 2023. https://angular.io/docs. Online, cit. 2023-05-18.

[19] Vue.js Contributors. *Vue.js Guide*. 2023. https://vuejs.org/guide/introduction.html. Online, cit. 2023-05-18.

[20] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai a JF Bastien. Bringing the Web up to Speed with WebAssembly. *SIGPLAN Not.*. 2017, 52 (6), 185–200. DOI 10.1145/3140587.3062363.

[21] Emscripten Project Team. *About Emscripten*. 2023. https://docs.huihoo.com/emscripten/emscripten-site/docs/introducing_emscripten/about_emscripten.html. Online, cit. 2023-05-12.

[22] Jari-Pekka Voutilainen, Tommi Mikkonen a Kari Systa. *Synchronizing Application State Using Virtual DOM Trees.* In: Sven Casteleyn, Peter Dolog a Cesare Pautasso, eds. *Current Trends in Web Engineering*. Cham: Springer International Publishing, 2016. 142–154. ISBN 978-3-319-46963-8.

[23] Sing Li. *React: Create maintainable, high-performance UI components*. 2015. https://developer.ibm.com/tutorials/wa-react-intro/. Online, cit. 2023-05-23.

[24] Alex Banks a Eve Porcello. *Learning React: Functional Web Development with React and Redux*. O'Reilly Media, 2017. ISBN 978-1491954621.

[25] React Team. *React v16.8: The One With Hooks*. 2019. https://legacy.reactjs.org/blog/2019/02/06/react-v16.8.0.html. Online, cit. 2023-05-18.

[26] React Hook Form. *React Hook Form*. 2023. https://react-hook-form.com. Online, cit. 2023-05-18.

[27] React Training. *React Router - useNavigate*. 2023. https://reactrouter.com/en/main/hooks/use-navigate. Online, cit. 2023-05-18.

[28] React Flow. *React Flow - Concepts - Introduction*. 2022. https://reactflow.dev/docs/concepts/introduction/. Online, cit. 2023-05-18.

[29] React Training. *Declarative Routing for React*. 2023. https://reactrouter.com/en/main. Online, cit. 2023-05-18.

[30] GitHub. *About GitHub Pages*. 2023. https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages. Online, cit. 2023-05-20.

[31] Beier Luo. *React Hook Form - Simple Form Validation*. 2023. https://react-hook-form.com. Online, cit. 2023-05-18.

[32] Davide Morello. *Getting Started - React Awesome Reveal*. 2023. `https://react-awesome-reveal.morello.dev/docs/getting-started`. Online, cit. 2023-05-19.

[33] React Icons Contributors. *React Icons*. 2023. `https://react-icons.github.io/react-icons/`. Online, cit. 2023-05-19.

[34] Keith Grant. *CSS in Depth.* In: Simon and Schuster, 2018. 9. ISBN 978-1617293450.

[35] Dieter Schmalstieg a Robert F. Tobler. *Real-time bounding box area computation.* Institute of Computer Graphics and Algorithms, Vienna University of Technology. 1999. TR-186-2-99-05.

[36] J. Rubin a D. Chisnell. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. 2nd vyd.. Indianapolis, IN: Wiley Publishing, Inc., 2008. ISBN 978-0470185483.

Příloha A
Struktura projektu

A.1 Popis jednotlivých modulů

```
src // Kořenový adresář, který obsahuje hlavní zdrojové soubory aplikace.
|-components // Všechny komponenty React, které jsou v aplikaci použity.
|   |---form // Komponenty spojené s formuláři v aplikaci.
|   |   |---fromQuery // Komponenty pro zpracování FROM dotazů.
|   |   |   |---menu // Boční menu pro stránku "Tables".
|   |   |       |---joins // "Join" komponenty pro spojování tabulek.
|   |   |---selectQuery // Komponenty pro zpracování SELECT dotazů.
|   |       |---editor // Prostor pro vytváření a úpravu dotazů.
|   |       |---menu // Nabídka různých operací pro konfiguraci dotazu.
|   |---UI // Komponenty uživatelského rozhraní.
|   |   |---buttons // Tlačítka v aplikaci.
|   |   |---loader // Načítací obrazovka při připojení wasm.
|   |   |---navbar // Horní navigační panel.
|   |   |---popup // Zobrazující se okno pro import databáze.
|   |   |---table // Tabulky v aplikaci.
|   |       |---popup // Tabulka po najetí kurzorem na vizualizaci.
|   |       |---result // Tabulka s výsledky dotazů.
|   |           |---fakeResult // Fiktivní tabulková data.
|   |---visualisation // Komponenty pro vizualizaci dat.
|       |---circles // Vizualizační komponenta ve tvaru kruhu.
|       |---diagram // Komponenta diagramu tabulek a jejich vztahů.
|       |---nodes // Prvek diagramu pro zobrazení tabulek.
|---context // Kontexty React poskytující globální stav aplikace.
|---pages // Jednotlivé stránky v aplikaci.
|---utils // Podpůrné funkce a zdroje
    |---images // Uložení obrázků používaných v aplikaci.
```

A.2 Struktura modulů

```
|-joins
    |---JoinsMenu.jsx // Komponenta, zobrazující seznam typů spojení.
    |---JoinsMenu.module.css // Modulární styly pro komponentu.
    |---JoinsMenu.test.jsx // Soubor pro testování funkcí komponenty.
```

Příloha B
Otázky z dotazníku uživatelského testování

1. „Zkuste do aplikace importovat databázi. Narazili jste při tomto procesu na jakékoliv problémy? Bylo pro vás zřejmé, jak a kam data importovat?“.

2. „Po importu dat z předchozího kroku zkuste analyzovat vizualizovanou databázi. Zkuste změňit pozici diagramu pro větší přehlednost. Jaké cizí klíče podle vás spojují tabulky?“.

3. „Pomocí pole pro vytváření dotazů a pomocné vizualizace tabulek zkuste spojit tabulky „Registration“ a „Courses“ pomocí NATURAL JOIN. Narazili jste na nějaké chyby? Způsobilo vám přidání elementu JOIN problémy?“.

4. „Vyčistěte pole vašeho dotazu a zkuste znovu provést následující dotaz: „SELECT * FROM Registration AS r INNER JOIN Students AS s ON r.student_id = s.id“. Narazili jste při tom na nějaké problémy? Podařilo se vám získat výsledek dotazu? Po provedení dotazu pole nečistěte. Výsledek si prosím uložte pro příští úkol.“.

5. „Po provedení předchozího kroku a sestavení dotazu přejděte na stránku vytváření SELECT dotazů s různými operacemi z vedlejší nabídky operací. Zkuste sestavit jednoduchý SELECT dotaz s výběrem atributu first_name („SELECT first_name FROM...“) bez použití dalších operací. Byla vizualizace výsledků dotazu srozumitelná? Zkopírujte svůj dotaz a napište jej níže.“.

6. „Nyní vyčistěte pole dotazu a zkuste sestavit dotaz s podmínkou WHERE, kde identifikátor studenta je větší než 2 („s.id > 2“). Podařilo se vám to? Bylo přetahování prvků z menu jasné? Prohlédněte si všechny kroky vizualizace dotazu pomocí šipek nahoře. Došlo ke změně vizualizace?“.

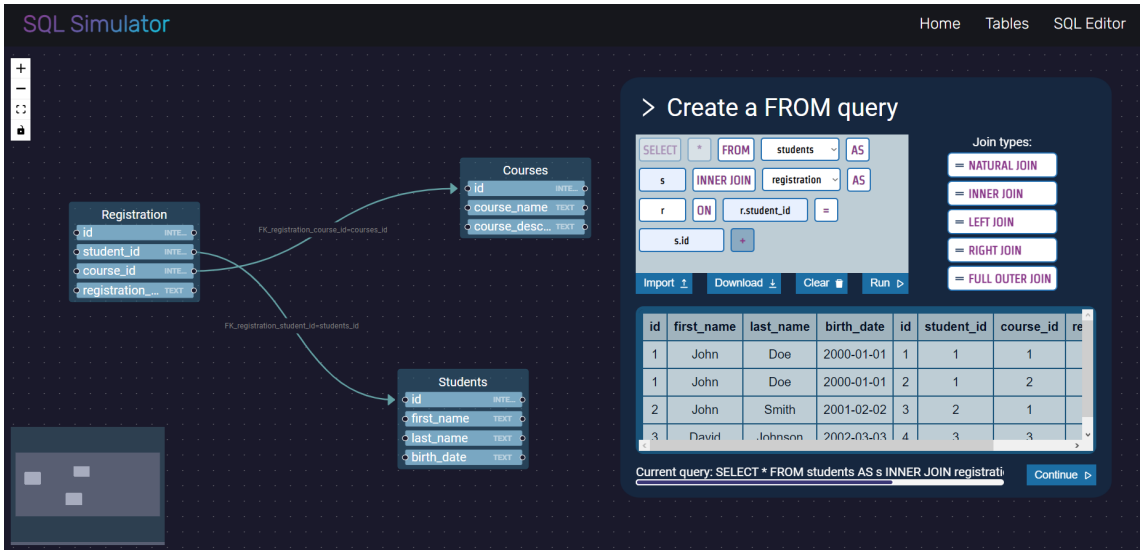
7. „Vyčistěte pole dotazu a zkuste sestavit následující dotaz: „SELECT last_name, COUNT(*) FROM registration AS r INNER JOIN students AS s ON r.student_id = s.id WHERE s.id < 3 GROUP BY last_name“. Zvažte všechny fáze vizualizace prvků. Jak se změnily?“.

8. „Pomohla vám vizualizace databáze a výsledků dotazů lépe porozumět dotazům SQL?“.

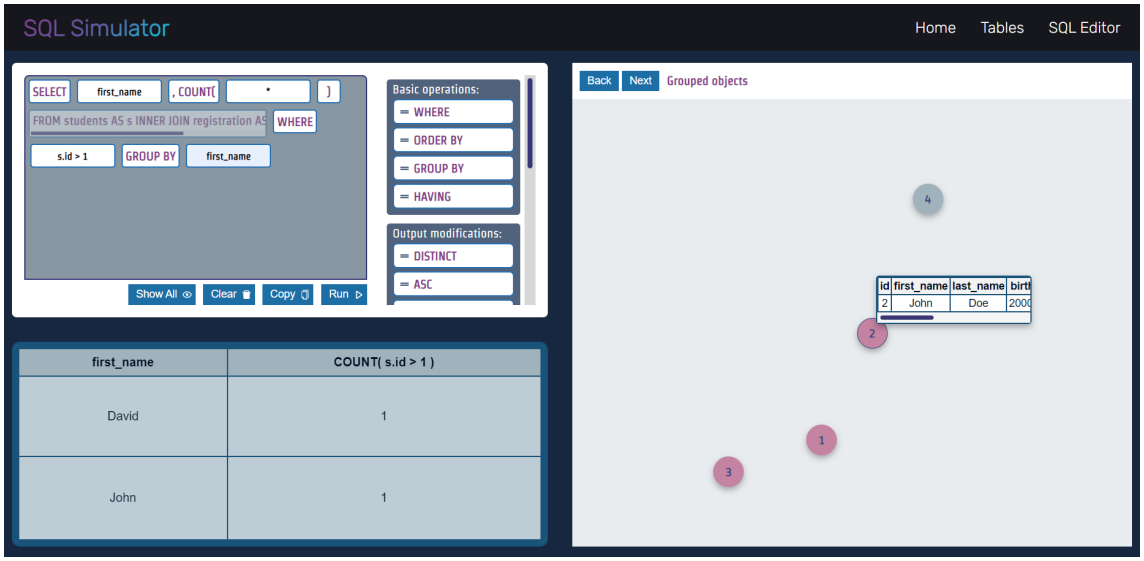
9. „Považujete aplikaci za užitečný a srozumitelný nástroj pro ty, kdo se teprve začínají učit SQL?“.

Příloha C
Seznam obrazovek implementované aplikace

Obrázek C.1. Úvodní obrazovka „Home“ aplikace.



Obrázek C.2. Obrazovka „Tables“ pro sloučení tabulek a vizualizaci databáze.



Obrázek C.3. Obrazovka „SQL editor“ pro vytváření a vizualizaci SQL dotazů.

Příloha D
Zkratky a symboly

D.1 Zkratky

DBMS	Database Management System - systém řízení báze dat.
DBS	Databázové systémy.
DDL	Data Definition Language - jazyk pro definici dat.
DML	Data Manipulation Language - jazyk pro manipulaci s daty.
TCL	Transaction Control Language - jazyk pro řízení transakcí.
SQL	Structured Query Language - strukturovaný dotazovací jazyk.
ER	Entity Relationship - entitně vztahový model.
ACID	Atomicity (atomicita), consistency (konzistence), isolation (izolovanost), durability (trvalost).
NPM	Node.js package manager - správce balíčků pro JavaScript.
Wasm	WebAssembly - binární formát instrukcí pro efektivní spuštění komplexních aplikací ve webových prohlížečích.
LLVM	Low Level Virtual Machine - nízkoúrovňový virtuální stroj optimalizující překladač.
API	Application Programming Interface - rozhraní pro programování aplikací.
DOM	Document Object Model - objektový model dokumentu.
XML	Extensible Markup Language - značkovací jazyk XML.
HTML	Hypertext Markup Language - značkovací jazyk HTML pro tvorbu webových stránek.
CSS	Cascading Style Sheets - kaskádové styly.
CRACO	Create React App Configuration Override - nástroj pro přizpůsobení konfigurace webových aplikací.
URL	Uniform Resource Locator - unifikovaný lokátor zdrojů.
AABB	Axis-Aligned Bounding Box - algoritmus vyhledávání kolizí mezi objekty.
RTL	React Testing Library - testovací knihovna React.
E2E	End-to-end testing - druh testování procesu v systému od začátku do konce.

D.2 Soubory, které jsou součástí

`SQL-DBSoperations.xlsx`	Analyzované SQL operace z předmětu Databázové systémy.
`Prototype.pdf`	Prototyp SQL simulátoru pro následnou implementaci.
`SQLsimulator.zip`	Implementace aplikace.
`Database.zip`	Data pro testování aplikace.