Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Системотехніки

**МІЖДИСЦИПЛІНАРНИЙ КУРСОВИЙ ПРОЕКТ**

**ПОЯСНЮВАЛЬНА** **ЗАПИСКА**

тема: «Вантажні перевезення»

з дисципліни «Проектування високонавантажених систем зберігання даних»

Керівник: Доцент кафедри СТ Коваленко А.І.

Студент: Антонюк М.В.

Робота захищена з оцінкою «\_\_\_\_\_\_\_\_\_» «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 р.

Комісія:

Доцент кафедри СТ Коваленко А.І.

Доцент кафедри СТ Решетнік В.М.

Асистент кафедри СТ Жернова П.Є

Харків 2018

Харківський національний університет радіоелектроніки

**Факультет** комп'ютерних наук

**Кафедра** системотехніки

**Напряму підготовки** 6.050101 – Комп'ютерні науки

**Курс** 3 **група** КН-15-2 **семестр** 6

**ЗАВДАННЯ**

**на курсове проектування**

**студентові** Антонюку Максиму Васильовичу

(прізвище, ім'я, по батькові)

**1. Тема роботи:** Інформаційна система «Вантажні перевезення»

**2. Строк здачі студентом закінченої роботи** 14.06.18

**3. Вихідні дані до проекту:** Розробити серверну частину інформаційної системи «Вантажні перевезення». Серверна частина повинна являти собою 2 варіанти реалізації бази даних, розробленої для платформи СУБД MySQL з використанням таблиць типу MyIsam і InnoDB. В системі існує 4 ролі: незареєстровані користувачі, зареєстровані користувачі(водій, менеджер, адміністратор). Бізнес-функції системи для незареєстрованних користувачів: подання заяви на позицію в компанії. Бізнес-функції системи для всіх зареєстрованих користувачів: зміна профілю (e-mail, пароль); вхід у систему з визначенням статусу; перегляд рейсів, фільтрація за статусом(«новий», «відмінений», «виконується», «виконаний»), датою створення/завершення, ціною, вантажопідйомність. Бізнес-функції системи для водіїв: подання заявки на рейс; підтвердження виконання рейсу; зміна статусу автомобіля(працює, не працює); подання заявки на зміну характеристик автомобіля. Бізнес-функції системи для менеджерів та адміністраторів: додавання нових рейсів; зміна статусу для створених рейсів(для менеджерів лише створених ним). Бізнес-функції системи для адміністраторів: підтвердження/відміна заявки для нових користувачів; перегляд всіх користувачів, їх присутність в системі, сортування їх по імені, прізвищу, даті реєстрації, ролі в системі; додавання/редагування автомобіля; списання автомобіля; редагування бази автомобілів, які можуть використовуватись в компанії; формування звіту за період; пошук користувачів. Операційна система – Linux Mint 16 або вище, програмне забезпечення: програмний пакет Workbench, сервер MySQL 5.7.

**4. Зміст пояснювальної записки (перелік питань, які підлягають розробці):** забезпечити цілісність даних, створивши необхідні тригери для таблиць типу MyIsam і виконавши відповідні інструкції SQL для таблиць типу InnoDB; описати функції інтерфейсу клієнтської частини високонавантаженої інформаційної системи «Вантажні перевезення», що реалізують основні бізнес-процеси; розробити SQL-запити у вигляді процедур, функцій, тригерів, представлень, необхідні для реалізації бізнес-процесів на стороні сервера MySQL (включаючи повнотекстовий пошук); розробити транзакцію для реалізації одного з основних бізнес-процесів на стороні сервера MySQL (для таблиць типу InnoDB); провести дослідження і прийняти обґрунтовані рішення по оптимізації доступу до високонавантажених баз даних за допомогою індексів (включаючи повнотекстовий пошук) і урахування специфіки використання таблиць типу MyIsam і InnoDB; здійснити масштабування високонавантажених баз даних (з використанням таблиць типу MyIsam і InnoDB) зі зміною структури даних для горизонтального і вертикального шардінга та обґрунтуванням прийнятих рішень. Розробити модифікації процедур, функцій, тригерів, транзакцій для кожного варіанта масштабування; провести порівняльний аналіз двох варіантів реалізації бази даних (з використанням таблиць типу MyIsam і InnoDB) високонавантаженої інформаційної системи «Вантажні перевезення» із прийняттям рішень і розробкою рекомендацій з їхнього використання.

**5. Перелік графічного матеріалу (з точним визначенням обов'язкових креслень):** фізична модель бази даних на платформі сервера MySQL у вигляді ER-діаграми згідно з нотацією IDEF1X (або у вигляді EER-діаграми, створеної за допомогою програмного пакета WorkBench) з обов'язковою вказівкою первинних і зовнішніх ключів, типу даних, атрибутів «NULL», «NOT NULL»; таблиці запитів на вибірку для обґрунтування і перевірки розроблених процедур, функцій, тригерів, представлень, транзакцій; таблиці планів EXPLANE виконання SQL-запитів для індексів, унікальних, кластерних і складених індексів; таблиці вибірки даних для визначення селективності складених індексів; таблиці вибірки даних для обґрунтування вибору довжини префікса при повнотекстовому пошуку; змінені структури даних фізичної моделі бази даних при проведенні масштабування (горизонтального і вертикального шардінга).

**6. Дата видачі завдання:**

**Керівник роботи Коваленко А. І.**

(підпис) (прізвище, ім'я, по батькові)

**Студент Антонюк М.В.**

(підпис) (прізвище, ім'я, по батькові)

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Назва етапів курсового проекту** | **Строк виконання** | **Примітка** |
|  | Аналіз предметної області |  |  |
|  | Визначення основних бізнес-функцій високонавантаженої інформаційної системи |  |  |
|  | Визначення функцій інтерфейсу клієнтської частини інформаційної системи |  |  |
|  | Розробка серверної частини інформаційної системи |  |  |
|  | Логічне й фізичне моделювання даних. |  |  |
|  | Створення і заповнення високонавантажених баз даних з таблицями типу MyIsam і InnoDB |  |  |
|  | Розробка підтримки цілісності даних |  |  |
|  | Реалізація бізнес-функцій інформаційної системи на стороні сервера MySQL (процедур, функцій, тригерів, представлень, транзакцій) |  |  |
|  | Оптимізація запитів до високонавантажених баз даних |  |  |
|  | Масштабування баз даних |  |  |
|  | Порівняльний аналіз двох варіантів реалізації бази даних (з використанням таблиць типу MyIsam і InnoDB) |  |  |
|  | Рекомендації з використання баз даних |  |  |

**Керівник роботи Коваленко А. І.**

(підпис) (прізвище, ім'я, по батькові)

**Студент Антонюк М.В.**

(підпис) (прізвище, ім'я, по батькові)

**«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г.**

РЕФЕРАТ

Пояснювальна записка з курсового проекту включає: 65 сторінок, 3 таблиці, 71 рисунок, 2 додатки та 4 джерела інформації.

БАЗА ДАНИХ, СИСТЕМА КЕРУВАННЯ БАЗОЮ ДАНИХ, MYSQL, МАШТАБУВАННЯ, ШАРДІНГ, ОПРИМІЗАЦІЯ SQL ЗАПИТІВ, ІНДЕКСИ, ІНФОРМАЦІЙНА СИСТЕМА

Об’єктом дослідження курсового проекту є автоматизація управління процесами для рішень бізнесу в рамках високонавантаженої системи.

Предметом дослідження є інформаційні технології, готові програмні рішення, які дозволяють програмними методами створити серверну частину високонавантаженої інформаційної системи команії вантажоперевезень.

Мета дослідження – розробка серверної частини високонавантаженої інформаційної системи.

Методи дослідження: системній підхід, методи структурного аналізу і моделювання баз даних на основі реляційної алгебри, тобто реляційної бази даних, методи реляційної алгебри та реляційного обчислення.

В роботі проведено проектування двох варіантів серверної частини високонавантаженої інформаційної системи інтернет магазину. Проведена оптимізація SQL запитів за критерієм мінімізації часу отримання даних з урахуванням специфіки високонавантажених інформаційних систем.

Галузь застосування – підтримка та автоматизація інтернет магазину.

ЗМІСТ

[ВСТУП 7](#_Toc516090707)

[1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8](#_Toc516090708)

[1.1 Аналіз предметної області та коротка постанова задачі 8](#_Toc516090709)

[1.2 Визначення основних бізнес функцій високонавантаженої інформаційної системи 8](#_Toc516090710)

[1.2.1 Бізнес функції адміністратора 8](#_Toc516090711)

[1.2.2 Бізнес функції всіх зареєстрованих користувачів 9](#_Toc516090712)

[1.2.3 Бізнес функції незареєстрованого користувача 9](#_Toc516090713)

[1.2.4 Бізнес функції доступні адміністраторам та менеджерам 9](#_Toc516090714)

[1.2.5 Бізнес функції доступні водіям 9](#_Toc516090714)

[1.3 Визначення функцій інтерфейсу клієнтської частини інформаційної системи 9](#_Toc516090715)

[2 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ВИСОКОНАВАНТАЖЕНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ 13](#_Toc516090716)

[2.1 Логічне та фізичне моделювання даних 13](#_Toc516090717)

[2.2 Забезпечення цілістності даних 14](#_Toc516090718)

[2.2.1 Цілісність даних для InnoDB 15](#_Toc516090719)

[2.2.2 Цілісність даних для MyISAM 15](#_Toc516090720)

[2.3 Реалізація бізнес функцій високонавантаженої інформаційної системи на стороні MySQL сервера 28](#_Toc516090721)

[2.4 Реалізація транзакції для InnoDB. 37](#_Toc516090722)

[3 ОПТИМІЗАЦІЯ ВИСОКОНАВАНТАЖЕНИХ БАЗ ДАНИХ 38](#_Toc516090723)

[3.1 Видалення підстановочних таблиць 38](#_Toc516090724)

[3.2 Зміна типів атрибутів 39](#_Toc516090725)

[3.3 Додавання індексів 39](#_Toc516090726)

[3.3.1 Додавання FULLTEXT індексів 39](#_Toc516090727)

[3.3.2 Аналіз використання кластерних індексів в InnoDB 40](#_Toc516090728)

[3.3.3 Додавання індексів типу INDEX 41](#_Toc516090729)

[3.3.4 Додавання префіксного індексу 42](#_Toc516090729)

[3.4 Денормалізація баз даних 44](#_Toc516090730)

[3.5 Шардинг 45](#_Toc516090731)

[3.5.1 Вертикальний шардинг 46](#_Toc516090732)

[3.5.2 Горизонтальний шардинг 46](#_Toc516090733)

[ВИСНОВКИ 47](#_Toc516090734)

[ПЕРЕЛІК ПОСИЛАНЬ 48](#_Toc516090735)

[ДОДАТОК А – КОД ГЕНЕРАЦІЇ НЕОПТИМІЗОВАНОЇ БД ТИПУ MyISAM 49](#_Toc516090736)

[ДОДАТОК Б – КОД ГЕНЕРАЦІЇ ОПТИМІЗОВАНОЇ БД ТИПУ InnoDB 56](#_Toc516090737)

# ВСТУП

Системи зберігання даних у сучасному світі орієнтовані на велику кількість користувачі, тому більшість таких систем є високонавантаженими. Системи масового обслуговування перш за все повинні бути надійними, тобто оперування даними не повинне ініціювати втрату певних даних, незважаючи на те, що кількість оброки даних може бути надзвичайно велика.

Надійність даних говорить про те що потрібно перевіряти дані, що може зробити високонавантажену систему не достатньо швидкою для великої кількості користувачів. Одним із рішень швидкості роботи такої бази є її оптимізація. Також слід не забувати про постійне підтримання консистентності даних, що буде досягатися за рахунок накладення обмежень на поля а також виконання певних операцій транзакційно.

У курсовому проекті розглядається розробка системи зберігання даних для компанії вантажних перевезень. Система дозволяє автоматизувати бізнес-процеси пов’язані зі створенням рейсів для перевезення вантажів, їх керуванню, також для відправлення та обробку заявок на отримання працевлаштування в компанії або отримання послуг з перевезення вантажів. Для заохочення використовування послуг саме цієї компанії є можливість відслідковувати перевезення клієнтського вантажу, тому в пікових режимах система буде перенавантажена запитами, що спонукає до створення інформаційної системи, яка буде стійка до великої кількості запитів від користувачів, що є невід’ємною частиною бізнесу.

Метою курсового проектування є поглиблення і узагальнення знань з курсу «Проектування високонавантажених систем зберігання даних», вивчення специфіки серверу СУБД MySQL.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## Аналіз предметної області та коротка постанова задачі

Компанія вантажних перевезень – підприємство, яке надає можливість перевозити вантажі в межах країни як для фізичних так і для юридичних осіб. Також компанія надає працевлаштування на позиції водія або менеджера. Клієнт може спостерігати за виконанням його перевезення, завдяки відслідковуванню переміщень автомобілю. Необхідно розробити систему зберігання даних для автоматизації інформаційної системи «Вантажні перевезення». Автоматизації підлягає процес оперування з рейсами, приймання заявок на отримання працевлаштування а також замовлень для перевезень вантажів від фізичних та юридичних осіб.

## Визначення основних бізнес функцій високонавантаженої інформаційної системи

В системі присутні 4 основні ролі, кожна з яких характеризується певним набором бізнес-процесів, замовник відноситься до ролі незареєстрованого користувача, оскільки персонального кабінету в системі для нього не існує.

### Бізнес функції адміністратора

Пошук користувачів за ім’ям, прізвищем, електронною поштою

Списання автомобілю з використання

Додання до системи нового автомобілю

Зміна характеристик автомобілів

Реєстрація нового співробітника

Переведення працівника до категорії неактивних(при звільненні)

### Бізнес функції всіх зареєстрованих користувача

Огляд рейсів, фільтрація за датою створення та завершення, за ідентифікатором, статусом

Зміна електронної пошти та паролю

Вхід

### Бізнес функції незареєстрованого користувача

Подання заяви на працевлаштування в компанії

Замовлення на перевезення вантажу

### Бізнес функції доступні адміністраторам та менеджерам

Створення рейсу

Підтвердження автомобіля на рейс(для менеджерів тільки ним створених)

Відміна рейсу(для менеджерів тільки ним створених)

### Бізнес функції доступні водіям

Підписання на рейс

Перегляд автомобілів доступних для нього

Переведення автомобіля в положення «Зламаний»/«Справний»

Підтвердження виконання рейсу

## Визначення функцій інтерфейсу клієнтської частини інформаційної системи

У таблицях 1.3.1 та 1.3.2 встановлена відповідність між бізнес-функціями та відповідним елементам інтерфейсу інформаційної системи, яка буде взаємодіяти з високонавантаженою системою зберігання даних яка розроблюється.

Таблиця 1.3.1 – Відповідність бізнес функцій до елементів інтерфейсу

|  |  |
| --- | --- |
| Бізнес функція | Опис інтерфейсу інформаційної системи |
| Пошук користувачів за ім’ям, прізвищем, електронною поштою | Таблиця з інформацією про користувача, поля для вводу імені, прізвища, електронної пошти та кнопка знайти |
| Списання автомобілю з використання | Таблиця з інформацією про автомобілі, один зі стовбців таблиці містить кнопку при натискання на яку автомобіль списується з використання і стає неактивним для водіїв |
| Додання до системи нового автомобілю | Поля із введенням номеру автомобіля, вибором марки та моделі автомобіля, поле для вказання вантажопідйомності, та поле для нотатку(необов’язкове) |
| Зміна характеристик автомобілів | Таблиця з інформацією про автомобілі, кожне значення в спадаючому списку опцій має властивість після активування якої можна змінити значення |
| Реєстрація нового співробітника | Таблиця з заявами на отримання працевлаштування в компанії, в кожному рядку окремим стовбцем присутня кнопка підтвердження та відхилення заяви |
| Переведення працівника до категорії неактивних(при звільненні) | Таблиця з інформацією про користувачів системи, окремим стовбцем кнопки при натисканні на які користувач стає неактивним і більше не може увійти в систему |

Продовження таблиці 1.3.1

|  |  |
| --- | --- |
| Бізнес функція | Опис інтерфейсу інформаційної системи |
| Огляд рейсів, фільтрація за датою створення та завершення, за ідентифікатором, статусом | Таблиця з інформацією про рейси, над таблицею форма з параметрами для фільтрації та кнопка «Фільтрувати» |
| Зміна електронної пошти та паролю | Форма на сторінці особистого кабінету в якій присутні три поля, нова електронна пошта, новий пароль, поле підтвердження нового паролю та кнопка «Змінити» |
| Вхід | Модальне вікно з полями для вводу електронної пошти та паролю та кнопка «Увійти» |
| Подання заяви на працевлаштування в компанії | Сторінка з формою, на якій знаходяться поля для вводу імені, прізвища, електронної пошти, очікувана заробітня плата, текст резюме, позиція в компанії(водій, менеджер) та кнопка «Відправити» |
| Замовлення на перевезення вантажу | Сторінка з формою, на якій знаходяться поля для вводу імені, прізвища, назви компанії(якщо юридична особа), контактний номер, електронна пошта, описання вантажу який необхідно перевезти та кнопка «Замовити» |
| Створення рейсу | Модальне вікно з формою, на якій знаходяться поля для вводу координат пункту відправки, координат пункту призначення, об’єму вантажу, додаткової інформації про рейс та кнопка «Створити» |

Продовження таблиці 1.3.1

|  |  |
| --- | --- |
| Бізнес функція | Опис інтерфейсу інформаційної системи |
| Створення рейсу | Модальне вікно з формою, на якій знаходяться поля для вводу координат пункту відправки, координат пункту призначення, об’єму вантажу, додаткової інформації про рейс та кнопка «Створити» |
| Підтвердження автомобіля на рейс(для менеджерів тільки ним створених) | Таблиця з інформацією про рейси, один стовбець має кнопку при натисканні якої з’являється модальне вікно зі списком автомобілів підписаних на рейс, біля кожного кнопка «Підтвердити» |
| Відміна рейсу(для менеджерів тільки ним створених) | Таблиця з інформацією про рейси, на якій в окремому стовбці для водіїв доступна кнопка яка дозволяє вибрати автомобіль який доступний та підписатися на рейс |
| Перегляд автомобілів доступних для нього | Сторінка з автомобілями, які доступні йому для використання |
| Підписання на рейс | Таблиця з інформацією про рейси, на якій в окремому стовбці кнопка |
| Переведення автомобіля в положення «Зламаний»/«Справний» | Сторінка з автомобілями, які доступні йому для використання, біля кожного автомобіля кнопка яка, при натисканні на яку автомобіль переходить в положення «Зламаний»/«Справний» |
| Підтвердження виконання рейсу | Таблиця з інформацією про рейси, на якій в окремому стовбці для водіїв доступна кнопка яка дозволяє підтвердити завершення рейсу |

# 2 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ВИСОКОНАВАНТАЖЕНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 2.1 Логічне та фізичне моделювання даних

Проаналізувавши та дослідивши предметну область були виділені наступні основні сутності:

* Employee співпрацівник компанії. Основні атрибути – name, password, email, surname, role, date\_registration, salary.
* Potential Employee кандидат на позицію в компанію. Основні атрибути – name, email, surname, role, salary\_expected, resume.
* Car автомобіль. Основні атрибути – number, mark, model, carrying, is\_fixed, current\_position.
* Journey рейс. Основні атрибути – from, destination, date\_created, date\_start, date\_finish, progress, status.
* Contract контракт перевезення вантажу. Основні атрибути – date, description, price, key.
* Customer замовник. Основні атрибути – name, surname, date\_cooperation, company(optional), address, email, phone.

Згідно з основними сутностями була складена ERD, котра відповідає 3-й нормальній формі. 3-тя нормальна форма була досягнута завдяки введенню додаткових таблиць, створенню сурогатних ключів для підтримання повної функціональної залежності неключових елементів від ключових та іншим правилам побудови 3-ї нормальної форми. ERD-діаграма відображена на рисунку 2.1

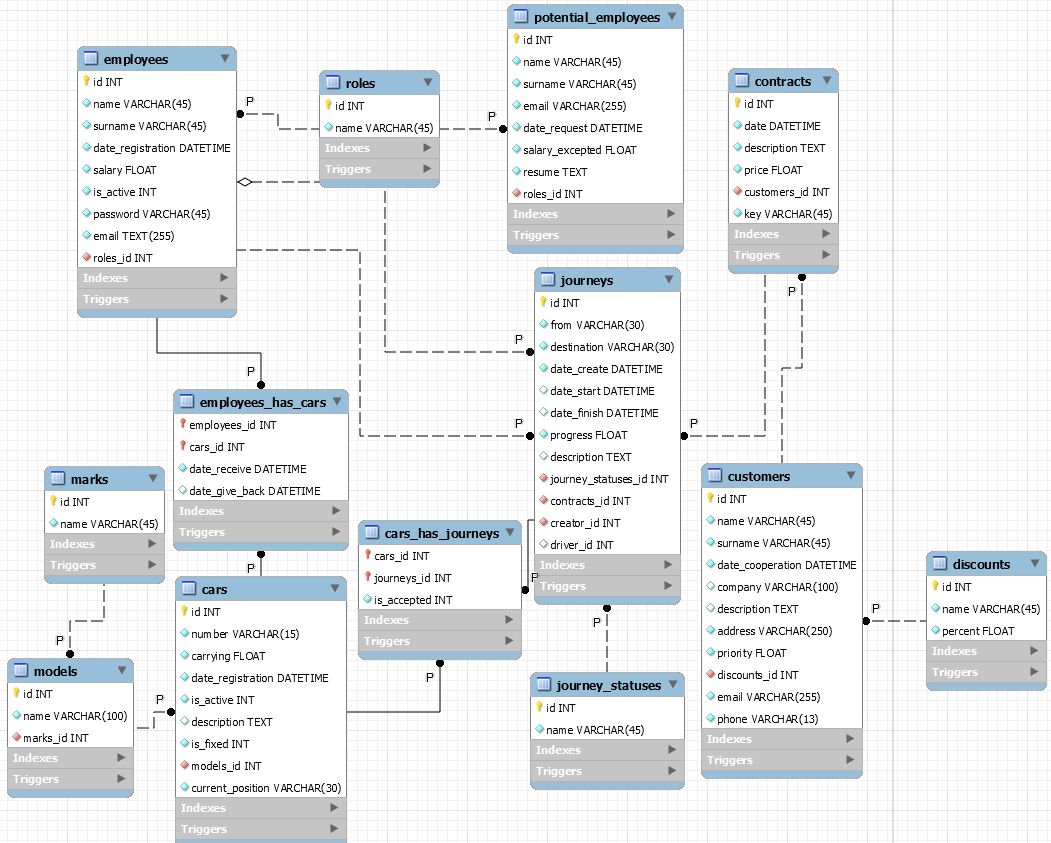


Рисунок 2.1 – ERD-діаграма для предметної області

## 2.2 Забезпечення цілісності даних

Однією із основних вимог до бази даних є цілісність даних, котра завжди буде відображати актуальні дані високонавантаженої системи. У розробленій ERD ми бачимо наявність зовнішніх зв’язків, щоб цілісність даних зберігалася, потрібно оброблювати такі операції як UPDATE та DELETE тих первинних ключів, які є зовнішніми для іншої таблиці.

Для даної предметної області нема наявності видаляти будь які записи з усіх таблиць окрім «potential\_employees», чи десь змінювати первинні ключі. Причиною даного рішення є необхідність зберігання всіх даних впродовж функціонування системи, оскільки вся інформація необхідна для звітності та підтримання статистики, до того ж компанії необхідно відслідковувати всю історію операцій. Проте деякі операції видалення можна все ж робити. Наприклад можна видалити марку або модель автомобіля, якщо вона ніде не використовувалась. Теж саме стосується знижок для клієнтів. Видалення потенціальних працівників можливе, адже можливе навмисне навантаження сервера, і тому адміністраторам буде доступна функція видалення заяв від потенціальних працівників, якщо адміністратор вважає цю заяву непотрібною. Після прийняття заяви, заяву теж видаляють, при цьому вважається що резюме користувача більше не потрібне для бізнес-процесів. До того ж видалення цих записів жодним способом не буде впливати на цілісність даних, бо зовнішніх ключів до цієї таблиці ніде нема.

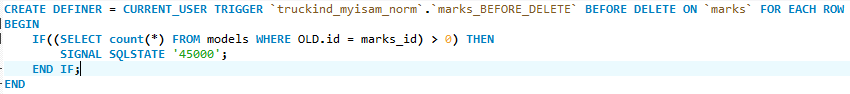
### 2.2.1 Цілісність даних для InnoDB

Так як було прийняте рішення, що первинні ключі не будуть оновлюватися, то потрібно таблицям у базі даних із таблицями типу InnoDB встановити обмеження NO ACTION або RESTRICT (є синонімами у MySQL, RESTRICT – стандарт SQL, NO ACTION – специфіка СУБД MySQL), для таких подій як ON UPDATE, ON DELETE. Ці обмеження дадуть можливість видаляти записи з підставних таблиць, якщо відсутні зовнішні зв’язки з іншими таблицями.

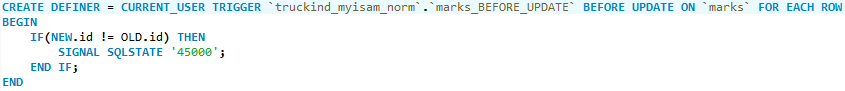
### 2.2.2 Цілісність даних для MyISAM

Цілісність даних для таблиць типу MyISAM реалізована за допомогою тригерів. Для кожної з таблиць необхідно передбачити ситуації які будуть впливати на цілісність даних.

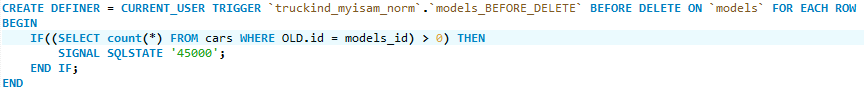
Для забезпечення цілісності даних для таблиці marks, при операції видалення ми перевіримо, що в таблиці models не присутній зовнішній ключ на цей запис, і якщо він присутній то ми спеціальною командою SIGNAL\_SQLSTATE та код сигналу сповістим, що видалення неможливе, якщо зовнішній зв’язок відсутній то ми дозволяємо видалення, даний тригер має наступний вигляд

 Рисунок 2.2 – Тригер для підтримання цілісності при видалення записів з таблиці marks

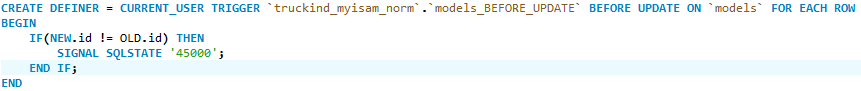
Для операції оновлення в таблиці marks потрібно заборонити оновлення, якщо змінюється первинний ключ, тому тригер перевіряє що при оновленні не змінюється первинний ключ, і якщо ця умова не виконується то сповіщаємо командою SIGNAL SQLSTATE, що оновлення неможливе, даний тригер має наступний вигляд

 Рисунок 2.3 – Тригер для підтримання цілісності при оновленні записів в таблиці marks

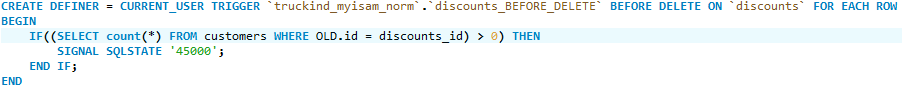
Для забезпечення цілісності даних для таблиці models, при операції DELETE ми перевіряємо, що в таблиці cars не існує зовнішнього ключа на даний запис, а якщо він присутній ми забороняємо видалення, при цьому командою SIGNAL SQLSTATE сповіщаємо що видалення неможливе. Даний тригер має наступний вигляд

 Рисунок 2.4 – Тригер для підтримання цілісності при видаленні записів в таблиці models

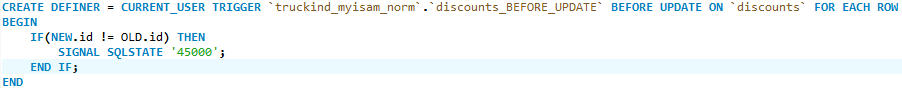
Для операції оновлення в таблиці models потрібно оновлення, якщо змінюється первинний ключ, для цього тригер перевіряє що первинний ключ не змінюється і тільки в цьому випадку дозволяє оновити дані, якщо ж змінюється первинний ключ то запороняється операція UPDATE завдяки команді SIGNAL SQLSTATE. Код даного тригера має наступний вигляд

 Рисунок 2.5 – Тригер для підтримання цілісності при оновленні записів в таблиці models

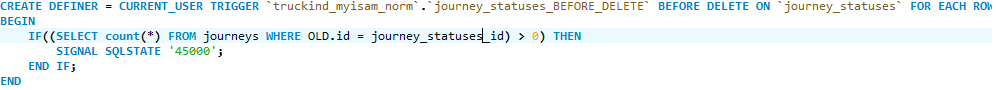
Для забезпечення цілісності даних для таблиці discount ми виконуємо дії аналогічні для інших підставних таблиць. А саме, для операції DELETE перед видаленням ми перевіряємо чи немає записів в таблиці customers за таким ідентифікатором, який є первинним ключем, і якщо він присутній то ми забороняємо видалення завдяки команді SIGNAL SQLSTATE. Код даного тригера має наступний вигляд

 Рисунок 2.6– Тригер для підтримання цілісності при видаленні записів в таблиці discounts

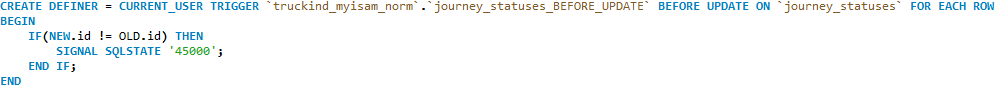
Для забезпечення цілісності при оновленні даних в таблиці discounts ми забороняємо оновлення запису в тому випадку, якщо змінюється первинний ключ в даній таблиці, для цього в разі такої ситуації ми посилаємо SIGNAL SQLSTATE. Код даного тригера має наступний вигляд

 Рисунок 2.7 – Тригер для підтримання цілісності при оновленні записів в таблиці discounts

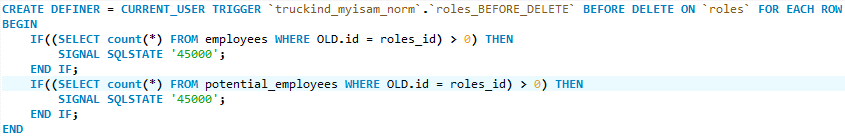
Для забезпечення цілісності даних при видаленні записів з таблиці journey\_statuses ми перевіряємо що в таблиці journeys немає записів з таким первинним ключем, а якщо є то забороняємо видалення завдяки команді SIGNAL SQLSTATE. Код даного тригера має наступний вигляд

 Рисунок 2.8 – Тригер для підтримання цілісності при видаленні записів з таблиці journey\_statuses

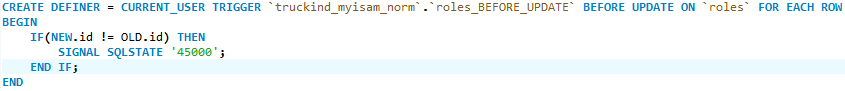
Для забезпечення цілісності даних при оновленні записів з таблиці journey\_statuses нам слід передбачити ситуацію коли будуть змінювати первинний ключ, і якщо його будуть змінювати то ми заборонимо операцію завдяки команді SIGNAL SQLSTATE. Код даного тригера має наступний вигляд

 Рисунок 2.9 – Тригер для підтримання цілісності при оновленні записів в таблиці journey\_statuses

Для забезпечення цілісності для таблиці roles при видаленні записів слід передбачити, що задані ролі уже використовуються в таблицях employees або potential\_employees, тому потрібно перевірити наявність зовнішніх ключів в цих таблицях і при їх наявності заборонити видалення завдяки команді SIGNAL SQLSTATE. Код даного тригера має наступний вигляд

 Рисунок 2.10 – Тригер для підтримання цілісності при видаленні записів з таблиці roles

Для забезпечення цілісності даних при оновленні даних в таблиці roles слід зупинити виконання операції UPDATE командою SIGNAL SQLSTATE, якщо при цій операції змінюється первинний ключ, що є недопустимим. Код даного тригера має наступний вигляд

 Рисунок 2.11 – Тригер для підтримання цілісності при оновленні записів в таблиці roles

Слід також згадати, що може виникнути ситуація при вставці нових записів в таблицю models, при якій ми намагатимемось вставити зовнішній ключ на таблицю marks, якого може не існувати, для цього нам слід заборонити вставку командою SIGNAL SQLSTATE якщо така ситуація сталася, тому перед вставкою будем перевіряти наявність такого ідентифікатора в таблиці marks. Код даного тригеру має наступний вигляд

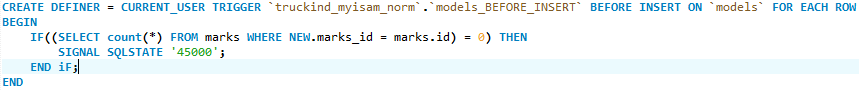
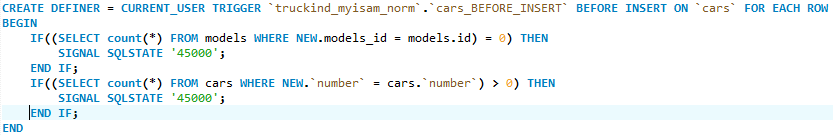


Рисунок 2.12 – Тригер для підтримання цілісності даних при вставці нових записів в таблицю models

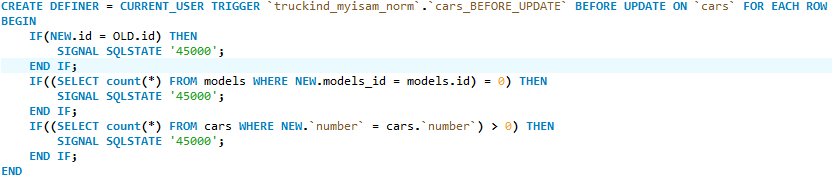
Для підтримання цілісності даних в таблиці cars нам потрібно заборонити вставку нових записів якщо не існує такого зовнішнього ключа на таблицю model, а також потрібно перевірити що в системі немає автомобіля за таким номером, адже номер автомобіля є унікальним в рамках бізнес-процесів, якщо одна з цих умов не буде виконуватися ми перервимо виконання вставки командою SIGNAL SQLSTATE. Код даного тригера має наступний вигляд

 Рисунок 2.13 – Тригер для підтримання цілісності при ставці нових записів в таблицю cars

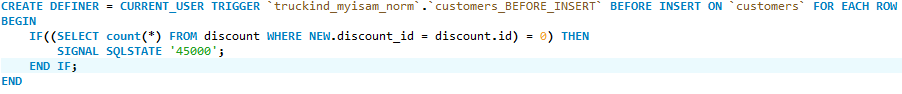
Для підтримання цілісності даних при виконанні операцій DELETE ми повністю забороняємо виконання даної операції, жоден автомобіль, який потрапив до системи не може бути видалений, адже це необхідно для звітності і статистики, тому в таблиці cars присутня колонка is\_active, яка повідомляє про те чи є автомобіль активним для бізнес-процесів чи він списаний з використання. Код тригеру для цієї операції має наступний вигляд

 Рисунок 2.14 – Тригер для підтримання цілісності при видаленні даних з таблиці cars

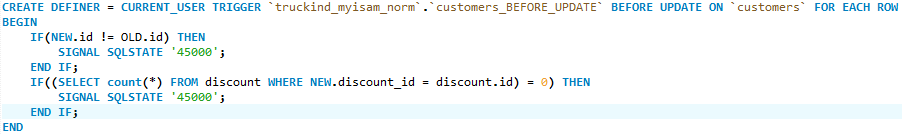
Для підтримання цілісності даних при виконанні операцій UPDATE на таблиці cars ми перевіряємо чи не змінюється первинний ключ, та якщо змінюється номер автомобілю чи немає такого номеру уже в системі, та якщо виконується зміна моделі автомобіля, слід перевірити, що існує сутність з таким зовнішнім ключем

 Рисунок 2.15 – Тригер для підтримання цілісності даних при оновленні даних в таблиці cars

Для підтримання цілісності даних при виконанні операцій INSERT до таблиці customers, нам необхідно перевірити що існує такий зовнішній ключ до таблиці discount, якщо існує ми забороняємо вставку командою SIGNAL SQLSTATE. Код даного тригера має вигляд

 Рисунок 2.16 – Тригер для підтримання цілісності при вставці до таблиці customers

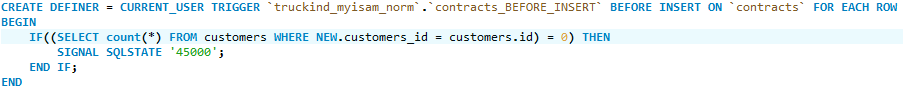
Для підтримання цілісності даних при оновленні записів в таблиці customers нам необхідно перевіряти що існує такий зовнішній ключ на таблицю discount, а також що ми не змінюємо первинний ключ, якщо одна з умов не виконується то виконуємо команду SIGNAL SQLSTATE. Код даного тригера має вигляд

 Рисунок 2.17 – Тригер для підтримання цілісності при оновленні записів в таблиці customers

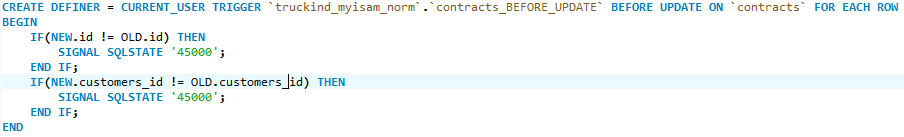
Для підтримання цілісності при видаленні записів з таблиці customers ми повністю забороняємо видалення клієнта, адже інформація про нього завжди необхідна для звітності. Тому код тригера має такий вигляд

 Рисунок 2.18 – Тригер для підтримання цілісності при операціях видалення з таблиці customers

Для підтримання цілісності даних при вставці даних в таблицю contracts ми перевіряємо наявність заданого зовнішнього ключа в таблиці customers, і якщо відсутній то сигналізуємо про помилку. Код тригера має наступний вигляд

 Рисунок 2.19 – Тригер для підтримання цілісності при вставці в таблицю contracts

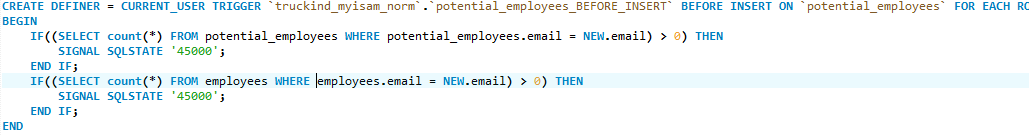
Щоб забезпечити цілісність при оновленні даних в таблиці contracts ми перевіряємо що не змінюється первинний ключ, а також що не змінюється клієнт, адже це не підлягає бізнес-процесам. Тому код тригера має наступний вигляд

 Рисунок 2.20 – Тригер для забезпечення цілісності при оновленні таблиці contracts

Для забезпечення цілісності даних при операціях DELETE на таблиці слід взагалі заборонити виконання цієї операції. Тому код тригера матиме наступний вигляд

 Рисунок 2.21 – Тригер для підтримання цілісності при видаленні записів з таблиці contracts

Для забезпечення цілісності при операціях вставки до таблиці potential\_employees нам слід переглянути чи немає в системі заданої електронної пошти серед всіх працівників та потенційних працівників, адже електронна пошта є унікальною характеристикою для особи. Тому при спробі створити потенційного працівника з існуючою електронною поштою ми повинні сигналізувати про помилку, також слід сигналізувати про помилку якщо ми додаємо неіснуючу роль. Тому код тригера має вигляд

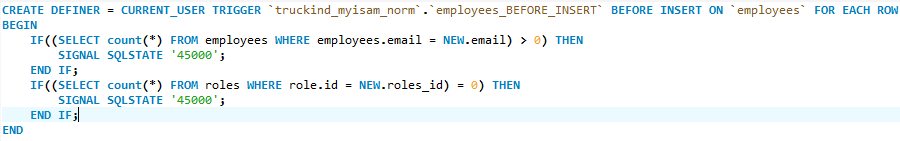
 Рисунок 2.22 – Тригер для підтримання цілісності при вставці записів до таблиці potential\_employees

В рамках бізнес-процесів компанії немає необхідності в оновленні записів таблиці potential\_employees, тому ми будемо сигналізувати про помилку при спробі зміни запису. Тому тригер має наступний вигляд

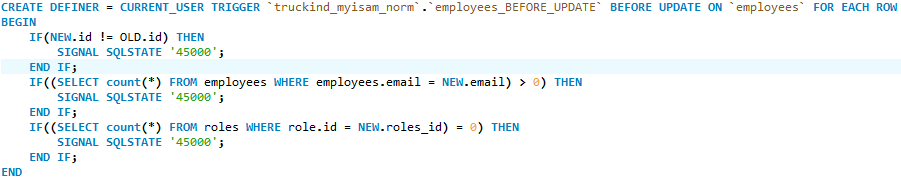
 Рисунок 2.23 – Тригер для підтримання цілісності при оновленні записів в таблиці potential\_employees

Для видалення записів з таблиці potential\_employees немає ніяких обмежень, тому в тригерові немає необхідності.

Для підтримання цілісності даних при вставці нових записів до таблиці employees на слід перевірити наявність зовнішнього ключа на таблицю role, а також перевірити відсутність такої електронної пошти в таблиці, оскільки це є унікальна характеристика працівника. Тому тригер має вигляд

 Рисунок 2.24 – Тригер для забезпечення цілісності при вставці записів до таблиці employees

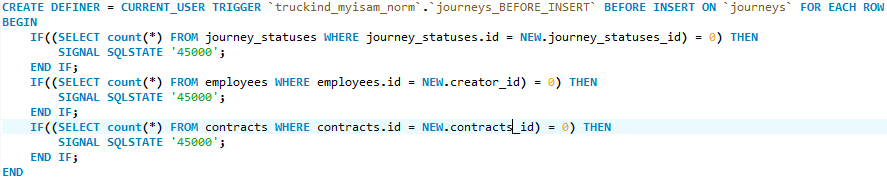
Для забезпечення цілісності при оновленні даних в таблиці employees слід передбачити можливість оновлення електронної пошти, яка все використовується в системі, або помилковий зовнішній ключ на таблицю roles, або намагаються змінити первинний ключ, в таких випадках слід сигналізувати про помилку при вставці. Тому тригер має вигляд

 Рисунок 2.25 – Тригер для забезпечення цілісності при оновленні таблиці employees

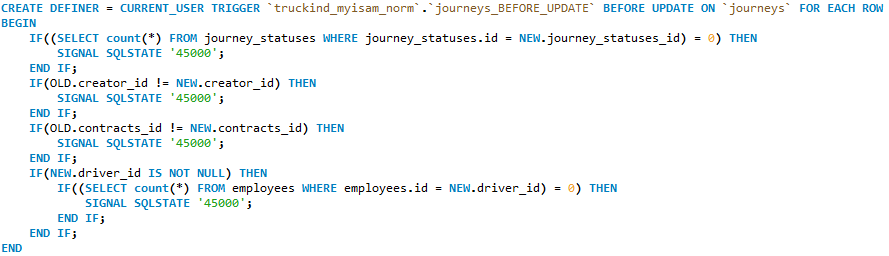
Операцій DELETE на таблиці employees слід заблоковувати, адже потрібно зберігати історію всіх працівників в компанії, якщо користувач більше не приймає участь в бізнес-процесах компанії, його слід зробити неактивним, але не видаляти запис з таблиці. Тому тригер має наступний вигляд

 Рисунок 2.26 – Тригер для заборони видалення записів з таблиці employees

Для забезпечення цілісності даних при вставці нових записів до таблиці journeys необхідно перевірити що існують такі зовнішні ключі на таблиці journey\_statuses, contracts\_id, creator\_id, якщо дані умови не виконуються то слід сигналізувати про виняткову ситуацію і зупинити вставку. Тому тригер має наступний вигляд

 Рисунок 2.27 – Тригер для підтримання цілісності при вставці записів в таблицю journeys

Для забезпечення цілісності даних при оновленні записів в таблиці journeys необхідно перевірити що існують такі зовнішні ключі на таблиці journey\_statuses, driver\_id, до того ж заборонити зміну contracts\_id, creator\_id, якщо дані умови не виконуються то слід сигналізувати про виняткову ситуацію і зупинити вставку. Тому тригер має наступний вигляд

 Рисунок 2.28 – Тригер для підтримання цілісності при оновленні записів в таблицю journeys

Для забезпечення цілісності даних при видаленні записів в таблиці journeys необхідно заборонити виконувати операції DELETE, оскільки це суперечить бізнес-процесам компанії. Тому тригер має наступний вигляд

 Рисунок 2.29 – Тригер для підтримання цілісності при видаленні записів з таблиці journeys

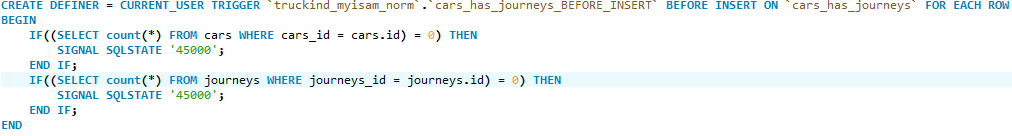
Для забезпечення цілісності даних при видаленні записів в таблиці journeys необхідно заборонити виконувати операції DELETE, оскільки це суперечить бізнес-процесам компанії. Тому тригер має наступний вигляд

 Рисунок 2.30 – Тригер для підтримання цілісності при видаленні записів з таблиці journeys

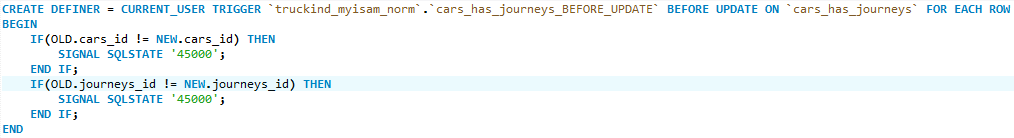
Для забезпечення цілісності даних при видаленні записів в таблиці cars\_has\_journeys необхідно заборонити виконувати операції DELETE, жоден запис з цієї таблиці не може бути видалений. Тому тригер має наступний вигляд

 Рисунок 2.31 – Тригер для підтримання цілісності при видаленні записів з таблиці cars\_has\_journeys

Для забезпечення цілісності даних при вставці записів до таблиці cars\_has\_journeys необхідно перевірити наявність таких зовнішніх ключів в таблицях cars та journeys. Тому тригер має наступний вигляд

 Рисунок 2.32 – Тригер для підтримання цілісності при вставці записів до таблиці cars\_has\_journeys

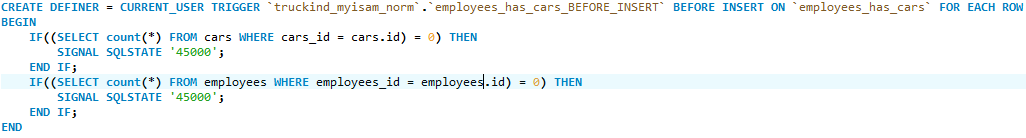
Для забезпечення цілісності даних при оновлення записів в таблиці cars\_has\_journeys необхідно заборонити виконувати операції UPDATE, якщо намагаються змінити зовнішні ключі на таблиці cars та journeys. Тому тригер має наступний вигляд

 Рисунок 2.33 – Тригер для підтримання цілісності при видаленні записів з таблиці cars\_has\_journeys

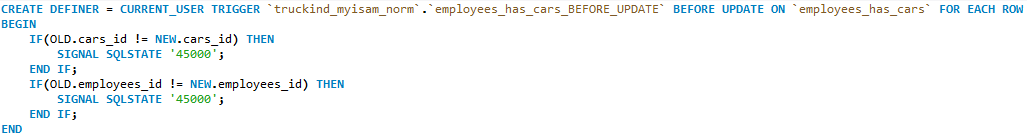
Для забезпечення цілісності даних при видаленні записів в таблиці employees\_has\_cars необхідно заборонити виконувати операції DELETE, жоден запис з цієї таблиці не може бути видалений. Тому тригер має наступний вигляд

 Рисунок 2.34 – Тригер для підтримання цілісності при видаленні записів з таблиці employees\_has\_cars

Для забезпечення цілісності даних при вставці записів до таблиці employees\_has\_cars необхідно перевірити наявність таких зовнішніх ключів в таблицях cars та employees. Тому тригер має наступний вигляд

 Рисунок 2.35 – Тригер для підтримання цілісності при вставці записів до таблиці employees\_has\_cars

Для забезпечення цілісності даних при оновлення записів в таблиці employees\_has\_cars необхідно заборонити виконувати операції UPDATE, якщо намагаються змінити зовнішні ключі на таблиці cars та journeys. Тому тригер має наступний вигляд

 Рисунок 2.36 – Тригер для підтримання цілісності при видаленні записів з таблиці employees\_has\_cars

## 2.3 Реалізація бізнес функцій високонавантаженої інформаційної системи на стороні MySQL сервера

Усі описані бізнес функції в розділі 1.2 були реалізовані за допомогою функцій і процедур. Відповідність бізнес функцій з функціями або процедурами можна побачити у таблиці 2.3.1. Реалізацію бізнес функцій можна побачити на рисунках 2.3.1-2.3.12

Таблиця 2.3.1 – Бізнес-функції та відповідні їм функції та процедури

|  |  |
| --- | --- |
| Бізнес функція | Назва функції або процедури |
| Пошук користувачів за ім’ям, прізвищем, електронною поштою | find\_employees\_by\_name\_surname\_email |

Продовження таблиці 2.3.1

|  |  |
| --- | --- |
| Бізнес функція | Назва функції або процедури |
| Списання автомобілю з використання | remove\_car |
| Додання до системи нового автомобілю | add\_car |
| Зміна характеристик автомобілів | edit\_car |
| Реєстрація нового співробітника | add\_employee |
| Переведення працівника до категорії неактивних(при звільненні) | remove\_employee |
| Огляд рейсів, фільтрація за датою створення та завершення, за ідентифікатором, статусом | get\_journeys |
| Зміна електронної пошти та паролю | update\_password\_and\_email |
| Вхід | try\_to\_login |
| Подання заяви на працевлаштування в компанії | send\_resume |
| Замовлення на перевезення вантажу | create\_order\_carrying |
| Створення рейсу | create\_journey |
| Підтвердження автомобіля на рейс(для менеджерів тільки ним створених) | accept\_driver\_journey |
| Відміна рейсу(для менеджерів тільки ним створених) | cancel\_journey |
| Перегляд автомобілів доступних для нього | get\_garage |
| Підписання на рейс | subscribe\_journey |
| Переведення автомобіля в положення «Зламаний»/«Справний» | set\_fix\_status\_car |
| Підтвердження виконання рейсу | confirm\_journey |

Для зручного доступу до інформації про працівників компанії доречно створити представлення, яке в потрібній формі буде надавати інформацію про працівників, для перегляду інформації необхідні наступні поля: ім’я, прізвище, електронна пошта, кількість днів в компанії, його роль, та заробітня платня(тільки для адміністраторів), до того ж для адміністраторів є можливість переглядати користувачів які неактивні, якщо він поставить відповідний флаг. Для реалізацій цих операцій досить зручно створити допоміжну функцію, яка надає ідентифікатор ролі за її текстовим значенням. Код даної функції має вигляд

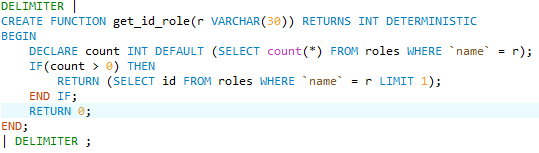


Рисунок 2.37 – Реалізація функції get\_id\_role

До того ж досить зручно створити оборотню функцыю, яка надає назву ролі за її ідентифікатором. Код такої функції має вигляд

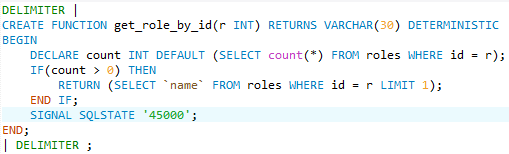
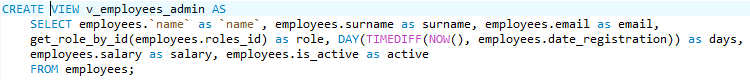


Рисунок 2.38 – Реалізація функції get\_role\_by\_id

Після таких підготовчих дій можливо створити представлення для отримання інформації про працівників компанії, представлення буде, одне для адміністратора, і для інших працівників. Представлення для адміністратора має вигляд

Рисунок 2.39 – Реалізація представлення v\_employees\_admin

Далі слід розробити представлення для користування будь-якими працівниками компанії, яке буде виключати поля заробітньої плати та присутності в системі, для цих користувачів будуть доступні працівники тільки активні. Тоді код представлення матиме вигляд

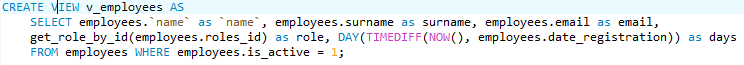
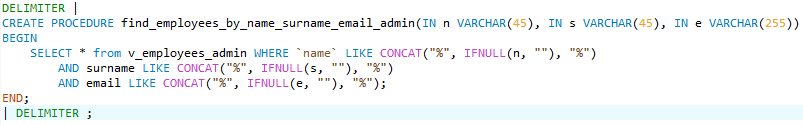
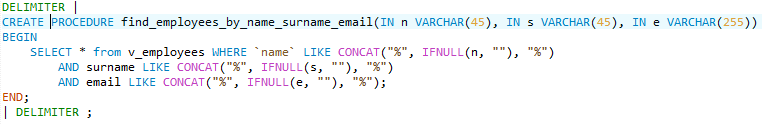


Рисунок 2.40 – Реалізація представлення v\_employees

Зробивши ці підготовчі дії можна приступити до реалізації процедури, котра знаходить користувачів заданим ім’ям, прізвищем, електронною поштою, для того щоб не шукати за одним із параметрів слід вказати це параметр як NULL. Тоді реалізація даної процедури матиме 2 форми, для адміністратора і для будь-якого користувача. Для адміністратора процедура має вигляд

 Рисунок 2.41 – Реалізація процедури find\_employees\_by\_name\_surname\_email\_admin

Для всіх користувачів дана процедура матиме аналогічний вигляд, за винятком використовування іншого представлення для отримання даних. Код даної процедури має наступний вигляд

 Рисунок 2.42 – Реалізація процедури find\_employees\_by\_name\_surname\_email

Для реалізації процедури котра списує автомобіль з використання ми використаємо в якості вхідного параметру ідентифікатор автомобілю і оновимо запис. Код даної процедури має вигляд

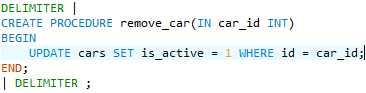
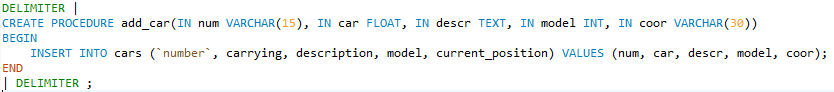
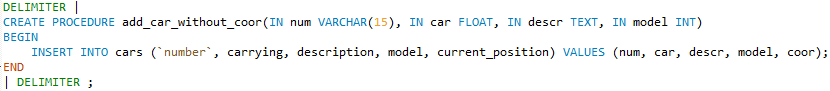


Рисунок 2.43 – Реалізація процедури remove\_car

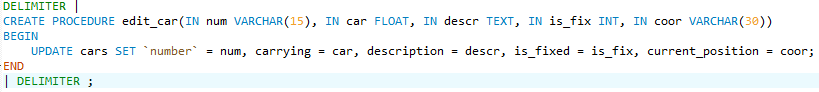
Для реалізації бізнес-процесу добавити автомобіль, створимо процедуру, в яку передамо всі необхідні параметри для сворення нового автомобіля в базі. Код даної процедури має вигляд

 Рисунок 2.44 – Реалізація процедури add\_car

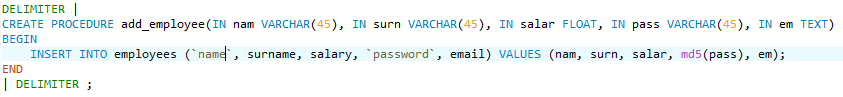
Було також створено аналогічну процедуру, але яка не приймате поточного положення автомобіля, а ставить його значенням за замовчуванням

 Рисунок 2.45 – Реалізація процедури add\_car\_without\_coor

Для реалізації бізнес-процесу зміни параметрів автомобілю ми реалізуємо процедуру edit\_car, котра приймає в себе основні параметри, які можна змінити: номер, вантажопідйомність, опис, справність, позицію. Дана процедура має наступний вигляд

 Рисунок 2.46 – Реалізація процедури edit\_car

Для реалізації бізнес-процесу створення працівника компанії було створено процедуру add\_employee яка створює новий запис в відповідній таблиці з усіма переданими даними як вхідні параметри до процедури, код такої процедури має наступний вигляд

 Рисунок 2.47 – Реалізація процедури add\_employee

Для реалізації бізнес-процесу по видаленню працівника при звільненні ми створюємо відповідну процедуру, яка прийме ідентифікатор працівника і встановить йому статус «неактивний». Код даної процедури має наступний вигляд

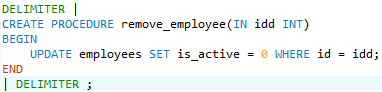


Рисунок 2.48 – Реалізація процедури remove\_employee

Для зручної роботи з інформацією про рейси було створено предсталення, яке з’єднує всі необхідні таблиці з даними, які можуть знадобитися для перегляду інформації. Таким чином дане представлення має вигляд

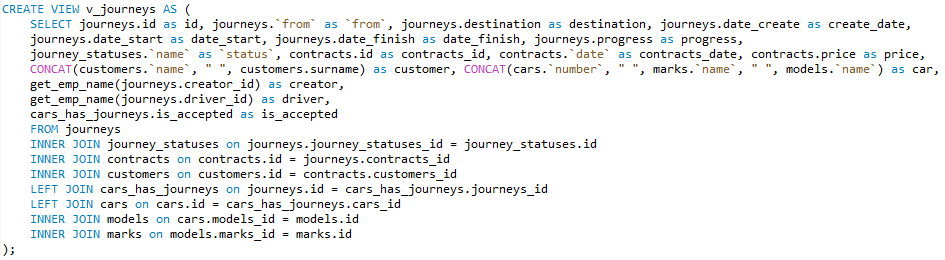


Рисунок 2.49 – Реалізація представлення v\_journeys

В ході реалізації даного представлення було прийняте рішення створити функцію, яка буде знаходити ім’я та прізвище працівника за його ідентифікатором. Код даної функції має вигляд

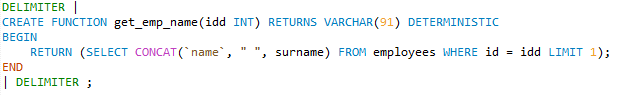


Рисунок 2.50 – Реалізація функції get\_emp\_name

Для реалізації бізнес-функції перегляд інформації про рейс було створено процедуру, яка приймає ідентифікатор рейсу і надає повну інформацію потрібну для звітності використовуючи створене раніше представлення. Таким чином дана процедура має вигляд

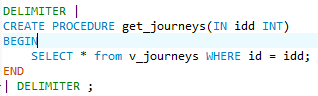


Рисунок 2.51 – Реалізація процедури get\_journeys

Для реалізації бізнес-функції зміни паролю та електронної пошти, для цього створемо функцію яка приймає в себе нову електронну пошту, новий пароль та підтверження нового паролю, якщо все успішно то повертаємо код 1, якщо паролі не співпадають то повертаемо 0. Таким чином дана функція має вигляд

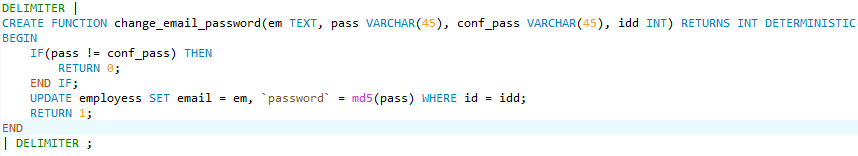


Рисунок 2.52 – Реалізація функції change\_email\_password

Для реалізації бізнес-функції подання резюме було створено процедуру, яка добавляє запис до таблиці potential\_employees, для цього до процедури передаються всі необхідні параметри. Таким чином дана функція має вигляд

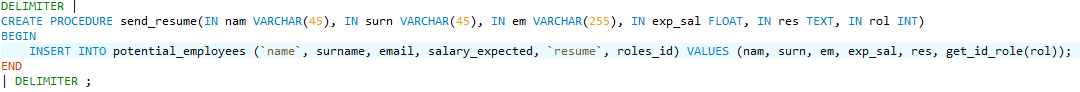


Рисунок 2.53 – Реалізація процедури send\_resume

Для реалізації бізнес-функції логіну створено функцію в яку передаємо електронну пошту і пароль, якщо невірна електронна пошта невірна то 0, якщо вірна електронна пошта вірна а пароль ні то код повертаємо 1, якщо все добре то код 2. Таким чином дана функція має вигляд

Для реалізації бізнес-функції подання резюме було створено процедуру, яка добавляє запис до таблиці potential\_employees, для цього до процедури передаються всі необхідні параметри. Таким чином дана функція має вигляд

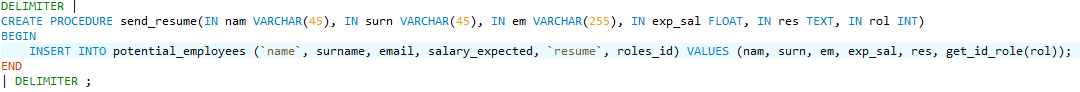


Рисунок 2.54 – Реалізація функції try\_to\_login

Для реалізації бізнес-функції відміни рейсу було створено процедуру котра приймає ідентифікатор рейсу та встановлює йому відповідний ідентифікатор статусу. Таким чином дана функція має вигляд

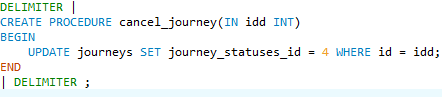


Рисунок 2.55 – Реалізація процедури cancel\_journey

Для реалізації бізнес-функції зміни статусу автомобіля справний чи несправний було реалізовано завдяки процедурі, котра приймає ідентифікатор автомобіля і код статусу і оновлює запис в таблиці з автомобілями. Таким чином дана функція має вигляд

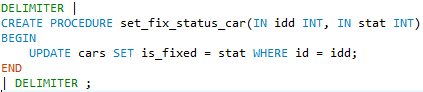


Рисунок 2.56 – Реалізація процедури set\_fix\_status\_car

Для реалізації бізнес-функції підтвердження виконання рейсу водієм виконується процедурою, котра приймає параметром ідентифікатор рейсу і оновлює відповідний запис. Таким чином дана функція має вигляд



Рисунок 2.57 – Реалізація процедури confirm\_journey

### 2.4 Реалізація транзакції для InnoDB.

Для InnoDB створення клієнта та його замовлення повинні добавитися атомарно, тобто ми ці дві операції вставки виконуємо транзакційно. Так як нам потрібно отримати згенерований ідентифікатор з вставки клієнта ми встановлюємо рівень ізоляції транзакції READ UNCOMMITTED, якщо при одній зі вставок виникне помилка сервер СУБД викине SQLEXCEPTION, тому ми його відловлюємо і в такому разі робимо ROLLBACK операцій, якщо все добре ми робимо COMMIT щоб зберегти дані. Так як ця транзакція може виконуватись на старих версіях сервера СУБД, до 5.3 то ми явним чином прописуємо виключення автоматичних збережень даних на час виконання транзакції, тому реалізація такої транзакції має вигляд

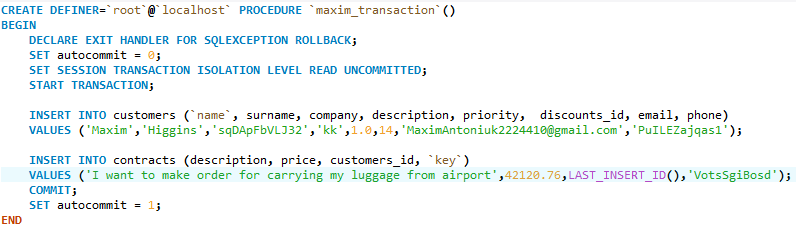


Рисунок 2.58 – Функція реєстрації з транзакцією

# 3 ОПТИМІЗАЦІЯ ВИСОКОНАВАНТАЖЕНИХ БАЗ ДАНИХ

## 3.1 Видалення підстановочних таблиць

Кожна підстановочна таблиця заздалегідь наштовхує користувача на використання операції JOIN. JOIN операція є складною для сервера MySQL. Для того щоб кількість операцій JOIN звести до мінімуму і це не впливало на роботу високонавантаженої системи бази даних потрібно всі підстановочні таблиці, кортежі якої будуть константими, винести у ENUM тієї таблиці де використовується підстановочна, та видалити ці таблиці, таким чином для отримання інформації з цих підстановочних таблиць нам досить взяти інформацію з основної таблиці. На рисунку 3.1 схема оптимізована схема даних без підстановочних таблиць.

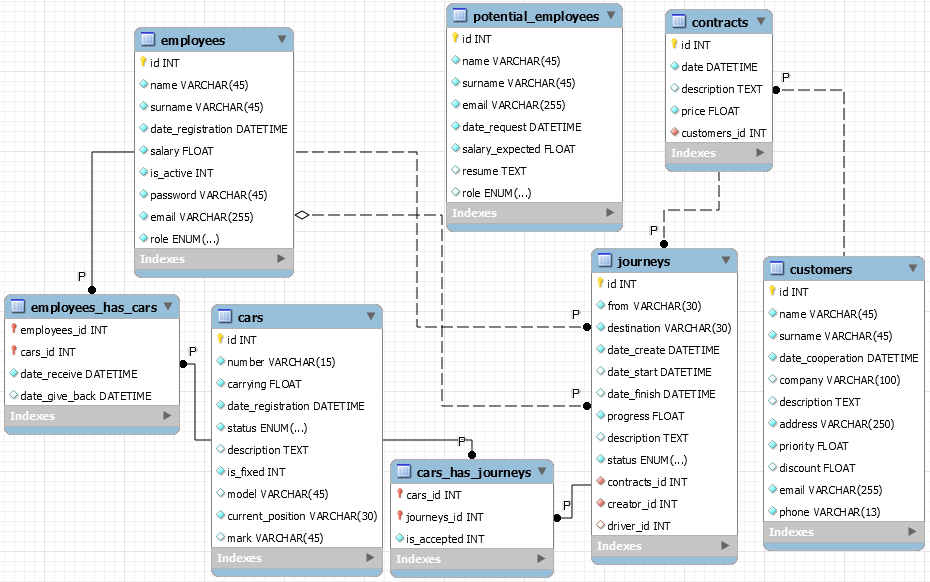


Рисунок 3.1 – Схема даних без підстановочних таблиць

У таблиці 3.1.1 наведено які підстановочні таблиці були замінені на назви нових атрибутів.

Таблиця 3.1.1 – Підстановочні таблиці та їх заміни

|  |  |  |
| --- | --- | --- |
| Таблиця | Підстановочна таблиця | Назва нового атрибута |
| cars | marks | mark |
| employees | roles | role |
| cars | models | model |
| customers | discounts | discount |
| potential\_employees | roles | role |
| journeys | journey\_statuses | status |

Слід зауважити, що в таблиці з клієнтами підстановочна таблиця зі знижками стала не перерахуванням а просто полем типу числа з плаваючою точкою.

## 3.2 Зміна типів атрибутів

Зміна типів атрибутів дозволяє в деяких місцях виділяти на один кортедж менше пам’яті. В розробленій базі даних не було таких типів, котрі можна б було зменшити за розміром. Але база даних має атрибути з типом TEXT. Деякі типи TEXT були замінені на VARCHAR(n), окрім описів, оскільки вони можуть займати великий розмір і зберігати його безпосередньо в пам’яті не вигідно.

## 3.3 Додавання індексів

### 3.3.1 Додавання FULLTEXT індексів

FULLTEXT INDEX дозволяють зберігати набір текстових даних у певному словнику. Для усіх конструкцій MATCH AGAINST буде застосовуватися BOOLEAN MODE, за для того , щоб визначати є такий запитуємий текстовий контент, чи він відсутній. BOOLEAN MODE працює декілька швидше, тому він і використовується, адже він просто перевіряє на наявність входження.

В системі найнавантаженішим запитом є запит до таблиці з рейсами та робота зі статистикою, ми об’єднали всі необхідні таблиці до одного представлення, для зручного доступу до інформації, слід переглянути хід виконання запросу якщо ми намагаємося знайти рейси з якоюсь фразою в описі

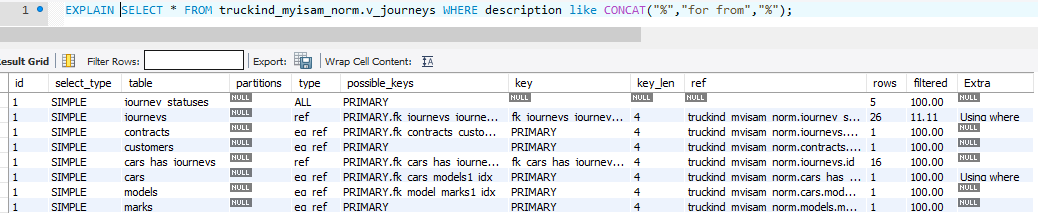


Рисунок 3.2 – EXPLAIN для неоптимізованого запиту

Після того як ми створили повтотекстовий індекс для опису в таблиці рейсів, та знову повторили операцію пошуку були виявлені покращення щодо роботи запиту. Результат виконання дослідження виконання запиту має такий вигляд

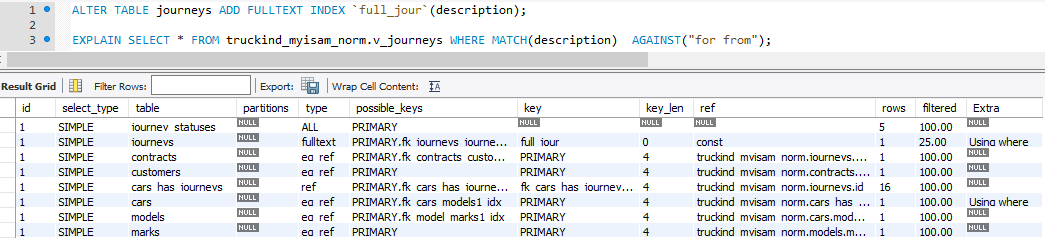


Рисунок 3.3 – EXPLAIN для оптимізованого запиту

### 3.3.2 Аналіз використання кластерних індексів в InnoDB

InnoDB є одним із найсильніших механізмів храніння даних в MySQL, однією із причин цього є наявність кластерних індексів в InnoDB. Кластерні індекси дозволяють зберігати дані у деревовидній структурі записів. При чому записи зберігаються не унікальним ідентифікатором, а повністю записує увесь кортеж даних. У момент пошуку записів СУБД рухається по дереву записів і шукає запитуєме значення, разом з яким зберігаються усі інші дані запису.

У InnoDB усі PRIMARY та UNIQUE індекси є кластерними. Якщо у таблиці немає ні PRIMARY індексів ні UNIQUE індексів, то MySQL виділяє на кожний кортеж по 6 байт на імітований ключ, який не буде видно при отриманні результатів, але який однозначно ідентифікує кортеж.

У розробленій високонавантаженій базі даних в усіх таблицях є PRIMARY індекс, тому додавати їх не потрібно.

### 3.3.3 Додавання індексів типу INDEX

У розробленій високонавантажені базі даних було добавлено індекси на всі зовнішні ключі для таблиць типу MyISAM, що надало приросту швидкості.

Повний перелік даних індексів наведений нижче

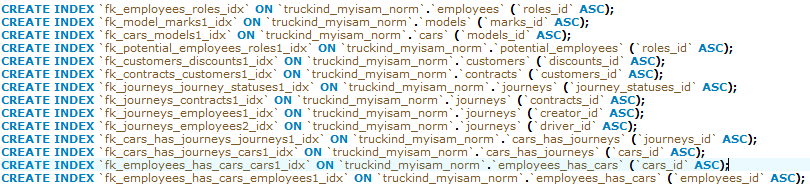
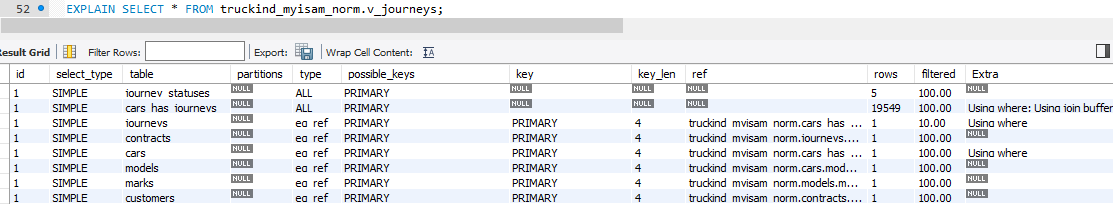


Рисунок 3.4 – Індекси для таблиць типу MyISAM

Для порівняння проаналізуємо запит без використання індексів та з ними. Варіант запиту без використання індексів

 Рисунок 3.5 – EXPLAIN без індексів

Після створення індексів оптимізатор серверу СУБД почав використовувати індекси для зв’язування таблиць і це призвело до збільшення швидкості. Результати виконання оптимізованого запиту мають наступний вигляд

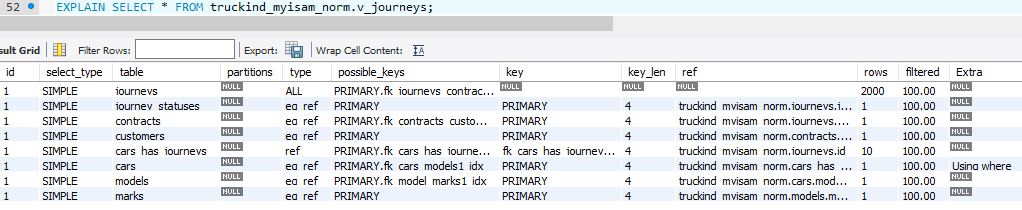


Рисунок 3.6 – EXPLAIN з індексами

### 3.3.4 Додавання префіксного індексу

Для розробленої системи було прийняте рішення пошуку працівників за ім’ям та прізвищем. Для прискорення цього процесу слід використати індекс, надавати унікальний індекс ми не можемо, адже імена можуть повторюватися, в такому випадку найкращим рішенням буде створити префіксний індекс. Для створення такого індексу слід спочатку дослідити довжину індексу. Так як імена мають довжину не довшу за 10-15 символів, то нам слід розглядати префікс до цих довжин, далі показані дослідження селективності для префіксу

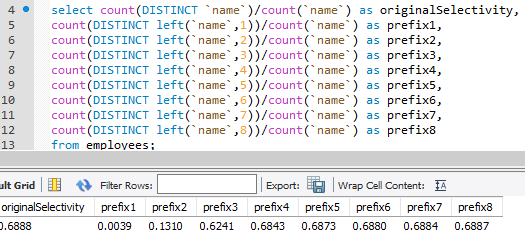


Рисунок 3.7 – Дослідження селективності для імені

Як видно з результатів дослідження майже абсолютної селективності префікс набуває при довжині в 8 символів, але можна помітити що величина селективності у префікса довжиною в 4 символи має майже таке ж значення, тому доцільно вибрати довжиною префіксу 4 символи. Слід перевірити приріст швидкодії з префіксом, на наступному рисунку зображено хід виконання вибірки без префіксного індексу

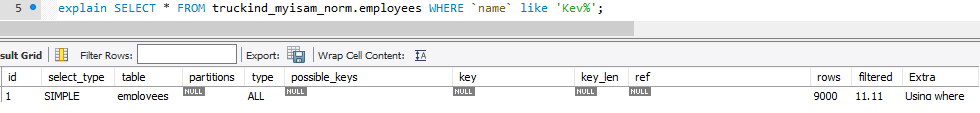


Рисунок 3.8 – EXPLAIN без префіксного індексу

Після цього ми виконали команду для створення префіксного індексу на 4 символи, код якого має наступний вигляд



Рисунок 3.9 – Створення префіксного індексу

Після цього була проведена ще одна операція EXPLAIN для перегляду приросту швидкості при використанні префіксного індексу. Результати мають такий вигляд

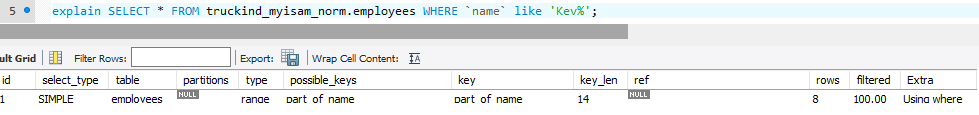


Рисунок 3.10 – EXPLAIN з префіксним індексом

## 3.4 Денормалізація баз даних

Денормалізація баз даних дуже важливий момент, який може збільшити швидкість певних запитів для високонавантаженої системи зберігання даних. У розроблені базі процедура пошуку інформації про автомобілі використовувала 2 операції INNER JOIN, як уже позначалося раніше було винесено значення таблиць marks і models як окремі поля в таблиці cars. Тому частина схеми даних яка раніше мала наступний вигляд

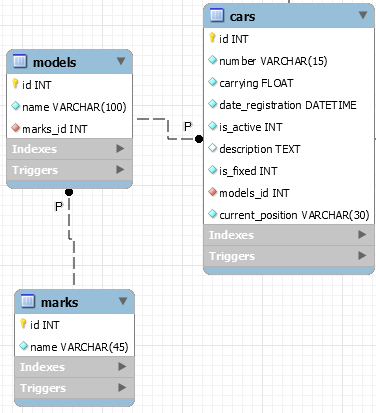


Рисунок 3.11 – Інформація про автомобілі до денормалізації

Для оптимізації виконання запиту ми всі ці випадаючі таблиці перенесли до основної, і ця частина схеми даних тепер має наступний вигляд

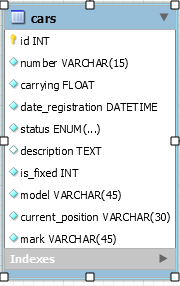


Рисунок 3.12 – Інформація про автомобілі після денормалізації

## 3.5 Шардинг

### 3.5.1 Вертикальний шардинг

Для вертикального шардингу можна рознести employees, на 2 таблиці, активних та завільнених, що прискорить роботу з працівниками, адже доступ до звільнених потрібен лише у виняткових ситуаціях. Якщо даного рішення недостатньо для оптимізації роботи, дану таблицю можна рознести по ролям на 3 таблиці, таким чином найбільшого навантаження отримають таблиці водіїв та менеджерів, і вони будуть розподілені що дасть прискорення в роботі. Тоді вибір працівників операцією SELECT не потрібно буде обмежувати оператором WHERE по типу.

Також можна рознести таблицю cars на 2 таблиці, активних та списаних автомобілів, це прискорить доступ до інформації про автомобілі, а враховуючи постійне оновлення даних про позицію автомобіля з GPS датчиків на автомобілі таке рознесення на таблиці буде оптимальним рішенням.

Таким чином схема шардування таблиці працівників може мати наступний вигляд, при розгляданні ситуації рознесення по різним таблицям активних і не активних працівників

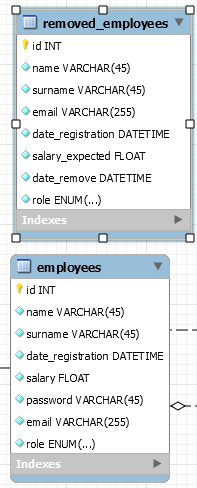


Рисунок 3.13 – Вертикальний шардинг таблиці працівників

### 3.5.2 Горизонтальний шардинг

Таблиці journeys, cars\_has\_journeys, cars є найбільш запитуємими, тому винести їх на окремий сервер, щоб для них був окремий пул з’єднань буде оптимальним рішенням.

# ВИСНОВКИ

У ході виконання курсового проекту була спроектована та розроблена високонавантажена система зберігання даних для інформаційної системи «Вантажні перевезення». База даних була спроектована у двох варіантах: InnoDB, MyIsam. Обидві БД були оптимізовані та заповнені тестовими даними. Усі висунуті бізнес-функції були реалізовані за допомогою функцій, процедур та тригерів, для зручності та обмеження доступу ми використали представлення для роботи з даними. Поставлена задача була виконана у повному обсязі.

Розроблена база даних може використовуватися у комерційних цілях компаніями вантажних перевезень. База досить гнучка завдяки широкому спектру задач націлених на неї, тому її просто інтегрувати до використання.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Методические указания к междисциплинарного курсового проекта по для студентов направления подготовки 6.050101 - Компьютерные науки [Электронное издание] / Сост .: Ю.В. Мищеряков, З.А. Имангулова., Л.В. Колесник - Харьков: ХНУРЭ, 2016. - 30 с.
2. Справочник по MySQL [Электронный ресурс] - Режим доступа URL: https://dev.mysql.com
3. Справочник по Workbench [Электронный ресурс] - Режим доступа URL: https:// dev.mysql.com/doc/workbench/en/
4. Работа с базами данных: учебное пособие / сост. О. Н. Евсеева, А. Б. Шамшев. – Ульяновск: УлГТУ, 2009. – 170 с.

# ДОДАТОК А

# КОД ГЕНЕРАЦІЇ НЕОПТИМІЗОВАНОЇ БД ТИПУ MYISAM

# -- MySQL Workbench Forward Engineering

# SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

# SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

# SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='TRADITIONAL,ALLOW\_INVALID\_DATES';

# -- -----------------------------------------------------

# -- Schema truckind\_myisam\_norm

# -- -----------------------------------------------------

# CREATE SCHEMA IF NOT EXISTS `truckind\_myisam\_norm` DEFAULT CHARACTER SET utf8 ;

# USE `truckind\_myisam\_norm` ;

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`roles`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`roles` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`roles` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `name` VARCHAR(45) NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM;

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`employees`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`employees` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`employees` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `name` VARCHAR(45) NOT NULL,

# `surname` VARCHAR(45) NOT NULL,

# `employeescol` VARCHAR(45) NOT NULL,

# `date\_registration` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

# `salary` FLOAT NOT NULL,

# `is\_active` INT NOT NULL DEFAULT 1,

# `password` VARCHAR(45) NOT NULL,

# `email` VARCHAR(255) NOT NULL,

# `role` INT NOT NULL,

# `roles\_id` INT NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`marks`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`marks` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`marks` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `name` VARCHAR(45) NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM;

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`models`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`models` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`models` (

# `id` INT NOT NULL,

# `name` VARCHAR(100) NOT NULL,

# `marks\_id` INT NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM;

# ----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`cars`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`cars` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`cars` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `number` VARCHAR(15) NOT NULL,

# `carrying` FLOAT NOT NULL,

# `date\_registration` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

# `is\_active` INT NOT NULL DEFAULT 1,

# `description` TEXT NULL,

# `is\_fixed` INT NOT NULL DEFAULT 1,

# `models\_id` INT NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM;

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`potential\_employees`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`potential\_employees` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`potential\_employees` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `name` VARCHAR(45) NOT NULL,

# `surname` VARCHAR(45) NOT NULL,

# `email` VARCHAR(255) NOT NULL,

# `date\_request` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

# `salary\_excepted` FLOAT NOT NULL,

# `resume` TEXT NOT NULL,

# `roles\_id` INT NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM;

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`journey\_statuses`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`journey\_statuses` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`journey\_statuses` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `name` VARCHAR(45) NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM;

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`discounts`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`discounts` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`discounts` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `name` VARCHAR(45) NOT NULL,

# `percent` FLOAT NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM;

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`customers`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`customers` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`customers` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `name` VARCHAR(45) NOT NULL,

# `surname` VARCHAR(45) NOT NULL,

# `date\_cooperation` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

# `company` VARCHAR(100) NULL,

# `description` TEXT NULL,

# `address` VARCHAR(250) NOT NULL,

# `priority` FLOAT NOT NULL DEFAULT 1,

# `discounts\_id` INT NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM;

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`contracts`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`contracts` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`contracts` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `date` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

# `description` TEXT NOT NULL DEFAULT 'no description',

# `price` FLOAT NOT NULL,

# `customers\_id` INT NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM

# -- -----------------------------------------------------

# -- Table `truckind\_myisam\_norm`.`journeys`

# -- -----------------------------------------------------

# DROP TABLE IF EXISTS `truckind\_myisam\_norm`.`journeys` ;

# CREATE TABLE IF NOT EXISTS `truckind\_myisam\_norm`.`journeys` (

# `id` INT NOT NULL AUTO\_INCREMENT,

# `from` VARCHAR(30) NOT NULL,

# `destination` VARCHAR(30) NOT NULL,

# `date\_create` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

# `date\_start` DATETIME NULL,

# `date\_finish` DATETIME NULL,

# `progress` FLOAT NOT NULL DEFAULT 0.0,

# `description` TEXT NULL,

# `journey\_statuses\_id` INT NOT NULL,

# `contracts\_id` INT NOT NULL,

# `creator\_id` INT NOT NULL,

# `driver\_id` INT NOT NULL,

# `cars\_id` INT NOT NULL,

# PRIMARY KEY (`id`))

# ENGINE = MyISAM;

# SET SQL\_MODE=@OLD\_SQL\_MODE;

# SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

# SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

# ДОДАТОК Б

# КОД ГЕНЕРАЦІЇ ОПТИМІЗОВАНОЇ БД ТИПУ InnoDB

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='TRADITIONAL,ALLOW\_INVALID\_DATES';

-- -----------------------------------------------------

-- Schema truckind\_innodb\_denorm

-- -----------------------------------------------------

DROP SCHEMA IF EXISTS `truckind\_innodb\_denorm` ;

-- -----------------------------------------------------

-- Schema truckind\_innodb\_denorm

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `truckind\_innodb\_denorm` DEFAULT CHARACTER SET utf8 ;

USE `truckind\_innodb\_denorm` ;

-- -----------------------------------------------------

-- Table `truckind\_innodb\_denorm`.`employees`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `truckind\_innodb\_denorm`.`employees` ;

CREATE TABLE IF NOT EXISTS `truckind\_innodb\_denorm`.`employees` (

`id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) NOT NULL,

`surname` VARCHAR(45) NOT NULL,

`date\_registration` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`salary` FLOAT NOT NULL,

`is\_active` INT NOT NULL DEFAULT 1,

`password` VARCHAR(45) NOT NULL,

`email` TEXT(255) NOT NULL,

`role` ENUM('driver', 'manager', 'admin') NOT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB;

CREATE UNIQUE INDEX `email\_UNIQUE` ON `truckind\_innodb\_denorm`.`employees` (`email` ASC);

-- -----------------------------------------------------

-- Table `truckind\_innodb\_denorm`.`cars`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `truckind\_innodb\_denorm`.`cars` ;

CREATE TABLE IF NOT EXISTS `truckind\_innodb\_denorm`.`cars` (

`id` INT NOT NULL AUTO\_INCREMENT,

`number` VARCHAR(15) NOT NULL,

`carrying` FLOAT NOT NULL,

`date\_registration` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`status` ENUM('fixed', 'broken', 'non\_active') NOT NULL DEFAULT 'fixed',

`description` TEXT NULL,

`is\_fixed` INT NOT NULL DEFAULT 1,

`model` VARCHAR(45) NULL,

`current\_position` VARCHAR(30) NOT NULL DEFAULT '50.018765,36.2276296',

`mark` VARCHAR(45) NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB;

CREATE UNIQUE INDEX `number\_UNIQUE` ON `truckind\_innodb\_denorm`.`cars` (`number` ASC);

-- -----------------------------------------------------

-- Table `truckind\_innodb\_denorm`.`potential\_employees`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `truckind\_innodb\_denorm`.`potential\_employees` ;

CREATE TABLE IF NOT EXISTS `truckind\_innodb\_denorm`.`potential\_employees` (

`id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) NOT NULL,

`surname` VARCHAR(45) NOT NULL,

`email` VARCHAR(255) NOT NULL,

`date\_request` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`salary\_expected` FLOAT NOT NULL,

`resume` TEXT NULL,

`role` ENUM('driver', 'manager') NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB;

CREATE UNIQUE INDEX `email\_UNIQUE` ON `truckind\_innodb\_denorm`.`potential\_employees` (`email` ASC);

-- -----------------------------------------------------

-- Table `truckind\_innodb\_denorm`.`customers`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `truckind\_innodb\_denorm`.`customers` ;

CREATE TABLE IF NOT EXISTS `truckind\_innodb\_denorm`.`customers` (

`id` INT NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) NOT NULL,

`surname` VARCHAR(45) NOT NULL,

`date\_cooperation` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`company` VARCHAR(100) NULL,

`description` TEXT NULL,

`address` VARCHAR(250) NOT NULL,

`priority` FLOAT NOT NULL DEFAULT 1,

`discount` FLOAT NULL,

`email` VARCHAR(255) NOT NULL,

`phone` VARCHAR(13) NOT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB;

CREATE UNIQUE INDEX `email\_UNIQUE` ON `truckind\_innodb\_denorm`.`customers` (`email` ASC);

-- -----------------------------------------------------

-- Table `truckind\_innodb\_denorm`.`contracts`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `truckind\_innodb\_denorm`.`contracts` ;

CREATE TABLE IF NOT EXISTS `truckind\_innodb\_denorm`.`contracts` (

`id` INT NOT NULL AUTO\_INCREMENT,

`date` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`description` TEXT NULL,

`price` FLOAT NOT NULL,

`customers\_id` INT NOT NULL,

PRIMARY KEY (`id`),

CONSTRAINT `fk\_contracts\_customers1`

FOREIGN KEY (`customers\_id`)

REFERENCES `truckind\_innodb\_denorm`.`customers` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_contracts\_customers1\_idx` ON `truckind\_innodb\_denorm`.`contracts` (`customers\_id` ASC);

-- -----------------------------------------------------

-- Table `truckind\_innodb\_denorm`.`journeys`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `truckind\_innodb\_denorm`.`journeys` ;

CREATE TABLE IF NOT EXISTS `truckind\_innodb\_denorm`.`journeys` (

`id` INT NOT NULL AUTO\_INCREMENT,

`from` VARCHAR(30) NOT NULL,

`destination` VARCHAR(30) NOT NULL,

`date\_create` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`date\_start` DATETIME NULL,

`date\_finish` DATETIME NULL,

`progress` FLOAT NOT NULL DEFAULT 0.0,

`description` TEXT NULL,

`status` ENUM('new', 'accepted', 'in\_progress', 'canceled', 'finished') NOT NULL DEFAULT 'new',

`contracts\_id` INT NOT NULL,

`creator\_id` INT NOT NULL,

`driver\_id` INT NULL,

PRIMARY KEY (`id`),

CONSTRAINT `fk\_journeys\_contracts1`

FOREIGN KEY (`contracts\_id`)

REFERENCES `truckind\_innodb\_denorm`.`contracts` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_journeys\_employees1`

FOREIGN KEY (`creator\_id`)

REFERENCES `truckind\_innodb\_denorm`.`employees` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_journeys\_employees2`

FOREIGN KEY (`driver\_id`)

REFERENCES `truckind\_innodb\_denorm`.`employees` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_journeys\_contracts1\_idx` ON `truckind\_innodb\_denorm`.`journeys` (`contracts\_id` ASC);

CREATE INDEX `fk\_journeys\_employees1\_idx` ON `truckind\_innodb\_denorm`.`journeys` (`creator\_id` ASC);

CREATE INDEX `fk\_journeys\_employees2\_idx` ON `truckind\_innodb\_denorm`.`journeys` (`driver\_id` ASC);

-- -----------------------------------------------------

-- Table `truckind\_innodb\_denorm`.`cars\_has\_journeys`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `truckind\_innodb\_denorm`.`cars\_has\_journeys` ;

CREATE TABLE IF NOT EXISTS `truckind\_innodb\_denorm`.`cars\_has\_journeys` (

`cars\_id` INT NOT NULL,

`journeys\_id` INT NOT NULL,

`is\_accepted` INT NOT NULL DEFAULT 0,

PRIMARY KEY (`cars\_id`, `journeys\_id`),

CONSTRAINT `fk\_cars\_has\_journeys\_cars1`

FOREIGN KEY (`cars\_id`)

REFERENCES `truckind\_innodb\_denorm`.`cars` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_cars\_has\_journeys\_journeys1`

FOREIGN KEY (`journeys\_id`)

REFERENCES `truckind\_innodb\_denorm`.`journeys` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_cars\_has\_journeys\_journeys1\_idx` ON `truckind\_innodb\_denorm`.`cars\_has\_journeys` (`journeys\_id` ASC);

CREATE INDEX `fk\_cars\_has\_journeys\_cars1\_idx` ON `truckind\_innodb\_denorm`.`cars\_has\_journeys` (`cars\_id` ASC);

-- -----------------------------------------------------

-- Table `truckind\_innodb\_denorm`.`employees\_has\_cars`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `truckind\_innodb\_denorm`.`employees\_has\_cars` ;

CREATE TABLE IF NOT EXISTS `truckind\_innodb\_denorm`.`employees\_has\_cars` (

`employees\_id` INT NOT NULL,

`cars\_id` INT NOT NULL,

`date\_receive` DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

`date\_give\_back` DATETIME NULL,

PRIMARY KEY (`employees\_id`, `cars\_id`),

CONSTRAINT `fk\_employees\_has\_cars\_employees1`

FOREIGN KEY (`employees\_id`)

REFERENCES `truckind\_innodb\_denorm`.`employees` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_employees\_has\_cars\_cars1`

FOREIGN KEY (`cars\_id`)

REFERENCES `truckind\_innodb\_denorm`.`cars` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_employees\_has\_cars\_cars1\_idx` ON `truckind\_innodb\_denorm`.`employees\_has\_cars` (`cars\_id` ASC);

CREATE INDEX `fk\_employees\_has\_cars\_employees1\_idx` ON `truckind\_innodb\_denorm`.`employees\_has\_cars` (`employees\_id` ASC);

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;