Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп’ютерні науки

Кафедра Системотехніки

**МІЖДИСЦИПЛІНАРНИЙ КУРСОВИЙ ПРОЕКТ**

**ПОЯСНЮВАЛЬНА** **ЗАПИСКА**

тема: Інформаційна система «Вступна кампанія»

(тема роботи)

з дисципліни Проектування високонавантажених систем зберігання даних

(назва дисципліни)

Керівник 06.06.2019 Коваленко А.І. доцент кафедри системотехніки

(підпис, дата, посада, прізвище, ініціали)

Студент ІТКНу-17-1 06.06.2018 Гончаренко В.О.

(група, підпис, дата, прізвище, ініціали)

Робота захищена з оцінкою «\_\_\_\_\_\_\_\_\_»

«06» червня 2019 р.

Комісія:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис, посада, прізвище, ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис, посада, прізвище, ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис, посада, прізвище, ініціали)

Харків 20 19

Харківський національний університет радіоелектроніки

**Факультет** комп'ютерних наук

**Кафедра** системотехніки

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

**Курс** 2 **група** ІТКНу-17 -1 **семестр** 4

**ЗАВДАННЯ**

**на курсове проектування**

**студентові** Гончаренко Василю Олександровичу

(прізвище, ім'я, по батькові)

**1. Тема роботи:** Інформаційна система «Вступна кампанія»

**2. Строк здачі студентом закінченої роботи** 06.06.2019

**3. Вихідні дані до проекту:** Розробити серверну частину інформаційної системи «Вступна кампанія». Серверна частина повинна являти собою 2 варіанти реалізації бази даних, розробленої для платформи СУБД MySQL з використанням таблиць типу MyIsam і InnoDB. Бізнес-функції системи для не зареєстрованних користувачів: реєстрація на сайті; авторизація на сайті. Бізнес-функції системи для зареєстрованих користувачів(абітурієнтів): додавання оцінок; реєстрація на факультет для участі у вступній кампанії; зв’язок з адміністратором; вихід із власного кабінету. Бізнес-функції системи для адміністраторів: вхід у систему з статусом «admin»; блокування та розблокування абітурієнтів; видалення факультетів; додавання факультетів; формування відомості по факультету; вихід із кабінету. Операційна система – Windows ХР або вище, програмне забезпечення: утиліта командного рядка MySQL Command Line Client; програмний пакет Workbench; програмний web-засіб phpAdmin; програмний засіб Devart dbForge Studio for MySQL, CASE-засіб All Fusion Data Modeler (ErWin).

**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 «Вступна кампанія», 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 «Вступна кампанія» 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:** 18.03.2019

**Керівник роботи Коваленко Андрій Іванович**

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

**Студент Гончаренко Василь Олександрович**

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

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Назва етапів курсового проекту** | **Строк виконання** | **Примітка** |
|  | Аналіз предметної області | 19.03.19 – 30.03.19 |  |
|  | Визначення основних бізнес-функцій високонавантаженої інформаційної системи | 31.03.19 – 10.04.19  3 |  |
|  | Визначення функцій інтерфейсу клієнтської частини інформаційної системи | 11.04.19 – 13.04.19 |  |
|  | Розробка серверної частини інформаційної системи | 14.04.19 – 25.04.19 |  |
|  | Логічне й фізичне моделювання даних. | 26.04.19 – 05.05.19 |  |
|  | Створення і заповнення високонавантажених баз даних з таблицями типу MyIsam і InnoDB | 06.05.19 – 08.05.19 |  |
|  | Розробка підтримки цілісності даних | 09.05.19 – 15.05.19 |  |
|  | Реалізація бізнес-функцій інформаційної системи на стороні сервера MySQL (процедур, функцій, тригерів, представлень, транзакцій) | 16.05.19 – 20.05.19 |  |
|  | Оптимізація запитів до високонавантажених баз даних | 21.05.19 – 25.05.19 |  |
|  | Масштабування баз даних | 26.05.19 – 30.05.19 |  |
|  | Порівняльний аналіз двох варіантів реалізації бази даних (з використанням таблиць типу MyIsam і InnoDB) | 31.05.19 – 03.06.19 |  |
|  | Рекомендації з використання баз даних | 04.06.19 – 05.06.19 |  |

**Керівник роботи** Коваленко Андрій Іванович

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

**Студент** Гончаренко Василь Олександрович

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

**«18» березня 2019 г.**

РЕФЕРАТ

***Пояснювальна записка до курсового проекту: с., рис., додатків, джерел інформації.***

БАЗА ДАНИХ, СИСТЕМА УПРАВЛІННЯ БАЗАМИ ДАНИХ, MYSQL, МАСШТАБУВАННЯ, ШАРДИНГ, ОПТИМІЗАЦІЯ SQL ЗАПИТІВ, ІНДЕКСИ, ІНТЕРФЕЙС ДОСТУПУ, СИСТЕМА ОБЛІКУ.

Об’єктом дослідження курсового проекту є процес університету – проведення вступної кампанії, облік абітурієнтів які приймають участь у конкурсі, формування відомостей щодо факультетів після закінчення конкурсу.

Предметом дослідження курсового проекту є інформаційні технології та програмні методи створення серверної частини високонавантаженої інформаційної системи автоматизації обліку вступу абітурієнтів.

Ціль дослідження: розробка серверної частини високонавантаженої інформаційної системи «Вступна кампанія».

Методи дослідження – системний підхід, методи структурованого аналізу та моделювання реляційних баз даних, методи реляційної алгебри та реляційного підрахунку.

В роботі проведено проектування двох варіантів серверної частини високонавантаженої інформаційної системи «Вступна кампанія». Проведена оптимізація SQL запитів по критеріям мінімізації часу доступу до даних з урахуванням специфікації високонавантажених систем. Відповідно до проведеного аналізу було проведено обґрунтування масштабування структур даних, проведено порівнюючи аналіз і розробка рекомендації по використанню кожного із варіантів серверної частини високонавантаженої інформаційної системи.

Область використання – університети.

ВСТУП

Обчислювальна техніка в наш час розвивається стрімкими темпами. Комп’ютери використовуються в усіх сферах життя суспільства: спілкування, пошук інформації в мережі Інтернет, придбання товарів та послуг, Інтернет-банкінг та інше. Сучасна людина не уявляє свого життя без електронних гаджетів.

В той же час вимоги до апаратного та програмного забезпечення невпинно розширюються. Якщо, наприклад, статичні веб-сторінки ще кілька років тому були досить поширеними, то зараз важко знайти сайт, де б не було інтерактивної складової.

Саме розвиток обчислювальної техніки, а особливо веб-технологій, дозволив перенести в Інтернет навчальний процес: в наш час майже кожна університет або школа має свій сайт. Це дозволяє економити кошти та час, залучати учасників навчального процесу до більш детального ознайомлення із навчальним закладом.

З поширенням мережі Інтернет виникли електронні сервіси, що надають різні можливості по інтерактивній взаємодії з навчальними закладами. У порівнянні із закладами які не мають власних сервісів вони мають безліч переваг, які сприяють зростанню популярності та доходів.

Метою даного курсового проекту є реалізація бази даних інформаційної системи «Вступна кампанія». Розроблена база даних має всі необхідні компоненти для виконання будь-яких завдань пов’язаних з пошуком і обробкою інформації.

Використання інтернету в навчальній сфері є досить перспективним напрямом, так як дозволяє за мінімальні затрати істотно підвищити рівень комунікація адміністрації навчального закладу та обліки, які використовуються в цьому напрямі.

Все вищезазначене підтверджує актуальність обраної теми - «Вступна кампанія».

# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

* 1. Аналіз предметної області і коротка постановка задачі

В якості підприємства було обрано Університет, головними функціями якого представленні :

* здійсненні освітньої;
* наукової діяльності.

Після аналізу підприємства було виділено наступні бізнес процеси, які можна автоматизувати з використанням програмних засобів ІС:

* управління розкладом;
* електронний журнал відвідування;
* онлайн портал для керування навчальним матеріалом;
* ведення обліку в бібліотеці по читачам\книгам;
* вступна кампанія яку проводить університет для вступу абітурієнтів до ВНЗ.

Серед вищезгаданих бізнес процесів було обрано саме бізнес процес проведення вступної кампанія. На сьогоднішній день вже існує аналог - «Інформаційна система Конкурс «Вступна кампанія»», але під час проведення цього конкурсу виникають проблеми в роботі, оскільки занадто велика кількість людей намагаються отримати доступ до порталу. Але якщо у кожного університету буде своя вступна кампанія це зменшить навантаження на кожний із серверів що в результаті приведе до стійкої роботи ІС «Вступна кампанія». В майбутньому буде розглядатися можливість синхронізації даних в спільній ноді(спільний сервер для всіх університетів, в якому будуть зберігатися вже оброблені дані), а сервер кожного університету буде виступати в якості під сервера.

Об'єктом дослідження є розробка серверної частина інформаційної системи «Вступна кампанія».

Користувачі даної системи зможуть самостійно у будь-який час переглянути актуальну інформацію по стану проведення конкурсу на різні факультети, самостійно подати заявку для участі у конкурсі після реєстрації та авторизації та інше. Можливість приймати участь у «Вступній кампанія» не виходячи з дому економить час користувачів, оскільки для використання системи непотрібно навіть виходити з дому. Системою можуть користуватися як представники вузу (педагогічний колектив, деканат) так і інші особи (абітурієнти; викладачі різних шкіл; різні організації, які займаються збором статистики). Також, система буде корисна абітурієнтам, які живуть в смт або селах.

Також використання даної системи вигідне ВНЗ – відсутність необхідних витрат на персонал, який буде проводити реєстрацію вручну в університеті.

В системі має бути можливість реєстрації. Зареєстровані користувачі матимуть свої переваги.

1.2 Визначення основних бізнес-функцій високонавантаженої інформаційної системи

У системі існує 3 ролі користувачів:

* незареєстрований користувач(guest);
* зареєстрований користувач(enrollee);
* головний адміністратор(admin);
* адміністратор факультету (facultyAdmin);

Головний адміністратор має можливість додавати\редагувати\видаляти факультет\користувачів, але головною бізнес функцією є формування відомості по обраному факультету. Також він має можливість керувати адміністраторами факультетів (додавати, редагувати, видаляти).

Адміністратор факультету має ті ж самі можливості що і головний адміністратор, окрім можливості керувати адміністраторами факультетів. Адміністратор факультетів має можливість редагувати інформацію про факультет, а також видаляти абітурієнтів з конкурсу на факультет, за яким він закріплений.

Незареєстровані користувачі можуть переглядати інформацію про факультети, інформацію про конкурс.

Зареєстровані користувачі можуть виконувати всі ті самі дії, що і незареєстровані плюс мають наступні переваги після виконання авторизації:

* створити заявку для участі у конкурсі;
* додати власну інформацію;
* додати оцінки.

1.3 Визначення функцій інтерфейсу клієнтської частини інформаційної системи

Під час заходу на сайт буде виводитись вікно реєстрації\авторизації в якому необхідно ввести дані для входу у власний кабінет або пройти процес реєстрації який потребує підтвердження емейлу.

Після переходу користувача на сайт (або після логуваня, реєстрації) в сесії браузера для кожного користувача зберігається його роль.

Далі описані елементи юзер-інтерфейсу для зареєстрованого та залогованого користувача.

Панель навігації містить наступні елементи:

* форма пошуку – текстове поле, в яке користувач вводить повну або часткову назву для факультету та кнопка «Пошук».
* посилання на каталог факультетів;
* кнопки : вхід, реєстрація.

Каталог факультетів містить список факультетів та панель фільтрів (за кількістю бюджетних місць, за загальною кількістю місць).

Сторінка факультету містить інформацію про факультет та кнопку «Подати заявку».

Форма додавання власних даних містить поля з даними користувача та кнопка «Зберегти».

Форма реєстрації містить текстові поля для введення імені, логіна, електронної пошти, пароля і кнопку «Реєстрація».

Клієнтська частина ІС «Вступна кампанія», яка базується на бізнес логікі відповідно до серверної частини повинна виконувати функції, які будуть надавати користувачеві необхідну інформацію і давати можливість взаємодіяти з системою:

* функція виведення результатів пошуку;
* функція виведення списку факультетів за фільтрами;
* функція виведення детальної інформації про факультет;
* функція оформлення заявки для участі у конкурсі;
* функція виведення списку всіх заявок;
* функція перегляду інформації про користувача.

У головного адміністратора та адміністраторів факультетів на сторінках детального перегляду інформації по факультету присутня кнопка «Сформувати відомість» – яка відповідає за початок процесу створення відомості для обраного факультету.

У незареєстрованого користувача присутня можливість переглядати лише сторінку реєстрації \ логування.

Повнотекстовий пошук на стороні юзер інтфрейсу буде використовуватися під час пошуку по полю «додаткова інформація» по абітурієнта та «додаткова інформація» по факультету.

# РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Фізичне моделювання даних

Основою для інформаційної системи є база даних. Саме БД дозволяє експлуатувати ІС, виконувати її поточне обслуговування, модифікувати і обробляти інформацію про роботу підприємства, приймати інформаційні потоки.

Для предметної області «Вступна кампанія» була розроблена модель бази даних у вигляді ER-діаграми (Рисунок 2.1).

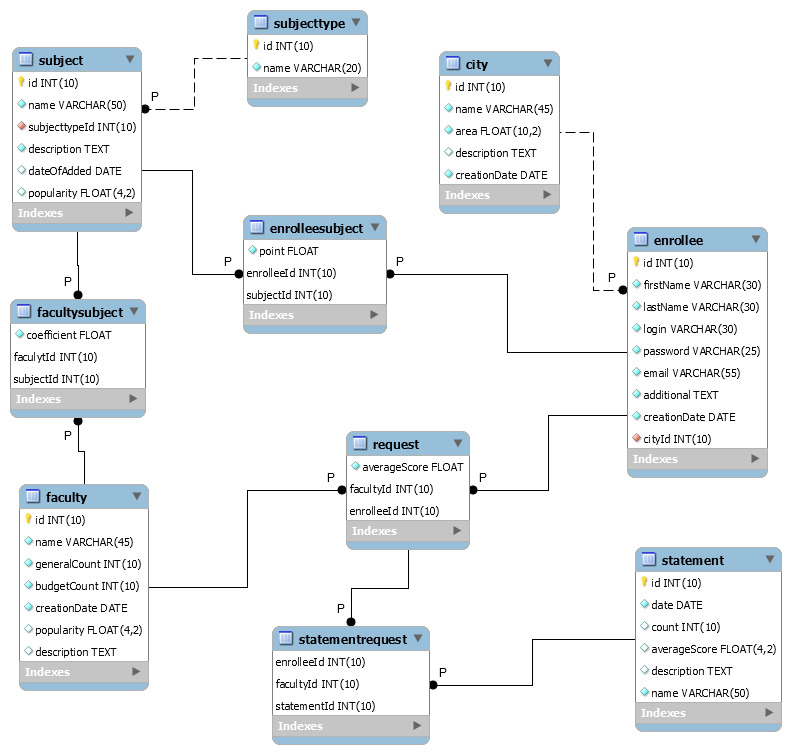


Рисунок 2.1 – EER діаграма бази даних з типом таблиць InnoDB

На EER діаграмі присутні наступні сутності:

* enrollee – містить наступну інформацію про абітурієнта: ім’я, прізвище, логін, пароль, емейл, додаткова інформація, id міста, дата реєстрації;
* city – містить інформацію про міста: назва міста, площа, опис, дата створення;
* subject – містить інформацію про предмети: назва предмета, id типу предмета, додаткова інформація про предмет, популярність предмету;
* subjectType – містить інформацію про тип предметів: назва (пр. екзамен, оцінка з атестату);
* faculty – містить інформацію про факультети: назва факультету, загальна кількість місць, бюджетна кількість місць, дата створення факультету, популярність факультету, опис факультету;
* request – містить інформацію про заявки на участь у конкурсі: середній бал, id факультету, id абітурієнт, дата створення;
* statement – містить інформацію про сформовані відомості: дата формування, кількість абітурієнтів, назва відомості (пр. Відомість для факультету …), середній бал.

2.2 Розробка підтримки цілісності даних

Логічна модель даних була перетворена в фізичну шляхом[1] вибору системи управління базами даних і заданням правил підтримки цілісності (Додаток А). Вони необхідні для опису змін відбуваються в сутності, якщо пов'язана з нею інша сутність була змінена або видалена.

Після побудови схеми фізичної моделі необхідно задати правила цілісності даних, тобто характеристики для кожного атрибута кожної сутності, щоб забезпечити відповідність інформації її внутрішній логікі, структурі і всім заданим правилам. Цілісність БД не гарантує достовірності інформації, що міститься в ній, але забезпечує, принаймні, узгодженість даних, відкидаючи свідомо неймовірні, неможливі значення.

Також було розроблено БД з типом таблиць MyISAM[2]. Схема БД наведена на рисунку 2.2.

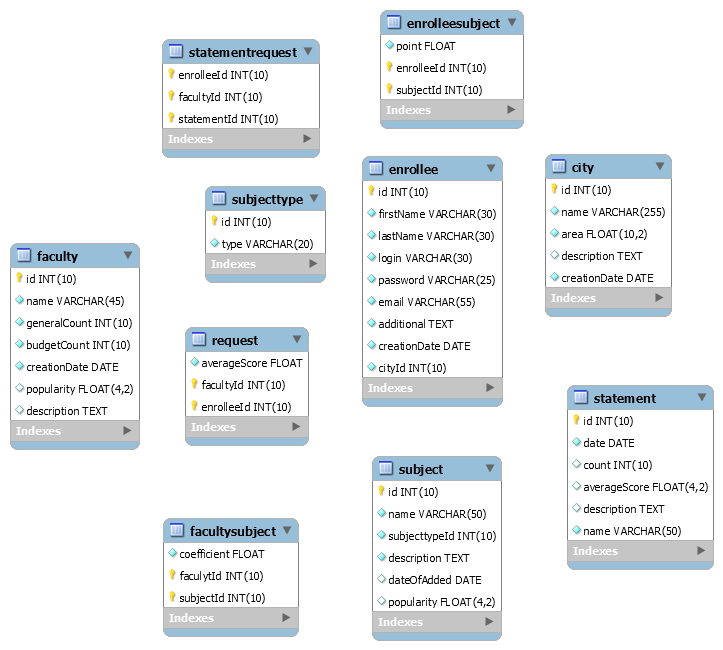


Рисунок 2.2 – EER діаграма бази даних з типом таблиць MyISAM

Оскільки таблиці типу MyISAM не підтримують зовнішні ключі, цілісність даних було реалізовано за допомогою тригерів (Додаток Б).

2.3 Розробка запитів які враховують специфіку високонавантаженої системи

Для того щоб в подальшому спроектувати необхідні індекси потрібно спочатку розробити необхідні високонавантажені запити які будуть частіше всього використовуватися. Високонавантажені запити це запити на вибірку із заданими умовами. Наприклад запит на вибірку із заданими умовами до зв’язних таблиць (за допомогою WHERE і JOIN). Запити із використанням GROUP BY а також об’єднуючі запити із використанням UNION та корельовані запити (запити із під запитами).

Реалізовані запити представлені в Додатку В.

2.4 Реалізація бізнес-функцій інформаційної системи на стороні сервера MYSQL

Для реалізації основних бізнес-функцій інформаційної системи, які були описані в п. 1.3 було використано процедури, функції та збережені процедури[3], які неведені в Додатку Г.

# Оптимізація високонавантажених систем

Для економії ресурсів та зменшення часу виконання запитів використовують різні способи по оптимізації бази даних. Найбільш популярні способи – зменшення кількості підстановлювальних таблиць, що зменшить алгоритми запитів та процедур збільшуючи швидкість їх виконання. Денормалізований вигляд таблиць на ядрі InnoD.

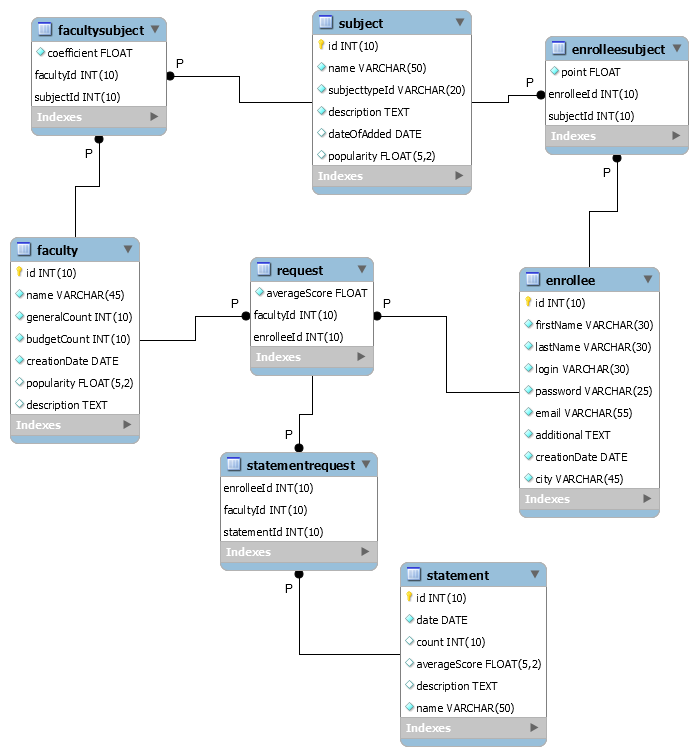


Рисунок 3.1 – Оптимізована ER діаграма для таблиць типу InnoDB

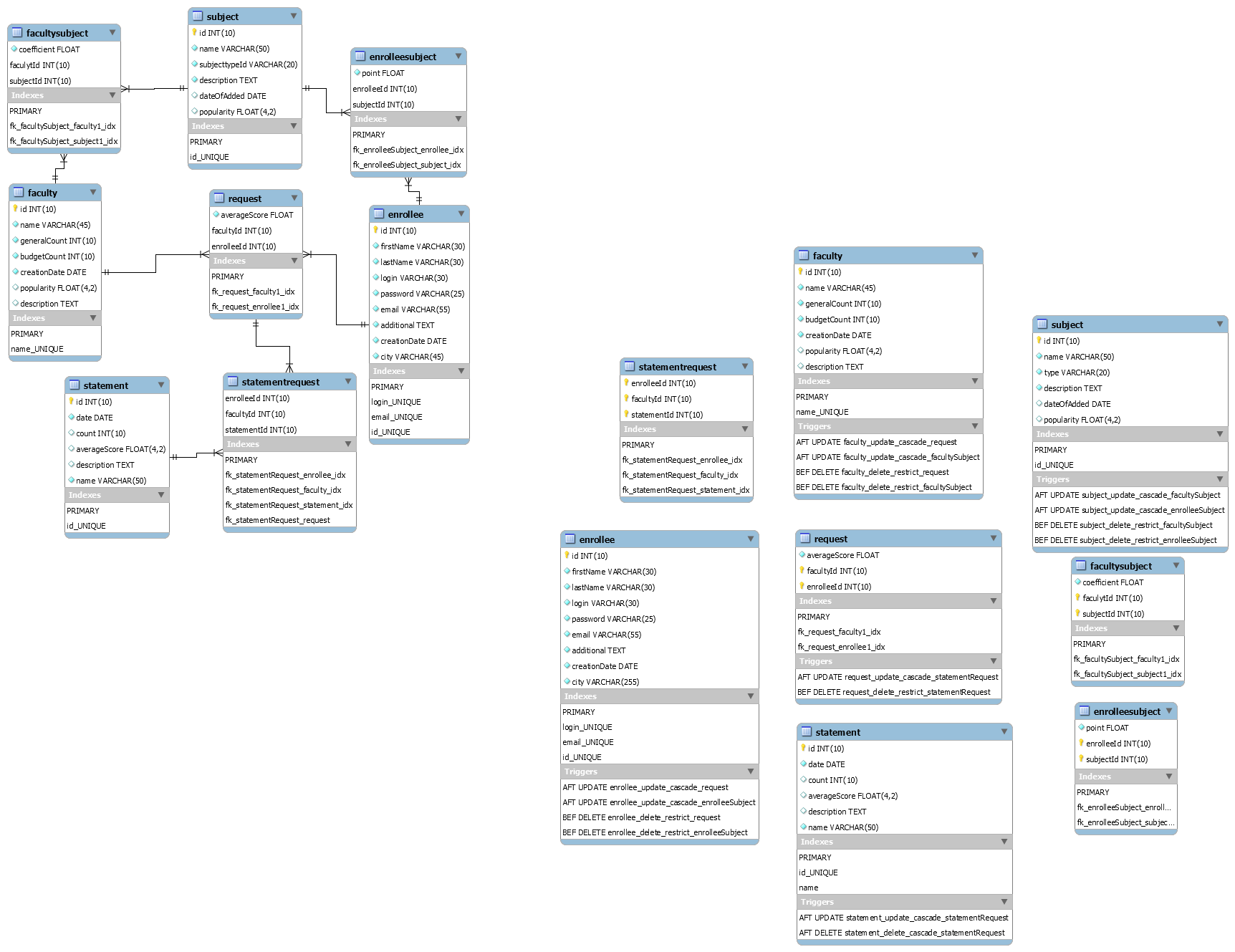


Рисунок 3.2 – Оптимізована ER діаграма для таблиць типу MyISAM

3.1 Після проведення детального аналізу даних кожного атрибуту вирішено що неможливо модифікувати(розширити або звузити) або зовсім змінити тип даних для атрибутів. Можливим рішенням щодо денормалізації стало видалення таблиці subjectType та таблиці city.

3.2 Так як більша частина індексів (unique and primary) була реалізована під час створення таблиці (окрім FullText) в Додатку Д буде наведено саме додавання індексів для повнотекстового пошуку.

Для аналізу було обрані наступні функції ІС (запити) для яких потрібно визначити індекси:

* запит для отримання всіх факультетів, на які було подано заявки абітурієнтом з прізвищем HOLBROOK;
* запит для отримання інформації про предмети, які додали абітурієнти з міста Princehaven;
* запит для отримання всіх абітурієнтів які подали заявку на факультет Education0;
* запит для перегляду абітурієнтів у яких є заявка з середнім балом вище 180;
* запит для отримання абітурієнтів, які згадали в додатковій інформації про себе слово «piano»;
* запит для отримання інформації про всіх абітурієнтів із міста Olafberg у яких є заявка із середнім балом вище 180;
* запит для отримання інформації абітурієнта за прізвищем із міста Olafberg.

Запит для отримання всіх факультетів, на які було подано заявки абітурієнтом з прізвищем HOLBROOK розроблено наступний індекс:

CREATE INDEX enrolleeLastNameIndex ON enrollee(lastName);

Запит для отримання інформації про предмети, які додали абітурієнти з міста Princehaven:

CREATE INDEX enrolleeCityIdIndex ON enrollee(cityId);

Для таблиці city індекс уже створено (UNIQUE INDEX `name\_UNIQUE` (`name` ASC)).

Запит для отримання всіх абітурієнтів які подали заявку на факультет Education0: UNIQUE INDEX `name\_UNIQUE` (`name` ASC) присутній в CREATE.

Запит для перегляду абітурієнтів у яких є заявка з середнім балом вище 180:

CREATE INDEX requestEnrolleIdAverageScoreIndex

ON request(enrolleeId, averageScore)

Індекс (averageScore, enrolleeId) буде не таким ефективним оскільки нас цікавлять саме абітурієнти а потім вже оцінка. Тобто по індексу який було розроблено спочатку буде обрано всі 12 заявок абітурієнта а потом з 12-ти буде обрано саме ті, в яких оцінка більше 180. Якщо ж розглядати іншу варіацію цього індекса то спочатку буде обрано приблизно 200 заявок які більше 180 а потім із цих 200 відберуться за абітурієнтом id. Як результат, було обрано саме перший індекс оскільки у нього краща селективність.

Запит для отримання інформації абітурієнта за прізвищем із міста Olafberg:

CREATE INDEX enrolleeLastNameCityId

ON enrollee(lastName, cityId);

Розглядаючи предметну область було прийнято рішення використання саме такого індексу оскільки у нього селективність краще. Тобто якщо розглядати як приклад Україну. В 2018 році на ЗНО було зареєстровано 335 тис. абітурієнтів. Можна вважати що з кожного міста було зареєстровано приблизно 335 / 45 (міст з чисельністю населення більше 100 тис.) = 7.5 тис абітурієнтів на кожне місто. Оскільки варіативність прізвищ вища ніж назв міст (45) то можна сказати що селективність індексу lastName, cityId буде краща.

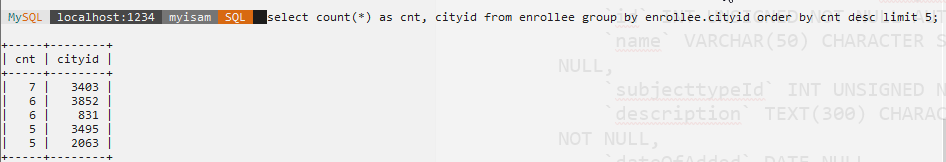


Рисунок 3.3 – group by по cityId.

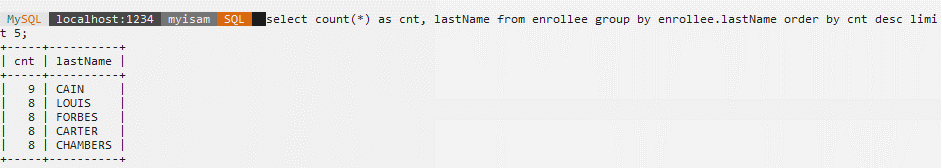


Рисунок 3.4 – group by по lastName.

Виходячи з результаті вище потрібно було обрати інший порядок для комплексного індексу але говорячи про реальні умови порядок повинен бути саме таким.

Запит для отримання абітурієнтів, які згадали в додатковій інформації про себе слово «piano»:

CREATE FULLTEXT INDEX enrolleAdditionalInformationFullTextIndex ON enrollee(additional);

Пошук потрібно буде виконувати у BOOLEAN MODE. Якщо використовувати стандартний режим то можемо не отримати ніяких даних оскільки в IN NATURAL MODE існує так званий трешхолд в 50%, і якщо кількість записів буде перевищувати цей трешхолд то ми не отримаємо жодного запису.

По способу зберігання індекси розподіляють на кластерні та звичайні.

Звичайні індекси – це відсортований набір даних колонки, по якій він будується. Індекс буде зберігати посилання на дані кожного запису для кожного значення id.

Кластерний індекс (кластерний ключ) – зберігає не тільки значення колонки в відсортованому вигляді але і дані усього рядка. Це дозволяє мінімізувати кілкість операція читання із диска при роботі з таким індексом. В таблиці може бути тільки один кластерний індекс. Первинний ключ таблиці має бути кластерним. Якщо в таблиці відсутній первинний ключ то в якості кластерного індекса MySQL обере перший стовпець який буде мати UNIQUEU INDEX.

3.3 Вертикальний шардінг. Вертикальний шардінг – одна із технік масштабування даних, яка полягає в розподілу таблиць по різним серверам. У цьому випадку декілька таблиць будуть знаходитись на першому сервері – а інші на другому. Сервера з різними таблицями називають шардами. Вертикальний шардінг для розроблюваної бази даних зображено на наступних рисунках.

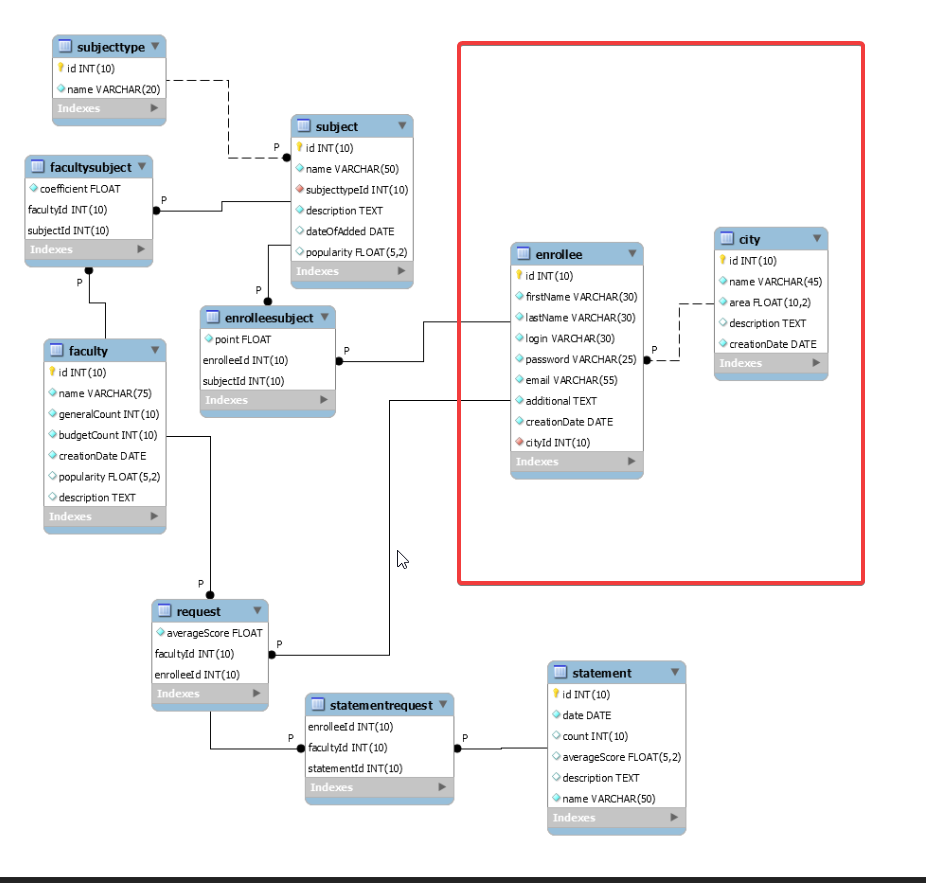


Рисунок 3.5 – виділення слейву для вертикального шардінгу

Горизонтальний шардінг – одна із технік оптимізації баз даних, яка базується на розподіленні даних однієї таблиці на декілька слейвів. Тобто дані для однієї таблиці зберігаються на різних серверах. Горизонтальний шардінг можливий в двох реалізаціях. Перша реалізація полягає в виборі атрибуту або атрибутів, які будуть виступати вхідними даними для обробки деякими математичними операціями і цей результат буде номером сервера, на який потрібно записати запис. Наприклад: можна обрати таблицю request для горизонтального масштабування та розраховувати номер сервера для збереження за формулою (facultyId + enrolleeId) % 2. Якщо ми маємо більше ніж два слейви то замість «2» потрібно використати їх кількість.

Другий спосіб реалізації горизонтального масштабування полягає в створенні словника – іще однієї таблиці в якій буде вказаний зв’язок між ID enrollee та номером шарда, в якому він зберігається.

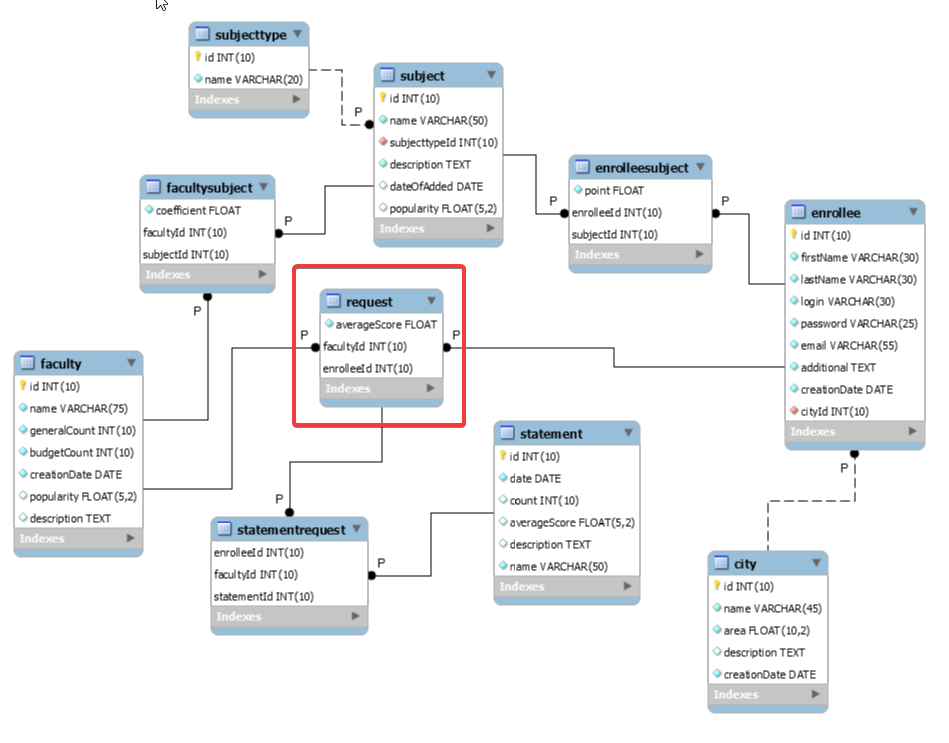


Рисунок 3.6 – горизонтального масштабування для таблиці request використовуючи математичну послідовність над facultyId та enrolleeId

3.4 Порівняльний аналіз MyiSAM & InnoDB.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Опис | MyISAM | InnoDB |
| 1 | Підтримка транзакій | – | + |
| 2 | Підтримка зв'язків по звонішнім ключам | – | + |
| 3 | Підтримка цілістності зв'язків для інструкцій update, delete, insert | – | + |
| 4 | Блокування на рівні таблиці\запису\відсутня | Таблиця | Запис |
| 5 | Одночасний запит до різних записів однієї таблиці | Повільніше | Скоріше |
| 6 | При змішаному навантаженні на запити  SELECT, UPDATE, DELETE, INSERT | Повільніше | Скоріше |
| 7 | Однотипні операції Insert | Скоріше | Повільніше |
| 8 | Однотипні операції SELECT | Скоріше | Повільніше |
| 9 | Запит COUNT(\*) | Скоріше | Повільніше |
| 10 | Взаємне блокування (DeadLock) | Неможливі | Можливі |
| 11 | Підтримка повнотекстового пошуку | + | + |
| 12 | Підтримка індексування полів в запитах | + | + |
| 13 | Можливі типи індексів | PK+, FK–, UNIQUE+, INDEX+, FULLTEXT+ | PK+,FK+,UNIQUE+ INEX+,FULLTEXT+ |
| 14 | Можливість бінарного копіювання таблиць | + | – |
| 15 | Розмір таблиці БД | Менше | Більше |
| 16 | Можливість відновлення у випадку збою | – | + |
| 17 | Фалйи за замовчуванням, що створе СУБД MySQL | .MYD(MYData), .MYI(MyINdex) |  |

ВИСНОВКИ

В ході виконання роботи було виконано проектування серверної частини високонавантаженої бази даних «Вступна кампанія» відповідно до методичних вказівок[4]. Для створення БД було використане середовище візуального проектування баз даних «MySQL Workbench» - інструмент, який дозволяє наочно уявити модель бази даних в графічному вигляді, а також є зручним редактором SQL запитів, що дозволяє відразу ж відправляти їх на сервер і отримати відповідь у вигляді таблиці.

Була розроблена база даних для двох типів таблиць «MyISAM» і «InnoDB». Для здійснення цілісності БД в «MyISAM» були реалізовані тригери, які забезпечать узгодженість даних, в «InnoDB» були описані обмеження у вигляді зв’язків (зовнішніх ключів). Для всіх типів таблиць були використані обмеження NOT NULL (виходячи з вимог системи).В ході виконання курсового проекту були поглиблені знання і закріплені практичні навички роботи з програмою Workbench, а також знайдені практичні навички роботи з високонавантажених базами даних.

Для усунення із бази даних надлишкових функціональних залежностей між атрибутами (полями таблиці) було проведено оптимізації шляхом скорочення кількості таблиць і індексування деяких атрибутів.

В ході виконання курсового проекту були поглиблено та закріплено навички роботи з програмою Workbench, а також набуто практичні навики роботи з високонавантаженими базами даних.

Отримані знання та навички будуть використані при виконанні подальших робіт в програмі університету, а також у подальшій професійній діяльності.

ПЕРЕЛІК ПОСИЛАНЬ

1. Довідник по MySQL [Електронний ресурс] - Режим доступу URL: <https://dev.mysql.com>
2. Довідник по Workbench [Електронний ресурс] - Режим доступу URL:

https:// dev.mysql.com/doc/workbench/en/

1. Работа с базами данных: учебное пособие / уклад. О. Н. Євсєєва, О. Б. Шамшев. - Ульяновськ: УлГТУ, 2009. - 170 с.
2. Методичні вказівки до міждисциплінарного курсового проекту по для студентів напряму підготовки 6.050101 - Комп'ютерні науки [Електронне видання] / Упоряд.: Ю.В. Міщеряков, З.А. Імангулова., Л.В. Колесник - Харків: ХНУРЕ, 2016. - 30 с.

ДОДАТОК А

Правила цілісності даних для таблиць з типом InnoDB

CREATE TABLE IF NOT EXISTS `City` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

`area` FLOAT(10, 2) UNSIGNED NOT NULL,

`description` TEXT(500) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NULL,

`creationDate` DATE NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `id\_UNIQUE` (`id` ASC),

UNIQUE INDEX `name\_UNIQUE` (`name` ASC))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

CREATE TABLE IF NOT EXISTS `enrollee` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`firstName` VARCHAR(30) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

`lastName` VARCHAR(30) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

`login` VARCHAR(30) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

`password` VARCHAR(25) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

`email` VARCHAR(55) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

`additional` TEXT(1000) NOT NULL,

`creationDate` DATE NOT NULL,

`cityId` INT UNSIGNED NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `login\_UNIQUE` (`login` ASC),

UNIQUE INDEX `email\_UNIQUE` (`email` ASC),

UNIQUE INDEX `id\_UNIQUE` (`id` ASC),

INDEX `fk\_enrollee\_city\_index` (`cityId` ASC),

CONSTRAINT `fk\_enrollee\_city\_index`

FOREIGN KEY (`cityId`)

REFERENCES `City` (`id`)

ON DELETE RESTRICT

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

CREATE TABLE IF NOT EXISTS `subjectType` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(20) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `id\_UNIQUE` (`id` ASC))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

CREATE TABLE IF NOT EXISTS `subject` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(50) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

`subjecttypeId` INT UNSIGNED NOT NULL,

`description` TEXT(300) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

`dateOfAdded` DATE NULL,

`popularity` FLOAT(4,2) NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `id\_UNIQUE` (`id` ASC),

UNIQUE INDEX `name\_UNIQUE` (`name` ASC),

INDEX `fk\_subject\_subjectType1\_idx` (`subjecttypeId` ASC),

CONSTRAINT `fk\_subject\_subjectType1`

FOREIGN KEY (`subjecttypeId`)

REFERENCES `subjectType` (`id`)

ON DELETE RESTRICT

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

CREATE TABLE IF NOT EXISTS `faculty` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(45) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NOT NULL,

`generalCount` INT UNSIGNED NOT NULL,

`budgetCount` INT UNSIGNED NOT NULL,

`creationDate` DATE NOT NULL,

`popularity` FLOAT(4,2) NULL,

`description` TEXT(500) CHARACTER SET 'utf8' COLLATE 'utf8\_unicode\_ci' NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `name\_UNIQUE` (`name` ASC))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

CREATE TABLE IF NOT EXISTS `statement` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`date` DATE NOT NULL,

`count` INT UNSIGNED NULL,

`averageScore` FLOAT(4,2) NULL,

`description` TEXT(500) NULL,

`name` VARCHAR(50) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `id\_UNIQUE` (`id` ASC))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

CREATE TABLE IF NOT EXISTS `enrolleeSubject` (

`point` FLOAT NOT NULL,

`enrolleeId` INT UNSIGNED NOT NULL,

`subjectId` INT UNSIGNED NOT NULL,

PRIMARY KEY (`enrolleeId`,`subjectId`),

INDEX `fk\_enrolleeSubject\_enrollee\_idx` (`enrolleeId` ASC),

INDEX `fk\_enrolleeSubject\_subject\_idx` (`subjectId` ASC),

CONSTRAINT `fk\_enrolleeSubject\_enrollee1`

FOREIGN KEY (`enrolleeId`)

REFERENCES `enrollee` (`id`)

ON DELETE RESTRICT

ON UPDATE CASCADE,

CONSTRAINT `fk\_enrolleeSubject\_subject1`

FOREIGN KEY (`subjectId`)

REFERENCES `subject` (`id`)

ON DELETE RESTRICT

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

CREATE TABLE IF NOT EXISTS `facultySubject` (

`coefficient` FLOAT NOT NULL,

`faculytId` INT UNSIGNED NOT NULL,

`subjectId` INT UNSIGNED NOT NULL,

PRIMARY KEY (`faculytId`, `subjectId`),

INDEX `fk\_facultySubject\_faculty1\_idx` (`faculytId` ASC),

INDEX `fk\_facultySubject\_subject1\_idx` (`subjectId` ASC),

CONSTRAINT `fk\_facultySubject\_faculty1`

FOREIGN KEY (`faculytId`)

REFERENCES `faculty` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk\_facultySubject\_subject1`

FOREIGN KEY (`subjectId`)

REFERENCES `subject` (`id`)

ON DELETE NO ACTION

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

CREATE TABLE IF NOT EXISTS `request` (

`averageScore` FLOAT NOT NULL,

`facultyId` INT UNSIGNED NOT NULL,

`enrolleeId` INT UNSIGNED NOT NULL,

PRIMARY KEY (`facultyId`, `enrolleeId`),

INDEX `fk\_request\_faculty1\_idx` (`facultyId` ASC),

INDEX `fk\_request\_enrollee1\_idx` (`enrolleeId` ASC),

CONSTRAINT `fk\_request\_faculty1`

FOREIGN KEY (`facultyId`)

REFERENCES `faculty` (`id`)

ON DELETE NO ACTION

ON UPDATE CASCADE,

CONSTRAINT `fk\_request\_enrollee1`

FOREIGN KEY (`enrolleeId`)

REFERENCES `enrollee` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

CREATE TABLE IF NOT EXISTS `statementRequest` (

`enrolleeId` INT UNSIGNED NOT NULL,

`facultyId` INT UNSIGNED NOT NULL,

`statementId` INT UNSIGNED NOT NULL,

PRIMARY KEY (`statementId`, `enrolleeId`,`facultyId`),

INDEX `fk\_statementRequest\_enrollee\_idx` (`enrolleeId` ASC),

INDEX `fk\_statementRequest\_faculty\_idx` (`facultyId` ASC),

INDEX `fk\_statementRequest\_statement\_idx` (`statementId` ASC),

CONSTRAINT `fk\_statementRequest\_request`

FOREIGN KEY (`enrolleeId`,`facultyId`)

REFERENCES `request` (`enrolleeId`,`facultyId`)

ON DELETE NO ACTION

ON UPDATE CASCADE,

CONSTRAINT `fk\_statementRequest\_statement`

FOREIGN KEY (`statementId`)

REFERENCES `statement` (`id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

COLLATE = utf8\_unicode\_ci;

Додаток Б

Тригери для таблиць типу MyISAM

Для забезпечення цілісності даних з таблицею enrollee при видаленні запису:

CREATE TRIGGER city\_delete\_restrict BEFORE DELETE ON city

FOR EACH ROW

BEGIN

IF OLD.id IN (SELECT cityId FROM enrollee)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can`t delete city, which used in enrollee table.';

END IF;

END;//

Для забезпечення цілісності даних з таблицею enrollee при оновленні запису:

CREATE TRIGGER city\_update\_after AFTER UPDATE ON city

FOR EACH ROW

BEGIN

UPDATE enrolee SET cityid = NEW.id WHERE cityid = OLD.id;

END;//

Для забезпечення цілісності даних з таблицею subject при видаленні запису:

CREATE TRIGGER subject\_type\_delete\_restrict\_subject BEFORE DELETE ON subjectType

FOR EACH ROW

BEGIN

IF OLD.id IN (SELECT subjecttypeId FROM subject)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can`t delete subject type, which used in subject table.';

END IF;

END;//

Для забезпечення цілісності даних з таблицею subject при оновленні запису:

CREATE TRIGGER subject\_type\_update\_cascade\_subject AFTER UPDATE ON subjectType

FOR EACH ROW

BEGIN

UPDATE subject SET subjecttypeId = NEW.id WHERE subjecttypeId = OLD.id;

END;//

Для забезпечення цілісності даних з таблицею enrolleeSubject при видаленні запису:

CREATE TRIGGER enrollee\_delete\_restrict\_enrolleeSubject BEFORE DELETE ON enrollee

FOR EACH ROW

BEGIN

IF OLD.id IN (SELECT enrolleeId FROM enrolleeSubject)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can`t delete enrollee, who used in subject table.';

END IF;

END;//

Для забезпечення цілісності даних з таблицею enrolleeSubject при оновленні запису:

CREATE TRIGGER enrollee\_update\_cascade\_enrolleeSubject AFTER UPDATE ON enrollee

FOR EACH ROW

BEGIN

UPDATE enrolleeSubject SET enrolleeId = NEW.id WHERE enrolleeId = OLD.id;

END;//

Для забезпечення цілісності даних з таблицею request при видаленні запису:

CREATE TRIGGER enrollee\_delete\_cascade\_request BEFORE DELETE ON enrollee

FOR EACH ROW

BEGIN

IF OLD.id IN (SELECT enrolleeId FROM request)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can`t delete enrollee, who has at least one request.';

END IF;

END;//

Для забезпечення цілісності даних з таблицею request при оновленні запису:

CREATE TRIGGER enrollee\_update\_cascade\_request AFTER UPDATE ON enrollee

FOR EACH ROW

BEGIN

UPDATE request SET enrolleeId = NEW.id WHERE enrolleeId = OLD.id;

END;//

Для забезпечення цілісності даних з таблицею enrolleeSubject при видаленні запису:

CREATE TRIGGER subject\_delete\_restrict\_enrolleeSubject BEFORE DELETE ON subject

FOR EACH ROW

BEGIN

IF OLD.id IN (SELECT subjectId FROM enrolleeSubject)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can`t delete enrollee, who used in subject table.';

END IF;

END;//

Для забезпечення цілісності даних з таблицею enrolleeSubject при оновленні запису:

CREATE TRIGGER subject\_update\_cascade\_enrolleeSubject AFTER UPDATE ON subject

FOR EACH ROW

BEGIN

UPDATE enrolleeSubject SET subjectId = NEW.id WHERE subjectId = OLD.id;

END;//

Для забезпечення цілісності даних з таблицею facultySubecjt при видаленні запису:

CREATE TRIGGER subject\_delete\_restrict\_facultySubject BEFORE DELETE ON subject

FOR EACH ROW

BEGIN

IF OLD.id IN (SELECT subjectId FROM facultySubject)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can`t delete enrollee, who used in subject table.';

END IF;

END;//

Для забезпечення цілісності даних з таблицею enrolleeSubject при оновленні запису:

CREATE TRIGGER subject\_update\_cascade\_facultySubject AFTER UPDATE ON subject

FOR EACH ROW

BEGIN

UPDATE facultySubject SET subjectId = NEW.id WHERE subjectId = OLD.id;

END;//

Для забезпечення цілісності даних з таблицею facultySubject при видаленні запису:

CREATE TRIGGER faculty\_delete\_restrict\_facultySubject BEFORE DELETE ON faculty

FOR EACH ROW

BEGIN

IF OLD.id IN (SELECT subjectId FROM facultySubject)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can`t delete enrollee, who used in subject table.';

END IF;

END;//

Для забезпечення цілісності даних з таблицею facultySubject при оновленні запису:

CREATE TRIGGER faculty\_update\_cascade\_facultySubject AFTER UPDATE ON faculty

FOR EACH ROW

BEGIN

UPDATE facultySubject SET facultyId = NEW.id WHERE facultyId = OLD.id;

END;//

Для забезпечення цілісності даних з таблицею request при видаленні запису:

CREATE TRIGGER faculty\_delete\_restrict\_request BEFORE DELETE ON faculty

FOR EACH ROW

BEGIN

IF OLD.id IN (SELECT enrolleeId FROM request)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can`t delete faculty, which has at least one request.';

END IF;

END;//

Для забезпечення цілісності даних з таблицею request при оновленні запису:

CREATE TRIGGER faculty\_update\_cascade\_request AFTER UPDATE ON faculty

FOR EACH ROW

BEGIN

UPDATE request SET facultyId = NEW.id WHERE facultyId = OLD.id;

END;//

Для забезпечення цілісності даних з таблицею statementRequest при видаленні запису:

CREATE TRIGGER request\_delete\_restrict\_statementRequest BEFORE DELETE ON request

FOR EACH ROW

BEGIN

IF (SELECT statementId FROM statementRequest WHERE enrolleeId = OLD.enrolleeId AND facultyId = OLD.facultyId) IS NOT NULL

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can`t delete request, which used in any statement.';

END IF;

END;//

Для забезпечення цілісності даних з таблицею statementRequest при оновленні запису:

CREATE TRIGGER request\_update\_cascade\_statementRequest AFTER UPDATE ON request

FOR EACH ROW

BEGIN

UPDATE statementRequest SET enrolleeId = NEW.enrolleeId, facultyId = NEW.facultyId

WHERE enrolleeId = OLD.enrolleeId AND facultyId = OLD.facultyId;

END;//

Для забезпечення цілісності даних з таблицею statementRequest при видаленні запису:

CREATE TRIGGER statement\_delete\_cascade\_statementRequest AFTER DELETE ON statement

FOR EACH ROW

BEGIN

DELETE FROM statementRequest WHERE statementId = OLD.id;

END;//

Для забезпечення цілісності даних з таблицею statementRequest при оновленні запису:

CREATE TRIGGER statement\_update\_cascade\_statementRequest AFTER UPDATE ON statement

FOR EACH ROW

BEGIN

UPDATE statementRequest SET statementId = NEW.id WHERE statementId = OLD.id;

END;//

Додатку В

Запити що враховують специфіку високонавантаженої системи:

Запит для отримання всіх абітурієнтів які подали заявку на факультет:

SELECT enrollee.\* FROM enrollee

INNER JOIN request ON request.enrolleeId = enrollee.id

INNER JOIN faculty ON request.facultyId = faculty.id

WHERE faculty.name = ‘Education0’;

Запит для отримання інформації про предмети, які додали абітурієнти з міста:

SELECT subject.\* FROM subject

INNER JOIN enrolleeSubject ON enrolleeSubject.subjectId = subject.id

INNER JOIN enrollee ON enrollee.id = enrolleeSubject.enrolleeId

INNER JOIN city ON enrollee.cityId = city.id

WHERE city.name = ‘Princehaven’;

Запит для отримання всіх факультетів, на які було подано заявки абітурієнтом:

SELECT faculty.\* FORM faculty

INNER JOIN request ON request.facultyId = faculty.id

INNER JOIN enrollee ON request.enrollee = enrollee.id

WHERE enrollee.lastName = ‘HOLBROOK’;

Запит для відображення інформації про заявки, а точніше назву факультету, прізвище та ім’я абітурієнта та його бал для заявки:

SELECT firstName, lastName, name, averageScore FROM request

INNER JOIN faculty ON faculty.id = request.facultyId

INNER JOIN enrollee ON enrollee.id = request.enrolleeId;

Запит для перегляду кількості абітурієнтів з кожного міста:  
SELECT count(\*), city.name FROM city

INNER JOIN enrollee ON enrollee.cityId = city.id

GROUP BY city.name;

Запит для перегляду кількості заявок по кожному з факультетів:

SELECT count(\*), faculty.name FROM faculty

INNER JOIN request ON request.facultyId = faculty.id

GROUP BY faculty.name;

Запит для перегляду кількості абітурієнтів з кожного міста які подали заявку на факультет:

SELECT count(\*), faculty.name, city.name FROM faculty

INNER JOIN request ON request.facultyId = faculty.id

INNER JOIN enrollee ON request.enrolleeId = enrollee.id

INNER JOIN city ON enrollee.cityId = City.id

GROUP BY faculty.name, city.name;

Запит для перегляду кількості предметів які є в абітурієнтів по факультету:

SELECT count(\*), faculty.name, city.name FROM faculty

INNER JOIN request ON request.facultyId = faculty.id

INNER JOIN enrollee ON request.enrolleeId = enrollee.id

INNER JOIN city ON enrollee.cityId = City.id

GROUP BY faculty.name, city.name;

Запит для перегляду абітурієнтів у яких є заявка з середнім балом вище 180.

SELECT \* FROM enrollee

INNER JOIN request ON enrollee.id = request.enrolleeId

WHERE request.averageScore > 180;

Запит для отримання абітурієнтів, які згадали в додатковій інформації про себе слово «piano»:

SELECT \* FROM enrollee WHERE additional LIKE ‘%piano%’;

Запит для отримання факультетів, на які заявок було більше 10:

SELECT faculty.\*, count(\*) as cnt FROM faculty

INNER JOIN request ON request.facultyId = faculty.id

INNER JOIN statementRequest ON statementRequest.facultyId = faculty.id

GROUP BY faculty.name HAVING cnt > 10;

Запит для отримання всіх відомостей, для яких середній бал більше 180:

SELECT statement.\*, avg(request.averageScore) as avgStat

FROM statement INNER JOIN statementRequest ON statementRequest.statementId = statement.Id INNER JOIN request ON statementRequest.facultyId = request.facultyId GROUP BY statement.id HAVING avgStat > 180;

Запит для отримання інформації про всіх абітурієнтів із міста Olafberg у яких є заявка із середнім балом вище 180:

SELECT enrollee.\* FROM request

INNER JOIN enrollee ON enrollee.id = request.enrolleeId

INNER JOIN city ON city.id = enrollee.cityId

WHERE city.name = ‘Olafberg’ AND averageScore > 180;

Додатку Г

Процедури, функції та збережені процедури бізнес-функцій БД.

Функція обрахування балу:

CREATE FUNCTION multiply(coef FLOAT, point FLOAT) RETURNS FLOAT

DETERMINISTIC

BEGIN

RETURN coef \* point;

END//

Функція для отримання id предмета по індексу, котрий потребує факультет:

CREATE FUNCTION get\_faculty\_subject\_id\_by\_index(facultyId INT, index INT) RETURNS INT

DETERMINISTIC

BEGIN

DECLARE subjectIdForReturn INT DEFAULT 0;

SELECT subjectId INTO subjectIdForReturn FROM facultySubject WHERE facultyId = facultyId ORDER BY id LIMIT 1 OFFSET index;

RETURN subjectIdForReturn;

END//

Функція отримання оцінки по id абітурієнта та id предмета:

CREATE FUNCTION get\_enrollee\_point\_by\_enrollee\_id\_subject\_id(enrolleeId INT, subjectId INT)

RETURNS FLOAT

DETERMINISTIC

BEGIN

DELCARE pnt FLOAT DEFAULT 0;

SELECT point INTO pnt FROM enrolleeSubject WHERE enrolleeSubject.enrolleeId = enrolleeId AND enrolleeSubject.subjectId = subjectId;

RETURN pnt;

END//

Функція обрахування середнього балу абітурієнта враховуючи коефіцієнти факультета:

CREATE FUNCTION get\_average\_point\_for\_enrollee\_id\_faculty\_id(enrolleeId INT, facultyId INT) RETURNS FLOAT

DETERMINISTIC

BEGIN

DECLARE firstPoint FLOAT DEFAULT -1;

DECLARE secondPoint FLOAT DEFAULT -1;

DECLARE thirdPoint FLOAT DEFAULT -1;

DECLARE firstCoef FLOAT DEFAULT -1;

DECLARE secondCoef FLOAT DEFAULT -1;

DECLARE thirdCoef FLOAT DEFAULT -1;

DECLARE avgPoint FLOAT DEFAULT 0;

SELECT point INTO firstPoint FROM enrolleeSubject WHERE subjectId = get\_faculty\_subject\_id\_by\_index(facultyId, 0) AND enrolleeId = enrolleeId;

SELECT point INTO secondPoint FROM enrolleeSubject WHERE subjectId = get\_faculty\_subject\_id\_by\_index(facultyId, 1) AND enrolleeId = enrolleeId;

SELECT point INTO thirdPoint FROM enrolleeSubject WHERE subjectId = get\_faculty\_subject\_id\_by\_index(facultyId, 2) AND enrolleeId = enrolleeId;

IF firstPoint = -1 OR secondPoint = -1 OR thirdPoint = -1 THEN

RETURN -1;

ELSE

BEGIN

SELECT coefficient INTO firstCoef FROM facultySubject WHERE facultyId = facultyId AND subjectId = get\_faculty\_subject\_id\_by\_index(facultyId, 0);

SELECT coefficient INTO secondCoef FROM facultySubject WHERE facultyId = facultyId AND subjectId = get\_faculty\_subject\_id\_by\_index(facultyId, 1);

SELECT coefficient INTO thirdCoef FROM facultySubject WHERE facultyId = facultyId AND subjectId = get\_faculty\_subject\_id\_by\_index(facultyId, 2);

SET avgPoint = avgPoint + multiply(firstCoef, firstPoint);

SET avgPoint = avgPoint + multiply(secondCoef, secondPoint);

SET avgPoint = avgPoint + multiply(thirdCoef, thirdPoint);

END;

END IF;

RETURN avgPoint;

END//

Процедура пошуку за прізвищем абітурієнта:

CREATE PROCEDURE find\_by\_lastName(IN lastName VARCHAR(50))

BEGIN

SELECT \* FROM enrollee WHERE enrollee.lastName = lastName;

END//

Процедура створення заявки для участі у конкурсі на факультет по id абітурієнта та id факультету:

CREATE PROCEDURE add\_request(IN enrolleeId INT, IN facultyId INT)

BEGIN

DECLARE avgPoint FLOAT DEFAULT 0;

SET avgPoint = get\_average\_point\_for\_enrollee\_id\_faculty\_id(enrolleeId, facultyId);

IF avgPoint = -1 THEN

SELECT 'Failed';

ELSE

INSERT INTO request(enrolleeId, facultyId, averageScore) VALUES (enrolleeId, facultyId, avgPoint);

SELECT 'Success added';

END IF;

END//

Процедура створення відомості по факультету по id факультету:

CREATE PROCEDURE createStatement(IN facultyId INT)

BEGIN

DECLARE averagePoint FLOAT DEFAULT 0;

DECLARE cnt INT DEFAULT 0;

DECLARE facultyName VARCHAR(50);

DECLARE total INT DEFAULT 0;

DECLARE statementId INT;

DECLARE isExists TINYINT DEFAULT 0;

SELECT count(\*) INTO cnt FROM request WHERE request.facultyId = facultyId GROUP BY facultyId;

SELECT generalCount INTO total FROM faculty WHERE faculty.id = facultyId;

SELECT avg(averageScore) INTO averagePoint FROM request WHERE request.facultyId = facultyId GROUP BY facultyId LIMIT total;

SELECT name INTO facultyName FROM faculty WHERE faculty.id = facultyId;

INSERT INTO statement(date, count, averageScore, description, name)

VALUES (now(), cnt, averagePoint, "somedescrpition", concat("Statement for ", facultyName));

SELECT count(\*) INTO isExists FROM statement

WHERE count = cnt AND averageScore = averagePoint AND description = "somedescrpition";

IF isExists = 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Could not create statement for faculty.';

END IF;

SELECT LAST\_INSERT\_ID() INTO statementId;

INSERT INTO statementrequest SELECT enrolleeId, facultyId, statementId from request where facultyId = 1;

END//

Представлення для відображення інформації про заявки, а точніше назву факультету, прізвище та ім’я абітурієнта та його бал для заявки:

CREATE OR REPLACE VIEW enrolleeRequestInfo AS

SELECT firstName, lastName, name, averageScore FROM request

INNER JOIN faculty ON faculty.id = request.facultyId

INNER JOIN enrollee ON enrollee.id = request.enrolleeId;\

Запит для отримання інформації про всіх абітурієнта за прізвищем із міста Olafberg:

SELECT \* FROM enrollee

WHERE enrollee.lastName = ‘VOGEL’ AND cityId = (SELECT city.id FROM city WHERE city.name = 'Olafberg');

Додатку Д

Додавання індексів, які ще не були накладені під час створення таблиць.

ALTER TABLE `enrollee`

ADD FULLTEXT INDEX `additionalFullText`

(`additional` ASC);

ALTER TABLE `subject`

ADD FULLTEXT INDEX `subjectDescriptionFullText`

(`description` ASC);

ALTER TABLE `faculty`

ADD FULLTEXT INDEX `facultyDescriptionFullText`

(`description` ASC);

ALTER TABLE `statement`

ADD FULLTEXT INDEX `statementDescriptionFullText`

(`description` ASC);