Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра СТ

Дисципліна: «Проектування високонавантажених баз даних та знань»

Лабораторна робота №3

«Реалізація програмної логіки та забезпечення цілісності зв’язків при допомозі тригерів для високонавантажених баз даних

На платформі СУБД MySQL»

|  |  |
| --- | --- |
| Виконав:  ст. гр. ІТКНу-17-1  Гончаренко В. О. | Перевірили:  Коваленко А.І. |

Харків 2018

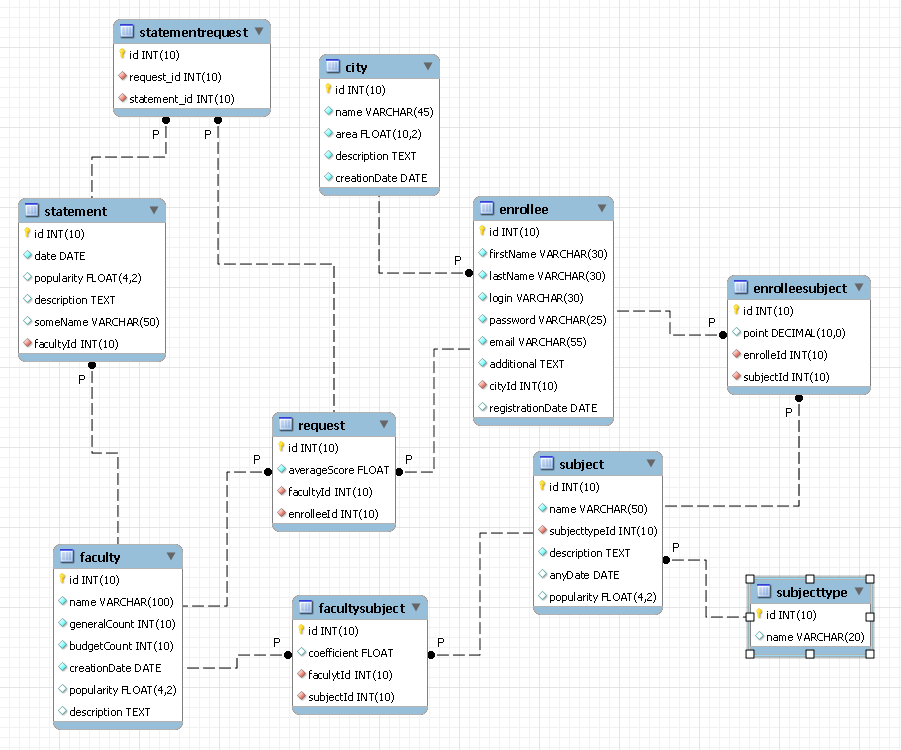
Ціль роботи – отримання практичних навичко по розробці тригерів для підтримки цілісності за’язків, модифікації даних та забезпечення основних бізнес-процесів високонавантеженної інформаційної системи.

## Тема індивідуального завдання та постановка задачі.

Тема індивідуального завдання є «Високонавантажена інформаційна система «Вступна кампанія»».

Вступна кампанія – це конкурс який проводиться для абітурієнтів, для того щоб сформувати списки абітурієнтів які поступили на факультет, які не поступили, їхні оцінки та інша інформація.

## Скріншот ЕРД.



## Перелік доступних операцій тригера:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Доступність | Before insert | After  Insert | Before  Update | After  Update | Before  Delete | After  Delete |
| New | Да | Да | Да | Да | Нет | Нет |
| Old | Нет | Нет | Да | Да | Да | Да |
| Можливість змінити поля | Да | Нет | Да | Нет | Нет | Нет |
| Значення автоінкременту | Нет | Да | Нет | Нет | Нет | Нет |
| Доступ до значення полів (та їх значення DEFAULT)  (SET NEW.a = DEFAULT() OR  GET VALUE FROM information\_schema.columns) | Да | Да | Да | Да | Да | Да |
| Можливість відміни операції | Нет | Нет | Нет | Нет | Нет | Нет |

## Перелік переваг використання тригерів

|  |  |  |  |
| --- | --- | --- | --- |
| № | Анализируемые параметры | Триггер | SQL – запрос |
| 1 | Принцип исполнения SQL кода в СУБД | Вызывается как хранимая процедура из скомпилированного вида | Проходит много стадий (оптимизация, кеширование, парсинг) |
| 2 | Влияние на ресуры СУБД | Компилируется один раз | Постоянно проходят стадии оптимизации и парсинга |
| 3 | Влияние на сетевой трафик | Сокращает потребление | Увеличивает |
| 4 | Возможность что либо автоматизировать | + | \_ |
| 5 | SQL – синтаксис(ограничения) | Есть некоторые ограничение в зависимости от вида. Может создавать только SUPER user. | Почти нет ограничений. |
| 6 | Принцип хранения SQL кода | Хранится скомпилированным на сервере | Могут хранится в кеше после первого вызова |

## Перелік типів цілісності:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Таблиця 1, зовнішній ключ | Таблиця 2, первинний ключ | Тип цілісності для таблиці 1 | Тип цілісності для таблиці 2 | Триггер |
| 1 | Enrollee, cityId | City, id | ON DELETE NO ACTION | ON DELETE NO ACTION | AFTER  DELETE |
| 2 | Enrollee, cityId | City, id | ON UPDATE CASCADE | ON UPDATE CASCADE | BEFORE UPDATE |
| 3 | Subject, subjectTypeId | SubjectType, id | ON DELETE  NO ACTION | ON DELETE  NO ACTION | AFTER DELETE |
| 4 | Subject, subjectTypeId | SubjectType, id | ON UPDATE CASCADE | ON UPDATE CASCADE | BEFORE UPDATE |
| 5 | EnrolleeSubject, subjectId | Subject, id | ON DELETE  NO ACTION | ON DELET  NO ACTION | AFTER DELETE |
| 6 | EnrolleeSubject, subjectId | Subject, id | ON UPDATE CASCADE | ON UPDATE CASCADE | BEFORE UPDATE |
| 7 | EnrolleeSubject,  enrolleeId | Enrollee, id | ON DELETE CASCADE | ON DELETE  CASCADE | BEFORE DELETE |
| 8 | EnrolleeSubject, enrolleeId | Enrollee,id | ON UPDATE CASCADE | ON UPDATE CASCADE | BEFORE DELETE |
| 9 | Request, enrolleeId | Enrollee,Id | ON DELETE CASCADE | ON DELETE CASCADE | AFTER DELETE |
| 10 | Reqeuest, enrolleId | Enrollee,id | ON UPDATE CASCADE | ON UPDATE CASCADE | BEFORE UPDATE |
| 11 | Request, facultyId | Faculty, id | ON DELETE CASCADE | ON DELETE CASCADE | BEFORE DELETE |
| 12 | Request, Facultyid | Faculty, id | ON UPDATE CASCADE | ON UPDATE CASCADE | BEFORE UPDATE |

DELIMITER // -- Определяем наш делиметр для окончания обьявления тригера.

Создаем триггер для таблицы city BEFORE где будет проверка на уникальность вставляемого города и что его айди будет не меньше 0.

DROP TRIGGER IF EXISTS new\_city;

CREATE TRIGGER new\_city BEFORE INSERT ON city

FOR EACH ROW

BEGIN

IF NEW.id IN (SELECT id FROM city)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Id should be unique';

ELSEIF NEW.id < 0

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Id should not be less than 0';

END IF;

END;//

Создаем триггер update before для проверки айди на то, что новы айди, который хотят апдейтнуть уже существует или же что он меньше 0.

DROP TRIGGER IF EXISTS city\_update;

CREATE TRIGGER city\_update BEFORE UPDATE ON city

FOR EACH ROW

BEGIN

IF NEW.id IN (SELECT id FROM city)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'The ID already exists';

ELSEIF NEW.id < 0

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Id should not be less than 0';

END IF;

END;//

Создаем триггер для замены айди в таблице enrollee со старого на новый.

DROP TRIGGER IF EXISTS city\_update\_after;

CREATE TRIGGER city\_update\_after AFTER UPDATE ON city

FOR EACH ROW

BEGIN

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

END;//

Триггер для проверки города перед удаление на то, привязан ли к нему хотя бы один абитуриент:

DROP TRIGGER IF EXISTS city\_delete\_before;

CREATE TRIGGER city\_delete\_before 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 use in enrollee table.';

ELSEIF OLD.id NOT IN (SELECT id FROM city)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can\'t delete city, which is not exists.';

END IF;

END;//

Тригер before insert для абитуриентов. Валидирует айди на >0 и уникальность.

DROP TRIGGER IF EXISTS enrolee\_before\_insert;

CREATE TRIGGER enrolee\_before\_insert BEFORE INSERT ON enrolee

FOR EACH ROW

BEGIN

IF NEW.id < 0

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'ID should not be less than 0.';

ELSEIF NEW.id IN (SELECT id FROM enrolee)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'ID should be unique.';

END IF;

END;//

Триггер before update для таблицы абитуриентов. Проверяет новый айди на уникальность и на то, что он больше 0.

DROP TRIGGER IF EXISTS enrolee\_before\_update;

CREATE TRIGGER enrolee\_before\_update BEFORE UPDATE ON enrolee

FOR EACH ROW

BEGIN

IF NEW.id < 0

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'ID should not be less than 0.';

IF NEW.id IN (SELECT id FROM enrolee)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'ID should be unique.';

END IF;

END;//

Тригер update after для таблицы абитуриентов. Обновляет значения в таблицах ernoleleSubject & request на новый айди абитуриента.

DROP TRIGGER IF EXISTS enrolee\_after\_update;

CREATE TRIGGER enrolee\_after\_update AFTER UPDATE ON enrolee

FOR EACH ROW

BEGIN

UPDATE request, enrolleeSubject

SET request.enrolleeId = NEW.id, enrolleeSubject.enrolleeId = NEW.id

WHERE request.enrolleeId = OLD.id AND enrolleeSubject.enrolleeId = OLD.id;

END;//

Триггер before delete для таблицы абитуриентов. Проверяет есть ли у абитуриента, которого пытаемся удалить хотя бы одна заявка или же есть хотя бы одна оценка ИЛИ ЖЕ если айди удаляемого пользователя меньше ноля или же если абитуриент с этим айди не существует. В каждом из этих случае выкидывается ошибка.

DROP TRIGGER IF EXISTS enrolee\_before\_delete;

CREATE TRIGGER enrolee\_before\_delete BEFORE DELETE ON enrolee

FOR EACH ROW

BEGIN

IF EXISTS (SELECT \* FROM reuqest WHERE enrollee.id = OLD.id)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can\'t delete enrollee who has any request.';

ELSEIF EXISTS (SELECT enrolleeId FROM enrolleeSubject WHERE enrolleeId = OLD.id)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can\'t delete enrollee who has any points in table enrolleesubject.';

ELSEIF OLD.id < 0

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can\'t delete enrollee with id which less than 0.';

ELSEIF NOT EXISTS(SELECT \* FROM enrolee WHERE id = OLD.id)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can\'t delete enrolee by not existing id.';

END IF;

END;//

Триггер before delete для таблицы subject. Проверяет есть ли хотя бы у одноого абитуриента оценка по этому предмету или же используется он хотя бы одним факультетов. Если да то будет вызвана ошибка.

DROP TRIGGER IF EXISTS subject\_before\_delete;

CREATE TRIGGER subject\_before\_delete BEFORE DELETE ON subject

FOR EACH ROW

BEGIN

IF EXISTS (SELECT \* FROM enrolleeSubject WHERE subjectId = OLD.id)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can\'t delete subject which use in enrolleeSubject.';

IF EXISTS (SELECT \* FROM facultySubject WHERE subjectId = OLD.id)

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can\'t delete subject which use in facultySubject.';

END IF;

END;//

Триггер before delete для таблицы факультетов. Проверяет если количество заявок для этого факультета больше 0 - если да то бросаем ошибку.

DROP TRIGGER IF EXISTS faculty\_before\_delete;

CREATE TRIGGER faculty\_before\_delete BEFORE DELETE ON faculty

FOR EACH ROW

BEGIN

IF (SELECT COUNT(\*) FROM request WHERE facultyId = OLD.id) > 0

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'You can\'t delete faculty which use in request.';

END;//

Триггер after delete для таблицы факультетов. Удаляет все предметы которые регистрировал факультет и ведомость, которая связана с ним.

DROP TRIGGERS IF EXISTS faculty\_after\_delete;

CREATE TRIGGER faculty\_after\_delete AFTER DELETE ON faculty

BEGIN

DELETE \* FROM facultySubject WHERE facultyId = OLD.id;

DELETE \* FROM statement WHERE facultyId = OLD.id;

END;//

Триггер для автоматизации высоконагруженной системы. Триггер который при добавлении заявки для участия в конкурсе на факультет высчитывает бал абитуриента.

DROP TRIGGER IF EXISTS request\_before\_insert;

CREATE TRIGGER request\_before\_insert BEFORE INSERT ON request

FOR EACH ROW

BEGIN

SET @firstSubject = (SELECT subjectId FROM facultySubject WHERE facultyId = NEW.facultyId LIMIT 1);

SET @secondSubject = (SELECT subjectId FROM facultySubject WHERE facultyId = NEW.facultyId LIMIT 1 OFFSET 1);

SET @thirdSubject = (SELECT subjectId FROM facultySubject WHERE facultyId = NEW.facultyId LIMIT 1 OFFSET 2);

SET @firstPoint = (SELECT point FROM enrolleeSubject WHERE enrolleeId = NEW.enrolleeId AND subjectId = @firstSubject);

SET @secondPoint = (SELECT point FROM enrolleeSubject WHERE enrolleeId = NEW.enrolleeId AND subjectId = @secondSubject);

SET @thirdPoint = (SELECT point FROM enrolleeSubject WHERE enrolleeId = NEW.enrolleeId AND subjectId = @thirdSubject);

SET @errorMessage = NULL;

IF @firstPoint IS NULL

OR @secondPoint IS NULL

OR @thirdPoint IS NULL

THEN SET @errorMessage = 'Enrollee has not next points by subject with id : ';

END IF;

IF @firstPoint IS NULL

THEN SET @errorMessage = concat(@errorMessage, @firstSubject);

END IF;

IF @secondPoint IS NULL

THEN SET @errorMessage = concat(@errorMessage, concat(',', @secondSubject));

END IF;

IF @thirdPoint IS NULL

THEN SET @errorMessage = concat(@errorMessage, concat(',', @thirdSubject));

END IF;

IF @errorMessage IS NOT NULL

THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = @errorMessage;

END IF;

SET @firstCoefficient = (SELECT coefficient FROM facultySubject WHERE facultyId = NEW.facultyId AND subjectId = @firstSubject);

SET @secondCoefficient = (SELECT coefficient FROM facultySubject WHERE facultyId = NEW.facultyId AND subjectId = @secondSubject);

SET @thirdCoefficient = (SELECT coefficient FROM facultySubject WHERE facultyId = NEW.facultyId AND subjectId = @thirdSubject);

SET @result = @firstPoint \* @firstCoefficient + @secondPoint \* @secondCoefficient + @thirdPoint \* @thirdCoefficient;

SET NEW.averageScore = @result;

END;//

Триггеры для InnoDB.

Триггер after delete для таблицы факультет. Удаляем из таблиця заявок, ведомостей и оценок все, что было связанно с удаленным факультетом.

DROP TRIGGERS IF EXISTS faculty\_after\_delete;

CREATE TRIGGER faculty\_after\_delete AFTER DELETE ON faculty

BEGIN

DELETE \* FROM facultySubject WHERE facultyId = OLD.id;

DELETE \* FROM statement WHERE facultyId = OLD.id;

DELETE \* FROM request WHERE facultyId = OLD.id

END;//

Триггер для удаления всех заявок абитуриента и всех его оценок после удаления самого абитуриента.

DROP TRIGGER IF EXISTS enrollee\_after\_delete;

CREATE TRIGGER enrolee\_after\_delete AFTER DELETE ON enrolee

FOR EACH ROW

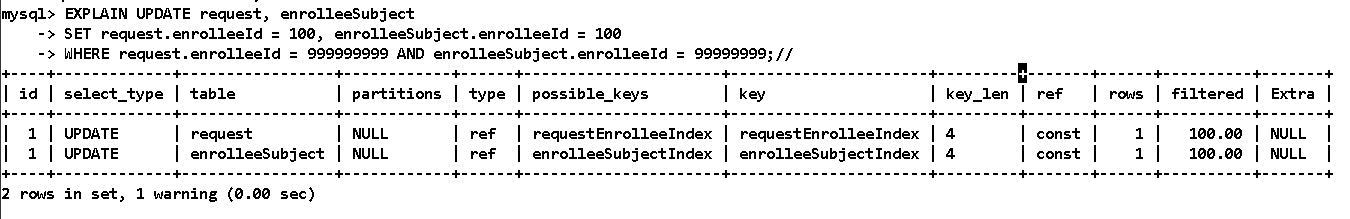
BEGIN

DELETE FROM request WHERE enrolleeId = OLD.id;

DELETE FROM enrolleeSubject WHERE enrolleeId = OLD.id;

END;//

Запросы с командой експлейн



## ВЫВОДЫ

На лабораторной работе были получены и закреплены навыки работы с построением триггеров для поддержки целостности связей, модификации данных и обеспечения основных бизнес-процессов высоконагруженной ИС. Были сформированы необходимые практические умения для анализа плана выполнения SQL запросов. Сформированы необходимые практические умения для создания триггеров с учетом особенностей реализации логики работы интерфейса высоконагруженной ИС хранения данных.

Изучены цели и назначения использования триггеров. Был изучен синтаксис создания триггера CREATE TRIGGER, изученны особенности его активации (BEORE, AFTER) а также изучены особенности создания триггера для всех возможных типов операций (INSER, UPDATE, DELETE). Изучены особенности использования предложений NEW and OLD в исполняемом SQL коде триггера для различных условий и событий его активации. Изучены особенности использования операторов EXISTS(NOT EXISTS), IS NULL(IS NOT NULL) IFNULL, COALESCE, NULLIF, IF, COUNT. Изучено использования предложения SIGNAL которое используется для прерывания на этапе BEFORE в случае невыполнения какого-либо из необходимых условий.