ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____

ПРЕПОДАВАТЕЛЬ

Доцент				Бржезовский А. В.
должность, уч. степень, звание		подпись, дата		инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7

ТРИГГЕРЫ

Вариант 5

по курсу: Методы и средства проектирования информационных систем и технологий

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4128				Воробьев В.А.
			подпись, дата		инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Постановка задачи

1.1 Задание

Реализовать:

— триггеры каждого вида (after, instead of) для каждой из операций (insert, update, delete), обеспечивающие активную целостность данных;

— триггер, реализующий вычисления или формирование статистики или ведение истории изменений в БД.

1.2 Содержание отчета

— тексты триггеров (с пояснениями/комментариями); — SQL операторы, вызывающие запуск триггеров, и наборы данных, иллюстрирующие их работу.

5 Вариант:

Создайте базу данных для хранения следующих сведений: ВУЗ, студент, группа, факультет, конференция, тема доклада, программа конференции. Составьте запросы, позволяющие выбрать:

а) студентов первого факультета, выступавших на конференции Информатика;

б) темы докладов студентов для заданной группы;

в) выступления, подготовленные двумя студентами различных факультетов;

г) количество докладов для каждой конференции;

д) среднее количество докладов, сделанных студентами третьего факультета на конференциях;

е) студентов, выступивших на трех или большем числе конференций;

ж) студентов четвертого факультета, не выступавших на конференциях;

з) студентов, выступивших на всех конференциях;

и) пары студентов, всегда выступающие вместе.

2 Выполнение работы

Исходные данные взяты из лабораторной работы №2, отчет для которой есть на GitHub (URI - https://github.com/vladcto/suai-labs/blob/d8c7a508971967641d8638ebcd107539c8fd618e/6_semester/%D0%9C%D0%A1%D0%9F%D0%98%D0%A1%D0%A2/%D0%BC%D1%81%D0%B8%D0%BF%D0%B8%D1%81%D1%82_2.pdf).

Текст запросов представлен в Приложении, а также в репозитории GitHub (URI - https://github.com/vladcto/suai-labs/tree/1feac804866a924523979b3a271d293076b96bdf/6_semester/%D0%9C%D0%A1%D0%9F%D0%98%D0%A1%D0%A2/7).

2.1 Демонстрация работы триггеров

Так как MySQL не может представить все триггеры, то было решено часть задания реализовать используя PostgreSQL.

2.1.1 after

Реализован триггер after_university_insert, активирующийся после операции INSERT в таблице university. Функционал данного триггера предусматривает вставку новой записи в таблицу faculty с university_id, равным вновь вставленному id, и начальным номером факультета 99 для каждой вставленной строки.

Далее, создан триггер after_uni_group_update, срабатывающий после операции UPDATE в таблице uni_group. Данный триггер обеспечивает обновление столбца name в таблице student путем конкатенации текущего значения name со новым значением name группы, заключенным в круглые скобки, для всех студентов, относящихся к обновленной группе.

```
1  USE conference_db_lab1;
2
3  CREATE TRIGGER after_university_insert
4    AFTER INSERT
5    ON university
6    FOR EACH ROW
7  BEGIN
8    INSERT INTO faculty(university_id, number)
9      VALUES (NEW.id, 99);
10 END;
```

```
11
12  CREATE TRIGGER after_uni_group_update
13    AFTER UPDATE
14    ON uni_group
15    FOR EACH ROW
16  BEGIN
17    UPDATE student
18    SET name = CONCAT(name, ' (', NEW.name, ')')
19      WHERE group_id = NEW.id;
20  END;
```

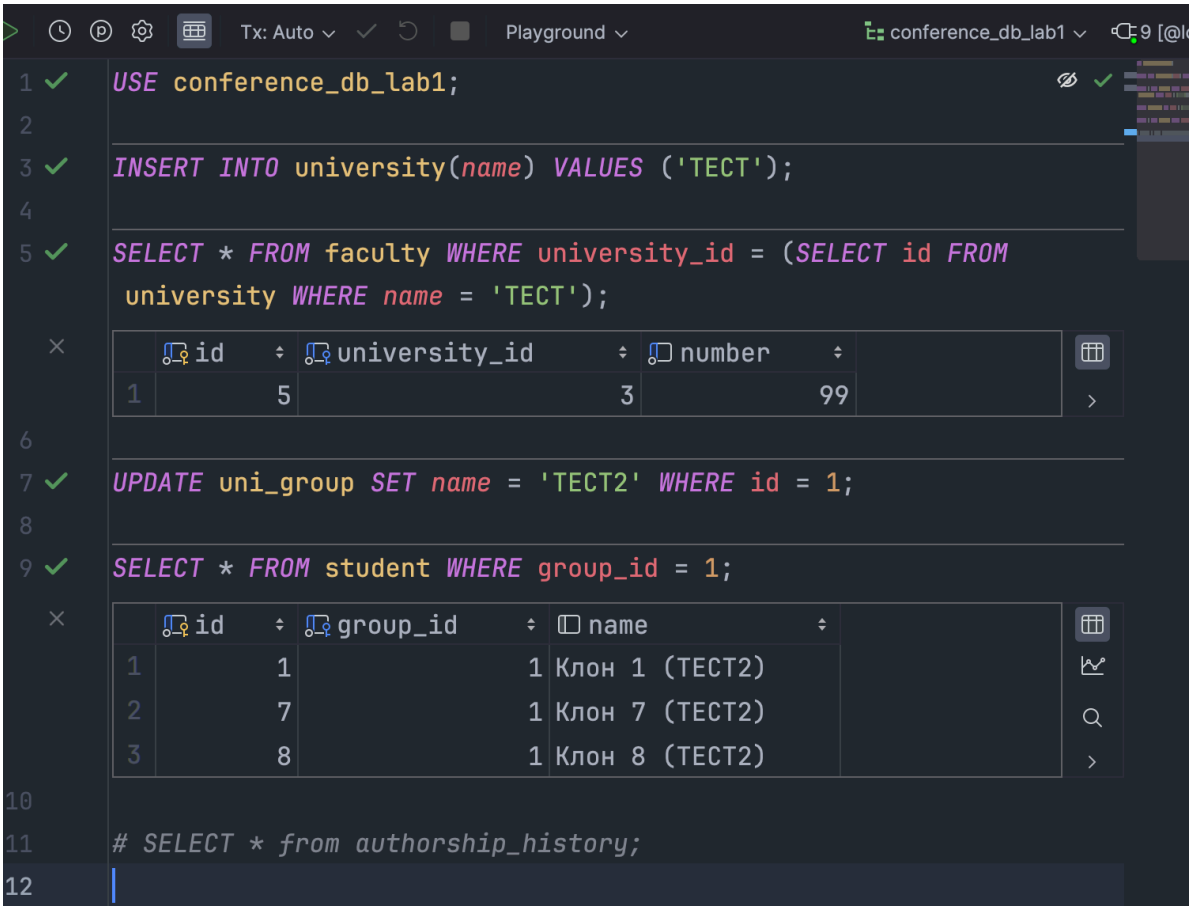

Рисунок 2.1 - Демонстрация работы

2.1.2 instead of

Для демонстрации работы триггера `instead of` был выбран `PostgreSQL`.

Скрипт инициализации представлен ниже.

```
1  INSERT INTO university (name)
2    VALUES ('СПбГУ');
3
```

```
4  INSERT INTO faculty
5    VALUES (1, 1, 1);
6
7  INSERT INTO uni_group (faculty_id, name)
8    VALUES (1, 'Группа 1');
9
10 INSERT INTO student (group_id, name)
11   VALUES (1, 'Клон 1');
12
13 INSERT INTO conference (name, place, theme)
14   VALUES ('Научная конференция', 'ГУАП', 'Искусственный
       интеллект');
15
16 INSERT INTO conference_session (conference_id, start_time,
     end_time)
17   VALUES (1, '2022-01-01 10:00:00', '2022-01-01 18:00:00');
18
19 INSERT INTO topic (title, session_id)
20   VALUES ('Тема 1', 1);
21
22 INSERT INTO authorship (author_id, topic_id)
23   VALUES (1, 1);
```

```
1  CREATE TABLE IF NOT EXISTS university
2  (
3    id   SERIAL PRIMARY KEY,
4    name VARCHAR(100) NOT NULL UNIQUE
5  );
6
7  CREATE TABLE IF NOT EXISTS faculty
8  (
9    id            SERIAL PRIMARY KEY,
10   university_id INT NOT NULL,
11   number        INT NOT NULL UNIQUE
12 );
13
14 CREATE TABLE IF NOT EXISTS uni_group
15 (
16   id         SERIAL PRIMARY KEY,
17   faculty_id INT          NOT NULL,
18   name       VARCHAR(50) NOT NULL UNIQUE
19 );
```

```
20
21  CREATE TABLE IF NOT EXISTS student
22  (
23    id         SERIAL PRIMARY KEY,
24    group_id INT NOT NULL,
25    name       VARCHAR(100)
26  );
27
28  CREATE TABLE IF NOT EXISTS conference
29  (
30    id     SERIAL PRIMARY KEY,
31    name  VARCHAR(100) NOT NULL DEFAULT 'Научная конференция',
32    place VARCHAR(100) NOT NULL DEFAULT 'ГУАП',
33    theme VARCHAR(255) NOT NULL
34  );
35
36  CREATE TABLE IF NOT EXISTS conference_session
37  (
38    id              SERIAL PRIMARY KEY,
39    conference_id INT        NOT NULL,
40    start_time     TIMESTAMP NOT NULL,
41    end_time       TIMESTAMP NOT NULL
42  );
43
44  CREATE TABLE IF NOT EXISTS topic
45  (
46    id           SERIAL PRIMARY KEY,
47    title        VARCHAR(255) NOT NULL,
48    session_id INT            NOT NULL
49  );
50
51  CREATE TABLE IF NOT EXISTS authorship
52  (
53    author_id INT NOT NULL,
54    topic_id  INT NOT NULL,
55    PRIMARY KEY (author_id, topic_id)
56  );
```

Реализованы представления student_view и groups_view, предоставляющие полный набор данных из таблиц student и uni_group соответственно. Созданы функции student_insert() и student_update() для вставки и об-

новления данных в таблице student, а также триггеры student_insert_trigger и student_update_trigger, активирующие эти функции вместо операций INSERT и UPDATE в представлении student_view.

Для удаления групп вместе с относящимися к ним студентами создана функция group_remove(). Реализован триггер group_delete_trigger, который срабатывает вместо операции DELETE в представлении groups_view и выполняет функцию group_remove() для каждой удаляемой строки.

```
1  CREATE OR REPLACE VIEW student_view AS
2  SELECT *
3    FROM student;
4
5  CREATE OR REPLACE VIEW groups_view AS
6  SELECT *
7    FROM uni_group;
8
9  CREATE OR REPLACE FUNCTION student_insert() RETURNS TRIGGER
       AS
10 \begin{equation}\begin{gathered}
11 BEGIN
12   INSERT INTO student(group_id, name) VALUES (new.group_id,
         new.name);
13   RETURN new;
14 END;
15 \end{gathered}\end{equation} LANGUAGE plpgsql;
16
17 CREATE OR REPLACE TRIGGER student_insert_trigger
18   INSTEAD OF INSERT
19   ON student_view
20   FOR EACH ROW
21 EXECUTE PROCEDURE student_insert();
22
23 CREATE OR REPLACE FUNCTION student_update() RETURNS TRIGGER
       AS
24 \begin{equation}\begin{gathered}
25 BEGIN
26   UPDATE student SET group_id = new.group_id, name = new.name
         WHERE id = old.id;
27   RETURN new;
28 END;
29 \end{gathered}\end{equation} LANGUAGE plpgsql;
```

```
30
31 CREATE OR REPLACE TRIGGER student_update_trigger
32   INSTEAD OF UPDATE
33   ON student_view
34   FOR EACH ROW
35 EXECUTE PROCEDURE student_update();
36
37 CREATE OR REPLACE FUNCTION group_remove() RETURNS TRIGGER AS
38 \begin{equation}\begin{gathered}
39 BEGIN
40   DELETE FROM student WHERE student.group_id = old.id;
41   DELETE FROM uni_group WHERE uni_group.id = old.id;
42   RETURN old;
43 END;
44 \end{gathered}\end{equation} LANGUAGE plpgsql;
45
46 CREATE OR REPLACE TRIGGER group_delete_trigger
47   INSTEAD OF DELETE
48   ON groups_view
49   FOR EACH ROW
50 EXECUTE PROCEDURE group_remove();
```

```sql
INSERT INTO student_view (group_id, name)
  VALUES (1, 'Клон 11');

UPDATE student_view
SET name = 'Клон 12'
  WHERE name = 'Клон 11';

DELETE
  FROM groups_view
  WHERE id = 1;

SELECT *
  FROM student;
```

	id	group_id	name
1	5	1	Клон 12
2	6	1	Клон 12
3	7	1	Клон 12
4	8	1	Клон 12
5	9	1	Клон 12

```sql
SELECT *
  FROM uni_group;
```

id	faculty_id	name

Рисунок 2.2 - Демонстрация работы

2.2 Логирование операций БД

Скрипт реализует логирование операций в таблице authorship.

Создается таблица authorship_history с колонками: id, author_id,

topic_id, change_type и change_time. Далее, создаются триггеры для фиксации операций INSERT, DELETE и UPDATE в таблице authorship: authorship_after_insert, authorship_after_delete и authorship_after_update. При выполнении этих операций данные о текущем изменении автором и темой, а также тип изменения (‘INSERT’, ‘DELETE’ или ‘UPDATE’) сохраняются в таблицу authorship_history.

```
1  USE conference_db_lab1;
2
3  CREATE TABLE IF NOT EXISTS authorship_history
4  (
5    id          INT PRIMARY KEY AUTO_INCREMENT,
6    author_id   INT          NOT NULL,
7    topic_id    INT          NOT NULL,
8    change_type VARCHAR(10) NOT NULL,
9    change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
10 );
11
12 DELIMITER //
13 CREATE TRIGGER authorship_after_insert
14   AFTER INSERT
15   ON authorship
16   FOR EACH ROW
17 BEGIN
18   INSERT INTO authorship_history (author_id, topic_id,
       change_type)
19     VALUES (NEW.author_id, NEW.topic_id, 'INSERT');
20 END;
21 //
22
23 CREATE TRIGGER authorship_after_delete
24   AFTER DELETE
25   ON authorship
26   FOR EACH ROW
27 BEGIN
28   INSERT INTO authorship_history (author_id, topic_id,
       change_type)
29     VALUES (OLD.author_id, OLD.topic_id, 'DELETE');
30 END;
31 //
32
```

```
33  CREATE TRIGGER authorship_after_update
34    AFTER UPDATE
35    ON authorship
36    FOR EACH ROW
37  BEGIN
38    INSERT INTO authorship_history (author_id, topic_id,
          change_type)
39      VALUES (NEW.author_id, NEW.topic_id, 'UPDATE');
40  END;
41  //
42  DELIMITER ;
```

```
SELECT * from authorship_history;
```

	id	author_id	topic_id	change_type	change_time
1	1	1	1	INSERT	2024-05-23 21:07:34
2	2	2	1	INSERT	2024-05-23 21:07:34
3	3	1	2	INSERT	2024-05-23 21:07:34
4	4	3	3	INSERT	2024-05-23 21:07:34
5	5	1	4	INSERT	2024-05-23 21:07:34
6	6	4	5	INSERT	2024-05-23 21:07:34
7	7	1	6	INSERT	2024-05-23 21:07:34
8	8	5	7	INSERT	2024-05-23 21:07:34
9	9	7	8	INSERT	2024-05-23 21:07:34
10	10	8	8	INSERT	2024-05-23 21:07:34
11	11	1	9	INSERT	2024-05-23 21:07:34
12	12	1	10	INSERT	2024-05-23 21:07:34
13	13	1	11	INSERT	2024-05-23 21:07:34

Рисунок 2.3 - Демонстрация работы

3 Вывод

В соответствии с поставленной задачей реализованы триггеры для обеспечения активной целостности данных и ведения истории изменений в базе данных.

Для операций INSERT, UPDATE и DELETE в таблице authorship созданы триггеры after insert, after update и after delete, которые обеспечивают сохранение информации об изменениях в таблице authorship_history. Эти триггеры фиксируют тип изменения, идентификатор автора и темы, а также время изменения.

Также реализован триггер instead of insert для представления student_view, который обеспечивает вставку данных в таблицу student при попытке вставки в представление. Аналогичным образом реализован триггер instead of update для этого представления, который обеспечивает обновление данных в таблице student. Для представления groups_view реализован триггер instead of delete, который обеспечивает удаление строк из таблиц student и uni_group при попытке удаления из представления.

Таким образом, реализованные триггеры обеспечивают активную целостность данных и ведение истории изменений в базе данных.

ПРИЛОЖЕНИЕ

```
1
2  autorship_history.sql
3  USE conference_db_lab1;
4
5  CREATE TABLE IF NOT EXISTS authorship_history
6  (
7    id          INT PRIMARY KEY AUTO_INCREMENT,
8    author_id   INT          NOT NULL,
9    topic_id    INT          NOT NULL,
10   change_type VARCHAR(10) NOT NULL,
11   change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
12 );
13
14 -- Создаем триггеры для отслеживания изменений в таблице
     authorship
15 DELIMITER //
16 CREATE TRIGGER authorship_after_insert
17   AFTER INSERT
18   ON authorship
19   FOR EACH ROW
20 BEGIN
21   INSERT INTO authorship_history (author_id, topic_id,
       change_type)
22     VALUES (NEW.author_id, NEW.topic_id, 'INSERT');
23 END;
24 //
25
26 CREATE TRIGGER authorship_after_delete
27   AFTER DELETE
28   ON authorship
29   FOR EACH ROW
30 BEGIN
31   INSERT INTO authorship_history (author_id, topic_id,
       change_type)
32     VALUES (OLD.author_id, OLD.topic_id, 'DELETE');
33 END;
34 //
35
36 CREATE TRIGGER authorship_after_update
37   AFTER UPDATE
```

```
38    ON authorship
39    FOR EACH ROW
40  BEGIN
41    INSERT INTO authorship_history (author_id, topic_id,
          change_type)
42      VALUES (NEW.author_id, NEW.topic_id, 'UPDATE');
43  END;
44  //
45  DELIMITER ;
46  check_trigger.sql
47  USE conference_db_lab1;
48
49  -- Проверка триггера after_university_insert
50  INSERT INTO university(name) VALUES ('ТЕСТ');
51
52  SELECT * FROM faculty WHERE university_id = (SELECT id FROM
      university WHERE name = 'ТЕСТ');
53
54  UPDATE uni_group SET name = 'ТЕСТ2' WHERE id = 1;
55
56  SELECT * FROM student WHERE group_id = 1;
57
58  SELECT * from authorship_history;
59
60  fill.sql
61  INSERT INTO university (name)
62    VALUES ('СПбГУ');
63
64  INSERT INTO faculty
65    VALUES (1, 1, 1);
66
67  INSERT INTO uni_group (faculty_id, name)
68    VALUES (1, 'Группа 1');
69
70  INSERT INTO student (group_id, name)
71    VALUES (1, 'Клон 1');
72
73  INSERT INTO conference (name, place, theme)
74    VALUES ('Научная конференция', 'ГУАП', 'Искусственный
        интеллект');
75
```

```
76  INSERT INTO conference_session (conference_id, start_time,
        end_time)
77    VALUES (1, '2022-01-01 10:00:00', '2022-01-01 18:00:00');
78
79  INSERT INTO topic (title, session_id)
80    VALUES ('Тема 1', 1);
81
82  INSERT INTO authorship (author_id, topic_id)
83    VALUES (1, 1);
84  check_trigger.sql
85  INSERT INTO student_view (group_id, name)
86    VALUES (1, 'Клон 11');
87
88  UPDATE student_view
89  SET name = 'Клон 12'
90    WHERE name = 'Клон 11';
91
92  DELETE
93    FROM groups_view
94    WHERE id = 1;
95
96  SELECT *
97    FROM student;
98
99  SELECT *
100   FROM uni_group;
101 instead_of.sql
102 CREATE OR REPLACE VIEW student_view AS
103 SELECT *
104   FROM student;
105
106 CREATE OR REPLACE VIEW groups_view AS
107 SELECT *
108   FROM uni_group;
109
110 CREATE OR REPLACE FUNCTION student_insert() RETURNS TRIGGER
        AS
111 \begin{equation}\begin{gathered}
112 BEGIN
113   INSERT INTO student(group_id, name) VALUES (new.group_id,
        new.name);
```

```
114     RETURN new;
115 END;
116 \end{gathered}\end{equation} LANGUAGE plpgsql;
117
118 CREATE OR REPLACE TRIGGER student_insert_trigger
119     INSTEAD OF INSERT
120     ON student_view
121     FOR EACH ROW
122 EXECUTE PROCEDURE student_insert();
123
124 CREATE OR REPLACE FUNCTION student_update() RETURNS TRIGGER
        AS
125 \begin{equation}\begin{gathered}
126 BEGIN
127     UPDATE student SET group_id = new.group_id, name = new.name
          WHERE id = old.id;
128     RETURN new;
129 END;
130 \end{gathered}\end{equation} LANGUAGE plpgsql;
131
132 CREATE OR REPLACE TRIGGER student_update_trigger
133     INSTEAD OF UPDATE
134     ON student_view
135     FOR EACH ROW
136 EXECUTE PROCEDURE student_update();
137
138 CREATE OR REPLACE FUNCTION group_remove() RETURNS TRIGGER AS
139 \begin{equation}\begin{gathered}
140 BEGIN
141     DELETE FROM student WHERE student.group_id = old.id;
142     DELETE FROM uni_group WHERE uni_group.id = old.id;
143     RETURN old;
144 END;
145 \end{gathered}\end{equation} LANGUAGE plpgsql;
146
147 CREATE OR REPLACE TRIGGER group_delete_trigger
148     INSTEAD OF DELETE
149     ON groups_view
150     FOR EACH ROW
151 EXECUTE PROCEDURE group_remove();
152 init.sql
```

```
153 CREATE TABLE IF NOT EXISTS university
154 (
155   id    SERIAL PRIMARY KEY,
156   name VARCHAR(100) NOT NULL UNIQUE
157 );
158
159 CREATE TABLE IF NOT EXISTS faculty
160 (
161   id             SERIAL PRIMARY KEY,
162   university_id INT NOT NULL,
163   number         INT NOT NULL UNIQUE
164 );
165
166 CREATE TABLE IF NOT EXISTS uni_group
167 (
168   id           SERIAL PRIMARY KEY,
169   faculty_id INT          NOT NULL,
170   name        VARCHAR(50) NOT NULL UNIQUE
171 );
172
173 CREATE TABLE IF NOT EXISTS student
174 (
175   id         SERIAL PRIMARY KEY,
176   group_id INT NOT NULL,
177   name      VARCHAR(100)
178 );
179
180 CREATE TABLE IF NOT EXISTS conference
181 (
182   id     SERIAL PRIMARY KEY,
183   name  VARCHAR(100) NOT NULL DEFAULT 'Научная конференция',
184   place VARCHAR(100) NOT NULL DEFAULT 'ГУАП',
185   theme VARCHAR(255) NOT NULL
186 );
187
188 CREATE TABLE IF NOT EXISTS conference_session
189 (
190   id             SERIAL PRIMARY KEY,
191   conference_id INT       NOT NULL,
192   start_time     TIMESTAMP NOT NULL,
193   end_time       TIMESTAMP NOT NULL
```

```
194 );
195
196 CREATE TABLE IF NOT EXISTS topic
197 (
198   id          SERIAL PRIMARY KEY,
199   title       VARCHAR(255) NOT NULL,
200   session_id  INT          NOT NULL
201 );
202
203 CREATE TABLE IF NOT EXISTS authorship
204 (
205   author_id INT NOT NULL,
206   topic_id  INT NOT NULL,
207   PRIMARY KEY (author_id, topic_id)
208 );
209 after.sql
210 USE conference_db_lab1;
211
212 CREATE TRIGGER after_university_insert
213   AFTER INSERT
214   ON university
215   FOR EACH ROW
216 BEGIN
217   INSERT INTO faculty(university_id, number)
218     VALUES (NEW.id, 99);
219 END;
220
221 CREATE TRIGGER after_uni_group_update
222   AFTER UPDATE
223   ON uni_group
224   FOR EACH ROW
225 BEGIN
226   UPDATE student
227   SET name = CONCAT(name, ' (', NEW.name, ')')
228     WHERE group_id = NEW.id;
229 END;
```