Изменение данных, Представления Процедуры и Триггеры

Базы данных

Юдинцев В. В.

Кафедра математических методов в экономике

20 апреля 2022 г.

САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

Содержание

INSERT

Шаблон оператора INSERT

- Оператор **INSERT** предназначен для добавления строк в таблицу
- Вставка одной строки:

```
INSERT INTO ИМЯ_ТАБЛИЦЫ
  (ИМЯ_СТОЛБЦА_1, ИМЯ_СТОЛБЦА_2, ИМЯ_СТОЛБЦА_3, ...)
VALUES
  (ЗНАЧЕНИЕ_1, ЗНАЧЕНИЕ_2, ЗНАЧЕНИЕ_3, ...);
```

- Вставка нескольких новых строк:

```
INSERT INTO ИМЯ_ТАБЛИЦЫ
  (ИМЯ_СТОЛБЦА_1, ИМЯ_СТОЛБЦА_2, ИМЯ_СТОЛБЦА_3, ...)
VALUES
  (ЗНАЧЕНИЕ_11, ЗНАЧЕНИЕ_12, ЗНАЧЕНИЕ_13, ...),
  (ЗНАЧЕНИЕ_21, ЗНАЧЕНИЕ_22, ЗНАЧЕНИЕ_23, ...),
  (ЗНАЧЕНИЕ_31, ЗНАЧЕНИЕ_32, ЗНАЧЕНИЕ_33, ...);
```

Пример

Таблица **students** состоит из трех столбцов: идентификатор (**id**), имя (**name**), номер зачетной книжки (**grade_book**):

```
CREATE TABLE students (
  id BIGINT UNSIGNED PRIMARY KEY NOT NULL,
  name VARCHAR(100),
  grade_book CHAR(10)
);
```

Вставка новой строки в таблицу **students**:

```
INSERT INTO students
  (id, name, grade_book)
VALUES
  (1, 'Раскольников Р. Р.', '1860-00044');
```

AUTO_INCREMENT

Если в таблице содержится поле с атрибутом **AUTO_INCREMENT**, то значение этого поля можно не указывать.

```
CREATE TABLE students (
  id BIGINT UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
  name VARCHAR(100),
  grade_book CHAR(10)
);
```

Вставка нескольких строк в таблицу **students**:

```
INSERT INTO students (name, grade_book)
VALUES
  ('Раскольников Р. Р.', '1860-00044'),
  ('Разумихин Д. С.', '1860-00046');
```

Таблица **students**

```
id|name                |grade_book|
--|--------------------|----------|
 1|Раскольников Р. Р.  |1860-00044|
 2|Разумихин Д. С.     |1860-00046|
```

INSERT INTO ... SELECT

Новые строки могут быть вставлены из другой таблицы. В этом случае **VALUES** заменяется на **SELECT**. Например, для вставки информации о новых студентах из таблицы абитуриентов может использоваться следующий SQL-сценарий:

```
INSERT INTO students (name)
  SELECT
    name
  FROM
    enrollees;
```

UPDATE

Шаблон оператора UPDATE

- Оператор **UPDATE** используется для обновления данных в таблице
- Шаблон

```
UPDATE
  ИМЯ_ТАБЛИЦЫ
SET
  СТОЛБЕЦ_1 = ЗНАЧЕНИЕ, СТОЛБЕЦ_2 = ЗНАЧЕНИЕ
WHERE
  УСЛОВИЕ
```

Пример

Изменение номера зачетной книжки с 1860-00046 на 1860-00010:

```
UPDATE
  students
SET
  grade_book = '1860-00010'
WHERE
  STRCMP(grade_book,'1860-00046') = 0;
```

Функция RELPACE

Использование функции **REPLACE** для замены части строки столбца:

```
REPLACE(СТРОКА, ЧТО_НАЙТИ, НА_ЧТО_ЗАМЕНИТЬ)
```

Исправить номера зачетных книжек с 1860-***** на 1861-*****:

```
UPDATE
  students
SET
  grade_book = REPLACE(grade_book,'1860','1861')
WHERE
  grade_book LIKE '%1860%';
```

DELETE

Шаблон оператора DELETE

- Оператор **DELETE** предназначен для удаления строк из таблицы
- Шаблон

```
DELETE FROM
   ИМЯ_ТАБЛИЦЫ
WHERE
   УСЛОВИЯ
```

Пример

Удаление записей с заданным значением поля **id**:

```
DELETE FROM
  students
WHERE
  id = 1;
```

Удаление записей с заданным значением строкового поля, использую функцию **STRCMP**:

```
DELETE FROM
  students
WHERE
  STRCMP(grade_boook,'1860-00001') = 0;
```

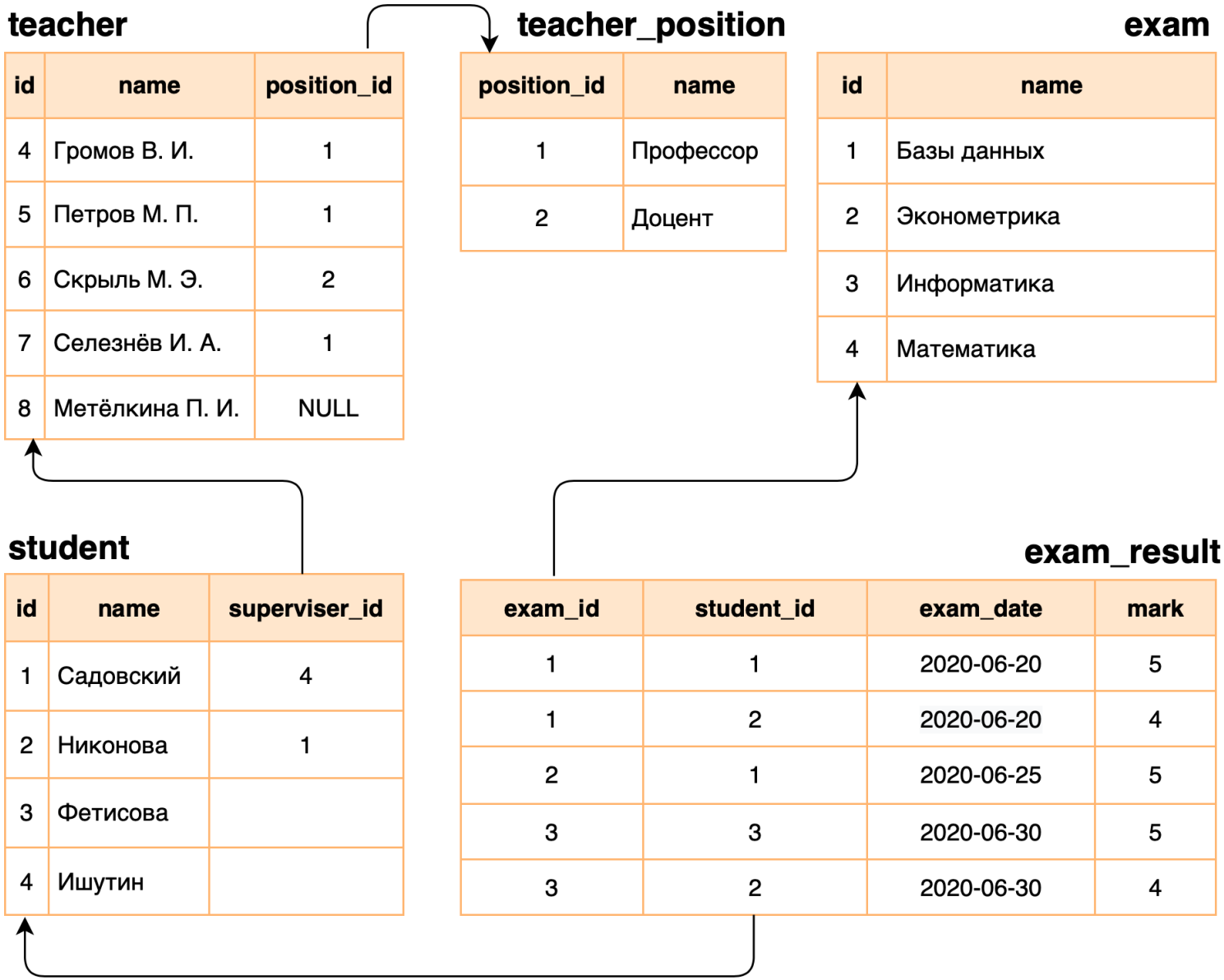
VIEW

Представление это...

- Представление **VIEW** это поименованный запрос (**SELECT**)
- Представление можно рассматривать как виртуальную таблицу, которая формируются из данных других таблиц при обращении к представлению
- Шаблон

```
CREATE VIEW
  ИМЯ_ПРЕДСТАВЛЕНИЯ
AS
  ВЫРАЖЕНИЕ_с_SELECT
```

Пример базы данных



teacher

id	name	position_id
4	Громов В. И.	1
5	Петров М. П.	1
6	Скрыль М. Э.	2
7	Селезнёв И. А.	1
8	Метёлкина П. И.	NULL

teacher_position

position_id	name
1	Профессор
2	Доцент

exam

id	name
1	Базы данных
2	Эконометрика
3	Информатика
4	Математика

student

id	name	superviser_id
1	Садовский	4
2	Никонова	1
3	Фетисова	
4	Ишутин	

exam_result

exam_id	student_id	exam_date	mark
1	1	2020-06-20	5
1	2	2020-06-20	4
2	1	2020-06-25	5
3	3	2020-06-30	5
3	2	2020-06-30	4

Запрос к двум таблицам

Запрос, возвращающий список преподавателей с наименованиями должностей

```
SELECT
  name as teacher_name,
  position.name as position_name
FROM
  teacher
LEFT JOIN
  teacher_position ON teacher_position.id = teacher.position_id
```

Результат выполнения запроса

```
teacher_name    |position_name|
----------------|-------------|
Громов В. И.    |Профессор    |
Петров М. П.    |Профессор    |
Скрыль М. Э.    |Доцент       |
Селезнёв И. А.  |Профессор    |
Метелкина П. И.|             |
```

Представление на основе запроса

Создание представления **teacher_position_name**:

```
CREATE VIEW
  teacher_position_name (teacher_name, position_name)
AS SELECT
     teacher.name, teacher_position.name
   FROM
     teacher
   LEFT JOIN
     teacher_position
   ON teacher_position.id = teacher.position_id;
```

Представление как виртуальная таблица

Для оператора **SELECT** Представление – это обычная таблица

```
SELECT * FROM teacher_position;
```

Результат:

```
teacher_name    |position_name|
----------------|-------------|
Громов В. И.    |Профессор    |
Петров М. П.    |Профессор    |
Скрыль М. Э.    |Доцент       |
Селезнёв И. А.  |Профессор    |
Метелкина П. И.|             |
```

Одно из назначений Представления – возможность скрытия часто используемых сложных запросов, собирающих данные из нескольких таблиц.

Пример

Создание представления **exam_progress**, формирующее список экзаменов с количеством студентов, которые сдали каждый экзамен:

```
CREATE VIEW
  exam_progress
AS SELECT
     exam.title as exam_title,
     COUNT(exam_result.student_id) as count_student
   FROM
     exam_result
   LEFT JOIN exam
     ON exam.id = exam_result.exam_id
   GROUP BY
     exam.title
   ORDER BY
     count_student DESC;
```

Пример

Использование представления

```
SELECT
  *
FROM
  exam_progress
ORDER BY
  count_student ASC;
```

Результат

```
exam_title  |count_student|
------------|-------------|
Эконометрика|            1|
Базы данных |            2|
Информатика |            2|
```

Удаление Представления

- Удалить представление **exam_progress** (должно существовать)

```
DROP VIEW exam_progress;
```

- Удалить, если существует (если представления нет, то не возникнет сообщения об ошибке)

```
DROP VIEW IF EXISTS exam_progress;
```

Использование

- Скрытие сложности структуры базы данных, ”сборка”данных после нормализации
- Повышение безопасности: пользователю открывается доступ к ограниченному набору данных из таблицы

Процедуры и функции

Процедура

- Процедура подобна подпрограмме в языках программирования общего назначения
- Процедуры, как и Представления – это именованые наборы выражений языка SQL
- Процедура компилируется один раз и хранится на сервере
- В отличие от Представления, в Процедуру можно передать параметры
- Процедура может принимать переменные, возвращать результаты или изменять переменные и возвращать их
- Процедура вызывается при помощи команды **CALL**

Функция

- Функция – это именованые блоки выражений языка SQL
- В отличие от Процедуры, Функция возвращает одно значение и может использоваться в SQL-выражениях также, как встроенные функции (STRCMP, FORMAT_DATE, ...)

Шаблон

```
DELIMITER $$

CREATE PROCEDURE ИМЯ_ПРОЦЕДУРЫ(СПИСОК_ПАРАМЕТРОВ)
BEGIN
    ВЫРАЖЕНИЕ 1;
    ВЫРАЖЕНИЕ 2;
    ...
END $$

DELIMITER ;
```

Разделитель

- Процедура – составное выражение, которое может включать из нескольких SQL-выражений, каждое из которых заканчивается точкой с запятой.
- Для объявления процедуры, как составного выражения, необходимо определить символ окончания этого составного выражения, который должен отличаться от символа, которые разделяет SQL-выражения внутри самой процедуры.
- Для переопределения разделителя используется директива DELIMITER.

```
DELIMITER $$
```

Определен символ-разделитель **$$**, которым закончится определение Процедуры или Функции

Создание Процедуры

Определяется временный символ-разделитель:

```
DELIMITER $$
```

Объявление процедуры с именем **GetTeachers** без аргументов:

```
CREATE PROCEDURE GetTeachers()
BEGIN
  SELECT
     teacher.name, teacher_position.name
  FROM
     teacher
  LEFT JOIN teacher_position
  ON teacher_position.id = teacher.position_id;
END $$
```

Определение процедуры заканчивается символом **$$**, который обозначает окончание составного выражения – определения процедуры. После завершения определения процедуры символом-разделителем снова становится точка с запятой:

```
DELIMITER ;
```

Вызов процедуры

```
CALL GetTeachers();

teacher_name    |position_name|
----------------|-------------|
Громов В. И.    |Профессор    |
Петров М. П.    |Профессор    |
Скрыль М. Э.    |Доцент       |
Селезнёв И. А.  |Профессор    |
Метелкина П. И.|             |
```

Процедура с аргументами

В скобках после имени процедуры перечисляются входные (**IN**) аргументы через запятую с указанием их типов:

```
DELIMITER $$

CREATE PROCEDURE ИМЯ_ПРОЦЕДУРЫ(IN имя1 тип1, IN имя2 тип2)
BEGIN
   ВЫРАЖЕНИЕ 1;
   ВЫРАЖЕНИЕ 2;
   ...
END $$

DELIMITER ;
```

Пример

```
DELIMITER $$

CREATE PROCEDURE GetStudentBetterThan(IN mark FLOAT)
BEGIN
  SELECT
    student.name, AVG(exam_result.mark) as mean_mark
  FROM
    exam_result
  LEFT JOIN student
    ON student.id = exam_result.student_id
  GROUP BY
    student.name
  HAVING mean_mark > mark;
END $$

DELIMITER ;
```

Вызов процедуры с аргументами

Список студентов со средней оценкой больше 4,5:

```
CALL GetStudentBetterThan(4.5);
```

Результат

```
name      |mean_mark|
----------|---------|
Садовский|   5.0000|
Фетисова |   5.0000|
```

Процедура с возвращаемым аргументом

Процедура возвращает среднее значение оценок полученных на экзамене, наименование которого указанного первым аргументом

```
DELIMITER $$;
CREATE PROCEDURE
GetMeanExamMark(IN exam_title VARCHAR(50), OUT mean_mark FLOAT)
BEGIN
  SELECT
    AVG(exam_result.mark)
  FROM
    exam_result
  LEFT JOIN exam
    ON exam.id = exam_result.exam_id
  WHERE
    STRCMP(exam.title,exam_title) = 0
  GROUP BY
    exam.id
  INTO mean_mark;
END $$
DELIMITER ;
```

Пример

Средняя оценка по экзамену Базы данных:

```
CALL GetMeanExamMark('Базы данных',@res);
```

Результат записывается в переменную **@res**. Значение переменной

```
SELECT @res;
```

Шаблон определения функции

```
CREATE FUNCTION имя_функции(агрумент1 тип1, аргумент2 тип2)
  RETURNS тип_возвращаемого_значения
    BEGIN
    ...
    ...
    RETURN значение;
    END
```

Пример. Объявление функции

Перевод оценки из пятибальной шкалы в 100-бальную:

```
DELIMITER $$

CREATE FUNCTION SCALE_MARK_100(mark FLOAT, max_mark FLOAT)
  RETURNS FLOAT
BEGIN
  DECLARE v1 FLOAT;
  SET v1 = 100*mark/max_mark;
  RETURN v1;
END $$

DELIMITER ;
```

Пример. Использование функции

```
SELECT
  exam.title as exam_title,
  student.name as student_name,
  SCALE_MARK_100(exam_result.mark,5) as mark100
FROM
  exam_result
LEFT JOIN exam
  ON exam.id = exam_result.exam_id
LEFT JOIN student
  ON exam_result.student_id = student.id
ORDER BY
  exam_title;
```

```
exam_title  |student_name|mark100|
------------|------------|-------|
Базы данных |Садовский   |    100|
Базы данных |Никонова    |     80|
Информатика |Никонова    |     80|
Информатика |Фетисова    |    100|
Эконометрика|Садовский   |    100|
```

Пример. Без использования функции

Можно формировать математические выражения со значениями столбцов

```
SELECT
  exam.title as exam_title,
  student.name as student_name,
  100*exam_result.mark/5 as mark100
FROM
  exam_result
LEFT JOIN exam
  ON exam.id = exam_result.exam_id
LEFT JOIN student
  ON exam_result.student_id = student.id
ORDER BY
  exam_title;
```

IF ... THEN ... ELSE

Функция возвращает «Зачёт», если оценка больше или равна заданной границы (второй аргумент) и «Незачёт», если оценка меньше границы или отсутсвует:

```
DELIMITER $$;

CREATE FUNCTION PassFail(mark FLOAT,margin FLOAT) RETURNS CHAR(7)
BEGIN
  DECLARE RES CHAR(7);
  IF (mark < margin OR mark is NULL) THEN
    RETURN 'Незачет';
  ELSE
    RETURN 'Зачет';
  END IF;
END $$

DELIMITER ;
```

Использование функции PassFail

```
SELECT
  student.name as student_name,
  exam.title as exam_title,
  PassFail(exam_result.mark,3.5) as PF
FROM
  exam_result
LEFT JOIN exam
  ON exam.id = exam_result.exam_id
RIGHT JOIN student
  ON exam_result.student_id = student.id
ORDER BY
  exam_title;
```

student_name	exam_title	PF
Ишутин		Незачет
Садовский	Базы данных	Зачет
Никонова	Базы данных	Зачет
Фетисова	Информатика	Зачет
Никонова	Информатика	Зачет
Садовский	Эконометрика	Зачет

Альтернативный вариант

```
SELECT
  student.name as student_name,
  exam.title as exam_title,
  IF(exam_result.mark>=3.5,'Зачёт','Незачёт') as PF
FROM
  exam_result
LEFT JOIN exam
  ON exam.id = exam_result.exam_id
RIGHT JOIN student
  ON exam_result.student_id = student.id
ORDER BY
  exam_title;
```

```
student_name|exam_title  |PF      |
------------|------------|--------|
Ишутин      |            |Незачет|
Садовский   |Базы данных |Зачет   |
Никонова    |Базы данных |Зачет   |
Фетисова    |Информатика |Зачет   |
Никонова    |Информатика |Зачет   |
Садовский   |Эконометрика|Зачет   |
```

Удаление процедуры или функции

```
DROP PROCEDURE имя_процедуры;

DROP PROCEDURE IF EXISTS имя_процедуры;

DROP FUNCTION имя_функции;

DROP FUNCTION IF EXISTS имя_функции;
```

Триггеры

Триггеры это ...

- Хранимая процедура, которая исполняется при наступлении определенного события
- Триггер связывается с таблицей, на события в которой он реагирует
- Триггер может срабатывать при следующих изменениях в таблице (тип триггера):
  - `BEFORE INSERT, AFTER INSERT`
  - `BEFORE UPDATE, AFTER UPDATE`
  - `BEFORE DELETE, AFTER DELETE`

Шаблон

```
DELIMITER $$
CREATE TRIGGER имя_триггера тип_триггера ON таблица
  FOR EACH ROW
  BEGIN
    выражение1;
    выражение2;
  END; $$
DELIMITER ;
```

Пример

При вставке новой записи в таблицу **students** необходимо генерировать автоматически номер зачетной книжки, который состоит из года поступления и идентификатора студента

```
CREATE TABLE students (
  id BIGINT UNSIGNED DEFAULT 0 PRIMARY KEY,
  name VARCHAR(50),
  grade_book CHAR(10),
  enrollment_date DATE
);
```

Пример триггера BEFORE UPDATE

Триггер срабатывает перед обновлением таблицы, и формирует значение для атрибута **grade_book**:

```
DELIMITER $$

CREATE TRIGGER ON_UPDATE_STUDENT BEFORE UPDATE ON students
  FOR EACH ROW
  BEGIN
    SET NEW.grade_book = CONCAT(DATE_FORMAT(NEW.enrollment_date,
    ↪  "%Y-"),LPAD(NEW.id, 5, 0));
  END; $$

DELIMITER ;
```

Фукнция CONCAT

Функция **CONCAT** склеивает две строки

```
SELECT CONCAT('Строка1-','Строка2') AS res;
```

```
res              |
-----------------|
Строка1-Строка2|
```

Функция FORMAT

Функция **DATE_FORMAT** возвращает тектовое представление даты в соотетсвии с заданным форматом:

```
DATE_FORMAT('2020-04-26','%Y')          => 2020
DATE_FORMAT('2020-04-26','%d.%m.%Y')    => 26.04.2020
DATE_FORMAT('2020-04-26','%d.%m.%y')    => 26.04.20
```

https://www.w3schools.com/sql/func_mysql_date_format.asp

Функция LPAD

Функция **LPAD** дополняет заданную строку слева символами до заданной длины строки

```
SELECT LPAD('TEXT',10,'*');

LPAD('TEXT',10,'*')|
-------------------|
******TEXT         |
```

Пример триггера BEFORE INSERT

```
DELIMITER $$

CREATE TRIGGER ON_NEW_STUDENT BEFORE INSERT ON students
  FOR EACH ROW
  BEGIN
    SELECT MAX(id) FROM students into @lastid;
    SET NEW.id = COALESCE(@lastid + 1, 1);
    IF NEW.enrollment_date IS NOT NULL THEN
      SET NEW.grade_book = CONCAT(DATE_FORMAT(NEW.enrollment_date,
      ↪  "%Y-"),LPAD(NEW.id, 5, 0));
    END IF;
  END; $$

DELIMITER ;
```

До вставки записи (в триггере **BEFORE INSERT**) значение автоикрементного поля еще неизвестно, поэтому **id** студента объявлено не как **AUTO_INCREMENT** и новое значение **id** определяется внутри триггера

Пример

```
INSERT INTO students (name, enrollment_date)
   values('Лопотухин В. Д','2016-10-20');

INSERT INTO students (name, enrollment_date)
   values('Малахова  О. С','2016-10-21');
```

Результат:

```
SELECT * FROM students;

id|name           |grade_book|enrollment_date|
--|---------------|----------|---------------|
 1|Лопотухин В. Д|2016-00001|     2016-10-20|
 2|Малахова  О. С|2016-00002|     2016-10-24|
```

Удаление триггера

```
DROP TRIGGER имя_триггера;
DROP TRIGGER IF EXISTS имя_триггера;
```

Список использованных источников

- SQL Учебник

  `https://schoolsw3.com/sql/index.php`

- SQL Tutorial

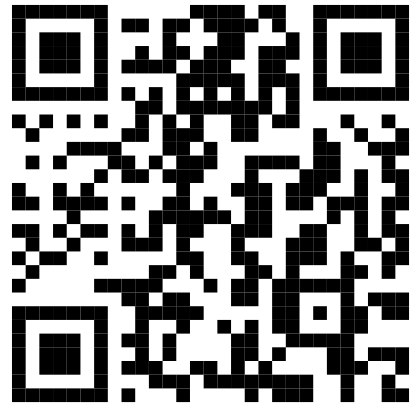
  `https://www.tutorialspoint.com/sql/index.htm`

- MySQL 8.0 Reference Manual

  `https://dev.mysql.com/doc/refman/8.0/en/introduction.html`

- Basic MySQL Tutorial

  `https://www.mysqltutorial.org/basic-mysql-tutorial.aspx`



https://classmech.ru/pages/databases/main