

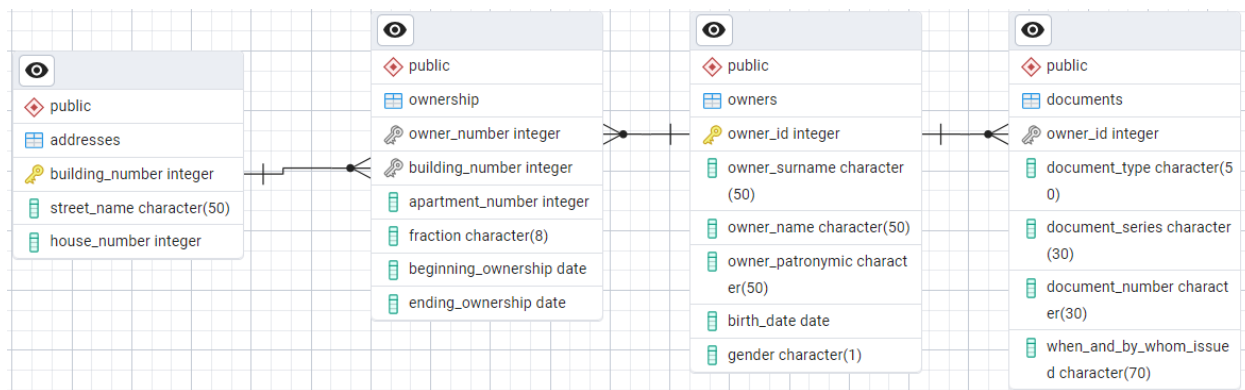
ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

ВАРИАНТ 7. СОБСТВЕННИКИ КВАРТИР.

HW-1. Анализ Subject Area, концептуальное проектирование БД.

HW-2. Логическое проектирование БД.

HW-3. Физическое проектирование, создание и заполнение отношений БД.



1 **SELECT** * **FROM** public.ownership

Data Output Сообщения Notifications

	owner_number integer	building_number integer	apartment_number integer	fraction character (8)	beginning_ownership date	ending_ownership date
1	1	1	10	1	2015-01-10	[null]
2	2	1	15	1/2	2016-05-20	[null]
3	3	2	20	1	2017-03-15	2020-12-31
4	4	3	25	1/3	2018-07-10	[null]
5	5	4	30	1	2019-09-05	[null]
6	1	2	35	1/4	2020-02-15	[null]
7	2	3	40	1	2021-04-20	[null]

1 **SELECT** * **FROM** public.owners

Data Output Сообщения Notifications

	owner_id [PK] integer	owner_surname character (50)	owner_name character (50)	owner_patronymic character (50)	birth_date date	gender character (1)
1	1	Иванов	Иван	Иванович	2008-05-15	м
2	2	Петрова	Мария	Сергеевна	1975-08-22	ж
3	3	Сидоров	Алексей	[null]	1990-03-10	м
4	4	Кузнецова	Елена	Владимировна	1988-11-05	ж
5	5	Смирнов	Дмитрий	Александрович	1972-07-30	м

1 **SELECT** * **FROM** public.documents

Data Output Сообщения Notifications

	owner_id integer	document_type character (50)	document_series character (30)	document_number character (30)	when_and_by_whom_issued character (70)
1	1	Паспорт ...	4501 ...	123456	ОВД Центрального района г. Москвы 2010-05-2...
2	2	Паспорт ...	4502 ...	654321	ОВД Северного района г. Санкт-Петербурга 201...
3	3	Паспорт ...	4503 ...	789012	ОВД Западного района г. Казани 2018-03-15 ...
4	4	Паспорт ...	4504 ...	345678	ОВД Восточного района г. Новосибирска 2017-...
5	5	Паспорт ...	4505 ...	901234	ОВД Южного района г. Екатеринбурга 2016-11-...
6	1	Паспорт ...	4501 ...	123456	ОВД Центрального района г. Москвы 2010-05-2...
7	2	Паспорт ...	4502 ...	654321	ОВД Северного района г. Санкт-Петербурга 201...
8	3	Паспорт ...	4503 ...	789012	ОВД Западного района г. Казани 2018-03-15 ...
9	4	Паспорт ...	4504 ...	345678	ОВД Восточного района г. Новосибирска 2017-...
10	5	Паспорт ...	4505 ...	901234	ОВД Южного района г. Екатеринбурга 2016-11-...

1 **SELECT** * **FROM** public.addresses

Data Output Сообщения Notifications

	building_number [PK] integer	street_name character (50)	house_number integer
1	1	Ленина ...	10
2	2	Пушкина ...	15
3	3	Гагарина ...	20
4	4	Советская ...	25
5	5	Мира ...	30

HW-4. Выборка данных.

Один из запросов надо написать двумя способами и объяснить, какой из вариантов будет работать быстрее и почему.

Создать упорядоченные списки:

- единоличных владельцев помещений с указанием адреса квартиры, которой он владеет;
- собственников с перечнем их;
- квартир, у которых более 10 собственников;
- текущих собственников квартир по улице Чкалова;
- квартир, у которых в настоящее время нет собственников.

-- Упорядоченный список единоличных владельцев помещений (доля которых равна 1) с указанием адреса

-- квартиры, которой он владеет

-- 1 способ

```
CREATE OR REPLACE VIEW sole_owners AS
```

```
SELECT o.owner_id,
```

```
       o.owner_surname || ' ' || o.owner_name || COALESCE(' ' ||  
o.owner_patronymic, '') AS full_name,
```

```
       a.street_name || ', д.' || a.house_number || ', кв.' || ow.apartment_number  
AS full_address
```

```
FROM ownership ow
```

```
JOIN owners o ON ow.owner_number = o.owner_id
```

```
JOIN addresses a ON ow.building_number = a.building_number
```

```
WHERE ow.fraction = '1' AND (ow.ending_ownership IS NULL OR  
ow.ending_ownership > CURRENT_DATE)
```

```
ORDER BY o.owner_surname, o.owner_name;
```

-- 2 способ

```
CREATE OR REPLACE VIEW sole_owners_subquery AS
```

```

SELECT o.owner_id,
       o.owner_surname || ' ' || o.owner_name || COALESCE(' ' ||
o.owner_patronymic, '') AS full_name,
       (SELECT a.street_name || ' д.' || a.house_number || ' кв.' ||
ow.apartment_number
       FROM addresses a
       WHERE a.building_number = ow.building_number) AS full_address
FROM ownership ow
JOIN owners o ON ow.owner_number = o.owner_id
WHERE ow.fraction = '1' AND (ow.ending_ownership IS NULL OR
ow.ending_ownership > CURRENT_DATE)
ORDER BY o.owner_surname, o.owner_name;

```

```
SELECT * FROM sole_owners;
```

```
SELECT * FROM sole_owners_subquery;
```

-- Вывод:

-- Первый способ с join быстрее, так как второй способ с подзапросом
проходится по каждой строке

	owner_id integer	full_name text	full_address text
1	1	Иванов Иван Иванович	Ленина, д.10, кв.10
2	2	Петрова Мария Сергеевна	Гагарина, д.20, кв.40
3	5	Смирнов Дмитрий Александрович	Советская, д.25, кв.30

-- Упорядоченный список собственников с перечнем их документов

-- (в том числе тех, у которых нет документов);

```
CREATE OR REPLACE VIEW owners_with_documents AS
```

```

SELECT o.owner_id,
       o.owner_surname || ' ' || o.owner_name || COALESCE(' ' ||
o.owner_patronymic, '') AS full_name,

```

```

STRING_AGG(
    COALESCE(d.document_type || ' ' || d.document_series || ' ' ||
d.document_number, 'Нет документов'),
    ','
) AS documents_list
FROM owners o
LEFT JOIN documents d ON o.owner_id = d.owner_id
GROUP BY o.owner_id, full_name
ORDER BY o.owner_surname, o.owner_name;

```

```
SELECT * FROM owners_with_documents;
```

	owner_id integer	full_name text	documents_list text
1	1	Иванов Иван Иванович	Паспорт 4501 123456, Паспорт 4501 123456
2	4	Кузнецова Елена Владимировна	Паспорт 4504 345678, Паспорт 4504 345678
3	2	Петрова Мария Сергеевна	Паспорт 4502 654321, Паспорт 4502 654321
4	3	Сидоров Алексей	Паспорт 4503 789012, Паспорт 4503 789012
5	5	Смирнов Дмитрий Александрович	Паспорт 4505 901234, Паспорт 4505 901234

-- Квартиры с более чем 10 собственниками

```

CREATE OR REPLACE VIEW apartments_with_many_owners AS
SELECT  a.street_name || ' д.' || a.house_number || ' кв.' ||
ow.apartment_number AS full_address,
        COUNT(ow.owner_number) AS owners_count
FROM ownership ow
JOIN addresses a ON ow.building_number = a.building_number
        WHERE  ow.ending_ownership IS NULL OR ow.ending_ownership >
CURRENT_DATE
GROUP BY a.street_name, a.house_number, ow.apartment_number
HAVING COUNT(ow.owner_number) > 10

```

ORDER BY owners_count DESC;

SELECT * FROM apartments_with_many_owners;

full_address	owners_count
text	bigint

-- Текущие собственники квартир на улице Чкалова

CREATE OR REPLACE VIEW pushkina_street_owners AS

SELECT o.owner_id,

o.owner_surname || ' ' || o.owner_name || ' ' || o.owner_patronymic AS full_name,

a.house_number || ', кв.' || ow.apartment_number AS address,

ow.fraction AS ownership_share

FROM ownership ow

JOIN owners o ON ow.owner_number = o.owner_id

JOIN addresses a ON ow.building_number = a.building_number

WHERE a.street_name = 'Пушкина'

AND (ow.ending_ownership IS NULL OR ow.ending_ownership > CURRENT_DATE)

ORDER BY a.house_number, ow.apartment_number, o.owner_surname;

SELECT * FROM pushkina_street_owners;

	owner_id integer	full_name text	address text	ownership_share character (8)
1	1	Иванов Иван Иванович	15, кв.35	1/4

-- Квартиры, у которых в настоящее время нет собственников

CREATE OR REPLACE VIEW apartments_without_owners AS

SELECT a.street_name || ', д.' || a.house_number || ', кв.' || ow.apartment_number AS full_address

```

FROM ownership ow
JOIN addresses a ON ow.building_number = a.building_number
WHERE NOT EXISTS (
    SELECT 1 FROM ownership ow2
    WHERE ow2.building_number = ow.building_number
    AND ow2.apartment_number = ow.apartment_number
    AND (ow2.ending_ownership IS NULL OR ow2.ending_ownership >
CURRENT_DATE)
);

SELECT * FROM apartments_without_owners;

```

	full_address text	
1	Пушкина, д.15, кв.20	

HW-5. Работа с представлениями.

Для созданных представлений необходимо проверить с помощью запросов UPDATE,

DELETE и INSERT, являются ли они обновляемыми, и объяснить полученный результат.

1. Представление "Квартиры, в числе собственников которых в настоящее время есть

несовершеннолетние дети".

2. Представление "Количество собственников по домам": номер здания – улица – номер

дома – количество текущих собственников.

3. Представление "Количество текущих собственников" по всем квартирам, включая

квартиры, у которых нет собственников.

-- 1. Квартиры с несовершеннолетними собственниками

```

CREATE OR REPLACE VIEW apartments_with_minors AS
SELECT DISTINCT
    a.building_number,
    a.street_name,
    a.house_number,
    ow.apartment_number
FROM ownership ow
JOIN owners o ON ow.owner_number = o.owner_id
JOIN addresses a ON ow.building_number = a.building_number
WHERE (ow.ending_ownership IS NULL OR ow.ending_ownership >
CURRENT_DATE)
    AND AGE(o.birth_date) < INTERVAL '18 years'
ORDER BY a.street_name, a.house_number, ow.apartment_number;

SELECT * FROM apartments_with_minors;

```

	building_number integer	street_name character (50)	house_number integer	apartment_number integer
1	1	Ленина	10	10
2	2	Пушкина	15	35

-- Проверка обновляемости

-- INSERT (не работает, так как представление объединяет несколько таблиц)

```

INSERT INTO apartments_with_minors (building_number, street_name,
house_number, apartment_number)
VALUES (6, 'Новая', 5, 10);

```

ERROR: Представления с DISTINCT не обновляются автоматически. вставить данные в представление "apartments_with_minors" нельзя

ОШИБКА: вставить данные в представление "apartments_with_minors" нельзя

SQL-состояние: 55000

Подробности: Представления с DISTINCT не обновляются автоматически.

Подсказка: Чтобы представление допускало добавление данных, установите триггер INSTEAD OF INSERT или безусловное правило ON INSERT DO INSTEAD.

-- UPDATE (не работает, так как представление объединяет несколько таблиц)

```
UPDATE apartments_with_minors SET apartment_number = 11 WHERE building_number = 1;
```

ERROR: Представления с DISTINCT не обновляются автоматически.изменить данные в представлении "apartments_with_minors" нельзя

ОШИБКА: изменить данные в представлении "apartments_with_minors" нельзя

SQL-состояние: 55000

Подробности: Представления с DISTINCT не обновляются автоматически.

Подсказка: Чтобы представление допускало изменение данных, установите триггер INSTEAD OF UPDATE или безусловное правило ON UPDATE DO INSTEAD.

-- DELETE (не работает, так как представление объединяет несколько таблиц)

```
DELETE FROM apartments_with_minors WHERE building_number = 1;
```

ERROR: Представления с DISTINCT не обновляются автоматически.удалить данные из представления "apartments_with_minors" нельзя

ОШИБКА: удалить данные из представления "apartments_with_minors" нельзя

SQL-состояние: 55000

Подробности: Представления с DISTINCT не обновляются автоматически.

Подсказка: Чтобы представление допускало удаление данных, установите триггер INSTEAD OF DELETE или безусловное правило ON DELETE DO INSTEAD.

-- 2. Количество собственников по домам

```
CREATE OR REPLACE VIEW owners_per_building AS
```

```
SELECT
```

```
    a.building_number,
```

```
    a.street_name,
```

```
    a.house_number,
```

```
    COUNT(DISTINCT ow.owner_number) AS owners_count
```

```
FROM ownership ow
```

```
JOIN addresses a ON ow.building_number = a.building_number
```

```
WHERE ow.ending_ownership IS NULL OR ow.ending_ownership >
CURRENT_DATE
```

```
GROUP BY a.building_number, a.street_name, a.house_number
```

```
ORDER BY a.street_name, a.house_number;
```

```
SELECT * FROM owners_per_building;
```

	building_number integer	street_name character (50)	house_number integer	owners_count bigint
1	3	Гагарина ...	20	2
2	1	Ленина ...	10	2
3	2	Пушкина ...	15	1
4	4	Советская ...	25	1

-- Проверка обновляемости

-- INSERT (не работает из-за соединения нескольких таблиц, GROUP BY, COUNT, DISTINCT)

```
INSERT INTO owners_per_building (building_number, street_name, house_number)
```

```
VALUES (6, 'Новая', 5); -- Ошибка
```

-- UPDATE (не работает из-за соединения нескольких таблиц, GROUP BY, COUNT, DISTINCT)

```
UPDATE owners_per_building SET house_number = 6 WHERE building_number = 1; -- Ошибка
```

-- DELETE (не работает из-за соединения нескольких таблиц, GROUP BY, COUNT, DISTINCT)

```
DELETE FROM owners_per_building WHERE building_number = 1; -- Ошибка
```

ERROR: Представления с GROUP BY не обновляются автоматически. вставить данные в представление "owners_per_building" нельзя

ОШИБКА: вставить данные в представление "owners_per_building" нельзя

SQL-состояние: 55000

Подробности: Представления с GROUP BY не обновляются автоматически.

Подсказка: Чтобы представление допускало добавление данных, установите триггер INSTEAD OF INSERT или безусловное правило ON INSERT DO INSTEAD.

-- 3. Количество текущих собственников по квартирам

```
CREATE OR REPLACE VIEW owners_per_apartment AS
```

```
SELECT
```

```
    a.building_number,
```

```
    a.street_name,
```

```

a.house_number,
ow.apartment_number,
COUNT(ow.owner_number) AS owners_count
FROM addresses a
LEFT JOIN ownership ow ON a.building_number = ow.building_number
AND (ow.ending_ownership IS NULL OR ow.ending_ownership >
CURRENT_DATE)
GROUP BY a.building_number, a.street_name, a.house_number,
ow.apartment_number
ORDER BY a.street_name, a.house_number, ow.apartment_number;

SELECT * FROM owners_per_apartment;

```

	building_number integer	street_name character (50)	house_number integer	apartment_number integer	owners_count bigint
1	3	Гагарина ...	20	25	1
2	3	Гагарина ...	20	40	1
3	1	Ленина ...	10	10	1
4	1	Ленина ...	10	15	1
5	5	Мира ...	30	[null]	0
6	2	Пушкина ...	15	35	1
7	4	Советская ...	25	30	1

-- Проверка обновляемости

-- INSERT (не работает из-за LEFT JOIN, GROUP, COUNT)

```

INSERT INTO owners_per_apartment (building_number, street_name,
house_number, apartment_number)
VALUES (6, 'Новая', 5, 10); -- Ошибка

```

-- UPDATE (не работает из-за LEFT JOIN, GROUP, COUNT)

```

UPDATE owners_per_apartment SET apartment_number = 11 WHERE
building_number = 1; -- Ошибка

```

-- DELETE (не работает из-за LEFT JOIN, GROUP, COUNT)

DELETE FROM owners_per_apartment WHERE building_number = 1; -- Ошибка

ERROR: Представления с GROUP BY не обновляются автоматически,изменить данные в представлении "owners_per_apartment" нельзя

ОШИБКА: изменить данные в представлении "owners_per_apartment" нельзя

SQL-состояние: 55000

Подробности: Представления с GROUP BY не обновляются автоматически.

Подсказка: Чтобы представление допускало изменение данных, установите триггер INSTEAD OF UPDATE или безусловное правило ON UPDATE DO INSTEAD.

СОЗДАНИЕ ПРЕДСТАВЛЕНИЙ

Задание: Создайте представления и проверьте с помощью запросов INSERT, UPDATE, DELETE являются ли они модифицируемыми (обновляемыми).

Создайте представления и проверьте с помощью запросов INSERT, UPDATE, DELETE являются ли они модифицируемыми (обновляемыми).

Создание представлений в MySQL. База данных «Книжное дело».

1. Создайте представление, которое показывает код поставки, наименование

книги, дату поставки, наименование поставщика, стоимость поставки,

объем поставки.

2. Создайте представление, которое показывает все сведения об издательствах

из города Москва.

3. Создайте представление, которое показывает код книги, наименование

книги, автора, количество книг на складе, стоимость книг (максимальная

стоимость).

4. Создайте представление, которое показывает топ 5 книг с максимальным

количеством на складе (используйте предыдущее представление).

-- 1. Представление, которое показывает код поставки, наименование книги,
 -- дату поставки, наименование поставщика, стоимость поставки,
 -- объем поставки

```
CREATE OR REPLACE VIEW delivery_info as
SELECT d.Code_delivery as 'Код поставки',
       b.Title_book as 'Наименование книги',
       p.Date_order as 'Дата поставки',
       d.Name_delivery as 'Наименование поставщика',
       p.Cost as 'Стоимость поставки',
       p.Amount as 'Объем поставки'
FROM purchases p
JOIN
  books b ON p.Code_book = b.Code_book
JOIN
  deliveries d ON p.Code_delivery = d.Code_delivery
ORDER BY p.Date_order DESC;
```

Код поставки	Наименование книги	Дата поставки	Наименование поставщика	Стоимость поставки	Объем поставки
1	Труды по истории	2023-10-01	Волков В.В.	300	10
4	Преступление и наказание	2003-06-15	Орлов О.О.	550	80
2	Мемуары	2003-05-14	Соколов С.С.	400	110
5	Евгений Онегин	2003-05-10	Смирнов С.С.	275	140
3	Война и мир	2003-04-12	Кузнецов К.К.	600	50
6	Обломов	2003-03-15	Иванов И.И.	325	160
1	Труды по истории	2003-03-13	Волков В.В.	300	150
9	Герой нашего времени	2002-05-05	Федоров Ф.Ф.	475	190
8	Вишневый сад	2002-01-15	Григорьев Г.Г.	450	180

-- Попытка INSERT

```
INSERT INTO delivery_info VALUES (1, 1, '2023-01-01', 1, 100.0, 5);
```

-- Попытка UPDATE

```
UPDATE delivery_info SET `Стоимость поставки` = 150 WHERE `Код поставки` = 1;
```

-- Попытка DELETE

```
DELETE FROM delivery_info WHERE `Код поставки` = 1;
```

-- Вывод:

-- Представление не является обновляемым, так как содержит JOIN нескольких таблиц

-- 2. представление, которое показывает все сведения об издательствах

-- из города Москва

```
CREATE OR REPLACE VIEW moscow_publishers as
```

```
SELECT
```

```
    Code_publish as 'Код издательства',
```

```
    Publish as 'Издательство',
```

```
    City as 'Город'
```

```
FROM publishing_house
```

```
WHERE City = 'Москва';
```

Код издательства	Издательство	Город
4	Мир	Москва
6	Просвещение	Москва
7	Эксмо	Москва
8	АСТ	Москва
9	Молодая гвардия	Москва
11	Наука	Москва

-- Попытка INSERT

```
INSERT INTO moscow_publishers VALUES (10, 'Новое издательство', 'Москва');
```

-- Попытка UPDATE

```
UPDATE moscow_publishers SET `Издательство` = 'Издательство' WHERE `Код издательства` = 1;
```

-- Попытка DELETE

```
DELETE FROM moscow_publishers WHERE `Код издательства` = 1;
```

-- Вывод:

-- Представление является обновляемым, так как:

-- Основано на одной таблице

-- Не содержит GROUP BY, агрегатных функций или DISTINCT

-- 3. представление, которое показывает код книги, наименование

-- книги, автора, количество книг на складе, стоимость книг (максимальная

-- стоимость)

```
CREATE OR REPLACE VIEW books_inventory as
```

```
SELECT b.Code_book as 'Код книги',
```

```
       b.Title_book as 'Наименование книги',
```

```
       a.Name_author as 'Автор',
```

```
       SUM(p.Amount) as 'Количество на складе',
```

```
       MAX(p.Cost) as 'Максимальная стоимость'
```

```
FROM books b
```

```
JOIN
```

```
  authors a ON b.Code_author = a.Code_author
```

```
LEFT JOIN
```

```
  purchases p ON b.Code_book = p.Code_book
```

```
GROUP BY b.Code_book, b.Title_book, a.Name_author;
```


Код книги	Наименование книги	Автор	Количество на складе	Максимальная стоимость
1	Труды по истории	Булгаков М.А.	160	300
2	Мемуары	Толстой Л.Н.	110	400
3	Война и мир	Толстой Л.Н.	50	600
4	Преступление и наказание	Достоевский Ф.М.	80	550
5	Евгений Онегин	Пушкин А.С.	140	275
6	Обломов	Тургенев И.С.	160	325
8	Вишневый сад	Лермонтов М.Ю.	180	450
9	Герой нашего времени	Чехов А.П.	190	475
10	Труды по экономике	Горький М.	NULL	NULL
11	Наука. Техника. Инновации	Булгаков М.А.	NULL	NULL
16	Анна Коренина	Толстой Л.Н.	NULL	NULL
17	Новая книга	Булгаков М.А.	NULL	NULL

-- Попытка INSERT

```
INSERT INTO books_inventory VALUES (100, 'Новая книга', 'Новый автор', 10, 500);
```

-- Попытка UPDATE

```
UPDATE books_inventory SET `Количество на складе` = 20 WHERE `Код книги` = 1;
```

-- Попытка DELETE

```
DELETE FROM books_inventory WHERE `Код книги` = 1;
```

-- Вывод:

-- Представление не является обновляемым, так как:

-- Содержит JOIN нескольких таблиц

-- Использует агрегатные функции (SUM, MAX)

-- Содержит GROUP BY

-- 4. представление, которое показывает топ 5 книг с максимальным

-- количеством на складе (использовали предыдущее представление)

```
CREATE OR REPLACE VIEW top5_books as
```

```
SELECT 'Код книги',
```

```

        'Наименование книги',
        'Автор',
        'Количество на складе',
        'Максимальная стоимость'
FROM books_inventory
ORDER BY 'Количество на складе' DESC
LIMIT 5;

```

Код книги	Наименование книги	Автор	Количество на складе	Максимальная стоимость
9	Герой нашего времени	Чехов А.П.	190	475
8	Вишневый сад	Лермонтов М.Ю.	180	450
1	Труды по истории	Булгаков М.А.	160	300
6	Обломов	Тургенев И.С.	160	325
5	Евгений Онегин	Пушкин А.С.	140	275

-- Попытка INSERT

```
INSERT INTO top5_books VALUES (100, 'Новая книга', 'Новый автор', 100, 500);
```

-- Попытка UPDATE

```
UPDATE top5_books SET `Количество на складе` = 20 WHERE `Код книги` = 1;
```

-- Попытка DELETE

```
DELETE FROM top5_books WHERE `Код книги` = 1;
```

-- Вывод:

-- Представление не является обновляемым, так как:

-- Основано на другом представлении

-- Содержит LIMIT

Создание представлений в Microsoft SQL Server. База данных «Успеваемость».

1. Создайте представление, которое показывает преподавателей, не принимающих экзамены.

2. Создайте представление, которое показывает список групп специальности Программирование в компьютерных системах.

3. Создайте представление, которое показывает код студента, ФИО студента, наименование группы, средний балл за весь период обучения.

4. Создайте представление, которое показывает список из 5 студентов с наибольшим средним баллом за весь период обучения.

-- 1. Преподаватели, не принимающие экзамены

```
CREATE OR REPLACE VIEW inactive_lectors AS
```

```
SELECT
```

```
    I."Code_lector" AS "Код преподавателя",
```

```
    I."Name_lector" AS "ФИО преподавателя",
```

```
    I."Science" AS "Научная степень",
```

```
    I."Post" AS "Должность"
```

```
FROM
```

```
    "Lectors" I
```

```
WHERE
```

```
    NOT EXISTS (
```

```
        SELECT 1
```

```
        FROM "Progress" p
```

```
        WHERE p."Code_lector" = I."Code_lector"
```

```
    );
```

-- Просмотр преподавателей без экзаменов

```
SELECT * FROM inactive_lectors;
```

	Код преподавателя integer	ФИО преподавателя character (40)	Научная степень character (25)	Должность character (25)
1	2	Иванов И.И.	Кандидат наук	Доцент
2	100	Иванов Иван Иванович	Кандидат наук	Доцент

-- INSERT (сработало)

```
INSERT INTO inactive_lectors ("Код преподавателя", "ФИО преподавателя",
"Научная степень", "Должность")
VALUES (100, 'Иванов Иван Иванович', 'Кандидат наук', 'Доцент');
```

-- UPDATE (сработало)

```
UPDATE inactive_lectors SET "Должность" = 'Профессор' WHERE "Код
преподавателя" = 1;
```

-- DELETE (сработало)

```
DELETE FROM inactive_lectors WHERE "Код преподавателя" = 1;
```

-- Вывод: Обновляемое, так как:

-- Выборка из одной таблицы

-- Нет GROUP BY, агрегатных функций, DISTINCT

-- 2. Группы специальности "Программирование в
компьютерных системах"

```
CREATE OR REPLACE VIEW programming_groups AS
```

```
SELECT
```

```
  "Code_group" AS "Код группы",
```

```
  "Name_group" AS "Название группы",
```

```
  "Num_course" AS "Курс"
```

```
FROM
```

```
  "Groups"
```

WHERE

"Name_speciality" = 'Биотехнология'

ORDER BY

"Num_course", "Name_group";

-- Просмотр групп программистов

SELECT * FROM programming_groups;

	Код группы integer	Название группы character (25)	Курс integer
1	105	Аналитики	2

-- INSERT (сработало)

INSERT INTO programming_groups ("Код группы", "Название группы",
"Курс")

VALUES (111, 'ПКС-101', 1);

-- UPDATE (сработало)

UPDATE programming_groups SET "Название группы" = 'ПКС-101a' WHERE
"Код группы" = 111;

-- DELETE (сработало)

DELETE FROM programming_groups WHERE "Код группы" = 111;

-- Вывод: Обновляемое, так как:

-- Выборка из одной таблицы

-- Нет GROUP BY, агрегатных функций, DISTINCT

-- 3. Студенты с ФИО, группой и средним баллом

CREATE OR REPLACE VIEW student_gpa AS

SELECT s."Code_stud" AS "Код студента",

```

        CONCAT(s."Surname", ' ', s."Name", ' ', COALESCE(s."Lastname", '')) AS
"ФИО",

        g."Name_group" AS "Группа",

        ROUND(AVG(p."Estimate"), 2) AS "Средний балл"

FROM "Students" s

JOIN "Groups" g ON s."Code_group" = g."Code_group"

LEFT JOIN

        "Progress" p ON s."Code_stud" = p."Code_stud"

GROUP BY

        s."Code_stud", s."Surname", s."Name", s."Lastname", g."Name_group"

ORDER BY "Средний балл" DESC;

```

-- Просмотр успеваемости студентов

```
SELECT * FROM student_gpa;
```

	Код студента character (20) 🔒	ФИО text 🔒			Группа character (25) 🔒	Средний балл numeric 🔒
1	200108	Орлова	Арина	Арсентьевна	Филологи	[null]
2	200202	Чернова	Мария	Матвеевна	Социологи	[null]
3	199907	Лебедев	Владислав	Владиславович	Аналитики	[null]
4	200005	Соловьев	Данил	Данилович	Экономисты	[null]
5	200203	Алексеева	Тина	Тимуровна	Геологи	[null]
6	199809	Тимофеева	Ева	Евгеньевна	Психологи	[null]
7	200111	Кудрявцева	Ирина	Ильинична	Инженеры	[null]
8	200520	Петрова	Анна	Петровна	Биологи	5.00
9	199803	Смирнов	Алексей	Алексеевич	Аналитики	5.00
10	199909	Соколова	Екатерина	Егоровна	Психологи	5.00
11	200106	Федорова	Кира	Кирилловна	Астрономы	4.50
12	200004	Михайлова	Арина	Артемовна	Социологи	4.00

-- INSERT (не сработало)

```

INSERT INTO student_gpa ("Код студента", "ФИО", "Группа")

VALUES ('ST100', 'Петров Петр Петрович', 'ПКС-101');

```

-- UPDATE (не сработало)

```
UPDATE student_gpa SET "Группа" = 'ПКС-102' WHERE "Код студента" = 'ST001';
```

-- DELETE (не сработало)

```
DELETE FROM student_gpa WHERE "Код студента" = 'ST001';
```

-- Вывод: не обновляемое, так как:

-- Содержит JOIN нескольких таблиц

-- Использует агрегатные функции (AVG, COUNT)

-- Имеет GROUP BY и HAVING

-- 4. Топ-5 студентов с наибольшим средним баллом

```
CREATE OR REPLACE VIEW top5_students AS
```

```
SELECT "Код студента",
```

```
       "ФИО",
```

```
       "Группа",
```

```
       "Средний балл"
```

```
FROM student_gpa
```

```
ORDER BY "Средний балл" DESC
```

```
LIMIT 5;
```

-- Просмотр лучших студентов

```
SELECT * FROM top5_students;
```

	Код студента character (20) 🔒	ФИО text 🔒	Группа character (25) 🔒	Средний балл numeric 🔒
1	200108	Орлова Арина ...	Филологи ...	[null]
2	199809	Тимофеева Ева ...	Психологи ...	[null]
3	200202	Чернова Мария ...	Социологи ...	[null]
4	200005	Соловьев Данил...	Экономисты ...	[null]
5	200203	Алексеева Тина ...	Геологи ...	[null]

-- INSERT (не сработало)

```
INSERT INTO top5_students ("Код студента", "ФИО", "Группа", "Средний  
балл")  
VALUES ('ST100', 'Петров Петр Петрович', 'ПКС-101', 4.8);
```

-- UPDATE (не сработало)

```
UPDATE top5_students SET "Средний балл" = 5.0 WHERE "Код студента" =  
'ST001';
```

-- DELETE (не сработало)

```
DELETE FROM top5_students WHERE "Код студента" = 'ST001';
```

-- Вывод: не обновляемое, так как:

-- Основано на другом представлении

-- Содержит ограничение LIMIT

СОЗДАНИЕ ХРАНИМЫХ ПРОЦЕДУР

Создание хранимых процедур в MySQL. База данных «Книжное дело».

1. Вывести все сведения о поставке (все поля таблицы *Purchases*), а также название книги (поле *Title_book*) с максимальной общей стоимостью (использовать поля *Cost* и *Amount*).
2. Сосчитать количество книг определенного автора (ФИО автора является входным параметром).
3. Определить адрес определенного поставщика (Наименование поставщика является входным параметром, адрес поставщика – выходным параметром).
4. Выполните операцию вставки в таблицу *Books*. Код книги должен увеличиваться автоматически на единицу.
5. Определить поставки с минимальной и максимальной стоимостью книг. Отобразить список всех поставок. Если стоимость поставки – максимальная, то вывести сообщение «Максимальная стоимость», если стоимость – минимальная, то вывести сообщение «Минимальная стоимость», иначе – «Средняя стоимость».
6. Определить количество записей в таблице поставщиков. Пока записей меньше 10, делать в цикле добавление записи в таблицу с автоматическим наращиванием значения ключевого поля, а вместо названия поставщика ставить значение 'не известен'.
use db_books;

-- 1. Поставки с максимальной общей стоимостью книги

DELIMITER //

CREATE PROCEDURE get_max_cost_deliveries()

BEGIN

SELECT p.*, b.Title_book

FROM purchases p

JOIN books b ON p.Code_book = b.Code_book

WHERE (p.Cost * p.Amount) = (

```

SELECT MAX(Cost * Amount)
FROM purchases
);
END //
DELIMITER ;

```

Code_purchase	Code_book	Date_order	Code_delivery	Type_purchase	Cost	Amount	Title_book
9	9	2002-05-05	9	2	475	190	Герой нашего времени

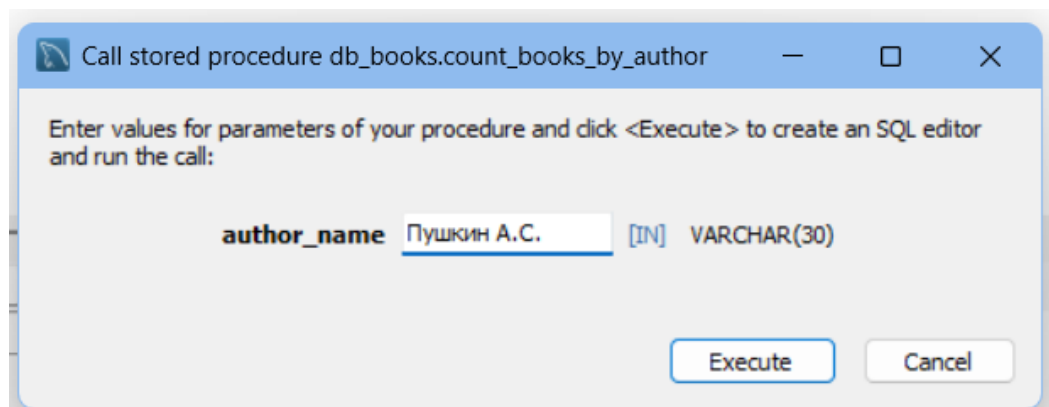
-- 2. Количество книг определенного автора

```

DELIMITER //
CREATE PROCEDURE count_books_by_author(IN author_name VARCHAR(30))
BEGIN
    SELECT COUNT(*) AS book_count
    FROM books b
    JOIN authors a ON b.Code_author = a.Code_author
    WHERE a.Name_author = author_name;
END //
DELIMITER ;

```

-- Вызов



book_count
1

-- 3. Адрес поставщика

```
DELIMITER //

CREATE PROCEDURE get_delivery_address(

    IN delivery_name VARCHAR(30),

    OUT delivery_address VARCHAR(100))

BEGIN

    SELECT Address INTO delivery_address

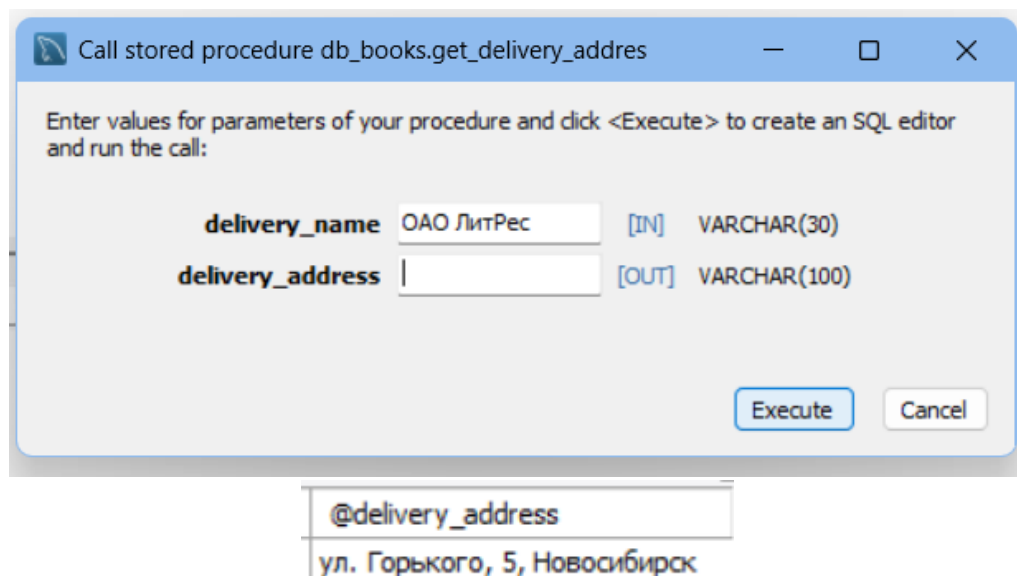
    FROM deliveries

    WHERE Name_company = delivery_name;

END //

DELIMITER ;
```

-- Вызов



-- 4. Вставка книги с автоинкрементом

```
DELIMITER //

CREATE PROCEDURE insert_book(

    IN title VARCHAR(40),

    IN author_id INT,
```

```

    IN pages INT,
    IN publish_id INT
)
BEGIN
    DECLARE next_code INT;
    SELECT IFNULL(MAX(Code_book), 0) + 1 INTO next_code FROM books;

    INSERT INTO books (Code_book, Title_book, Code_author, Pages, Code_publish)
    VALUES (next_code, title, author_id, pages, publish_id);
END //
DELIMITER ;

```

-- Вызов

Call stored procedure db_books.insert_book

Enter values for parameters of your procedure and click <Execute> to create an SQL editor and run the call:

title	Новая книга	[IN]	VARCHAR(40)
author_id	2	[IN]	INT
pages	200	[IN]	INT
publish_id	1	[IN]	INT

Execute Cancel

18	Новая книга	2	200	1
----	-------------	---	-----	---

-- 5. Анализ стоимости поставок

```

DELIMITER //
CREATE PROCEDURE analyze_purchase_costs()
BEGIN
    DECLARE min_cost FLOAT;
    DECLARE max_cost FLOAT;

```

```

SELECT MIN(Cost * Amount), MAX(Cost * Amount)

INTO min_cost, max_cost

FROM purchases;

SELECT

p.*,

CASE

    WHEN (p.Cost * p.Amount) = max_cost THEN 'Максимальная стоимость'

    WHEN (p.Cost * p.Amount) = min_cost THEN 'Минимальная стоимость'

    ELSE 'Средняя стоимость'

END AS cost_status

FROM purchases p;

END //

DELIMITER ;

```

-- Вызов

Code_purchase	Code_book	Date_order	Code_delivery	Type_purchase	Cost	Amount	cost_status
1	1	2003-03-13	1	2	300	150	Средняя стоимость
2	2	2003-05-14	2	1	400	110	Средняя стоимость
3	3	2003-04-12	3	2	600	50	Средняя стоимость
4	4	2003-06-15	4	1	550	80	Средняя стоимость
5	5	2003-05-10	5	2	275	140	Средняя стоимость
6	6	2003-03-15	6	1	325	160	Средняя стоимость
8	8	2002-01-15	8	1	450	180	Средняя стоимость
9	9	2002-05-05	9	2	475	190	Максимальная стоимость
11	1	2023-10-01	1	1	300	10	Минимальная стоимость

-- 6. Заполнение таблицы поставщиков

```

DELIMITER //

CREATE PROCEDURE fill_deliveries()

BEGIN

```

```
DECLARE delivery_count INT;

DECLARE next_code INT;

SELECT COUNT(*) INTO delivery_count FROM deliveries;

WHILE delivery_count < 10 DO

    SELECT IFNULL(MAX(Code_delivery), 0) + 1 INTO next_code FROM deliveries;

    INSERT INTO deliveries (

        Code_delivery,

        Name_delivery,

        Name_company,

        Address,

        Phone,

        INN

    ) VALUES (

        next_code,

        'не известен',

        'не известен',

        'не указан',

        0,

        '00000000000000'

    );

    SET delivery_count = delivery_count + 1;

END WHILE;

END //
```

DELIMITER ;

-- Проверка количества поставщиков

SELECT COUNT(*) FROM deliveries;

COUNT(*)
12

-- Вызов

CALL fill_deliveries();

Создание хранимых процедур в PostgreSQL. База данных «Успеваемость».

1. Вывести фамилии и имена студентов (поля *Surname*, *Name* из таблицы *Students*) с максимальным средним баллом за весь период обучения (условие по полю *Estimate* из таблицы *Progress*).
2. Определить средний балл определенного студента (ФИО студента является входным параметром).
3. Определить специальность и номер курса определенного студента (ФИО студента является входным параметром, Название специальности и Номер курса – выходными параметрами).
4. Выполните операцию вставки в таблицу *Students*. Код студента должен автоматически увеличиваться на единицу.
5. Определить средний возраст всех студентов. Вывести список всех студентов. Если возраст студента больше среднего возраста, то вывести сообщение «Вы старше среднего возраста всех студентов», если возраст – меньше, то вывести сообщение «Ваш возраст меньше среднего возраста всех студентов», а иначе – «Ваш возраст равен среднему возрасту всех студентов».
6. Определить количество записей в таблице дисциплин. Пока записей меньше 10, делать в цикле добавление записи в таблицу с автоматическим наращиванием значения ключевого поля, а вместо названия дисциплины ставить значение 'не известно'.

-- 1. Студенты с максимальным средним баллом

```
CREATE OR REPLACE FUNCTION get_top_students()
RETURNS TABLE (surname CHAR(25), name CHAR(25)) AS $$
BEGIN
    RETURN QUERY
    SELECT s."Surname", s."Name"
    FROM "Students" s
    JOIN (
        SELECT "Code_stud", AVG("Estimate") AS avg_score
        FROM "Progress"
```



```

GROUP BY "Code_stud"
ORDER BY avg_score DESC
LIMIT 1
) p ON s."Code_stud" = p."Code_stud";
END;
$$ LANGUAGE plpgsql;

```

```
SELECT * FROM get_top_students();
```

surname character	name character
Петрова	Анна

-- 2. Средний балл конкретного студента

```

CREATE OR REPLACE FUNCTION get_student_avg(fio VARCHAR)
RETURNS NUMERIC AS $$
DECLARE
    avg_score NUMERIC;
BEGIN
    SELECT AVG("Estimate") INTO avg_score
    FROM "Progress" p
    JOIN "Students" s ON p."Code_stud" = s."Code_stud"
    WHERE s."Surname" || ' ' || s."Name" = fio;

    RETURN avg_score;
END;
$$ LANGUAGE plpgsql;

```

-- Вызов

```
SELECT get_student_avg('Иванов Иван');
```

get_student_avg	numeric
4.0000000000000000	

-- 3. Специальность и курс студента

```
CREATE OR REPLACE FUNCTION get_student_info(fio VARCHAR)
RETURNS TABLE (speciality CHAR, course INTEGER) AS $$
BEGIN
    RETURN QUERY
        SELECT g."Name_speciality", g."Num_course"
        FROM "Students" s
        JOIN "Groups" g ON s."Code_group" = g."Code_group"
        WHERE s."Surname" || ' ' || s."Name" = fio;
END;
$$ LANGUAGE plpgsql;
```

-- Вызов

```
SELECT * FROM get_student_info('Иванов Иван');
```

speciality	course
character	integer
Прикладная информати...	1

-- 4. Вставка студента с автоинкрементом

```
CREATE OR REPLACE FUNCTION insert_student(
    s_surname VARCHAR,
    s_name VARCHAR,
    s_lastname VARCHAR,
    s_group INTEGER,
```

```

s_birthday DATE,
s_phone NUMERIC
) RETURNS VOID AS $$
BEGIN
    INSERT INTO "Students" (
        "Code_stud", "Surname", "Name", "Lastname", "Code_group",
"Birthday", "Phone"
    )
    VALUES (
        (
            SELECT 'ST' || LPAD(
                (COALESCE(MAX(SUBSTRING("Code_stud" FROM 3)::INTEGER), 0) +
1)::TEXT,
                5,
                '0'
            )
            FROM "Students"
        ),
        s_surname,
        s_name,
        s_lastname,
        s_group,
        s_birthday,
        s_phone
    );
END;
$$ LANGUAGE plpgsql;

```

-- Вызов

```
SELECT insert_student('Петров', 'Петр', 'Петрович', 101, '2000-01-01',  
89123456789);
```

id	surname	name	patronymic	age	birthday	phone	avg_age
81	Сидоров	Сергей	Сергеевич	103	2001-08-10	89345678901	3.00
82	Иванов	Павел	Сергеевич	[null]	[null]	[null]	4.00
910	Петров	Петр	Петрович	101	2000-01-01	89123456789	0.00

-- 5. Анализ возраста студентов

```
CREATE OR REPLACE FUNCTION analyze_age()  
RETURNS TABLE (  
    student VARCHAR(51),  
    age INTEGER,  
    message TEXT  
) AS $$  
DECLARE  
    avg_age NUMERIC;  
BEGIN  
    SELECT AVG(EXTRACT(YEAR FROM AGE(CURRENT_DATE, "Birthday"))) INTO  
avg_age  
    FROM "Students";  
  
    RETURN QUERY  
    SELECT  
        ("Surname" || ' ' || "Name")::VARCHAR(51) as student,  
        EXTRACT(YEAR FROM AGE(CURRENT_DATE, "Birthday"))::INTEGER as  
age,  
        CASE  
            WHEN EXTRACT(YEAR FROM AGE(CURRENT_DATE, "Birthday")) >  
avg_age THEN 'Вы старше среднего возраста'  
            WHEN EXTRACT(YEAR FROM AGE(CURRENT_DATE, "Birthday")) <  
avg_age THEN 'Ваш возраст меньше среднего'
```

```

ELSE 'Ваш возраст равен среднему'

END::TEXT as message

FROM "Students";

END;

$$ LANGUAGE plpgsql;

```

-- Вызов

```
SELECT * FROM analyze_age();
```

	student character varying	age integer	message text
1	Смирнов Алексей	27	Вы старше среднего возраста
2	Тимофеева Ева	26	Вы старше среднего возраста
3	Морозов Никита	26	Вы старше среднего возраста
4	Лебедев Владислав	25	Вы старше среднего возраста
5	Соколова Екатерина	25	Вы старше среднего возраста
6	Волкова Роза	25	Вы старше среднего возраста
7	Михайлова Арина	25	Вы старше среднего возраста
8	Соловьев Данил	25	Вы старше среднего возраста
9	Попов Максим	24	Ваш возраст меньше средне...
10	Федорова Кира	23	Ваш возраст меньше средне...
11	Орлова Арина	23	Ваш возраст меньше средне...
12	Кудрявцева Ирина	23	Ваш возраст меньше средне...
13	Кузнецова Дарья	24	Ваш возраст меньше средне...
14	Иванов Иван	25	Вы старше среднего возраста
15	Чернова Мария	23	Ваш возраст меньше средне...
16	Алексеева Тина	23	Ваш возраст меньше средне...
17	Новиков Даниил	22	Ваш возраст меньше средне...
18	Васильева Мария	22	Ваш возраст меньше средне...
19	Петрова Анна	26	Вы старше среднего возраста
20	Сидоров Сергей	23	Ваш возраст меньше средне...

-- 6. Заполнение таблицы дисциплин

```

CREATE OR REPLACE FUNCTION fill_subjects()

RETURNS VOID AS $$

```

```

DECLARE

    i INTEGER;

BEGIN

    FOR i IN 1..(10 - (SELECT COUNT(*) FROM "Subjects")) LOOP

        INSERT INTO "Subjects" ("Code_subject", "Name_subject",
"Count_hours")

            VALUES (

                (SELECT COALESCE(MAX("Code_subject"), 0) + 1 FROM "Subjects"),

                'не известно',

                0

            );

    END LOOP;

END;

$$ LANGUAGE plpgsql;

```

-- **Вызов**

```
SELECT fill_subjects();
```

	Code_subject [PK] integer	Name_subject character (25)	Count_hours integer
1	1001	Математичес...	120
2	1002	Физика ...	100
3	1003	Химия ...	90
4	1004	Биология ...	80
5	1005	Информатика...	140
6	1006	История ...	70
7	1007	Литература ...	60
8	1008	География ...	50
9	1009	Английский я...	110
10	1010	Физическая к...	40
11	1011	Экономика ...	130
12	1012	Психология ...	100
13	1013	Социология ...	90
14	1014	Правоведени...	80
15	1015	Изобразитель...	60
16	1016	Музыка ...	50
17	1017	Технология ...	70
18	1018	Биология ...	80
19	1019	География ...	50
20	1020	История ...	70

СОЗДАНИЕ ТРИГГЕРОВ. ТРАНЗАКЦИИ

Создание триггеров в MySQL. База данных «Книжное дело».

1. Создайте триггер, запускаемый при занесении новой строки в таблицу Авторы. Триггер должен увеличивать счетчик числа добавленных строк.

2. Добавьте в таблицу Авторы поле Количество книг (Count_books) целого типа со значением по умолчанию 0. Создайте хранимую процедуру, которая подсчитывает количество книг по каждому автору и заносит в поле Count_books эту информацию. Создайте триггер, запускаемый после внесения новой информации о книге.

3. Создайте триггер, запускаемый при внесении информации о новых поставках. Выполните проверку о количестве добавляемой книги в таблице Книги. Если количество экземпляров книг в таблице меньше 10, то необходимо увеличить стоимость книг на 20 %.

4. Запретить вставлять новые строки в таблицу Поставщики, выводя при этом сообщение «Вставка строк запрещена».

5. Проверьте выполнение команд транзакции при добавлении новой информации об издательствах.

-- 1. триггер, запускаемый при занесении новой строки в таблицу Авторы.

-- Триггер должен увеличивать счетчик числа добавленных строк.

-- Создаем таблицу-счетчик

```
create table if not exists authors_counter (  
    counter_name varchar(50) primary key,  
    counter_value int default 0  
);
```

-- Инициализируем счетчик

```
insert into authors_counter (counter_name, counter_value)  
values ('added_authors', 0)
```

on duplicate key update counter_value = counter_value;

counter_name	counter_value
added_authors	0

-- Вставка нового автора (триггер сработает автоматически)

insert into authors (Code_author, Name_author, Birthday)

values (1001, 'Иванов Иван Иванович', '1980-05-15');

counter_name	counter_value
added_authors	1

The screenshot shows the MySQL Workbench interface. At the top, there are settings for Charset/Collation (utf8mb4, utf8mb4_0900_ai_ci) and Engine (InnoDB). Below this is a list of trigger events: BEFORE INSERT, AFTER INSERT (selected), BEFORE UPDATE, AFTER UPDATE, BEFORE DELETE, and AFTER DELETE. The selected event 'after_insert_author' is highlighted. To the right, the SQL code for the trigger is displayed:

```
1 CREATE DEFINER='root'@'localhost' TRIGGER `after_insert_author` AFTER INSERT
2   UPDATE authors_counter
3   SET counter_value = counter_value + 1
4   WHERE counter_name = 'added_authors';
5 END
```

-- 2. Добавление в таблицу Авторы поля Количество книг (Count_books)

-- целого типа со значением по умолчанию 0. Создание хранимой

-- процедуры, которая подсчитывает количество книг по каждому автору

-- и вставка этой информации в поле Count_books. Создание триггера,

-- запускаемого после внесения новой информации о книге.

-- Добавляем поле

alter table authors add column Count_books int default 0;

Code_author	Name_author	Birthday	Count_books
-------------	-------------	----------	-------------

-- Процедура для обновления счетчика книг

DELIMITER //

create procedure update_books_count()

begin


```

update authors a
set Count_books = (
    select COUNT(*)
    from books b
    where b.Code_author = a.Code_author
);
end//
DELIMITER ;

```

-- Триггер при добавлении книги

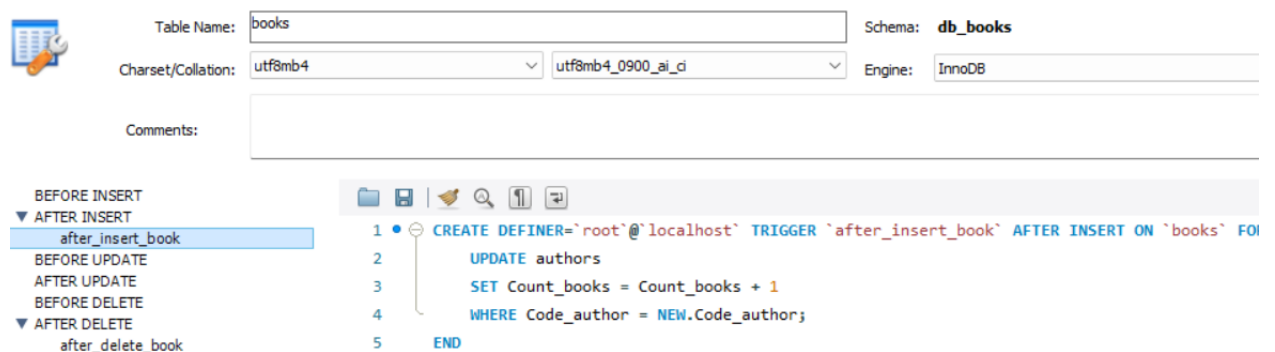
```

DELIMITER //

create trigger after_insert_book
after insert on books
for each row
begin
    update authors
    set Count_books = Count_books + 1
    where Code_author = NEW.Code_author;
end//

DELIMITER ;

```



The screenshot shows a database management interface. At the top, the 'books' table is selected, with schema 'db_books', charset 'utf8mb4', collation 'utf8mb4_0900_ai_ci', and engine 'InnoDB'. Below this, a tree view on the left shows the trigger 'after_insert_book' under the 'AFTER INSERT' category. The main area displays the SQL script for creating this trigger:

```

1 CREATE DEFINER='root'@'localhost' TRIGGER `after_insert_book` AFTER INSERT ON `books` FOR
2 UPDATE authors
3 SET Count_books = Count_books + 1
4 WHERE Code_author = NEW.Code_author;
5 END

```

-- Триггер при удалении книги

```

DELIMITER //

create trigger after_delete_book

after DELETE on books

for each row

begin

    update authors

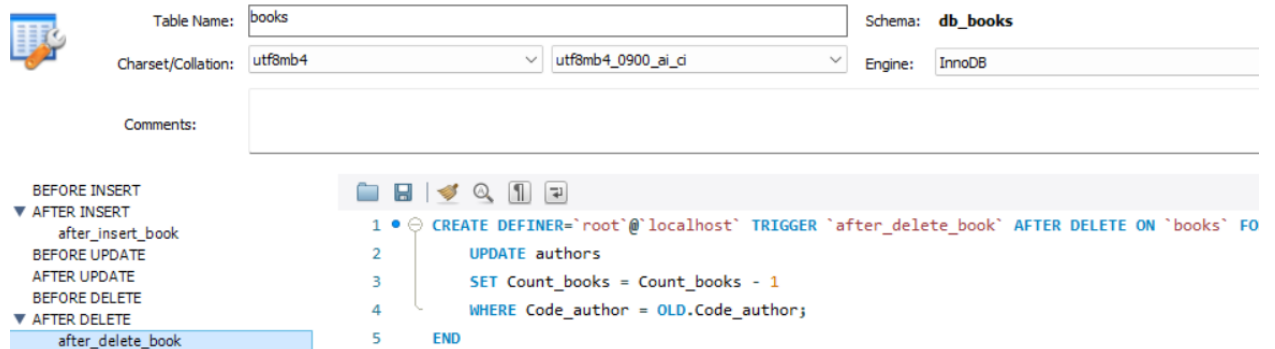
    set Count_books = Count_books - 1

    where Code_author = OLD.Code_author;

end//

DELIMITER ;

```



-- 3. Создание триггера, запускаемого при внесении информации о новых поставках. Выполнение проверки о количестве добавляемой книги в таблице Книги. Если количество экземпляров книг в таблице меньше 10, то стоимость книг увеличивается на 20 %.

```

DELIMITER //

create trigger before_insert_purchase

before INSERT ON purchases

for each row

begin

    declare book_count int;

```

-- Получаем общее количество экземпляров книги

```
select SUM(Amount) into book_count
```

```
from purchases
```

```
where Code_book = NEW.Code_book;
```

-- Если книг меньше 10, увеличиваем стоимость на 20%

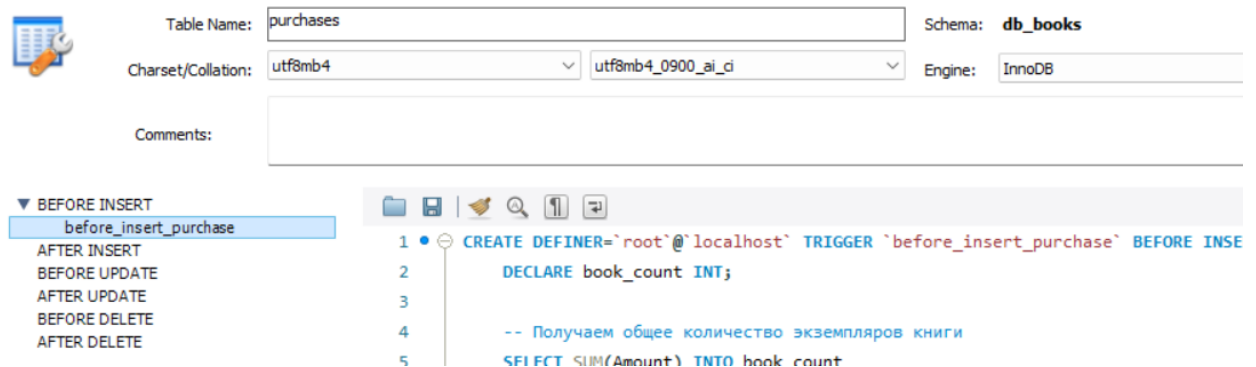
```
if book_count < 10 then
```

```
    set NEW.Cost = NEW.Cost * 1.2;
```

```
end if;
```

```
end//
```

```
DELIMITER ;
```



-- 4. Запрет вставлять новые строки в таблицу Поставщики, вывод при

-- этом сообщения «Вставка строк запрещена».

```
DELIMITER //
```

```
create trigger prevent_deliveries_insert
```

```
before INSERT ON deliveries
```

```
for each row
```

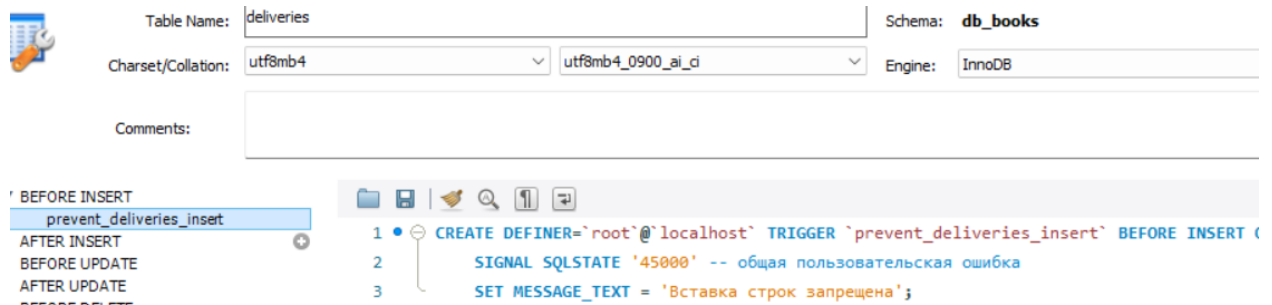
```
begin
```

```
    SIGNAL SQLSTATE '45000' -- общая пользовательская ошибка
```

```
    SET MESSAGE_TEXT = 'Вставка строк запрещена';
```

end//

DELIMITER ;



The screenshot shows the MySQL Workbench interface. At the top, the 'Table Name' is 'deliveries', the 'Schema' is 'db_books', the 'Charset/Collation' is 'utf8mb4', and the 'Engine' is 'InnoDB'. Below this, the 'Comments' field is empty. On the left, a list of triggers for the 'deliveries' table is shown, with 'prevent_deliveries_insert' selected. The main editor displays the SQL code for this trigger:

```
1 CREATE DEFINER='root'@'localhost' TRIGGER `prevent_deliveries_insert` BEFORE INSERT (
2     SIGNAL SQLSTATE '45000' -- общая пользовательская ошибка
3     SET MESSAGE_TEXT = 'Вставка строк запрещена';
```

-- 5. Проверка выполнения команд транзакции при добавлении новой
-- информации об издательствах.

start transaction;

-- Добавляем издательство

```
insert into publishing_house (Code_publish, Publish, City)
values (999, 'Новое издательство', 'Москва');
```

-- Проверяем добавление

```
select * from publishing_house where Code_publish = 999;
```

Code_publish	Publish	City
999	Новое издательство	Москва

-- Откатываем

ROLLBACK;

-- Проверка после отката

```
select * from publishing_houses where Code_publish = 999;
```

Code_publish	Publish	City
999	Новое издательство	Москва

Создание триггеров в PostgreSQL. База данных «Успеваемость».

1. Создайте триггер, запускаемый при занесении новой строки в таблицу

Преподаватели. Триггер должен увеличивать счетчик числа добавленных строк.

2. Добавьте в таблицу Студенты поле Средний балл (Avg_Estimate) вещественного типа со значением по умолчанию 0. Создайте хранимую процедуру, которая подсчитывает средний балл для каждого студента и заносит в поле Avg_Estimate эту информацию. Создайте триггер, запускаемый после внесения новой информации об оценках студента и автоматически обновляет информацию о среднем балле студента.

3. Создайте триггер, запускаемый при внесении информации о новых оценках. Выполните проверку наличия информации о добавляемом студенте в таблице Студенты. Если данная информация в таблице отсутствует, то необходимо запустить хранимую процедуру на вставку записи в таблицу Студенты (параметры можно задать произвольно).

4. Запретить вставлять новые строки в таблицу Группы, выводя при этом сообщение «Вставка строк запрещена».

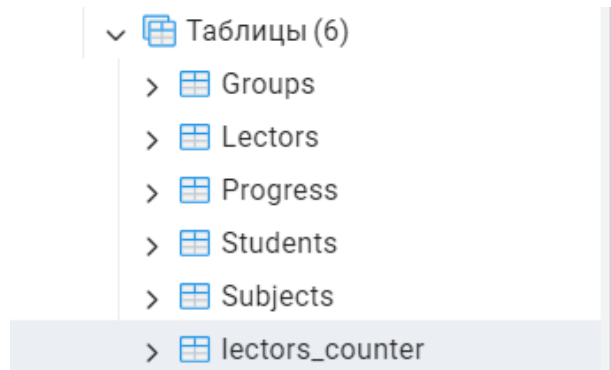
5. Проверьте выполнение команд транзакции при добавлении новой информации о преподавателях.

-- 1. Триггер, запускаемый при занесении новой строки в таблицу Преподаватели.

-- Триггер должен увеличивать счетчик числа добавленных строк.

-- Создаем таблицу-счетчик

```
CREATE TABLE IF NOT EXISTS lectors_counter (  
    counter_name VARCHAR(50) PRIMARY KEY,  
    counter_value INTEGER DEFAULT 0  
);
```



-- Инициализируем счетчик (если он пустой)

```
INSERT INTO lectors_counter (counter_name, counter_value)  
VALUES ('added_lectors', 0);
```

-- Создаем функцию-триггер, увеличивающую кол-во на 1

```
CREATE OR REPLACE FUNCTION increment_lectors_counter()  
RETURNS TRIGGER AS $$  
BEGIN  
    UPDATE lectors_counter  
    SET counter_value = counter_value + 1  
    WHERE counter_name = 'added_lectors';  
  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

-- Создаем триггер AFTER INSERT

```
CREATE TRIGGER tr_after_insert_lector  
AFTER INSERT ON "Lectors"  
FOR EACH ROW  
EXECUTE FUNCTION increment_lectors_counter();
```

-- ПРОВЕРКА

```
INSERT INTO "Lectors" ("Code_lector", "Name_lector", "Science", "Post", "Date_")  
VALUES (2, 'Иванов И.И.', 'Кандидат наук', 'Доцент', CURRENT_DATE);
```

```
SELECT * FROM lectors_counter;
```

counter_name [PK] character varying (50)	counter_value integer
added_lectors	3

-- 2. Триггер, запускаемый после внесения новой информации об

-- оценках студента и автоматическое обновление информации о среднем

-- балле студента.

-- Добавление поля Avg_Estimate в таблицу Students вещественного типа со значением по умолчанию 0.

```
ALTER TABLE "Students"
```

```
ADD COLUMN "Avg_Estimate" NUMERIC(5,2) DEFAULT 0.00;
```

Code_stud [PK] character (20)	Surname character (25)	Name character (25)	Lastname character (25)	Code_group integer	Birthday date	Phone numeric	Avg_Estimate numeric (5,2)
----------------------------------	---------------------------	------------------------	----------------------------	-----------------------	------------------	------------------	-------------------------------

-- Хранимая процедура, которая подсчитывает средний балл для

-- каждого студента и заносит в поле Avg_Estimate эту информацию

```
CREATE OR REPLACE PROCEDURE update_students_avg_estimate()
```

```
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE "Students" s
    SET "Avg_Estimate" = COALESCE((
        SELECT AVG(p."Estimate")::NUMERIC(5,2)
        FROM "Progress" p
        WHERE p."Code_stud" = s."Code_stud"
    ), 0.00);

    RAISE NOTICE 'Средние баллы всех студентов успешно обновлены';
END;
$$;
```

-- Триггерная функция, автоматически обновляющая информацию о среднем
-- балле студента.

```
CREATE OR REPLACE FUNCTION update_student_avg_trigger()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE "Students"
    SET "Avg_Estimate" = COALESCE((
        SELECT AVG("Estimate")::NUMERIC(5,2)
        FROM "Progress"
        WHERE "Code_stud" = NEW."Code_stud"
    ), 0.00)
```



```
WHERE "Code_stud" = NEW."Code_stud";
```

```
RETURN NEW;
```

```
END;
```

```
$$;
```

```
-- Триггер, запускаемый после внесения новой информации об
```

```
-- оценках студента
```

```
CREATE TRIGGER tr_update_student_avg
```

```
AFTER INSERT OR UPDATE OR DELETE ON "Progress"
```

```
FOR EACH ROW
```

```
EXECUTE FUNCTION update_student_avg_trigger();
```

```
CALL update_students_avg_estimate();
```

```
INSERT INTO "Progress" ("Code_stud", "Code_subject", "Code_lector",  
"Date_exam", "Estimate", "Code_progress")
```

```
VALUES ('201082', 1003, 1, CURRENT_DATE, 5, 100);
```

```
-- Проверка обновления среднего балла
```

```
SELECT "Code_stud", "Surname", "Name", "Avg_Estimate"
```

```
FROM "Students"
```

```
WHERE "Code_stud" = '201082';
```

```
UPDATE "Progress"
```

```
SET "Estimate" = 4
```

```
WHERE "Code_progress" = 21;
```

-- Проверка обновления среднего балла

```
SELECT "Code_stud", "Surname", "Name", "Avg_Estimate"
FROM "Students"
WHERE "Code_stud" = '201082';
```

	Code_stud [PK] character (20)	Surname character (25)	Name character (25)	Lastname character (25)	Code_group integer	Birthday date	Phone numeric	Avg_Estimate numeric (5,2)
1	199803	Смирнов ...	Алексей ...	Алексеевич ...	105	1998-03-05	89567890123	5.00
2	199809	Тимофеева ...	Ева	Евгеньевна ...	108	1998-09-01	89890123456	0.00
3	199810	Морозов ...	Никита ...	Никитич ...	102	1998-10-28	89234567890	4.00
4	199907	Лебедев ...	Владислав ...	Владиславов...	105	1999-07-20	89567890123	0.00
5	199909	Соколова ...	Екатерина ...	Егоровна ...	108	1999-09-18	89890123456	5.00
6	200001	Волкова ...	Роза ...	Романовна ...	103	2000-01-05	89345678901	3.00
7	200004	Михайлова ...	Арина ...	Артемовна ...	109	2000-04-22	89901234567	4.00
8	200005	Соловьев ...	Данил ...	Данилович ...	107	2000-05-12	89789012345	0.00
9	200102	Попов ...	Максим ...	Максимович ...	107	2001-02-12	89789012345	3.00
10	200106	Федорова ...	Кира ...	Кирилловна ...	101	2001-06-15	89123456789	4.50
11	200108	Орлова ...	Арина ...	Арсентьевна ...	110	2001-08-18	89012345678	0.00
12	200111	Кудрявцева ...	Ирина ...	Ильинична ...	106	2001-11-30	89678901234	0.00
13	200113	Кузнецова ...	Дарья ...	Дмитриевна ...	104	2000-11-30	89456789012	4.00
14	200115	Иванов ...	Иван ...	Иванович ...	101	2000-01-15	89123456789	4.00
15	200202	Чернова ...	Мария ...	Матвеевна ...	109	2002-02-25	89901234567	0.00
16	200203	Алексеева ...	Тина ...	Тимуровна ...	104	2002-03-10	89456789012	0.00
17	200212	Новиков ...	Даниил ...	Данилович ...	110	2002-12-01	89012345678	3.00
18	200225	Васильева ...	Мария ...	Андреевна ...	106	2002-07-25	89678901234	4.00
19	200520	Петрова ...	Анна ...	Петровна ...	102	1999-05-20	89234567890	5.00
20	201081	Сидоров ...	Сергей ...	Сергеевич ...	103	2001-08-10	89345678901	3.00
21	201082	Иванов ...	Павел ...	Сергеевич ...	[null]	[null]	[null]	4.00
22	ST09910	Петров ...	Петр ...	Петрович ...	101	2000-01-01	89123456789	0.00

-- 3. Триггер, запускаемый при внесении информации о новых

-- оценках. Проверка наличия информации о добавляемом

-- студенте в таблице Студенты. Если данная информация в таблице

-- отсутствует, то запускается хранимая процедура на

-- вставку записи в таблицу Студенты (параметры можно задать

-- произвольно).

-- Хранимая процедура для добавления нового студента

```
CREATE OR REPLACE PROCEDURE add_new_student(
    IN p_code_stud VARCHAR(20),
```

```

    IN p_surname VARCHAR(25) DEFAULT 'Неизвестно',
    IN p_name VARCHAR(25) DEFAULT 'Неизвестно',
    IN p_lastname VARCHAR(25) DEFAULT NULL,
    IN p_code_group INTEGER DEFAULT 0,
    IN p_birthday DATE DEFAULT CURRENT_DATE,
    IN p_phone NUMERIC DEFAULT 0
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO "Students" (
        "Code_stud", "Surname", "Name", "Lastname",
        "Code_group", "Birthday", "Phone", "Avg_Estimate"
    ) VALUES (
        p_code_stud, p_surname, p_name, p_lastname,
        p_code_group, p_birthday, p_phone, 0.00
    );

    RAISE NOTICE 'Добавлен новый студент: % %', p_surname, p_name;
END;
$$;

```

-- Триггерная функция с проверкой наличия информации о добавляемом
-- студенте в таблице Студенты

```

CREATE OR REPLACE FUNCTION check_student_before_insert_grade()
RETURNS TRIGGER
LANGUAGE plpgsql

```

AS \$\$

BEGIN

-- Проверяем существование студента

IF NOT EXISTS (SELECT 1 FROM "Students" WHERE "Code_stud" =
NEW."Code_stud") THEN

CALL add_new_student(

p_code_stud := NEW."Code_stud",

p_surname := 'Новый',

p_name := 'Студент',

p_code_group := 0,

p_birthday := CURRENT_DATE - INTERVAL '20 years',

p_phone := 0

);

RAISE WARNING 'Студент с кодом % не найден и был автоматически
добавлен', NEW."Code_stud";

END IF;

RETURN NEW;

END;

\$\$;

-- Триггер, запускаемый при внесении информации о новых
оценках

CREATE TRIGGER tr_check_student_before_insert

BEFORE INSERT ON "Progress"

FOR EACH ROW

EXECUTE FUNCTION check_student_before_insert_grade();

-- ПРОВЕРКА

```
INSERT INTO "Progress" (  
    "Code_stud", "Code_subject", "Code_lector",  
    "Date_exam", "Estimate", "Code_progress"  
) VALUES (  
    '9999', 1001, 1, CURRENT_DATE, 5, 100  
);
```

-- Проверим, что студент был добавлен

```
SELECT * FROM "Students" WHERE "Code_stud" = '9999';
```

-- Проверим, что оценка была добавлена

```
SELECT * FROM "Progress" WHERE "Code_stud" = '9999';
```

-- 4. Запрет вставки новых строк в таблицу Группы, вывод при этом

-- сообщения «Вставка строк запрещена».

-- Триггерная функция

```
CREATE OR REPLACE FUNCTION prevent_groups_insert()  
RETURNS TRIGGER  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    RAISE EXCEPTION 'Вставка строк запрещена';  
    RETURN NULL;  
END;  
$$;
```

-- Триггер BEFORE INSERT

```
CREATE TRIGGER tr_prevent_groups_insert  
BEFORE INSERT ON "Groups"  
FOR EACH ROW  
EXECUTE FUNCTION prevent_groups_insert();
```

-- Попытка вставки

```
INSERT INTO "Groups" ("Code_group", "Name_group", "Num_course",  
"Name_speciality")  
VALUES (999, 'TEST', 1, 'Тестовая специальность');
```

-- 5. Проверка выполнения команд транзакции при добавлении новой
-- информации о преподавателях.

-- Начало транзакции

```
BEGIN;
```

-- Добавление нового преподавателя

```
INSERT INTO "Lectors" ("Code_lector", "Name_lector", "Science", "Post",  
"Date_date")  
VALUES (999, 'Иванов И.И.', 'Кандидат наук', 'Доцент', CURRENT_DATE);
```

-- Проверка добавления (в рамках той же транзакции)

```
SELECT * FROM "Lectors" WHERE "Code_lector" = 999;
```

-- Откат транзакции (для проверки)

```
ROLLBACK;
```

-- Проверка, что после ROLLBACK записи нет

```
SELECT * FROM "Lectors" WHERE "Code_lector" = 999;
```

СОЗДАНИЕ ПОЛЬЗОВАТЕЛЕЙ MySQL

Создайте пользователей базы данных «Книжное дело» в MySQL.

1. Администратор – обладает всеми правами
2. Диспетчер – просматривает, заполняет и изменяет справочники: книги, авторы, издательства, поставщики.
3. Менеджер по работе с поставщиками – просматривает и добавляет новую информацию в справочники, оформляет поставки.
4. Поставщики – просматривают только свои поставки

-- 1. Администратор – обладает всеми правами

```
CREATE USER 'admin_db_books'@'localhost' IDENTIFIED BY 'admin_pass';  
GRANT ALL PRIVILEGES ON db_books.* TO 'admin_db_books'@'localhost' WITH  
GRANT OPTION;  
FLUSH PRIVILEGES;
```

-- 2. Диспетчер – просматривает, заполняет и изменяет справочники: книги,
-- авторы, издательства, поставщики.

```
CREATE USER 'dispatcher'@'localhost' IDENTIFIED BY 'dispatcher_pass';  
GRANT SELECT, INSERT, UPDATE ON db_books.authors TO  
'dispatcher'@'localhost';  
GRANT SELECT, INSERT, UPDATE ON db_books.books TO 'dispatcher'@'localhost';  
GRANT SELECT, INSERT, UPDATE ON db_books.publishing_house TO  
'dispatcher'@'localhost';  
GRANT SELECT, INSERT, UPDATE ON db_books.deliveries TO  
'dispatcher'@'localhost';  
FLUSH PRIVILEGES;
```

-- 3. Менеджер по работе с поставщиками – просматривает и добавляет
-- новую информацию в справочники, оформляет поставки.


```
CREATE USER 'supply_manager'@'localhost' IDENTIFIED BY 'manager_pass';
```

```
-- Права на справочники
```

```
GRANT SELECT, INSERT ON db_books.authors TO 'supply_manager'@'localhost';
```

```
GRANT SELECT, INSERT ON db_books.books TO 'supply_manager'@'localhost';
```

```
GRANT SELECT, INSERT ON db_books.publishing_houses TO  
'supply_manager'@'localhost';
```

```
GRANT SELECT, INSERT ON db_books.deliveries TO 'supply_manager'@'localhost';
```

```
-- Права на оформление поставок
```

```
GRANT SELECT, INSERT ON db_books.purchases TO 'supply_manager'@'localhost';
```

```
FLUSH PRIVILEGES;
```

```
-- 4. Поставщики – просматривают только свои поставки
```

```
-- Создаем представление для ограничения доступа
```

```
CREATE VIEW supplier_purchases_view AS
```

```
SELECT p.*
```

```
FROM purchases p
```

```
JOIN deliveries d ON p.Code_delivery = d.Code_delivery
```

```
WHERE d.Name_company = CURRENT_USER();
```

```
-- Создаем пользователя
```

```
CREATE USER 'supplier_knigaplust'@'localhost' IDENTIFIED BY 'supplier_pass';
```

```
GRANT SELECT ON db_books.supplier_purchases_view TO  
'supplier_knigaplust'@'localhost';
```

```
FLUSH PRIVILEGES;
```

```
SHOW GRANTS FOR 'dispatcher'@'localhost';
```

Grants for dispatcher@localhost
GRANT USAGE ON *.* TO `dispatcher`@`localhost`
GRANT SELECT, INSERT, UPDATE ON `db_books`.`authors` TO `dispatcher`@`localhost`
GRANT SELECT, INSERT, UPDATE ON `db_books`.`books` TO `dispatcher`@`localhost`
GRANT SELECT, INSERT, UPDATE ON `db_books`.`deliveries` TO `dispatcher`@`localhost`
GRANT SELECT, INSERT, UPDATE ON `db_books`.`publishing_house` TO `dispatcher`@`local...