



**«Московский государственный технический университет
имени Н.Э. Баумана»
(национальный исследовательский университет)
(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ

ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА

КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

О т ч е т

по лабораторной работе № 2

Название лабораторной работы: Создание БД для приложения

Дисциплина: Базы данных

Вариант 18

Студент гр. ИУ6-34Б

(Подпись, дата)

А. И. Мокшина

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

(И.О. Фамилия)

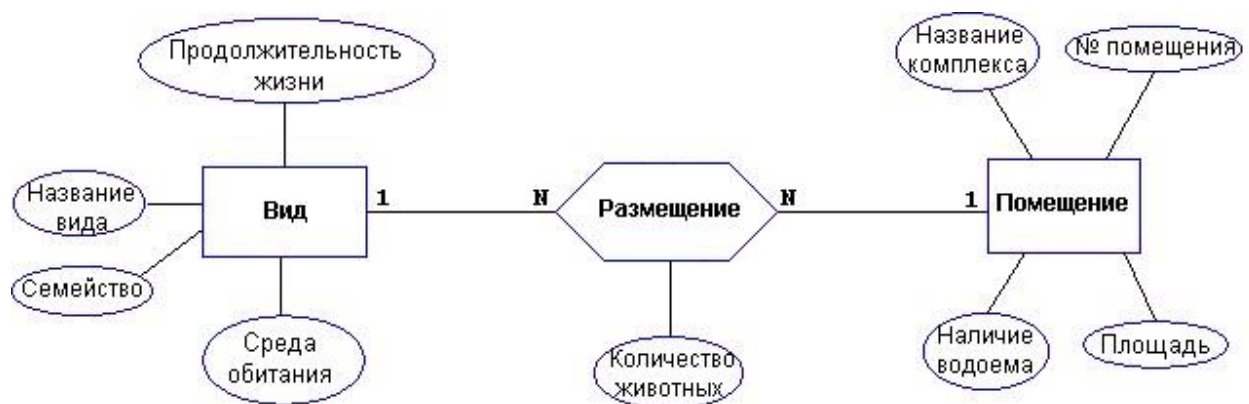
Цель:

Сформировать понимание особенностей хранения данных приложения в РСУБД, а также настройка и поддержка хранения данных.

Задачи:

- Получить теоретические знания по концептуальным картам.
- Ознакомиться с понятием нормализации в БД.
- Изучить типы связей между сущностями и таблицами БД.
- Ознакомиться с операторами создания БД.
- Изучение типов данных.
- Научится добавлять записи в таблицы.
- Научиться удалять и изменять записи в таблице.
- Ознакомиться с механизмами контроля согласованности БД, транзакциями и триггерами.

Предметная область для практических заданий: **Зоопарк**



1. Проектирование схемы базы данных

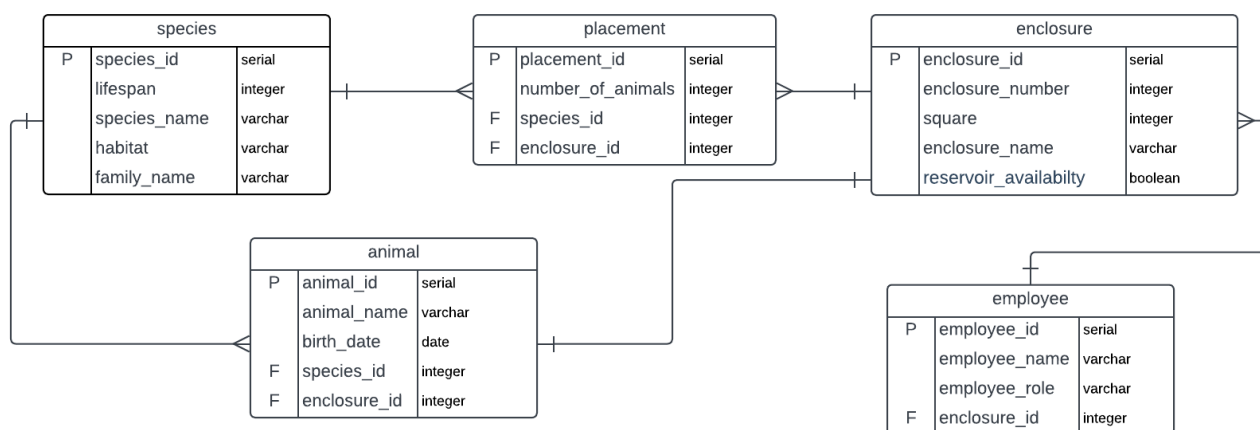


Рис. 1 – Схема базы данных

2. Создание и заполнения таблиц.

Создадим таблицы согласно схеме, полученной в предыдущем задании (с уточнением типов полей). Определить первичные и внешние ключи, а также ограничения полей (возможность принимать неопределенное значение, уникальные ключи, проверочные ограничения и т. д.).

```
create table species
(species_id serial PRIMARY KEY,
lifespan integer CHECK(lifespan > 0),
species_name varchar NOT NULL,
habitat varchar NOT NULL,
family_name varchar NOT NULL);

create table animal
(animal_id serial PRIMARY KEY,
animal_name varchar NOT NULL,
birth_date date NOT NULL,
species_id integer NOT NULL,
enclosure_id integer NOT NULL);

create table placement
(placement_id serial PRIMARY KEY,
number_of_animals integer CHECK(number_of_animals > 0),
species_id integer NOT NULL,
enclosure_id integer NOT NULL);

create table enclosure
(enclosure_id serial PRIMARY KEY,
enclosure_number integer CHECK(enclosure_number > 0),
square integer CHECK(square > 0),
enclosure_name varchar NOT NULL,
reservoir_availability boolean NOT NULL);

create table employee
(employee_id serial PRIMARY KEY,
```

```

employee_name varchar NOT NULL,
employee_role varchar NOT NULL,
enclosure_id integer NOT NULL);

ALTER TABLE animal
ADD FOREIGN KEY(species_id) REFERENCES species(species_id),
ADD FOREIGN KEY(enclosure_id) REFERENCES enclosure(enclosure_id);

ALTER TABLE employee
ADD FOREIGN KEY(enclosure_id) REFERENCES enclosure(enclosure_id);

ALTER TABLE enclosure
ADD UNIQUE(enclosure_number);

ALTER TABLE placement
ADD FOREIGN KEY(species_id) REFERENCES species(species_id),
ADD FOREIGN KEY(enclosure_id) REFERENCES enclosure(enclosure_id);

ALTER TABLE species
ADD UNIQUE(species_name);

ALTER TABLE species
RENAME COLUMN habibat TO habitat;

```

Заполним таблицы с помощью библиотеки Faker для генерации случайных данных и psycopg2 для работы с БД в Python.

```

from faker import Faker
import psycopg2
import random

# Создание подключения к базе данных
conn = psycopg2.connect(dbname="zoo", user="postgres", password="f8ysz789",
host="127.0.0.1")
cur = conn.cursor()

# Создание экземпляра Faker
fake = Faker()

# Генерация и вставка данных в таблицу species
number_of_species = 0
for _ in range(1, 50):
    species_name = fake.last_name()
    hab = ("grassland", "polar", "desert", "mountain", "freshwater", "ocean",
"rainforest" )
    habitat = random.choice(hab)
    family_name = fake.first_name()
    lifespan = random.randint(5, 90)

    # Проверка на уникальность значения поля "species_name"
    query = "SELECT count(*) FROM species WHERE species_name = %s"
    values = (species_name,)
    cur.execute(query, values)
    result = cur.fetchone()
    if result[0] > 0:
        continue
    number_of_species += 1
    species_id = number_of_species
    query = "INSERT INTO species (species_name, habitat, family_name, lifespan)

```

```

VALUES (%s, %s, %s, %s)"
    values = (species_name, habitat, family_name, lifespan)
    cur.execute(query, values)
# Генерация и вставка данных в таблицу enclosure
for _ in range(1, 50):
    enclosure_id = _
    enclosure_number = _ + 3
    square = random.randint(5, 100)
    enclosure_name = fake.company() # название комплекса
    b = (True, False)
    reservoir_availability = random.choice(b)

    query = "INSERT INTO enclosure (enclosure_number, square, enclosure_name,
reservoir_availability) VALUES (%s, %s, %s, %s)"
    values = (enclosure_number, square, enclosure_name, reservoir_availability)
    cur.execute(query, values)
# Генерация и вставка данных в таблицу animal
for _ in range(1, 1001):
    animal_id = _
    animal_name = fake.first_name()
    birth_date = fake.date()
    species_id = random.randint(1, number_of_species)
    enclosure_id = random.randint(1, 30)

    # SQL-запрос для вставки данных
    query = "INSERT INTO animal (animal_name, birth_date, species_id,
enclosure_id) VALUES (%s, %s, %s, %s)"
    values = (animal_name, birth_date, species_id, enclosure_id)
    # Выполнение SQL-запроса
    cur.execute(query, values)
# Генерация и вставка данных в таблицу employee
for _ in range(1, 51):
    employee_id = _
    employee_name = fake.name()
    role = ("clean", "wash", "photo", "video", "feed", "play", "guide")
    employee_role = random.choice(role)
    enclosure_id = random.randint(1, 30)

    query = "INSERT INTO employee (employee_name, employee_role, enclosure_id)
VALUES (%s, %s, %s)"
    values = (employee_name, employee_role, enclosure_id)
    cur.execute(query, values)
# Генерация и вставка данных в таблицу placement
for _ in range(1, 31):
    placement_id = _
    number_of_animals = random.randint(1,5)
    species_id = random.randint(1, number_of_species)
    enclosure_id = random.randint(1, 30)

    query = "INSERT INTO placement (number_of_animals, species_id, enclosure_id)
VALUES (%s, %s, %s)"
    values = (number_of_animals, species_id, enclosure_id)
    cur.execute(query, values)
# Подтверждение изменений и закрытие подключения
conn.commit()
cur.close()
conn.close()

```

Запрос к одной таблице, содержащий фильтрацию по нескольким полям.

- Получить план выполнения запроса без использования индексов.

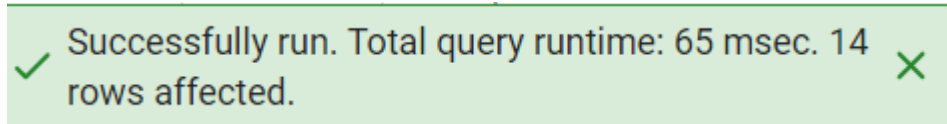
```
select *
```

```
from enclosure
```

```
where square > 50 and reservoir_availability is true
```

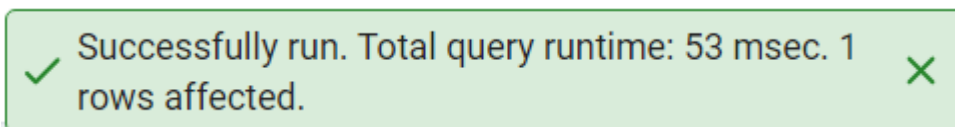
```
select count(*) from enclosure where square > 50 and  
reservoir_availability is true
```

- Получить статистику (IO и Time) выполнения запроса без использования индексов.



✓ Successfully run. Total query runtime: 65 msec. 14 rows affected. ✕

Рис. 2 – время первого запроса без использования индексов



✓ Successfully run. Total query runtime: 53 msec. 1 rows affected. ✕

Рис. 3 – время второго запроса без использования индексов

- Создать нужные индексы, позволяющие ускорить запрос.

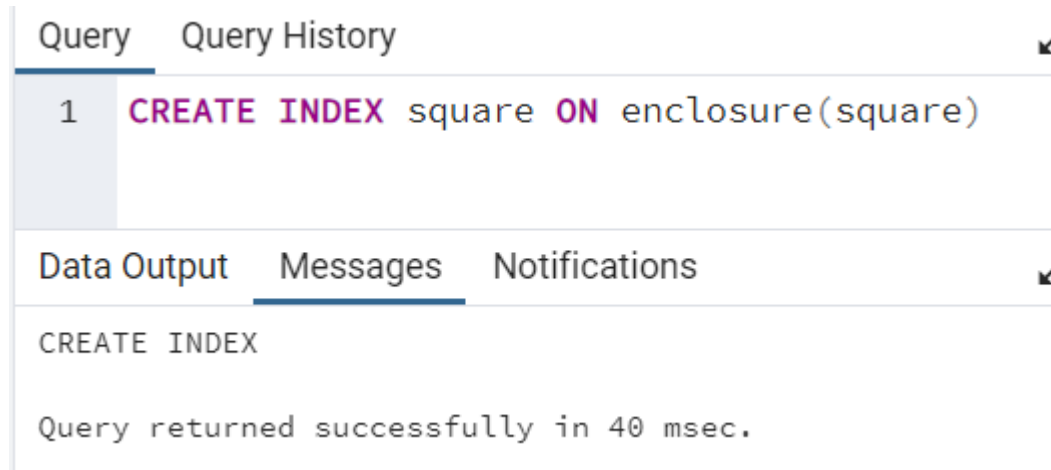
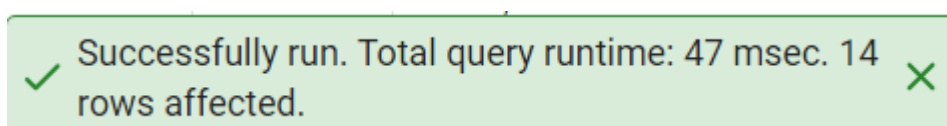


Рис. 4 – создание индекса

- Получить статистику выполнения запроса с использованием индексов и сравнить с первоначальной статистикой.



✓ Successfully run. Total query runtime: 47 msec. 14 rows affected. ✕

Рис. 5 – время первого запроса с использованием индексов

✓ Successfully run. Total query runtime: 50 msec. 1 rows affected. ✕

Рис. 6 – время второго запроса с использованием индексов

Запрос к нескольким связанным таблицам, содержащий фильтрацию по нескольким полям.

```
SELECT COUNT(DISTINCT animal.animal_id)
FROM employee
LEFT JOIN animal ON employee.enclosure_id =
animal.enclosure_id
WHERE employee.employee_role = 'feed'
```

✓ Successfully run. Total query runtime: 40 msec. 1 rows affected. ✕

Рис. 7 – время выполнения запроса без использования индексов

```
CREATE INDEX enclosure_id ON animal(enclosure_id);
```

✓ Successfully run. Total query runtime: 36 msec. 1 rows affected. ✕

Рис. 8 – время выполнения запроса с использованием индексов

2. Операторы манипулирования данными (DML)

В ходе выполнения задания необходимо:

Подготовить 3-4 выборки, которые имеют осмысленное значение для предметной области, и также составить для них SQL-скрипты.

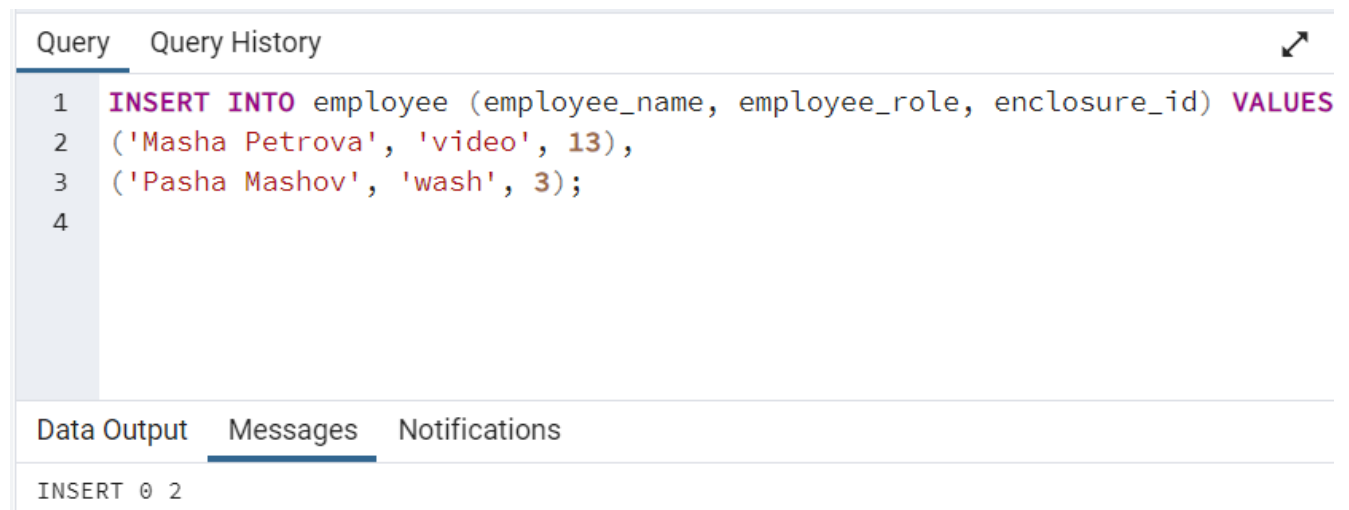


Рис. 9 – вставка значений

Query	Query History
1	INSERT INTO enclosure (enclosure_number, square, enclosure_name, reservoir_availability)
2	VALUES
3	(56, 100, 'Stauty', true),
4	(70, 10, 'Asasd', false);

Data Output	Messages	Notifications
INSERT 0 2		

Рис. 10 – вставка значений

Сформулировать 3-4 запроса на изменение и удаление из базы данных. Запросы должны быть сформулированы в терминах предметной области. Среди запросов обязательно должны быть такие, которые будут вызывать срабатывание ограничений целостности. Составить SQL-скрипты для выполнения этих запросов

Query	Query History
1	DELETE FROM animal
2	WHERE birth_date <= '1985-12-31'

Data Output	Messages	Notifications
DELETE 304		

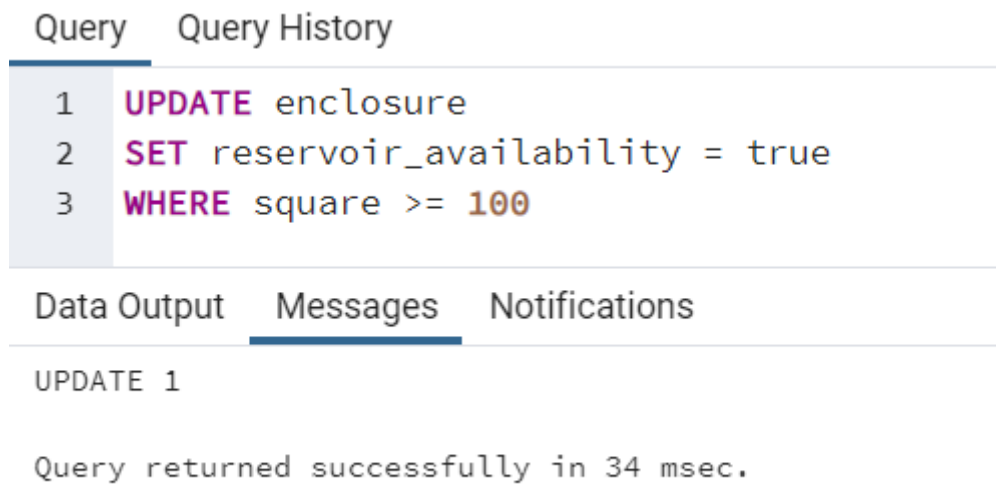
Query returned successfully in 32 msec.

Рис. 11 – удаление значений

Query	Query History
1	DELETE FROM employee
2	WHERE employee_name = 'Masha Petrova'

Data Output	Messages	Notifications
DELETE 1		

Рис. 12 – удаление значений



The screenshot shows a database query editor with two tabs: 'Query' and 'Query History'. The 'Query' tab is active, displaying an SQL UPDATE statement in three lines: '1 UPDATE enclosure', '2 SET reservoir_availability = true', and '3 WHERE square >= 100'. Below the query, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the text 'UPDATE 1' and 'Query returned successfully in 34 msec.'

```
Query Query History
```

```
1 UPDATE enclosure
2 SET reservoir_availability = true
3 WHERE square >= 100
```

```
Data Output Messages Notifications
```

```
UPDATE 1
```

```
Query returned successfully in 34 msec.
```

Рис. 13 – модификация значений

4. Представления

- Составить SQL-скрипты для создания нескольких представлений, которые позволяли бы упростить манипуляции с данными или позволяли бы ограничить доступ к данным, предоставляя только необходимую информацию.



The screenshot shows a database query editor with two tabs: 'Query' and 'Query History'. The 'Query' tab is active, displaying an SQL CREATE VIEW statement in five lines: '1 CREATE VIEW animals_age AS', '2 SELECT animal_name, species_name, age(birth_date) AS age', '3 FROM animal', '4 LEFT JOIN species ON animal.species_id = species.species_id', and '5 ORDER BY age;'. Below the query, there are three tabs: 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the text 'CREATE VIEW' and 'Query returned successfully in 34 msec.'

```
Query Query History
```

```
1 CREATE VIEW animals_age AS
2 SELECT animal_name, species_name, age(birth_date) AS age
3 FROM animal
4 LEFT JOIN species ON animal.species_id = species.species_id
5 ORDER BY age;
```

```
Data Output Messages Notifications
```

```
CREATE VIEW
```

```
Query returned successfully in 34 msec.
```

Рис. 14 – создание представления

Query	Query History
<pre> 1 CREATE VIEW group_of_workers AS 2 SELECT employee_role, count(*) as number_of_workers 3 FROM employee 4 GROUP BY employee_role 5 ORDER BY number_of_workers; </pre>	
Data Output	Messages
CREATE VIEW	Query returned successfully in 34 msec.

Рис. 15 – создание представления

Query	Query History
<pre> 1 CREATE VIEW id_name_animal AS 2 SELECT animal_id, animal_name 3 FROM animal 4 </pre>	
Data Output	Messages
CREATE VIEW	

Рис. 16 – создание представления

Query	Query History
1	CREATE VIEW guides AS
2	SELECT employee_id, employee_name,
3	employee_role, enclosure_id
4	FROM employee
5	WHERE employee_role = 'guide'
6	

Data Output	Messages	Notifications
CREATE VIEW		
Query returned successfully in 86 msec.		

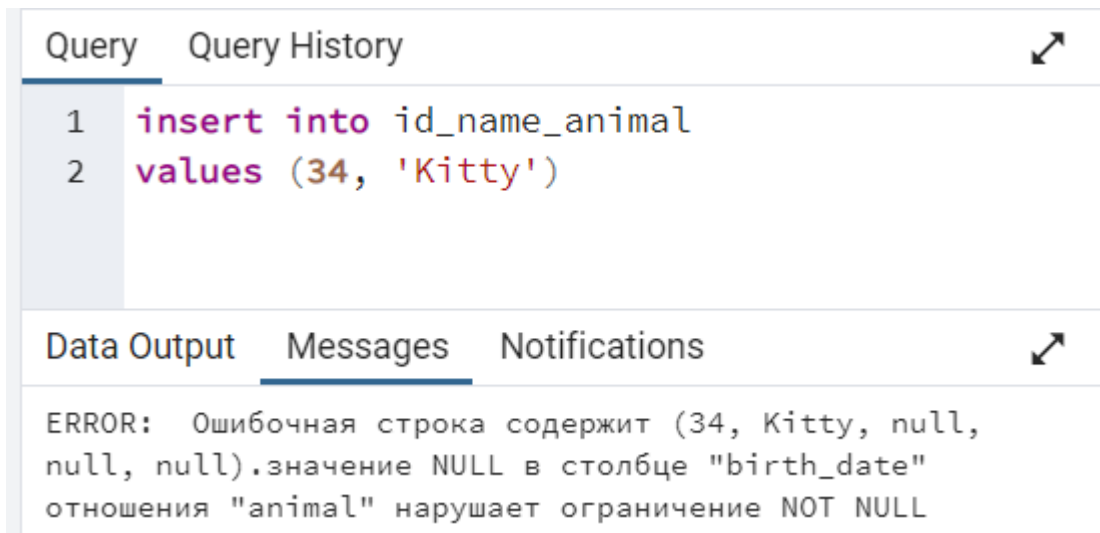
Рис. 17 – создание представления

- Продемонстрировать изменение и вставку данных через представления.

Query	Query History
1	insert into animals_age
2	values('Pupy', 'Fagty', INTERVAL '1 day')
3	

Data Output	Messages	Notifications
ERROR: Представления, выбирающие данные не из одной таблицы или представления, не обновляются автоматически.вставить данные в представление "animals_age" нельзя		
ОШИБКА: вставить данные в представление "animals_age" нельзя		
SQL state: 55000		

Рис. 18 – попытка вставки данных



The screenshot shows a database query interface with two tabs: "Query" and "Query History". The "Query" tab is active, displaying a SQL insert statement:

```
1 insert into id_name_animal
2 values (34, 'Kitty')
```

Below the query, there are three tabs: "Data Output", "Messages", and "Notifications". The "Messages" tab is active, showing an error message in Russian:

```
ERROR: Ошибочная строка содержит (34, Kitty, null,
null, null).значение NULL в столбце "birth_date"
отношения "animal" нарушает ограничение NOT NULL
```

Рис. 19 – попытка вставки данных

Не можем добавлять данные с помощью представлений из-за установленных нами ограничений



The screenshot shows a database query interface with two tabs: "Query" and "Query History". The "Query" tab is active, displaying a SQL insert statement:

```
1 insert into guides (employee_name, employee_role, enclosure_id)
2 values ('Adam Sandler', 'photo', 12)
3
```

Below the query, there are three tabs: "Data Output", "Messages", and "Notifications". The "Messages" tab is active, showing the result of the insert operation:

```
INSERT 0 1

Query returned successfully in 34 msec.
```

Рис. 20 – вставка данных

Вставили значения, которые не удовлетворяют начальным условиям WHERE. Поэтому нужно делать через WITH CHECK OPTION, использование этого предложения проверяет каждую вставляемую строку на удовлетворение условий предложения WHERE.

Query	Query History	
1	create view guides_check as	
2	select employee_id, employee_name,	
3	employee_role, enclosure_id	
4	from employee	
5	where employee_role = 'guide'	
6	WITH CHECK OPTION;	
Data Output	Messages	Notifications
CREATE VIEW		

Рис. 21 – создание представления с check option

Query	Query History
1	insert into guides_check (employee_name, employee_role, enclosure_id)
2	values ('Adam Sendler', 'photo', 10)
3	

Data Output	Messages	Notifications
ERROR: Ошибочная строка содержит (54, Adam Sendler, photo, 10).новая строка нарушает ограничение-проверку для представления "guides_check"		
ОШИБКА: новая строка нарушает ограничение-проверку для представления "guides_check"		
SQL state: 44000		
Detail: Ошибочная строка содержит (54, Adam Sendler, photo, 10).		

Рис. 22 – вставка данных с проверкой

Вставку строк в таблицу, на которой основано представление, нельзя выполнить, если это представление содержит одну из следующих возможностей:

- предложение FROM в определении представления содержит более чем одну таблицу, и список столбцов содержит столбцы более чем из одной таблицы;
- столбец в представлении создается из агрегатной функции;
- инструкция SELECT в представлении содержит предложение GROUP BY или параметр DISTINCT;
- столбец в представлении создается из константы или выражения.

Query	Query History
1	update id_name_animal
2	set animal_name = 'photo'
3	where animal_id = 15

Data Output	Messages	Notifications
UPDATE 1		

Рис. 23 – обновление данных

Логическое значение предложения WITH CHECK OPTION для инструкции UPDATE имеет такое же значение, как и для инструкции INSERT

- **Продемонстрировать невозможность изменения данных через представление.**

Query	Query History	
1	update guides_check	
2	set employee_role = 'photo'	
3	where employee_id > 15	

Data Output	Messages	Notifications	
ERROR: Ошибочная строка содержит (22, Kathryn Thompson, photo, 5).новая строка нарушает ограничение-проверку для представления "guides_check"			

Рис. 24 – невозможность обновления данных

Другие случаи были перечислены выше.

- **Продемонстрировать полезность материализованного представления.**

Материализованные представления основаны на системе правил, как и представления, но их содержимое сохраняется как таблица.

Основное отличие между:

CREATE MATERIALIZED VIEW mymatview AS SELECT * FROM mytab;

и этой командой:

CREATE TABLE mymatview AS SELECT * FROM mytab;

состоит в том, что материализованное представление впоследствии нельзя будет изменить непосредственно, а запрос, создающий материализованное

представление, сохраняется точно так же, как запрос представления, и получить актуальные данные в материализованном представлении можно так:

```
REFRESH MATERIALIZED VIEW mymatview;
```



Рис. 25 – создание материализованного представления

Скорость для SELECT запроса

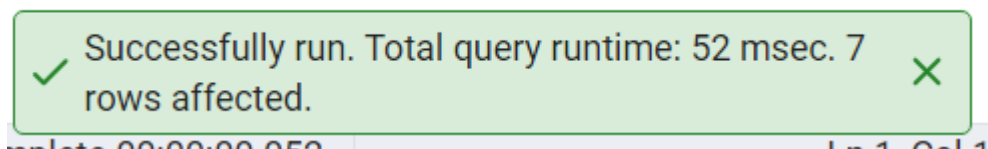


Рис. 26 – скорость для select запроса

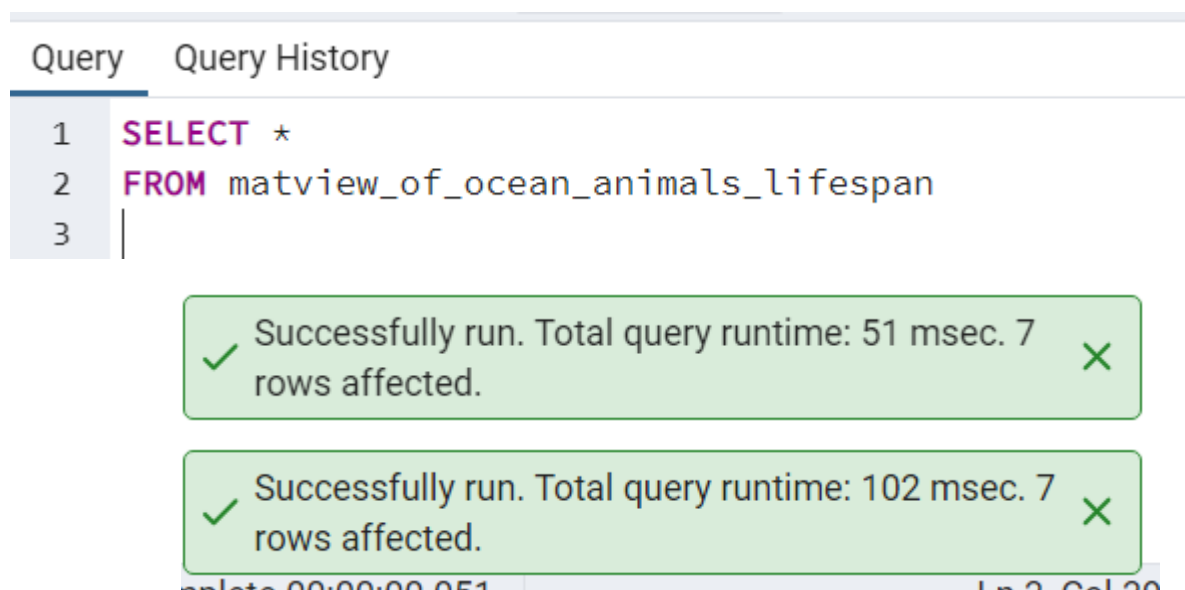


Рис. 27 – скорость для материализованного представления

5. Транзакции

- «грязное» чтение — чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится (откатится);
- неповторяющееся чтение — при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными;
- фантомное чтение — одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям. Другая транзакция в интервалах между этими выборками добавляет строки или изменяет столбцы некоторых строк, используемых в критериях выборки первой транзакции, и успешно заканчивается. В результате получится, что одни и те же выборки в первой транзакции дают разные множества строк.

Задание:

- Установить в обоих сеансах уровень изоляции READ UNCOMMITTED.

```
zoo=# BEGIN ISOLATION LEVEL READ UNCOMMITTED;
BEGIN
zoo=# UPDATE animal SET birth_date = DATE('2004-11-15') WHERE animal_id = 13;
UPDATE 1
zoo=# SELECT * FROM animal WHERE animal_id = 13;
 animal_id | animal_name | birth_date | species_id | enclosure_id
-----+-----+-----+-----+-----
        13 | Vicki      | 2004-11-15 |          21 |          15
(1 ÷÷÷÷÷)
```

Рис. 28 – Транзакция 1

```
zoo=# BEGIN ISOLATION LEVEL READ UNCOMMITTED;
BEGIN
zoo=# SELECT * FROM animal WHERE animal_id = 13;
 animal_id | animal_name | birth_date | species_id | enclosure_id
-----+-----+-----+-----+-----
        13 | Vicki      | 1990-08-07 |          21 |          15
(1 ÷÷÷÷÷)
```

Рис. 29 – Транзакция 2

Чтение «грязных» данных в PostgreSQL на этом уровне не допускается.

- Установить в обоих сеансах уровень изоляции READ COMMITTED.


```

zoo=# BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN
zoo=# UPDATE employee
zoo-*= SET employee_role = 'photo'
zoo-*= WHERE employee_role = 'clean'
zoo-*= RETURNING *;

```

employee_id	employee_name	employee_role	enclosure_id
18	Brandon Miller	photo	4
24	Taylor Smith	photo	4
31	Mrs. Lindsay Bauer	photo	25
35	Joshua Hanson	photo	14

```

(4 rows)

```

Рис. 37 – Транзакция 2

Изменение, произведенное в первой транзакции, вторая транзакция не видит, поскольку на уровне изоляции **SERIALIZABLE** каждая транзакция работает с тем снимком базы данных, которых был сделан в ее начале, т. е. непосредственно перед выполнением ее первого оператора.

Вывод: были освоены навыки проектирования БД, создания и заполнение базы. Научились добавлять, удалять и изменять записи в таблице. Были отработаны навыки работы с транзакциями.