**Приложение Б**

Листинг процедур

CREATE OR REPLACE PROCEDURE generate\_timetable(

film\_title VARCHAR(100),

hall\_name VARCHAR(100),

start\_date DATE,

end\_date DATE,

time\_coefficients JSONB,

status VARCHAR(20)

)

AS $$

DECLARE

film\_id INTEGER;

hall\_id INTEGER;

curr\_date DATE;

start\_time TIME;

time\_coefficient FLOAT;

details\_json JSONB;

BEGIN

-- Проверка существования фильма в таблице info\_films

SELECT id\_film INTO film\_id FROM info\_films WHERE title = film\_title;

IF NOT FOUND THEN

RAISE EXCEPTION 'Фильм с названием "%", указанным в параметрах, не найден в базе данных.', film\_title;

END IF;

-- Проверка существования зала в таблице halls

SELECT id\_hall INTO hall\_id FROM halls WHERE name = hall\_name;

IF NOT FOUND THEN

RAISE EXCEPTION 'Зал с названием "%", указанным в параметрах, не найден в базе данных.', hall\_name;

END IF;

curr\_date := start\_date;

-- Цикл по датам

WHILE curr\_date <= end\_date LOOP

-- Парсинг временных коэффициентов

FOR i IN 0..jsonb\_array\_length(time\_coefficients)-1 LOOP

start\_time := (time\_coefficients->i->>'time')::TIME;

time\_coefficient := (time\_coefficients->i->>'coefficient')::FLOAT;

-- Создание JSONB-объекта для поля details

details\_json := jsonb\_build\_object('date', to\_char(curr\_date, 'YYYY-MM-DD'), 'time', to\_char(start\_time, 'HH24:MI'), 'coefficient', time\_coefficient);

-- Вставка записи в таблицу "Расписание"

INSERT INTO timetable (id\_film, id\_hall, details, status)

VALUES (film\_id, hall\_id, details\_json, status);

END LOOP;

curr\_date := curr\_date + 1; -- Переход к следующей дате

END LOOP;

RAISE NOTICE 'Расписание успешно сгенерировано.';

EXCEPTION

WHEN OTHERS THEN

RAISE NOTICE 'Ошибка: %', SQLERRM;

END;

$$ LANGUAGE plpgsql;

Листинг Б.1 – процедура generate\_timetable

CREATE OR REPLACE PROCEDURE delete\_movie\_from\_timetable(

film\_title VARCHAR(100),

hall\_name VARCHAR(100),

theater\_name VARCHAR(100)

)

AS $$

DECLARE

film\_id INTEGER;

hall\_id INTEGER;

theater\_id INTEGER;

BEGIN

-- Получение идентификатора фильма по названию

SELECT id\_film INTO film\_id FROM info\_films WHERE title = film\_title;

-- Получение идентификатора зала и кинотеатра по их названиям

SELECT h.id\_hall, t.id\_theater INTO hall\_id, theater\_id

FROM halls h

INNER JOIN theaters t ON h.id\_theater = t.id\_theater

WHERE h.name = hall\_name AND t.name = theater\_name;

-- Проверка наличия записи о фильме в заданном зале и кинотеатре

IF NOT EXISTS (

SELECT 1 FROM timetable

WHERE id\_film = film\_id AND id\_hall = hall\_id

) THEN

RAISE NOTICE 'Некорректные данные. Фильм "%s" не показывается в зале "%s" кинотеатра "%s".', film\_title, hall\_name, theater\_name;

RETURN;

END IF;

-- Удаление связанных билетов из таблицы "tickets"

DELETE FROM tickets

WHERE id\_timetable IN (

SELECT id\_timetable FROM timetable

WHERE id\_film = film\_id AND id\_hall = hall\_id

);

-- Удаление записи из таблицы "timetable" по заданным условиям

DELETE FROM timetable

WHERE id\_film = film\_id AND id\_hall = hall\_id;

RAISE NOTICE 'Запись о фильме "%", показываемом в зале "%", кинотеатра "%", успешно удалена из расписания.', film\_title, hall\_name, theater\_name;

END;

$$ LANGUAGE plpgsql;

Листинг Б.2 – процедура delete\_movie\_from\_timetable

CREATE OR REPLACE PROCEDURE update\_timetable\_entry(

film\_title VARCHAR(100),

hall\_name VARCHAR(100),

start\_date DATE,

end\_date DATE,

time\_coefficients JSONB,

status VARCHAR(20)

)

AS $$

DECLARE

film\_id INTEGER;

hall\_id INTEGER;

curr\_date DATE;

start\_time TIME;

time\_coefficient FLOAT;

details\_json JSONB;

BEGIN

-- Проверка существования фильма в таблице info\_films

SELECT id\_film INTO film\_id FROM info\_films WHERE title = film\_title;

IF NOT FOUND THEN

RAISE EXCEPTION 'Фильм с названием "%", указанным в параметрах, не найден в базе данных.', film\_title;

END IF;

-- Проверка существования зала в таблице halls

SELECT id\_hall INTO hall\_id FROM halls WHERE name = hall\_name;

IF NOT FOUND THEN

RAISE EXCEPTION 'Зал с названием "%", указанным в параметрах, не найден в базе данных.', hall\_name;

END IF;

curr\_date := start\_date;

DELETE FROM tickets

WHERE id\_timetable IN (

SELECT id\_timetable FROM timetable

WHERE id\_film = film\_id AND id\_hall = hall\_id

AND details->>'date' >= to\_char(start\_date, 'YYYY-MM-DD')

AND details->>'date' <= to\_char(end\_date, 'YYYY-MM-DD')

);

-- Удаление существующих записей в таблице "Расписание" для указанного фильма, зала и периода

DELETE FROM timetable

WHERE id\_film = film\_id AND id\_hall = hall\_id AND details->>'date' >= to\_char(start\_date, 'YYYY-MM-DD') AND details->>'date' <= to\_char(end\_date, 'YYYY-MM-DD');

-- Цикл по датам

WHILE curr\_date <= end\_date LOOP

-- Парсинг временных коэффициентов

FOR i IN 0..jsonb\_array\_length(time\_coefficients)-1 LOOP

start\_time := (time\_coefficients->i->>'time')::TIME;

time\_coefficient := (time\_coefficients->i->>'coefficient')::FLOAT;

-- Создание JSONB-объекта для поля details

details\_json := jsonb\_build\_object('date', to\_char(curr\_date, 'YYYY-MM-DD'), 'time', to\_char(start\_time, 'HH24:MI'), 'coefficient', time\_coefficient);

-- Вставка записи в таблицу "Расписание"

INSERT INTO timetable (id\_film, id\_hall, details, status)

VALUES (film\_id, hall\_id, details\_json, status);

END LOOP;

curr\_date := curr\_date + 1; -- Переход к следующей дате

END LOOP;

RAISE NOTICE 'Расписание успешно обновлено.';

EXCEPTION

WHEN OTHERS THEN

RAISE NOTICE 'Ошибка: %', SQLERRM;

END;

$$ LANGUAGE plpgsql;

Листинг Б.3 – процедура update\_timetable\_entry

CREATE OR REPLACE PROCEDURE add\_movie\_to\_catalog(

movie\_title VARCHAR(255),

movie\_genre VARCHAR(255),

movie\_rating FLOAT,

movie\_description VARCHAR(2000)

)

security definer

AS $$

BEGIN

IF EXISTS (SELECT 1 FROM info\_films WHERE title = movie\_title) THEN

RAISE EXCEPTION 'Фильм с названием "%", уже присутствует в каталоге.', movie\_title;

END IF;

INSERT INTO info\_films (title, genre, rating, description)

VALUES (movie\_title, movie\_genre, movie\_rating, movie\_description);

RAISE NOTICE 'Фильм "%", успешно добавлен в каталог.', movie\_title;

END;

$$ LANGUAGE plpgsql;

Листинг Б.4 – процедура add\_movie\_to\_catalog

CREATE OR REPLACE PROCEDURE delete\_movie\_from\_catalog(

film\_title VARCHAR(255)

)

security definer

AS $$

BEGIN

IF NOT EXISTS (SELECT 1 FROM info\_films WHERE title = film\_title) THEN

RAISE EXCEPTION 'Фильм "%" не найден в каталоге.', film\_title;

END IF;

DELETE FROM info\_films

WHERE title = film\_title;

RAISE NOTICE 'Фильм "%", успешно удален из каталога.', film\_title;

END;

$$ LANGUAGE plpgsql;

Листинг Б.5 – процедура delete\_movie\_from\_catalog

CREATE OR REPLACE PROCEDURE add\_administrator(

admin\_name VARCHAR(255),

admin\_password VARCHAR(255),

admin\_email VARCHAR(255)

)

security definer

AS $$

BEGIN

IF EXISTS (SELECT 1 FROM administrators WHERE name = admin\_name) THEN

RAISE EXCEPTION 'Администратор с именем "%s" уже существует.', admin\_name;

END IF;

IF EXISTS (SELECT 1 FROM administrators WHERE email = admin\_email) THEN

RAISE EXCEPTION 'Администратор с почтой "%s" уже существует.', admin\_email;

END IF;

INSERT INTO administrators (name, password, email) VALUES (admin\_name, admin\_password, admin\_email);

RAISE NOTICE 'Администратор "%s" успешно добавлен.', admin\_name;

END;

$$ LANGUAGE plpgsql;

Листинг Б.6 – процедура add\_administrator

CREATE OR REPLACE PROCEDURE update\_hall\_def\_price(

p\_hall\_name VARCHAR(255),

p\_theater\_name VARCHAR(255),

p\_new\_def\_price NUMERIC(10, 2)

)

security definer

AS $$

DECLARE

hall\_id INTEGER;

theater\_id INTEGER;

BEGIN

-- Получаем id кинотеатра по его названию

SELECT id\_theater INTO theater\_id

FROM theaters

WHERE name = p\_theater\_name;

-- Проверяем, был ли найден кинотеатр

IF theater\_id IS NULL THEN

RAISE EXCEPTION 'Кинотеатр с названием "%" не найден.', p\_theater\_name;

END IF;

-- Получаем id зала по его названию и id кинотеатра

SELECT id\_hall INTO hall\_id

FROM halls

WHERE name = p\_hall\_name

AND id\_theater = theater\_id;

-- Проверяем, был ли найден зал

IF hall\_id IS NULL THEN

RAISE EXCEPTION 'Зал с названием "%" в кинотеатре "%" не найден.', p\_hall\_name, p\_theater\_name;

END IF;

-- Обновляем значение def\_price

UPDATE halls

SET def\_price = p\_new\_def\_price

WHERE id\_hall = hall\_id

AND id\_theater = theater\_id;

-- Выводим сообщение об успешном обновлении

RAISE NOTICE 'Значение def\_price для зала "%" в кинотеатре "%" успешно обновлено на %.', p\_hall\_name, p\_theater\_name, p\_new\_def\_price;

END;

$$ LANGUAGE plpgsql;

Листинг Б.7 – процедура update\_hall\_def\_price

CREATE OR REPLACE PROCEDURE delete\_admin (

admin\_name VARCHAR(255),

admin\_password VARCHAR(255),

admin\_email VARCHAR(255)

)

security definer

AS $$

BEGIN

IF EXISTS (SELECT 1 FROM administrators WHERE name = admin\_name) THEN

RAISE EXCEPTION 'Администратор с именем "%s" уже существует.', admin\_name;

END IF;

IF EXISTS (SELECT 1 FROM administrators WHERE email = admin\_email) THEN

RAISE EXCEPTION 'Администратор с почтой "%s" уже существует.', admin\_email;

END IF;

INSERT INTO administrators (name, password, email) VALUES (admin\_name, admin\_password, admin\_email);

RAISE NOTICE 'Администратор "%s" успешно добавлен.', admin\_name;

END;

$$ LANGUAGE plpgsql;

Листинг Б.8 – процедура delete\_admin

CREATE OR REPLACE PROCEDURE add\_administrator(

admin\_name VARCHAR(255),

admin\_password VARCHAR(255),

admin\_email VARCHAR(255)

)

security definer

AS $$

BEGIN

IF EXISTS (SELECT 1 FROM administrators WHERE name = admin\_name) THEN

RAISE EXCEPTION 'Администратор с именем "%s" уже существует.', admin\_name;

END IF;

IF EXISTS (SELECT 1 FROM administrators WHERE email = admin\_email) THEN

RAISE EXCEPTION 'Администратор с почтой "%s" уже существует.', admin\_email;

END IF;

INSERT INTO administrators (name, password, email) VALUES (admin\_name, admin\_password, admin\_email);

RAISE NOTICE 'Администратор "%s" успешно добавлен.', admin\_name;

END;

$$ LANGUAGE plpgsql;

Листинг Б.6 – процедура add\_administrator

CREATE OR REPLACE PROCEDURE add\_administrator(

admin\_name VARCHAR(255),

admin\_password VARCHAR(255),

admin\_email VARCHAR(255)

)

security definer

AS $$

BEGIN

IF EXISTS (SELECT 1 FROM administrators WHERE name = admin\_name) THEN

RAISE EXCEPTION 'Администратор с именем "%s" уже существует.', admin\_name;

END IF;

IF EXISTS (SELECT 1 FROM administrators WHERE email = admin\_email) THEN

RAISE EXCEPTION 'Администратор с почтой "%s" уже существует.', admin\_email;

END IF;

INSERT INTO administrators (name, password, email) VALUES (admin\_name, admin\_password, admin\_email);

RAISE NOTICE 'Администратор "%s" успешно добавлен.', admin\_name;

END;

$$ LANGUAGE plpgsql;

Листинг Б.6 – процедура add\_administrator

CREATE OR REPLACE PROCEDURE add\_administrator(

admin\_name VARCHAR(255),

admin\_password VARCHAR(255),

admin\_email VARCHAR(255)

)

security definer

AS $$

BEGIN

IF EXISTS (SELECT 1 FROM administrators WHERE name = admin\_name) THEN

RAISE EXCEPTION 'Администратор с именем "%s" уже существует.', admin\_name;

END IF;

IF EXISTS (SELECT 1 FROM administrators WHERE email = admin\_email) THEN

RAISE EXCEPTION 'Администратор с почтой "%s" уже существует.', admin\_email;

END IF;

INSERT INTO administrators (name, password, email) VALUES (admin\_name, admin\_password, admin\_email);

RAISE NOTICE 'Администратор "%s" успешно добавлен.', admin\_name;

END;

$$ LANGUAGE plpgsql;

Листинг Б.6 – процедура add\_administrator

CREATE OR REPLACE PROCEDURE add\_administrator(

admin\_name VARCHAR(255),

admin\_password VARCHAR(255),

admin\_email VARCHAR(255)

)

security definer

AS $$

BEGIN

IF EXISTS (SELECT 1 FROM administrators WHERE name = admin\_name) THEN

RAISE EXCEPTION 'Администратор с именем "%s" уже существует.', admin\_name;

END IF;

IF EXISTS (SELECT 1 FROM administrators WHERE email = admin\_email) THEN

RAISE EXCEPTION 'Администратор с почтой "%s" уже существует.', admin\_email;

END IF;

INSERT INTO administrators (name, password, email) VALUES (admin\_name, admin\_password, admin\_email);

RAISE NOTICE 'Администратор "%s" успешно добавлен.', admin\_name;

END;

$$ LANGUAGE plpgsql;

Листинг Б.6 – процедура add\_administrator

**Приложение А**

Листинг таблиц

CREATE TABLE info\_films (

id\_film SERIAL PRIMARY KEY,

title VARCHAR(255) NOT NULL,

genre VARCHAR(255) NOT NULL,

rating FLOAT NOT NULL,

description VARCHAR(2000)

);

Листинг А.1 – Таблица INFO\_FILMS

CREATE TABLE ratings (

id\_rating SERIAL PRIMARY KEY,

id\_film INTEGER REFERENCES info\_films (id\_film),

id\_customer INTEGER REFERENCES customers (id\_customer),

rating FLOAT NOT NULL

);

Листинг А.2 – Таблица RATINGS

CREATE TABLE administrators (

id\_admin SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

password VARCHAR(255) NOT NULL,

email VARCHAR(255)

);

Листинг А.3 – Таблица ADMINISTRATORS

CREATE TABLE customers (

id\_customer SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

Листинг А.4 – Таблица CUSTOMERS

CREATE TABLE theaters (

id\_theater SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

address VARCHAR(255) NOT NULL,

admin serial REFERENCES administrators (id\_admin)

);

Листинг А.5 – Таблица THEATERS

CREATE TABLE halls (

id\_hall SERIAL PRIMARY KEY,

id\_theater INTEGER REFERENCES theaters (id\_theater),

name VARCHAR(255) NOT NULL,

rows integer not null,

seats integer not null,

def\_price NUMERIC(10, 2) NOT NULL,

bestrows jsonb,

bestseats jsonb,

coefficient NUMERIC(10, 2) not null

);

Листинг А.6 – Таблица HALLS

CREATE TABLE timetable (

id\_timetable SERIAL PRIMARY KEY,

id\_film INTEGER REFERENCES info\_films (id\_film),

id\_hall INTEGER REFERENCES halls (id\_hall),

details JSONB,

status VARCHAR(20)

);

Листинг А.7 – Таблица TIMETABLE

CREATE TABLE orders (

id\_ord SERIAL PRIMARY KEY,

ord\_date TIMESTAMP without time zone,

id\_customer INTEGER REFERENCES customers (id\_customer)

);

Листинг А.8 – Таблица ORDERS

CREATE TABLE ord\_det (

id\_ord\_det SERIAL PRIMARY KEY,

id\_ord INTEGER REFERENCES orders (id\_ord),

id\_row INTEGER,

id\_seat INTEGER,

id\_timetable INTEGER REFERENCES timetable (id\_timetable),

FOREIGN KEY (id\_row, id\_seat, id\_timetable) REFERENCES tickets (id\_row, id\_seat, id\_timetable)

);

Листинг А.9 – Таблица ORD\_DET

CREATE TABLE tickets (

id\_row SERIAL,

id\_seat SERIAL,

id\_timetable SERIAL REFERENCES timetable (id\_timetable),

price NUMERIC(10, 2),

status INTEGER CHECK (status IN (0, 1)),

PRIMARY KEY (id\_row, id\_seat, id\_timetable)

);

Листинг А.10 – Таблица TICKETS

CREATE TABLE history (

id\_change SERIAL PRIMARY KEY,

change\_date DATE,

description VARCHAR(255),

previous\_admin VARCHAR(255),

new\_admin VARCHAR(255)

);

Листинг А.11 – Таблица HISTORY