**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

**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\_rating\_by\_title(

film\_title VARCHAR(255),

user\_name VARCHAR(255),

user\_rating FLOAT

)

AS $$

DECLARE

film\_id INTEGER;

user\_id INTEGER;

BEGIN

-- Получаем идентификатор фильма по его названию

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

-- Получаем идентификатор пользователя по его имени

SELECT id\_customer INTO user\_id FROM customers WHERE name = user\_name;

-- Проверяем, существует ли пользователь и фильм

IF film\_id IS NULL THEN

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

END IF;

IF user\_id IS NULL THEN

RAISE EXCEPTION 'Пользователь с именем "%", указанным в параметрах, не найден в базе данных.', user\_name;

END IF;

-- Проверяем, существует ли уже рейтинг для этого фильма от этого пользователя

IF EXISTS (SELECT 1 FROM ratings WHERE id\_film = film\_id AND id\_customer = user\_id) THEN

RAISE EXCEPTION 'Рейтинг для фильма "%", уже существует от пользователя "%".', film\_title, user\_name;

END IF;

-- Добавляем рейтинг в таблицу ratings

INSERT INTO ratings (id\_film, id\_customer, rating) VALUES (film\_id, user\_id, user\_rating);

RAISE NOTICE 'Рейтинг для фильма "%", добавлен успешно.', film\_title;

EXCEPTION

WHEN OTHERS THEN

RAISE NOTICE 'Ошибка: %', SQLERRM;

END;

$$ LANGUAGE plpgsql;

Листинг Б.9 – процедура add\_rating\_by\_title

CREATE OR REPLACE PROCEDURE buy\_ticket(

p\_movie\_title VARCHAR(255),

p\_theater\_name VARCHAR(255),

p\_hall\_name VARCHAR(255),

p\_date DATE,

p\_time TIME,

p\_row INTEGER,

p\_seat INTEGER,

p\_customer\_name VARCHAR(255)

)

AS $$

DECLARE

movie\_id INTEGER;

theater\_id INTEGER;

hall\_id INTEGER;

timetable\_id INTEGER;

ticket\_price NUMERIC(10, 2);

customer\_id INTEGER;

order\_id INTEGER;

BEGIN

-- Проверка доступности билета

SELECT t.id\_timetable, tk.price

INTO timetable\_id, ticket\_price

FROM timetable t

JOIN halls h ON t.id\_hall = h.id\_hall

JOIN theaters th ON h.id\_theater = th.id\_theater

JOIN info\_films f ON t.id\_film = f.id\_film

JOIN tickets tk ON tk.id\_timetable = t.id\_timetable

WHERE f.title = p\_movie\_title

AND th.name = p\_theater\_name

AND h.name = p\_hall\_name

AND t.details->>'date' = to\_char(p\_date, 'YYYY-MM-DD')

AND t.details->>'time' = to\_char(p\_time, 'HH24:MI')

AND tk.id\_row = p\_row

AND tk.id\_seat = p\_seat

AND tk.status = 1;

IF NOT FOUND THEN

RAISE EXCEPTION 'Билет на указанный фильм, кинотеатр, зал, дату, время, ряд и место недоступен или не существует.';

END IF;

-- Получение ID покупателя

SELECT id\_customer INTO customer\_id FROM customers WHERE name = p\_customer\_name;

IF NOT FOUND THEN

RAISE EXCEPTION 'Покупатель с именем "%", указанным в параметрах, не найден в базе данных.', p\_customer\_name;

END IF;

-- Создание заказа

INSERT INTO orders (ord\_date, id\_customer)

VALUES (CURRENT\_DATE, customer\_id)

RETURNING id\_ord INTO order\_id;

-- Создание записи о купленном билете

INSERT INTO ord\_det (id\_ord, id\_row, id\_seat, id\_timetable)

VALUES (order\_id, p\_row, p\_seat, timetable\_id);

-- Обновление статуса билета на "куплен"

UPDATE tickets

SET status = 0

WHERE id\_row = p\_row

AND id\_seat = p\_seat

AND id\_timetable = timetable\_id;

-- Вывод сообщения о стоимости билета

RAISE NOTICE 'Билет куплен. Стоимость: %', ticket\_price;

END;

$$ LANGUAGE plpgsql;

Листинг Б.10 – процедура buy\_ticket

CREATE OR REPLACE PROCEDURE return\_ticket(

p\_movie\_title VARCHAR(255),

p\_theater\_name VARCHAR(255),

p\_hall\_name VARCHAR(255),

p\_date DATE,

p\_time TIME,

p\_row INTEGER,

p\_seat INTEGER,

p\_customer\_name VARCHAR(255)

)

security definer

AS $$

DECLARE

\_customer\_id INTEGER;

\_ticket\_id\_row INTEGER;

\_ticket\_id\_seat INTEGER;

\_ticket\_id\_timetable INTEGER;

\_ticket\_price NUMERIC(10, 2);

\_order\_id INTEGER;

BEGIN

-- Получаем ID покупателя

SELECT id\_customer INTO \_customer\_id FROM customers WHERE name = p\_customer\_name;

IF NOT FOUND THEN

RAISE EXCEPTION 'Покупатель с именем "%" не найден в базе данных.', p\_customer\_name;

END IF;

-- Получаем информацию о билете

SELECT tk.id\_row, tk.id\_seat, tk.id\_timetable, tk.price, o.id\_ord

INTO \_ticket\_id\_row, \_ticket\_id\_seat, \_ticket\_id\_timetable, \_ticket\_price, \_order\_id

FROM tickets tk

JOIN ord\_det od ON tk.id\_row = od.id\_row AND tk.id\_seat = od.id\_seat AND tk.id\_timetable = od.id\_timetable

JOIN orders o ON od.id\_ord = o.id\_ord

JOIN timetable t ON tk.id\_timetable = t.id\_timetable

JOIN halls h ON t.id\_hall = h.id\_hall

JOIN theaters th ON h.id\_theater = th.id\_theater

JOIN info\_films f ON t.id\_film = f.id\_film

WHERE f.title = p\_movie\_title

AND th.name = p\_theater\_name

AND h.name = p\_hall\_name

AND t.details->>'date' = to\_char(p\_date, 'YYYY-MM-DD')

AND t.details->>'time' = to\_char(p\_time, 'HH24:MI')

AND tk.id\_row = p\_row

AND tk.id\_seat = p\_seat;

IF NOT FOUND THEN

RAISE EXCEPTION 'Билет на указанный фильм, кинотеатр, зал, дату, время, ряд и место недоступен или не существует.';

END IF;

-- Проверяем, был ли билет куплен данным пользователем

IF NOT EXISTS (

SELECT 1

FROM ord\_det od

JOIN orders o ON od.id\_ord = o.id\_ord

WHERE od.id\_row = \_ticket\_id\_row

AND od.id\_seat = \_ticket\_id\_seat

AND od.id\_timetable = \_ticket\_id\_timetable

AND o.id\_customer = \_customer\_id

) THEN

RAISE EXCEPTION 'Билет не может быть возвращен, так как он зарегестрирован не на Вас.';

END IF;

-- Удаляем запись о билете из таблицы ord\_det

DELETE FROM ord\_det

WHERE id\_row = \_ticket\_id\_row

AND id\_seat = \_ticket\_id\_seat

AND id\_timetable = \_ticket\_id\_timetable;

-- Обновляем статус билета на доступный

UPDATE tickets

SET status = 1

WHERE id\_row = \_ticket\_id\_row

AND id\_seat = \_ticket\_id\_seat

AND id\_timetable = \_ticket\_id\_timetable;

-- Удаляем заказ

DELETE FROM orders WHERE id\_ord = \_order\_id;

-- Выводим сообщение о стоимости билета

RAISE NOTICE 'Билет успешно возвращен. Вам возвращено % рублей.', \_ticket\_price;

END;

$$ LANGUAGE plpgsql;

Листинг Б.11 – процедура return\_ticket

CREATE OR REPLACE PROCEDURE cancel\_ticket(

p\_movie\_title VARCHAR(255),

p\_theater\_name VARCHAR(255),

p\_hall\_name VARCHAR(255),

p\_date DATE,

p\_time TIME,

p\_row INTEGER,

p\_seat INTEGER

)

security definer

AS $$

DECLARE

movie\_id INTEGER;

theater\_id INTEGER;

hall\_id INTEGER;

timetable\_id INTEGER;

ticket\_price NUMERIC(10, 2);

BEGIN

-- Проверка доступности билета

SELECT t.id\_timetable, tk.price

INTO timetable\_id, ticket\_price

FROM timetable t

JOIN halls h ON t.id\_hall = h.id\_hall

JOIN theaters th ON h.id\_theater = th.id\_theater

JOIN info\_films f ON t.id\_film = f.id\_film

JOIN tickets tk ON tk.id\_timetable = t.id\_timetable

WHERE f.title = p\_movie\_title

AND th.name = p\_theater\_name

AND h.name = p\_hall\_name

AND t.details->>'date' = to\_char(p\_date, 'YYYY-MM-DD')

AND t.details->>'time' = to\_char(p\_time, 'HH24:MI')

AND tk.id\_row = p\_row

AND tk.id\_seat = p\_seat

AND tk.status = 1;

IF NOT FOUND THEN

RAISE EXCEPTION 'Билет на указанный фильм, кинотеатр, зал, дату, время, ряд и место недоступен или не существует.';

END IF;

-- Обновление статуса билета на "куплен"

UPDATE tickets

SET status = 0

WHERE id\_row = p\_row

AND id\_seat = p\_seat

AND id\_timetable = timetable\_id;

-- Вывод сообщения о стоимости билета

RAISE NOTICE 'Билет удален из доступа.';

END;

$$ LANGUAGE plpgsql;

Листинг Б.12 – процедура cancel\_ticket

CREATE OR REPLACE PROCEDURE ExportCountTicketsFromTheaterToXML(

file\_path TEXT

)

security definer

AS $$

DECLARE

xml\_data TEXT := '<?xml version="1.0" encoding="UTF-8"?><TheatersTickets>';

theater\_movie\_ticket\_rec RECORD;

BEGIN

FOR theater\_movie\_ticket\_rec IN SELECT \* FROM temp\_theater\_movie\_tickets LOOP

xml\_data := xml\_data || '<theater\_movie\_ticket>';

xml\_data := xml\_data || '<theater\_name>' || theater\_movie\_ticket\_rec.theater\_name || '</theater\_name>';

xml\_data := xml\_data || '<movie\_title>' || theater\_movie\_ticket\_rec.movie\_title || '</movie\_title>';

xml\_data := xml\_data || '<total\_tickets\_sold>' || theater\_movie\_ticket\_rec.total\_tickets\_sold || '</total\_tickets\_sold>';

xml\_data := xml\_data || '<from\_date>' || theater\_movie\_ticket\_rec.first\_purchase\_date || '</from\_date>';

xml\_data := xml\_data || '<to\_date>' || theater\_movie\_ticket\_rec.last\_purchase\_date || '</to\_date>';

xml\_data := xml\_data || '</theater\_movie\_ticket>';

END LOOP;

xml\_data := xml\_data || '</TheaterTickets>';

EXECUTE format('COPY (SELECT %L) TO %L', xml\_data, file\_path);

RAISE NOTICE 'Данные успешно загружены в XML файл: %', file\_path;

EXCEPTION

WHEN OTHERS THEN

RAISE NOTICE 'Ошибка при экспортировке данных в XML: %', SQLERRM;

END;

$$ LANGUAGE plpgsql;

Листинг Б.13 – процедура ExportCountTicketsFromTheaterToXML

CREATE OR REPLACE PROCEDURE ImportFilmsFromXML(file\_path VARCHAR)

security definer

AS $$

DECLARE

xml\_data TEXT;

BEGIN

-- Read data from the file

xml\_data := pg\_read\_file(file\_path);

-- Check if data is read successfully

IF xml\_data IS NULL THEN

RAISE EXCEPTION 'Failed to read data from file %', file\_path;

END IF;

-- Display the data read from the file for debugging

RAISE INFO 'Данные проверяются из файла: %', xml\_data;

-- Create a temporary table for importing data

CREATE TEMP TABLE tmp\_films (

title VARCHAR(255),

genre VARCHAR(255),

rating INT,

description VARCHAR(2000)

);

-- Insert new data from XML into the temporary table

BEGIN

EXECUTE 'INSERT INTO tmp\_films (title, genre, rating, description)

SELECT

unnest(xpath(''/info\_films/info\_films/title/text()'',

xmlparse(document ''' || xml\_data || ''')))::text AS title,

unnest(xpath(''/info\_films/info\_films/genre/text()'',

xmlparse(document ''' || xml\_data || ''')))::text AS genre,

CAST(TRIM(unnest(xpath(''/info\_films/info\_films/rating/text()'',

xmlparse(document ''' || xml\_data || ''')))::text) AS INT) AS rating,

unnest(xpath(''/info\_films/info\_films/description/text()'',

xmlparse(document ''' || xml\_data || ''')))::text AS description';

EXCEPTION

WHEN OTHERS THEN

RAISE EXCEPTION 'Возникла ошибка с импортом данных из XML: %', SQLERRM;

END;

-- Insert data from the temporary table into info\_films

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

SELECT title, genre, rating, description FROM tmp\_films;

RAISE INFO 'Данный каталог был успешно перенесен из файла % во временную таблицу tmp\_films', file\_path;

END;

$$ LANGUAGE plpgsql;

Листинг Б.14 – процедура ImportFilmsFromXML

**Приложение В**

Листинг функций

CREATE OR REPLACE FUNCTION generate\_tickets()

RETURNS TRIGGER AS $$

DECLARE

hall\_row\_count INTEGER;

hall\_seat\_count INTEGER;

curr\_row INTEGER;

curr\_seat INTEGER;

ticket\_price NUMERIC(10, 2);

hall\_coefficient NUMERIC(10, 2);

timetable\_coefficient NUMERIC(10, 2);

BEGIN

-- Получение количества рядов и мест в зале

SELECT rows, seats, coefficient INTO hall\_row\_count, hall\_seat\_count, hall\_coefficient

FROM halls

WHERE id\_hall = (

SELECT id\_hall FROM timetable WHERE id\_timetable = NEW.id\_timetable

);

curr\_row := 1;

curr\_seat := 1;

-- Получение коэффициента из деталей расписания

SELECT (NEW.details->>'coefficient')::NUMERIC INTO timetable\_coefficient;

-- Генерация записей в таблицу "tickets" для каждого ряда и места

WHILE curr\_row <= hall\_row\_count LOOP

WHILE curr\_seat <= hall\_seat\_count LOOP

-- Получение базовой цены из таблицы halls

SELECT def\_price INTO ticket\_price

FROM halls

WHERE id\_hall = (

SELECT id\_hall FROM timetable WHERE id\_timetable = NEW.id\_timetable

);

-- Проверка совпадения с bestrows и bestseats

IF curr\_row = ANY(ARRAY(SELECT jsonb\_array\_elements\_text(bestrows)::INTEGER FROM halls WHERE id\_hall = NEW.id\_hall)::INTEGER[]) AND

curr\_seat = ANY(ARRAY(SELECT jsonb\_array\_elements\_text(bestseats)::INTEGER FROM halls WHERE id\_hall = NEW.id\_hall)::INTEGER[]) THEN

ticket\_price := ticket\_price \* hall\_coefficient \* timetable\_coefficient;

ELSE

ticket\_price := ticket\_price \* timetable\_coefficient;

END IF;

-- Вставка записи в таблицу "tickets"

INSERT INTO tickets (id\_row, id\_seat, id\_timetable, price, status)

VALUES (curr\_row, curr\_seat, NEW.id\_timetable, ticket\_price, 1);

curr\_seat := curr\_seat + 1; -- Переход к следующему месту

END LOOP;

curr\_row := curr\_row + 1; -- Переход к следующему ряду

curr\_seat := 1; -- Сброс счетчика мест

END LOOP;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

Листинг В.1 – функция generate\_tickets

CREATE OR REPLACE FUNCTION delete\_related\_tickets()

RETURNS TRIGGER AS $$

BEGIN

-- Удаление билетов, связанных с удаляемой записью из таблицы "timetable"

DELETE FROM tickets

WHERE id\_timetable = OLD.id\_timetable;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

Листинг В.2 – функция delete\_related\_tickets

CREATE OR REPLACE FUNCTION update\_tickets\_on\_timetable\_update()

RETURNS TRIGGER AS $$

DECLARE

hall\_row\_count INTEGER;

hall\_seat\_count INTEGER;

curr\_row INTEGER;

curr\_seat INTEGER;

ticket\_price NUMERIC(10, 2);

hall\_coefficient NUMERIC(10, 2);

timetable\_coefficient NUMERIC(10, 2);

BEGIN

-- Получение количества рядов и мест в зале

SELECT rows, seats, coefficient INTO hall\_row\_count, hall\_seat\_count, hall\_coefficient

FROM halls

WHERE id\_hall = (

SELECT id\_hall FROM timetable WHERE id\_timetable = NEW.id\_timetable

);

curr\_row := 1;

curr\_seat := 1;

-- Получение коэффициента из деталей расписания

SELECT (NEW.details->>'coefficient')::NUMERIC INTO timetable\_coefficient;

-- Обновление данных в таблице "tickets" для каждого ряда и места

WHILE curr\_row <= hall\_row\_count LOOP

WHILE curr\_seat <= hall\_seat\_count LOOP

-- Получение базовой цены из таблицы halls

SELECT def\_price INTO ticket\_price

FROM halls

WHERE id\_hall = (

SELECT id\_hall FROM timetable WHERE id\_timetable = NEW.id\_timetable

);

-- Проверка совпадения с bestrows и bestseats

IF curr\_row = ANY(ARRAY(SELECT jsonb\_array\_elements\_text(bestrows)::INTEGER FROM halls WHERE id\_hall = NEW.id\_hall)::INTEGER[]) AND

curr\_seat = ANY(ARRAY(SELECT jsonb\_array\_elements\_text(bestseats)::INTEGER FROM halls WHERE id\_hall = NEW.id\_hall)::INTEGER[]) THEN

ticket\_price := ticket\_price \* hall\_coefficient \* timetable\_coefficient;

ELSE

ticket\_price := ticket\_price \* timetable\_coefficient;

END IF;

-- Обновление записи в таблице "tickets"

UPDATE tickets

SET price = ticket\_price,

status = CASE WHEN NEW.status = 'В прокате' THEN 1 ELSE 0 END

WHERE id\_row = curr\_row

AND id\_seat = curr\_seat

AND id\_timetable = NEW.id\_timetable;

curr\_seat := curr\_seat + 1; -- Переход к следующему месту

END LOOP;

curr\_row := curr\_row + 1; -- Переход к следующему ряду

curr\_seat := 1; -- Сброс счетчика мест

END LOOP;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

Листинг В.3 – функция update\_tickets\_on\_timetable\_update

CREATE OR REPLACE FUNCTION search\_tickets\_by\_movie\_title(movie\_title\_param VARCHAR(255))

RETURNS TABLE (

movie\_title VARCHAR(255),

hall\_name VARCHAR(255),

session\_date JSONB,

session\_time JSONB,

available\_tickets BIGINT

)

security definer

AS $$

DECLARE

movie\_found BOOLEAN;

BEGIN

movie\_found := FALSE;

FOR movie\_title, hall\_name, session\_date, session\_time, available\_tickets IN

SELECT i.title AS movie\_title,

h.name AS hall\_name,

TO\_JSONB(t.details->>'date') AS session\_date,

TO\_JSONB(t.details->>'time') AS session\_time,

COUNT(tk.id\_timetable) AS available\_tickets

FROM info\_films i

INNER JOIN timetable t ON i.id\_film = t.id\_film

INNER JOIN halls h ON t.id\_hall = h.id\_hall

LEFT JOIN tickets tk ON t.id\_timetable = tk.id\_timetable AND tk.status = 1

WHERE i.title = movie\_title\_param

GROUP BY i.title, h.name, t.details

ORDER BY h.name

LOOP

movie\_found := TRUE;

RETURN NEXT;

END LOOP;

IF NOT movie\_found THEN

RAISE NOTICE 'Фильм с названием "%" не найден.', movie\_title\_param;

END IF;

RETURN;

END;

$$ LANGUAGE plpgsql;

Листинг В.4 – функция search\_tickets\_by\_movie\_title

CREATE OR REPLACE FUNCTION search\_tickets\_by\_session\_date(session\_date\_param DATE)

RETURNS TABLE (

movie\_title VARCHAR(255),

hall\_name VARCHAR(255),

session\_date JSONB,

session\_time JSONB,

available\_tickets BIGINT

)

security definer

AS $$

DECLARE

movies\_found BOOLEAN;

BEGIN

movies\_found := FALSE;

FOR movie\_title, hall\_name, session\_date, session\_time, available\_tickets IN

SELECT i.title AS movie\_title,

h.name AS hall\_name,

TO\_JSONB(t.details->>'date') AS session\_date,

TO\_JSONB(t.details->>'time') AS session\_time,

COUNT(tk.id\_timetable) AS available\_tickets

FROM info\_films i

INNER JOIN timetable t ON i.id\_film = t.id\_film

INNER JOIN halls h ON t.id\_hall = h.id\_hall

LEFT JOIN tickets tk ON t.id\_timetable = tk.id\_timetable AND tk.status = 1

WHERE t.details->>'date' = TO\_CHAR(session\_date\_param, 'YYYY-MM-DD')

GROUP BY i.title, h.name, t.details

ORDER BY h.name

LOOP

movies\_found := TRUE;

RETURN NEXT;

END LOOP;

IF NOT movies\_found THEN

RAISE NOTICE 'На % дату нет фильмов.', TO\_CHAR(session\_date\_param, 'YYYY-MM-DD');

END IF;

RETURN;

END;

$$ LANGUAGE plpgsql;

Листинг В.5 – функция search\_tickets\_by\_session\_date

CREATE OR REPLACE FUNCTION search\_tickets\_by\_genre(genre\_param VARCHAR(255))

RETURNS TABLE (

movie\_title VARCHAR(255),

hall\_name VARCHAR(255),

session\_date JSONB,

session\_time JSONB,

available\_tickets BIGINT

)

security definer

AS $$

DECLARE

movies\_found BOOLEAN;

BEGIN

movies\_found := FALSE;

FOR movie\_title, hall\_name, session\_date, session\_time, available\_tickets IN

SELECT i.title AS movie\_title,

h.name AS hall\_name,

TO\_JSONB(t.details->>'date') AS session\_date,

TO\_JSONB(t.details->>'time') AS session\_time,

COUNT(tk.id\_timetable) AS available\_tickets

FROM info\_films i

INNER JOIN timetable t ON i.id\_film = t.id\_film

INNER JOIN halls h ON t.id\_hall = h.id\_hall

LEFT JOIN tickets tk ON t.id\_timetable = tk.id\_timetable AND tk.status = 1

WHERE i.genre = genre\_params

GROUP BY i.title, h.name, t.details

ORDER BY h.name

LOOP

movies\_found := TRUE;

RETURN NEXT;

END LOOP;

IF NOT movies\_found THEN

RAISE NOTICE 'Фильмов жанра % не найдено.', genre\_param;

END IF;

RETURN;

END;

$$ LANGUAGE plpgsql;

Листинг В.6 – функция search\_tickets\_by\_genre

CREATE OR REPLACE FUNCTION sort\_films\_by\_rating()

RETURNS TABLE (

movie\_title VARCHAR(255),

genre VARCHAR(255),

rating FLOAT,

start\_date DATE,

end\_date DATE

)

security definer

AS $$

BEGIN

RETURN QUERY

WITH timetable\_dates AS (

SELECT id\_film,

MIN((details->>'date')::DATE) AS start\_date,

MAX((details->>'date')::DATE) AS end\_date

FROM timetable

GROUP BY id\_film

)

SELECT i.title AS movie\_title,

i.genre,

i.rating,

td.start\_date,

td.end\_date

FROM info\_films i

INNER JOIN timetable\_dates td ON i.id\_film = td.id\_film

ORDER BY i.rating DESC;

END;

$$ LANGUAGE plpgsql;

Листинг В.7 – функция sort\_films\_by\_rating

CREATE OR REPLACE FUNCTION find\_tickets\_by\_quantity(num\_tickets\_needed INTEGER)

RETURNS TABLE (

movie\_title VARCHAR(255),

film\_genre VARCHAR(255),

film\_rating FLOAT,

day DATE,

show\_time TIME,

available\_tickets BIGINT

)

security definer

AS $$

BEGIN

RETURN QUERY

SELECT

f.title AS movie\_title,

f.genre AS film\_genre,

f.rating AS film\_rating,

(t.details->>'date')::DATE AS day,

(t.details->>'time')::TIME AS show\_time,

COUNT(\*) AS available\_tickets

FROM timetable t

INNER JOIN info\_films f ON t.id\_film = f.id\_film

LEFT JOIN tickets ti ON t.id\_timetable = ti.id\_timetable

AND ti.status = 1

GROUP BY f.title, f.genre, f.rating, (t.details->>'date')::DATE, (t.details->>'time')::TIME

HAVING COUNT(\*) >= num\_tickets\_needed;

END;

$$ LANGUAGE plpgsql;

Листинг В.8 – функция find\_tickets\_by\_quantity

CREATE OR REPLACE FUNCTION get\_available\_tickets\_summary()

RETURNS TABLE (

movie\_title VARCHAR(255),

film\_genre VARCHAR(255),

film\_rating FLOAT,

theater\_name VARCHAR(255),

film\_status VARCHAR(20),

total\_available\_tickets BIGINT

)

security definer

AS $$

BEGIN

RETURN QUERY

SELECT

f.title AS movie\_title,

f.genre AS film\_genre,

f.rating AS film\_rating,

th.name AS theater\_name,

CASE

WHEN EXISTS (SELECT 1 FROM timetable t WHERE t.id\_film = f.id\_film) THEN 'В прокате'::VARCHAR(20)

ELSE 'Ожидается в прокате'::VARCHAR(20)

END AS film\_status,

COALESCE(SUM(CASE WHEN ti.status = 1 THEN 1 ELSE 0 END), 0) AS total\_available\_tickets

FROM info\_films f

LEFT JOIN timetable t ON f.id\_film = t.id\_film

LEFT JOIN halls h ON t.id\_hall = h.id\_hall

LEFT JOIN theaters th ON h.id\_theater = th.id\_theater

LEFT JOIN tickets ti ON t.id\_timetable = ti.id\_timetable

GROUP BY f.id\_film, th.name;

END;

$$ LANGUAGE plpgsql;

Листинг В.9 – функция get\_available\_tickets\_summary

CREATE OR REPLACE FUNCTION get\_tickets\_info()

RETURNS TABLE (

movie\_title VARCHAR(255),

film\_genre VARCHAR(255),

film\_rating FLOAT,

available\_tickets BIGINT,

ticket\_price\_range TEXT

)

security definer

AS $$

BEGIN

RETURN QUERY

SELECT

f.title AS movie\_title,

f.genre AS film\_genre,

f.rating AS film\_rating,

COUNT(t.id\_row)::BIGINT AS available\_tickets,

(MIN(t.price)::VARCHAR(10) || ' - ' || MAX(t.price)::VARCHAR(10)) AS ticket\_price\_range

FROM

info\_films f

LEFT JOIN

timetable tt ON f.id\_film = tt.id\_film

LEFT JOIN

tickets t ON tt.id\_timetable = t.id\_timetable AND t.status = 1

WHERE

tt.id\_film IS NOT NULL

GROUP BY

f.title, f.genre, f.rating;

END;

$$ LANGUAGE plpgsql;

Листинг В.10 – функция get\_tickets\_info

CREATE OR REPLACE FUNCTION available\_movie\_tickets\_price(price\_range NUMERIC) RETURNS TABLE (

film\_title VARCHAR(255),

film\_rating FLOAT,

film\_genre VARCHAR(255),

available\_tickets BIGINT,

ticket\_price\_range JSONB

)

security definer

AS $$

BEGIN

RETURN QUERY

SELECT

f.title AS film\_title,

f.rating AS film\_rating,

f.genre AS film\_genre,

COUNT(t.id\_row) AS available\_tickets,

jsonb\_build\_object(

'min\_price', MIN(t.price),

'max\_price', MAX(t.price)

) AS ticket\_price\_range

FROM

info\_films f

JOIN

timetable tt ON f.id\_film = tt.id\_film

JOIN

tickets t ON tt.id\_timetable = t.id\_timetable

WHERE

t.status = 1

GROUP BY

f.title, f.rating, f.genre

HAVING

COUNT(t.id\_row) > 0 -- Считаем количество строк по рандомному полю

AND MIN(t.price) <= price\_range AND MAX(t.price) >= price\_range;

END;

$$ LANGUAGE plpgsql;

Листинг В.11 – функция available\_movie\_tickets\_price

CREATE OR REPLACE FUNCTION update\_average\_rating()

RETURNS TRIGGER AS $$

BEGIN

UPDATE info\_films AS f

SET rating = (

SELECT AVG(rating)

FROM ratings

WHERE id\_film = NEW.id\_film

)

WHERE f.id\_film = NEW.id\_film;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

Листинг В.12 – функция update\_average\_rating

CREATE OR REPLACE FUNCTION add\_customer(

p\_customer\_name VARCHAR(255)

)

RETURNS VOID

security definer

AS $$

DECLARE

\_existing\_customer BOOLEAN;

BEGIN

-- Проверяем наличие покупателя с таким именем

SELECT EXISTS(SELECT 1 FROM customers WHERE name = p\_customer\_name) INTO \_existing\_customer;

-- Если покупатель уже существует, выводим сообщение

IF \_existing\_customer THEN

RAISE NOTICE 'Покупатель "%" уже существует.', p\_customer\_name;

ELSE

-- Вставляем данные о покупателе в таблицу customers

INSERT INTO customers (name) VALUES (p\_customer\_name);

-- Выводим сообщение об успешном добавлении

RAISE NOTICE 'Данные о покупателе "%" успешно добавлены в таблицу customers.', p\_customer\_name;

END IF;

END;

$$ LANGUAGE plpgsql;

Листинг В.13 – функция add\_customer

CREATE OR REPLACE FUNCTION get\_user\_ticket\_purchases(user\_name VARCHAR(255))

RETURNS TABLE (

user\_email VARCHAR(255),

purchase\_date TIMESTAMP without time zone,

total\_ticket\_price NUMERIC

)

security definer

AS $$

BEGIN

RETURN QUERY

SELECT

user\_ticket\_purchases.user\_email,

user\_ticket\_purchases.purchase\_date,

user\_ticket\_purchases.total\_ticket\_price

FROM

user\_ticket\_purchases

WHERE

user\_ticket\_purchases.user\_email = user\_name;

END;

$$ LANGUAGE plpgsql;

Листинг В.14 – функция get\_user\_ticket\_purchases

CREATE OR REPLACE FUNCTION get\_tickets\_sold\_per\_theater()

RETURNS TABLE (

theater\_name VARCHAR(255),

total\_tickets\_sold INTEGER

)

security definer

AS $$

BEGIN

RETURN QUERY

SELECT

th.name AS theater\_name,

COUNT(\*)::INTEGER AS total\_tickets\_sold

FROM

orders o

JOIN

ord\_det od ON o.id\_ord = od.id\_ord

JOIN

tickets t ON od.id\_row = t.id\_row AND od.id\_seat = t.id\_seat AND od.id\_timetable = t.id\_timetable

JOIN

timetable tt ON t.id\_timetable = tt.id\_timetable

JOIN

halls h ON tt.id\_hall = h.id\_hall

JOIN

theaters th ON h.id\_theater = th.id\_theater

GROUP BY

th.name;

END;

$$ LANGUAGE plpgsql;

Листинг В.15 – функция get\_tickets\_sold\_per\_theater

CREATE OR REPLACE FUNCTION get\_theater\_movie\_tickets(theater\_name VARCHAR(255), from\_date DATE, to\_date DATE)

RETURNS VOID

SECURITY DEFINER

AS $$

BEGIN

-- Drop the temporary table if it already exists

IF EXISTS (SELECT 1 FROM pg\_tables WHERE tablename = 'temp\_theater\_movie\_tickets') THEN

DROP TABLE temp\_theater\_movie\_tickets;

END IF;

-- Create the temporary table

CREATE TEMP TABLE temp\_theater\_movie\_tickets AS

SELECT

tm.theater\_name,

tm.movie\_title,

tm.total\_tickets\_sold,

tm.first\_purchase\_date,

tm.last\_purchase\_date

FROM

theaters\_movie\_tickets tm

WHERE

tm.theater\_name = get\_theater\_movie\_tickets.theater\_name

AND tm.first\_purchase\_date >= get\_theater\_movie\_tickets.from\_date

AND tm.last\_purchase\_date <= get\_theater\_movie\_tickets.to\_date;

END;

$$ LANGUAGE plpgsql;

Листинг В.16 – функция get\_theater\_movie\_tickets