**Тема**: Обработка исключительных ситуаций. Методы и способы идентификации сбоев и ошибок.

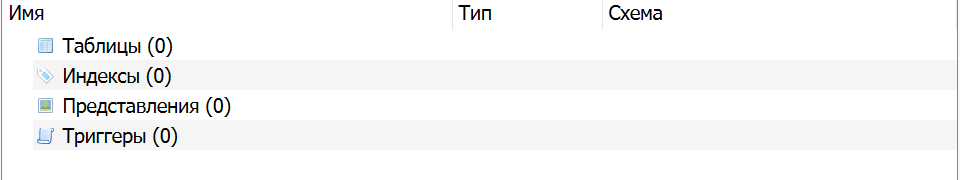
**Лекция:**

**Обработка исключительных ситуаций. Методы и способы идентификации сбоев и ошибок**

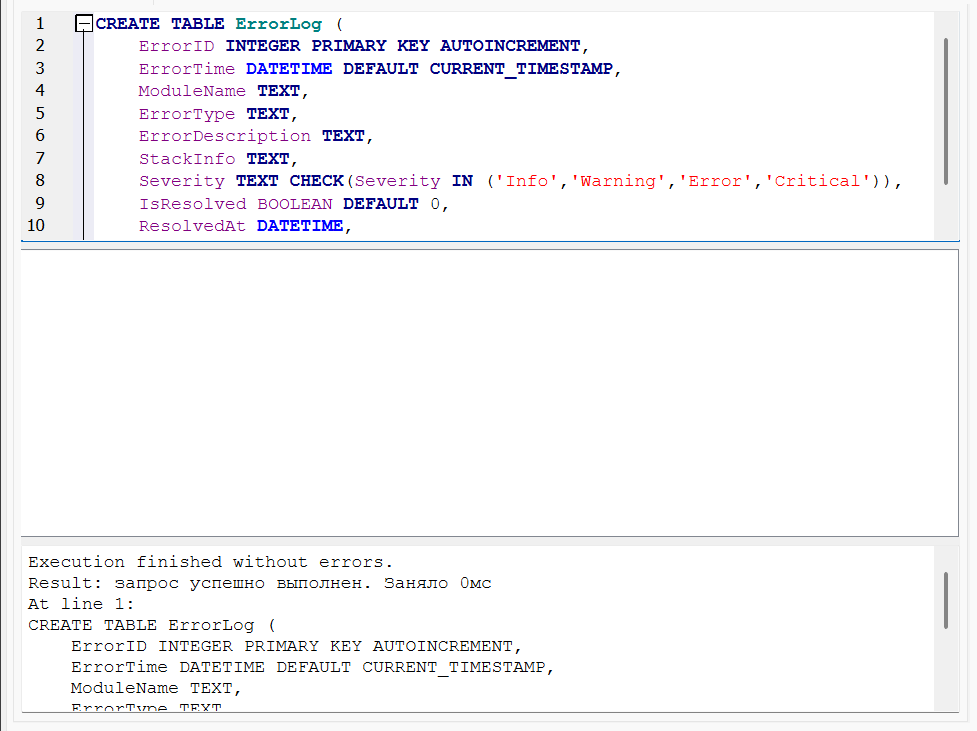
**Цель занятия**

Сформировать у студентов знания о типах исключительных ситуаций (ошибок и сбоев), методах их выявления и обработке в приложениях и базах данных. Научить проектировать механизмы логирования и реагирования (включая автоматическую идентификацию ошибок), реализовывать журналирование и тестирование в MS Access, а также визуализировать процессы обнаружения и обработки ошибок при помощи диаграмм.

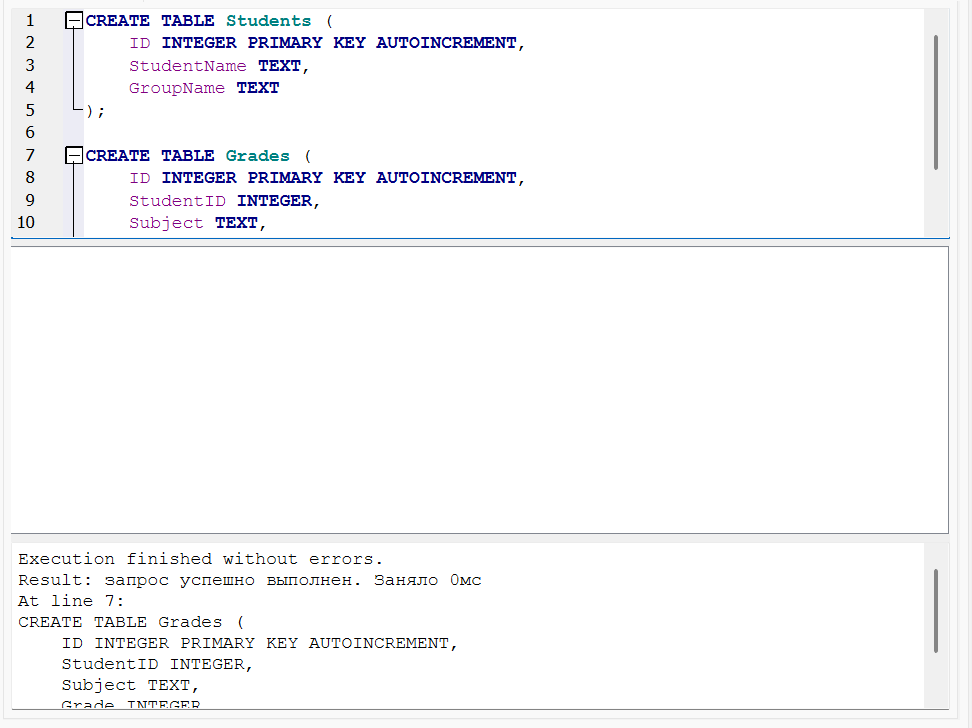
**ЗАДАНИЕ 1**: СОЗДАНИЕ СИСТЕМЫ ЖУРНАЛИРОВАНИЯ ОШИБОК



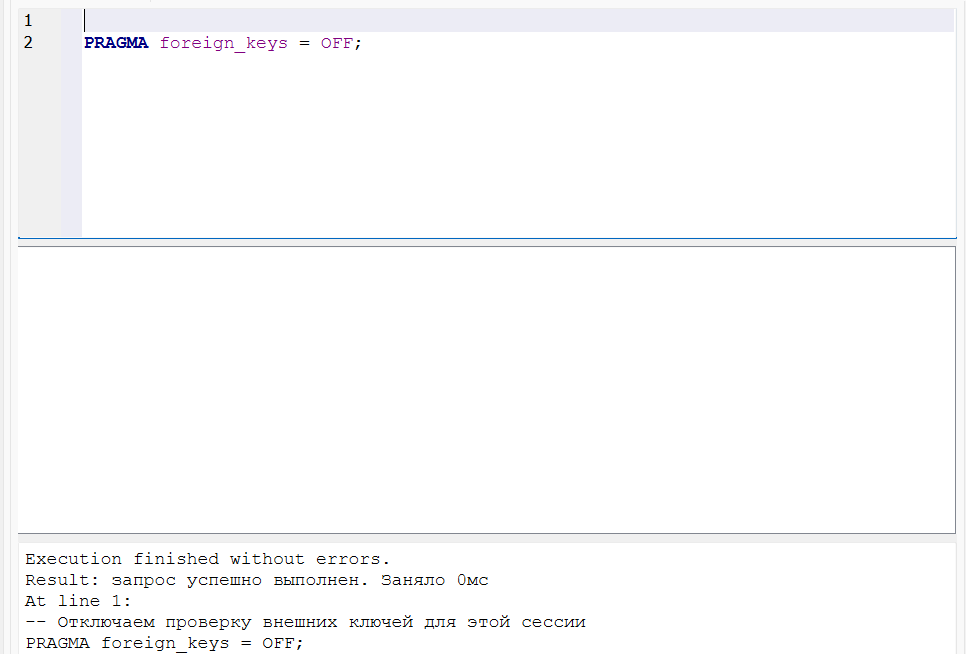
Шаг 1: **Создаем таблицу ErrorLog**

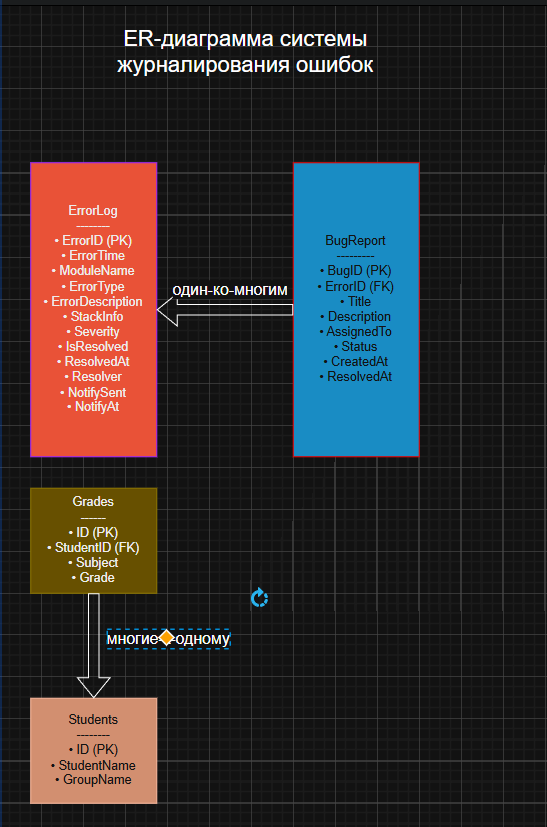


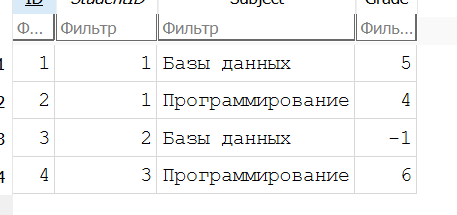
Шаг 2: **Создаем таблицы Students и Grades для тестирования**



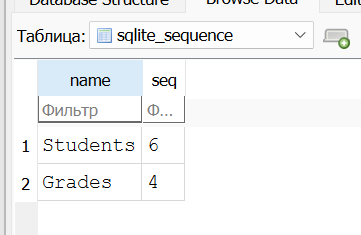
Шаг 3: **Добавляем тестовые данные**

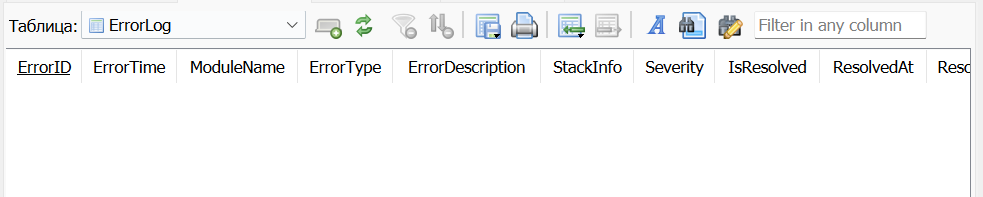






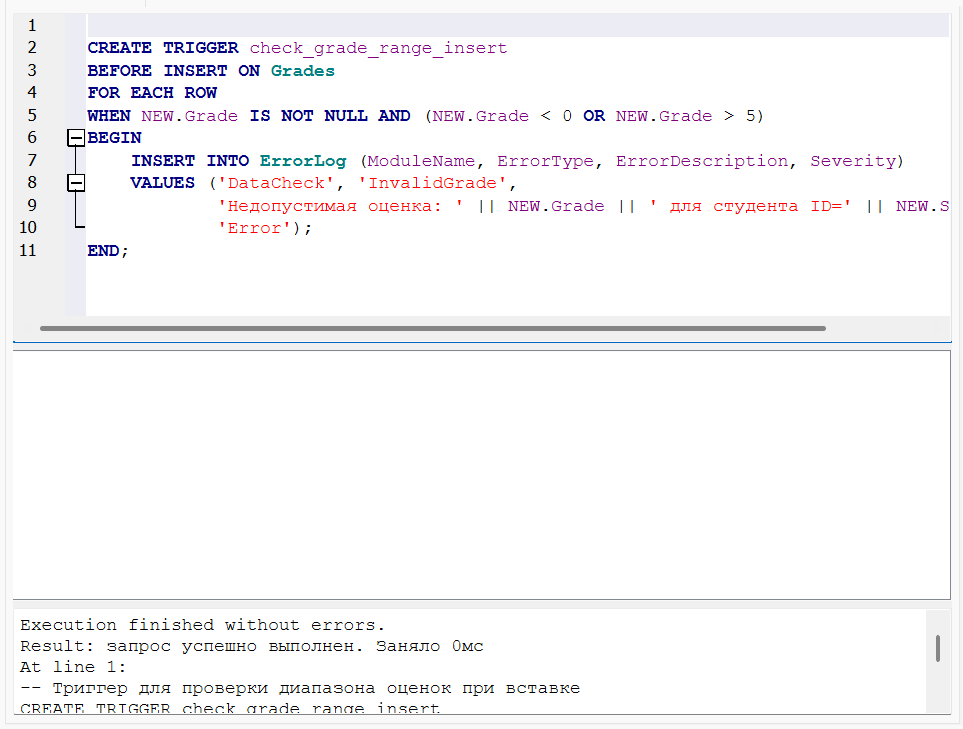


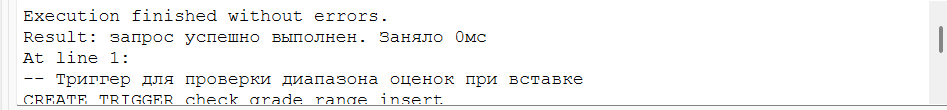


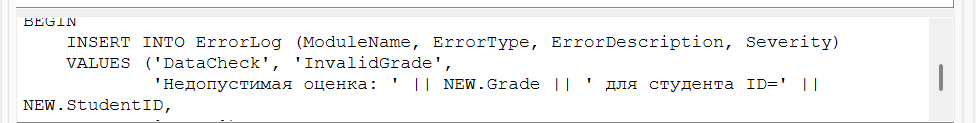


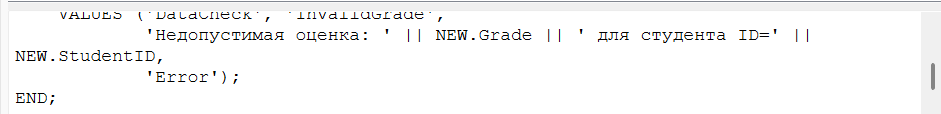
**ЗАДАНИЕ 2:** АВТОМАТИЧЕСКАЯ ИДЕНТИФИКАЦИЯ СБОЕВ

Шаг 4: **Создаем процедуру проверки данных через SQL-триггеры**

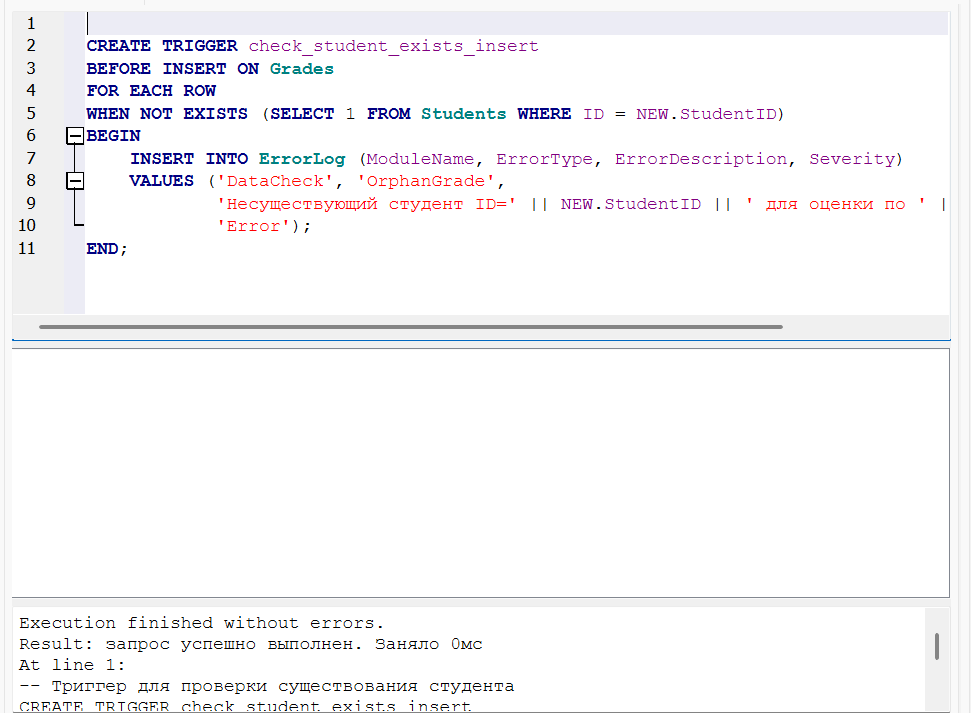


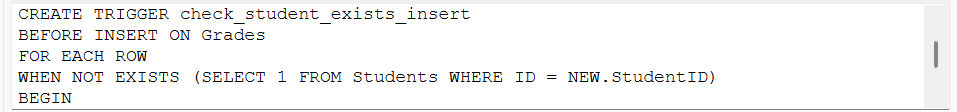


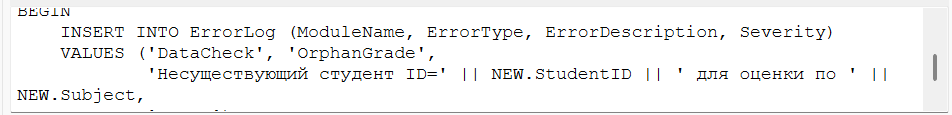


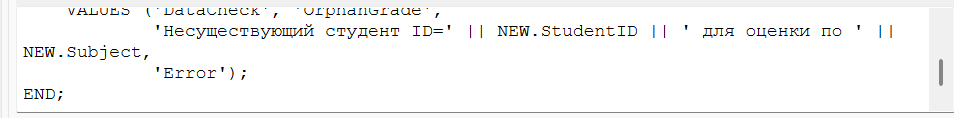


Создаем триггер для проверки существования студента:





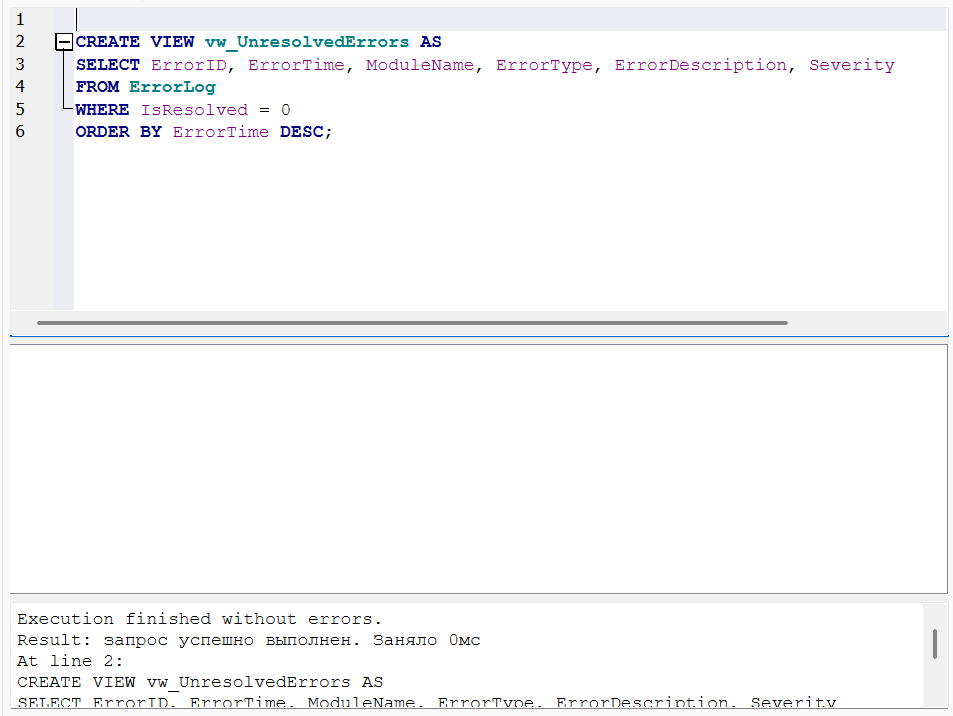


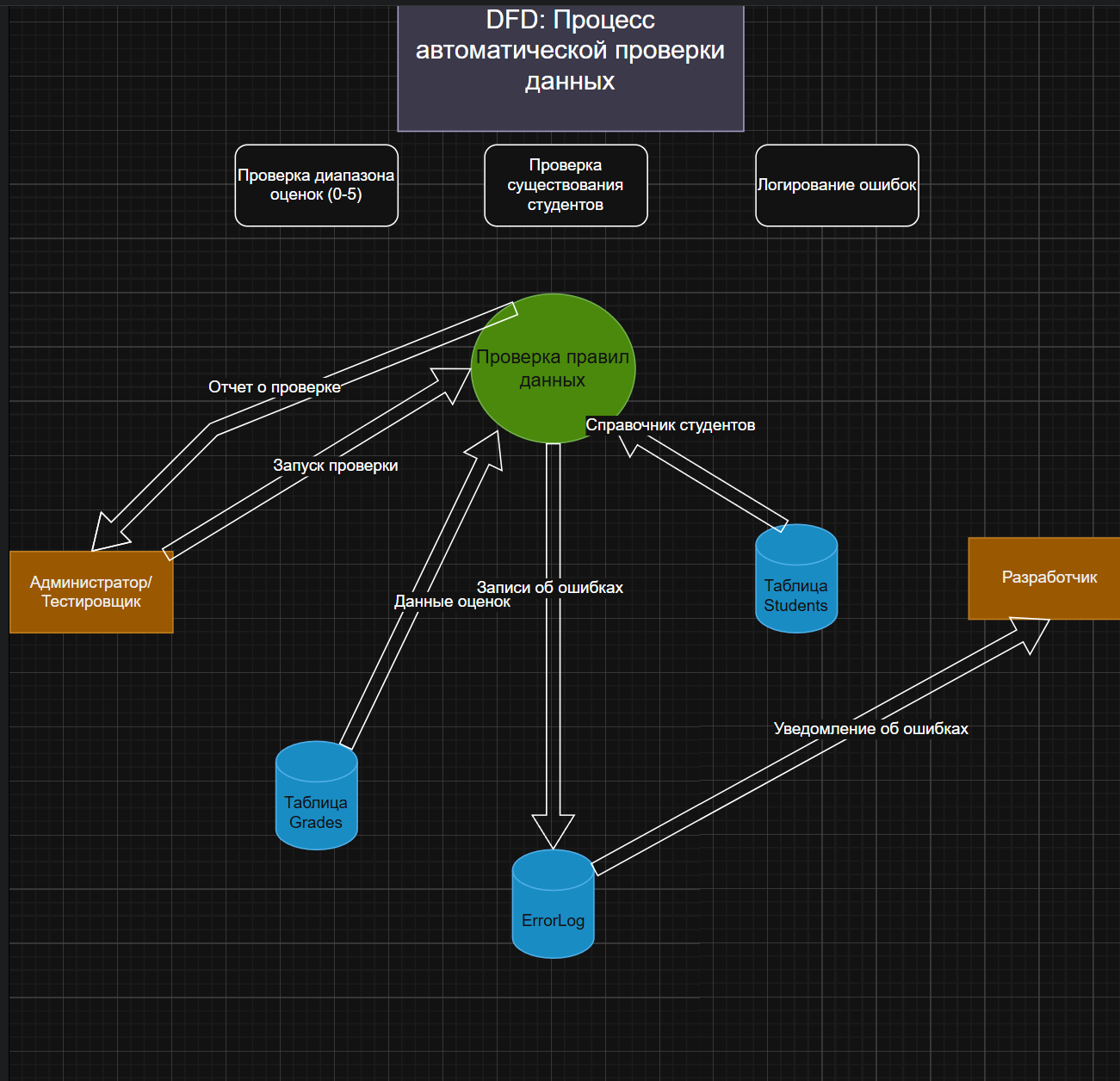


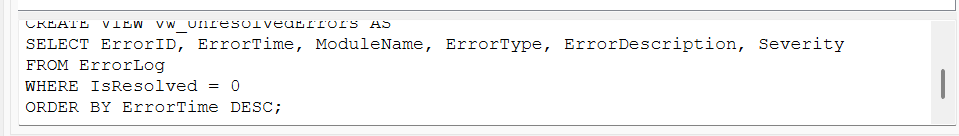
Шаг 5: **Тестируем автоматические проверки**



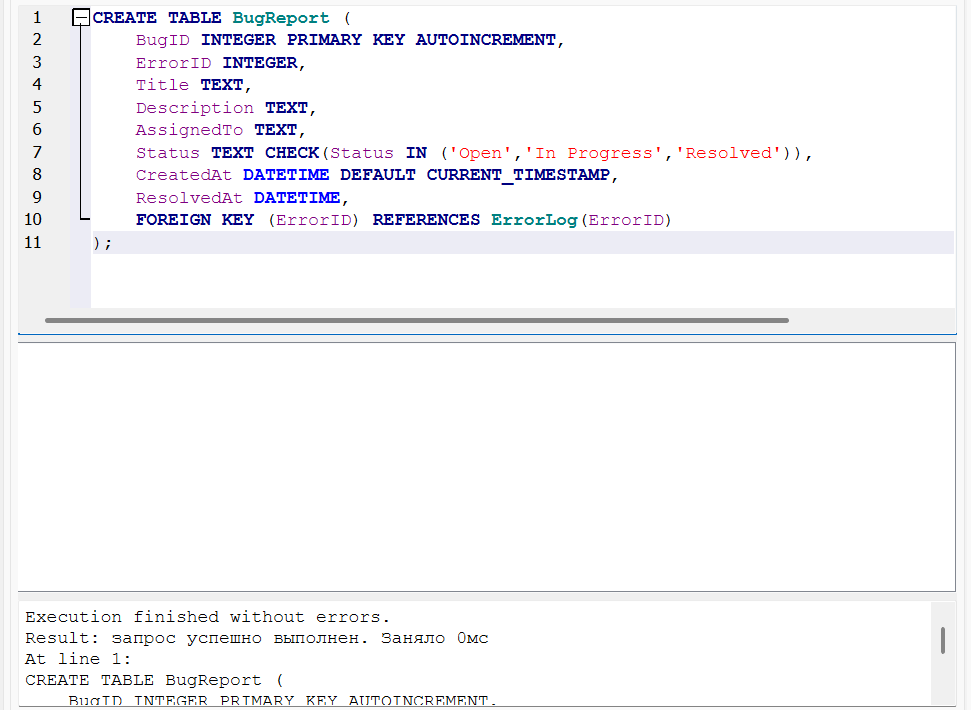
Шаг 6 **Создаем представление для нерешенных ошибок**

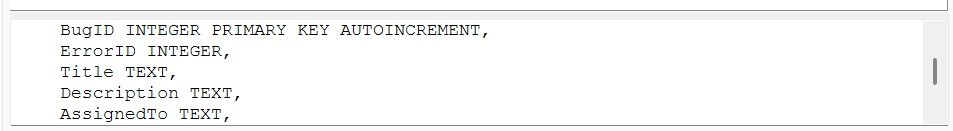


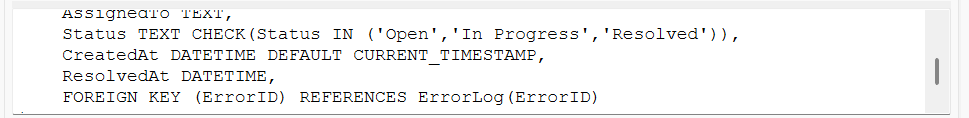


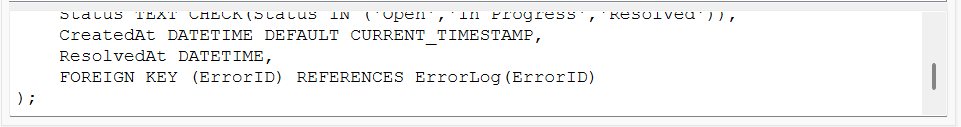


**ЗАДАНИЕ 3**: ПОЛНАЯ СХЕМА ОБРАБОТКИ ИСКЛЮЧЕНИЙ

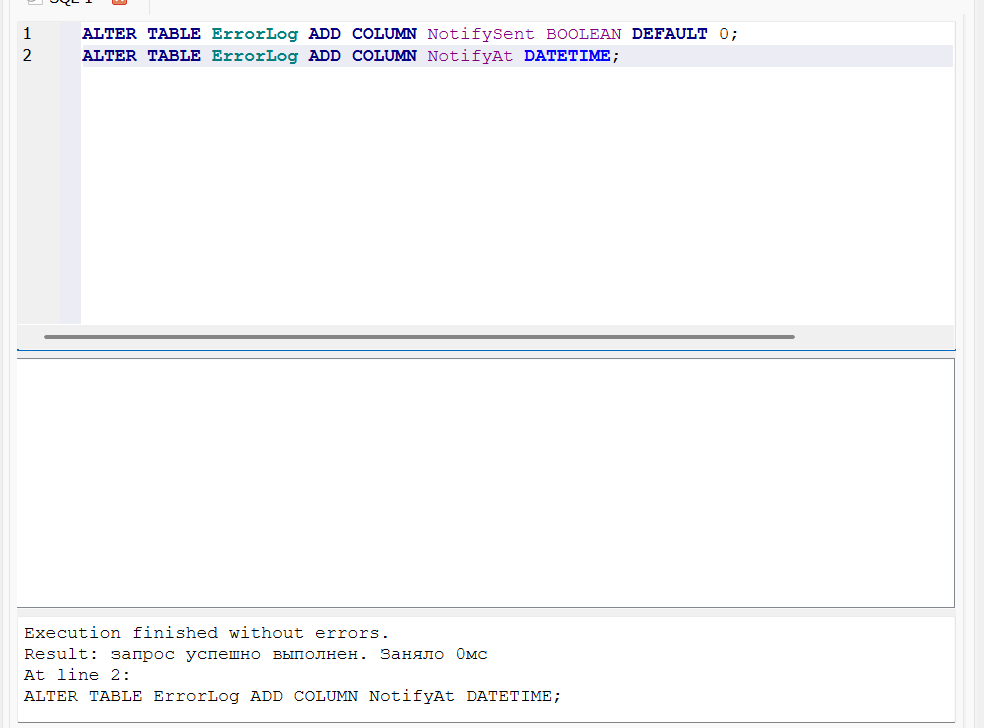


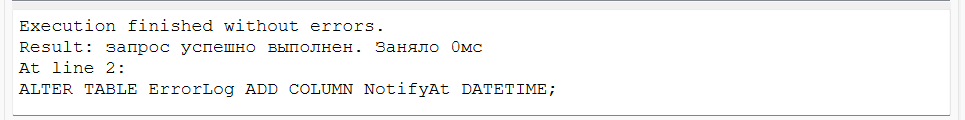




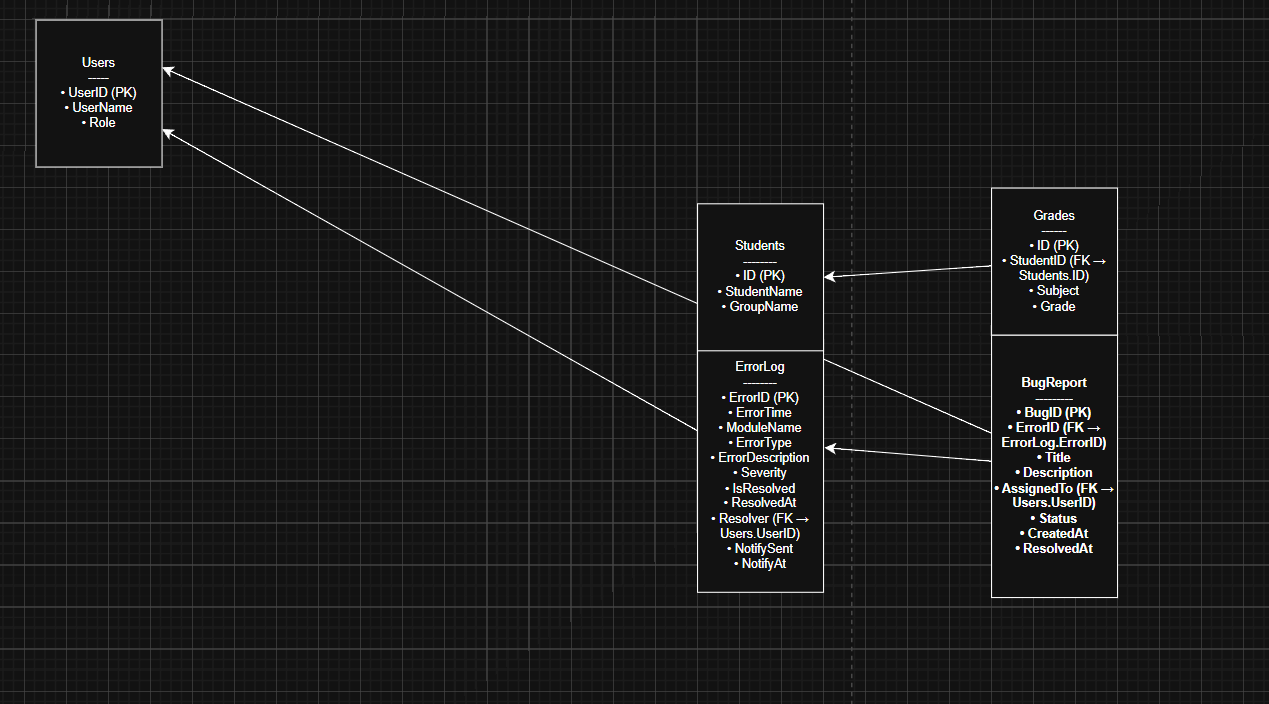


**Шаг 7**: Добавляем поля для уведомлений в ErrorLog

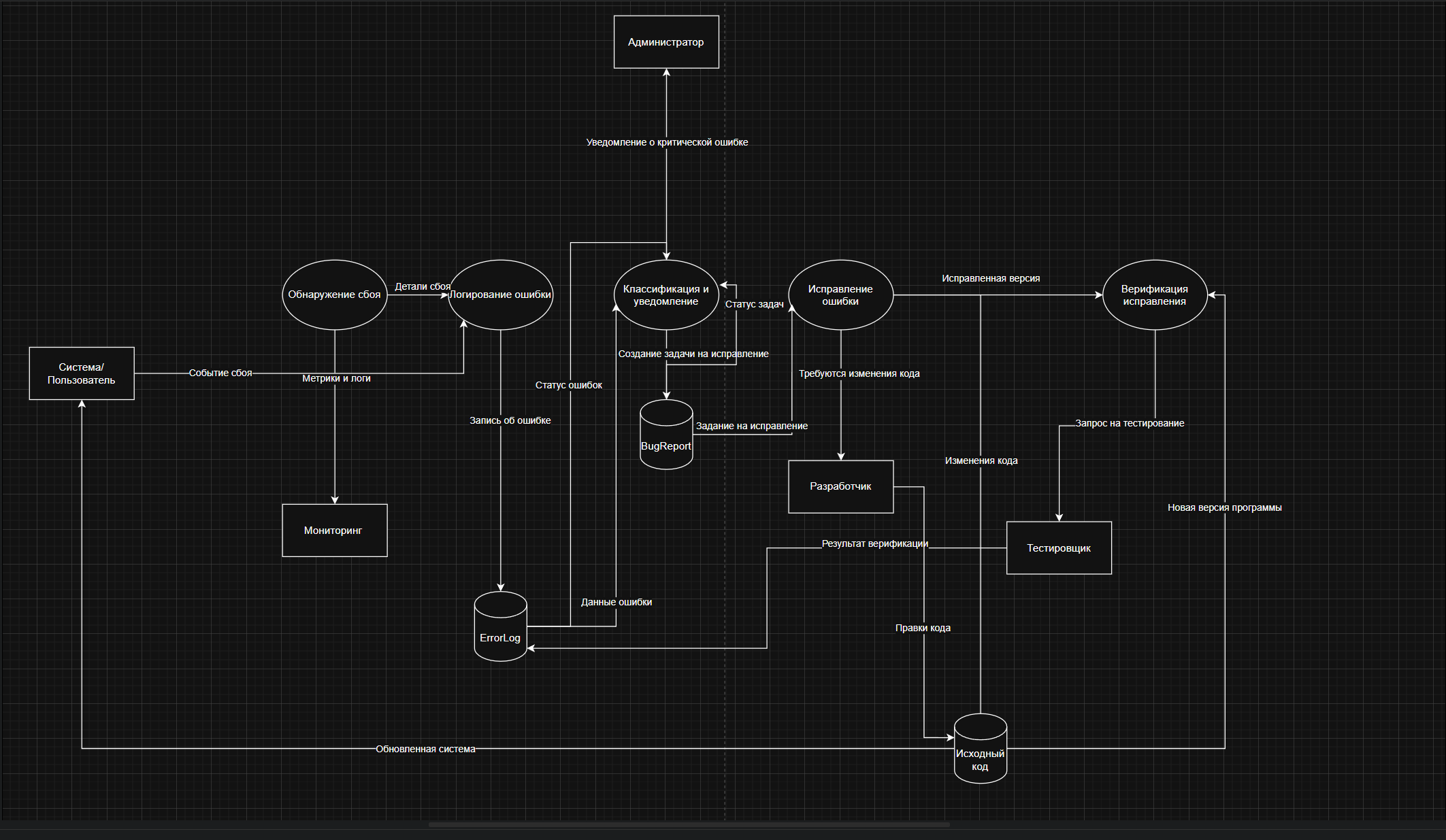




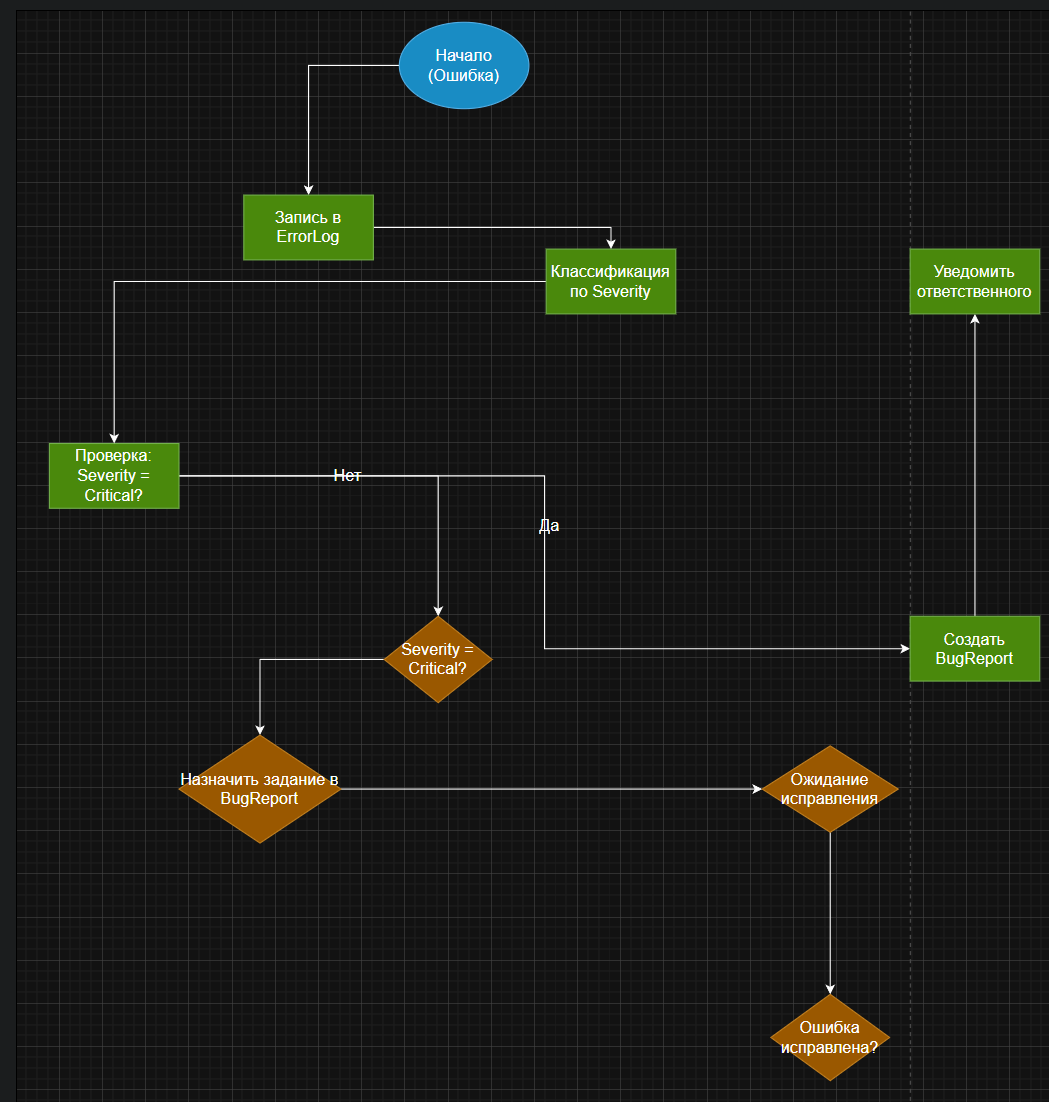
**ДИАГРАММА 1: ER-ДИАГРАММА СИСТЕМЫ**



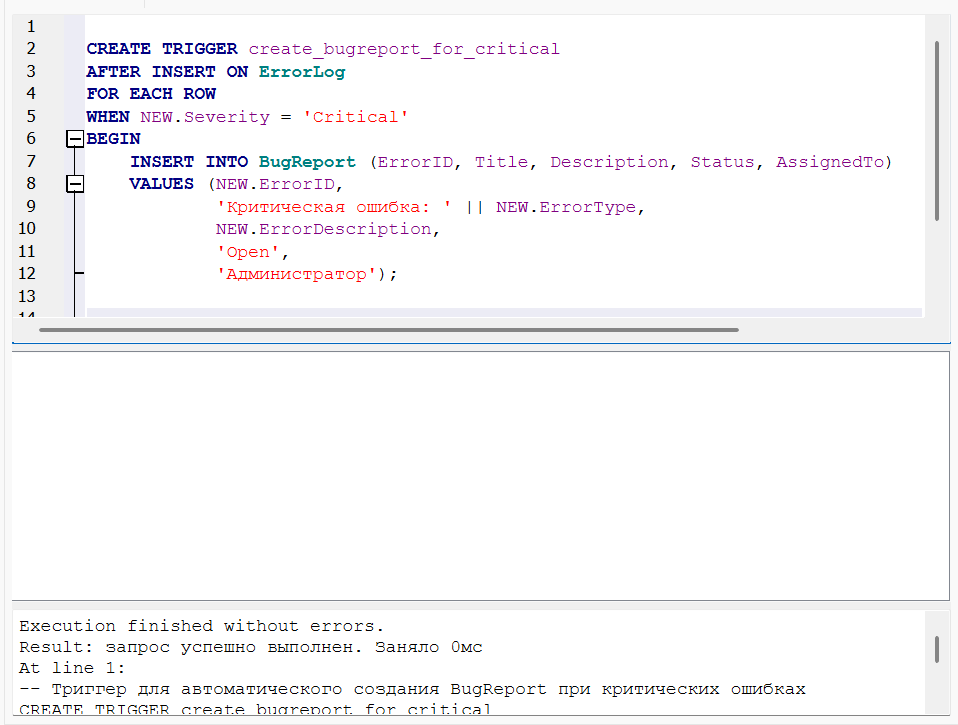
**ДИАГРАММА 2: DFD ПРОЦЕССА ОБРАБОТКИ СБОЕВ**

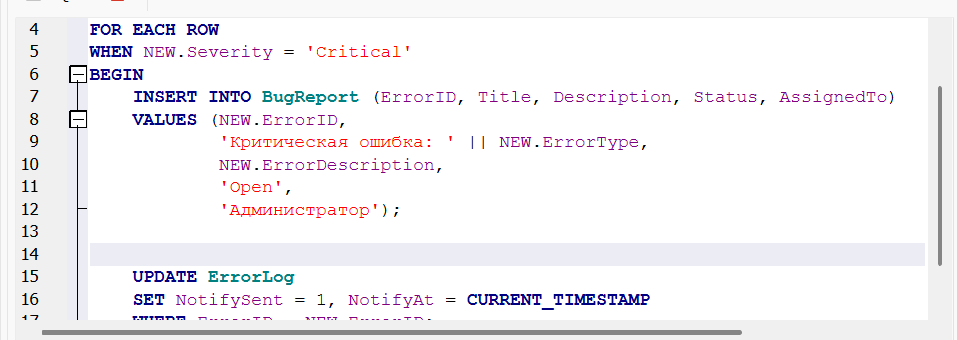
****

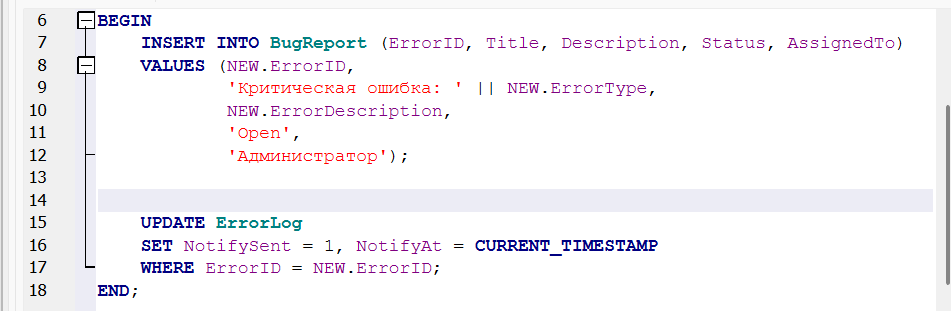
**ДИАГРАММА 3: БЛОК-СХЕМА ОБРАБОТЧИКА ИСКЛЮЧЕНИЙ**

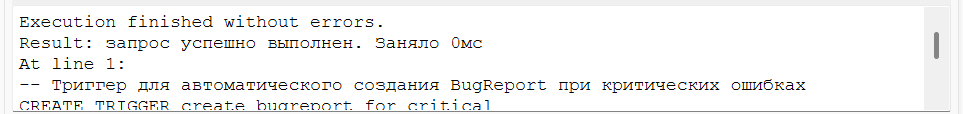
****

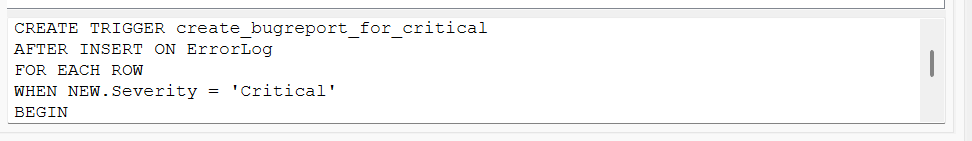
Шаг 8 **Создаем триггер для автоматического создания BugReport**

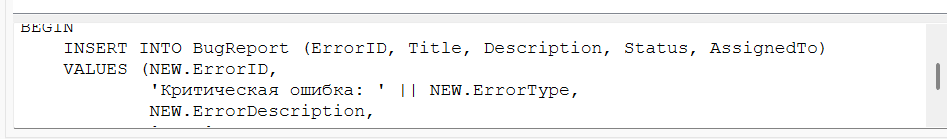


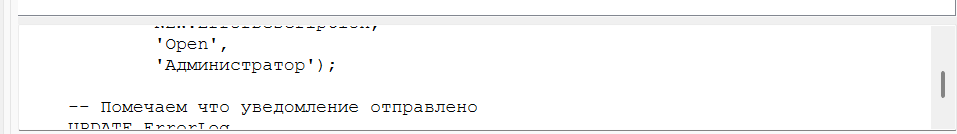


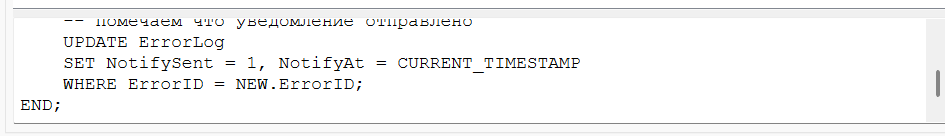






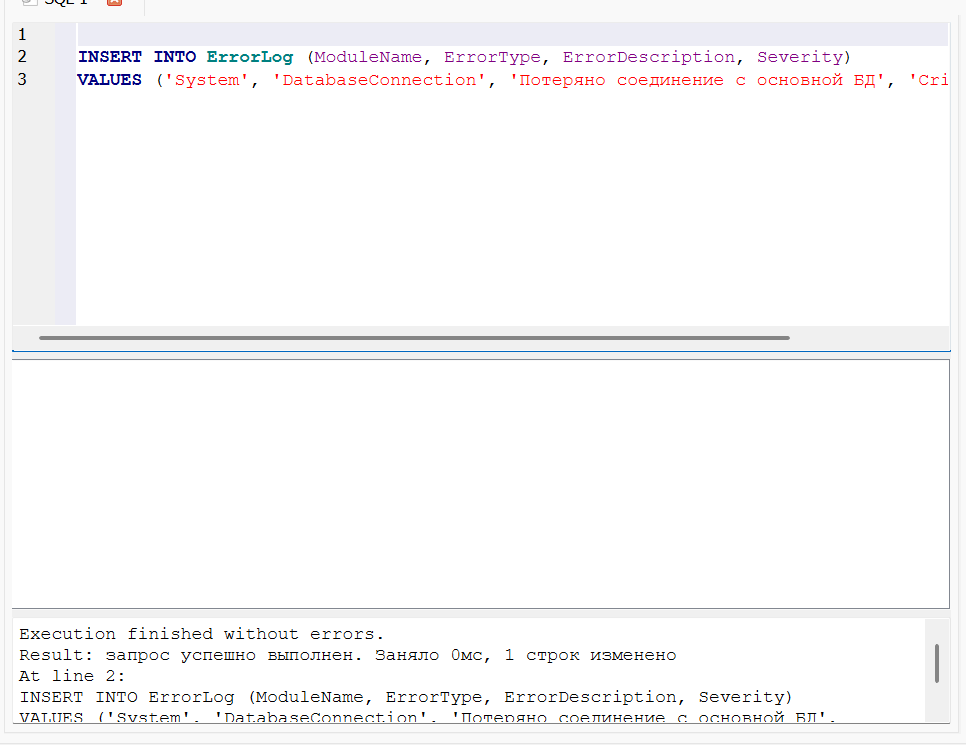


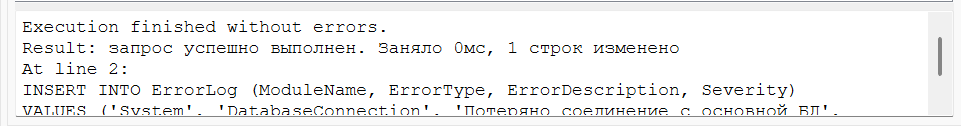
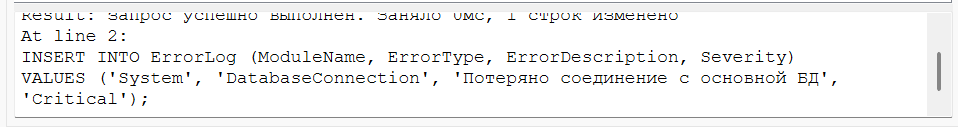


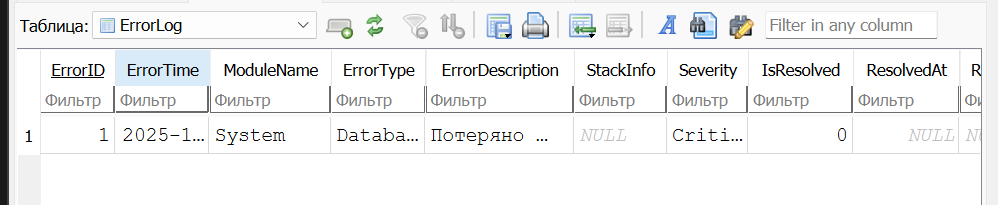


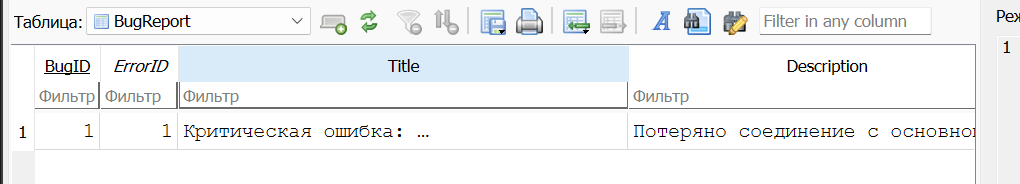
Шаг 9: **Тестируем полный цикл**

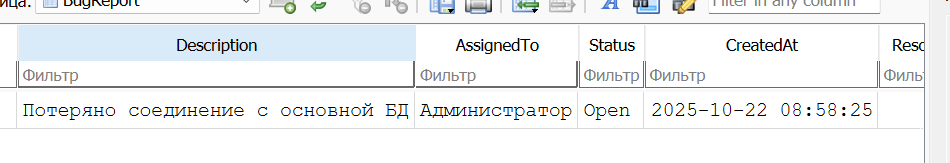
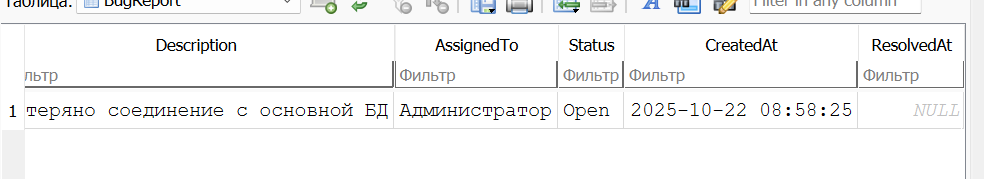
Добавляем критическую ошибку вручную



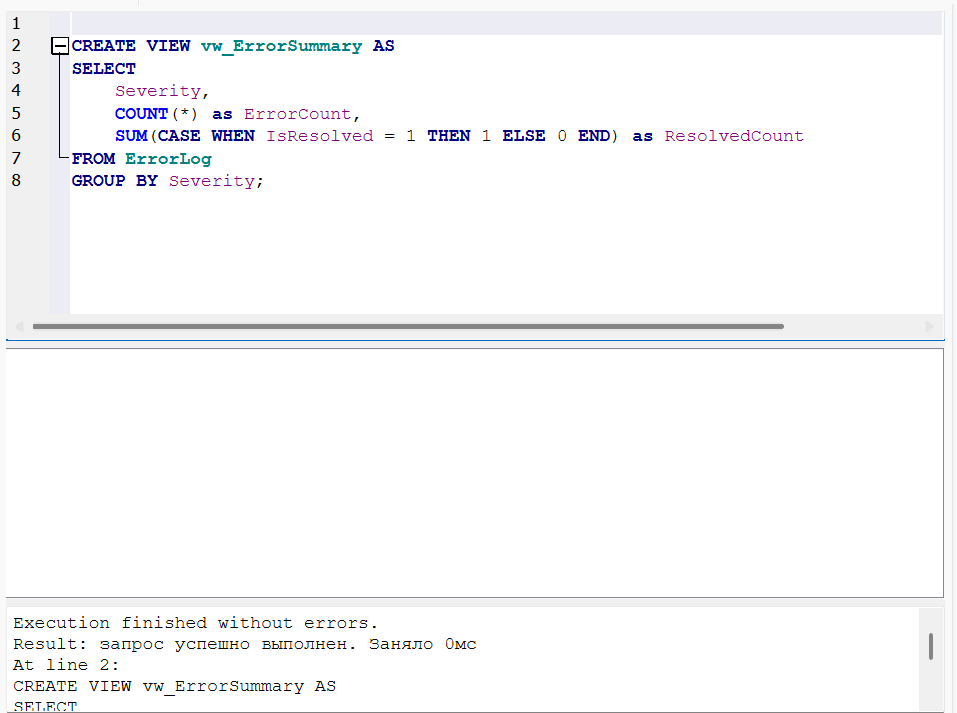


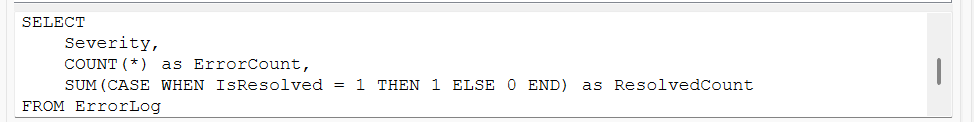
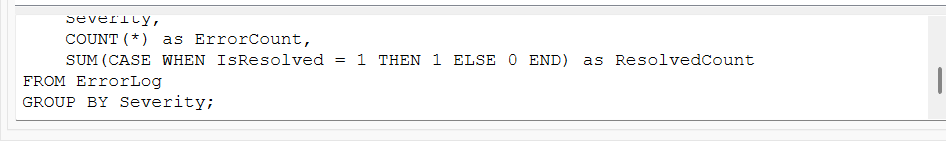


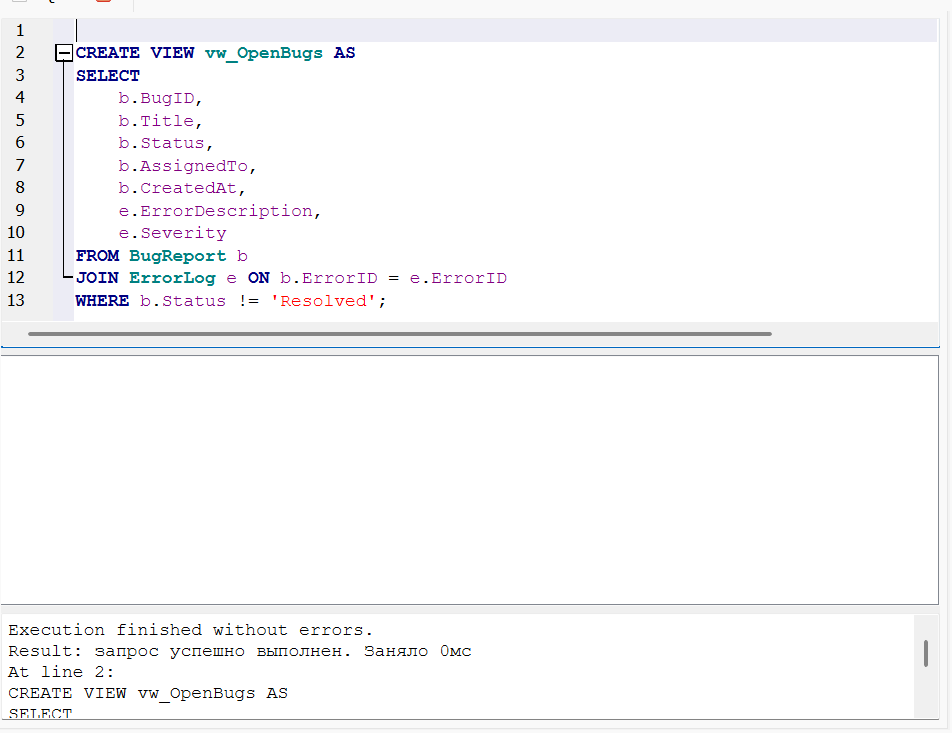
Шаг 10: **Создаем полезные запросы**

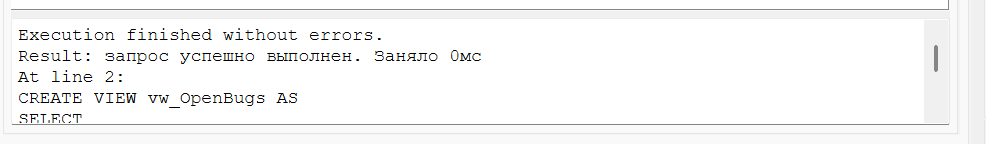
Запрос для отчета по ошибкам:

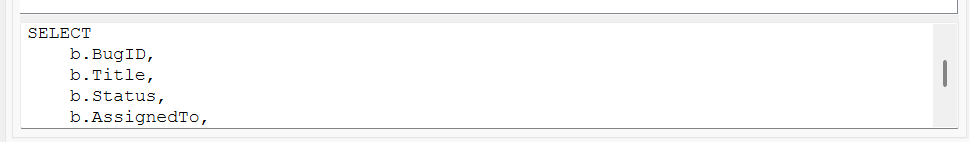


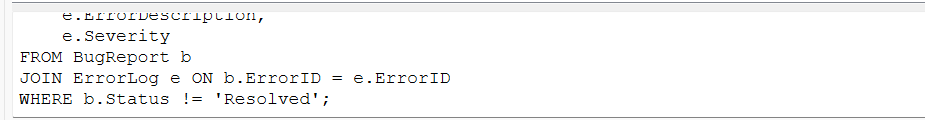
 

Запрос для открытых багов

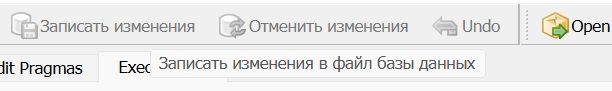


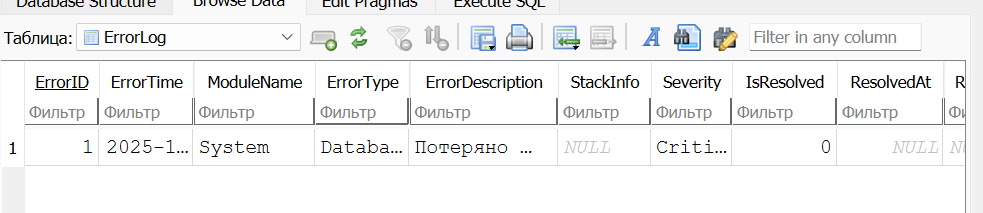


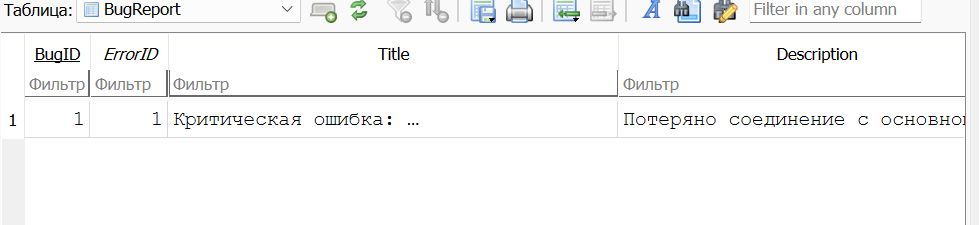


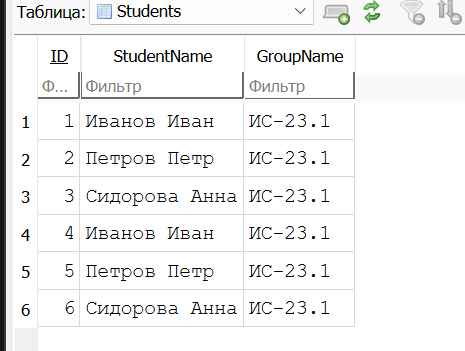


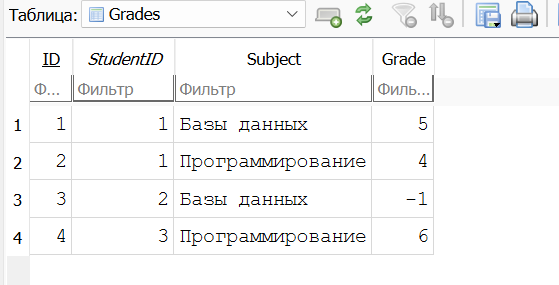
Шаг 11: **СОХРАНЕНИЕ И ПРОВЕРКА**







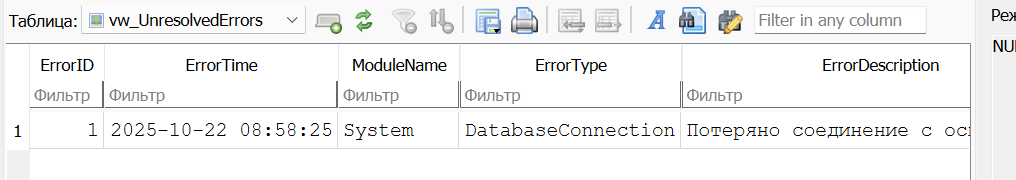


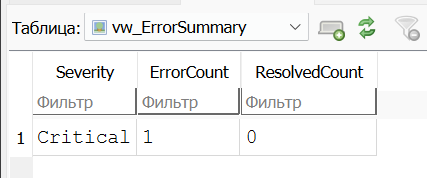


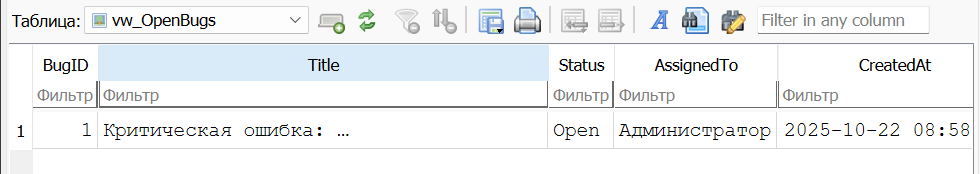
vw\_UnresolvedErrors - нерешенные ошибки

vw\_ErrorSummary - сводка по ошибкам

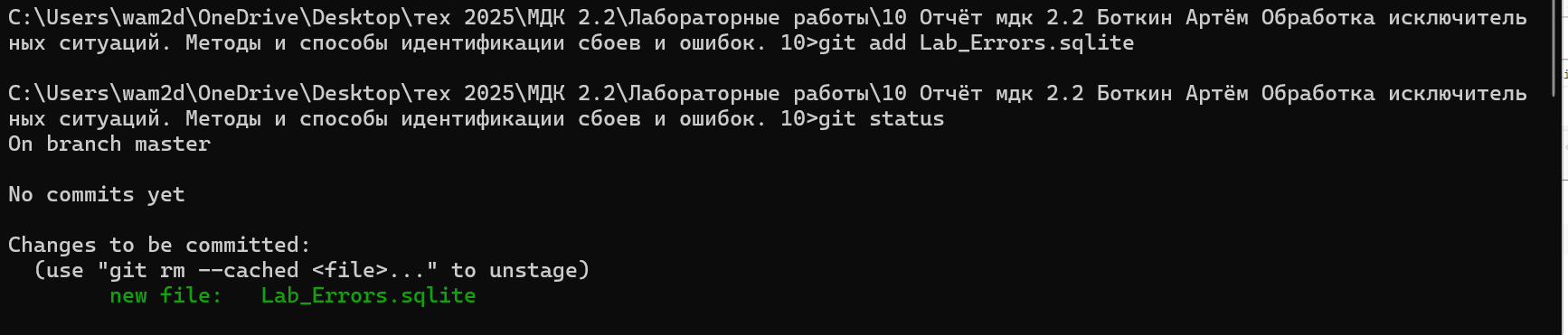
vw\_OpenBugs - открытые баги

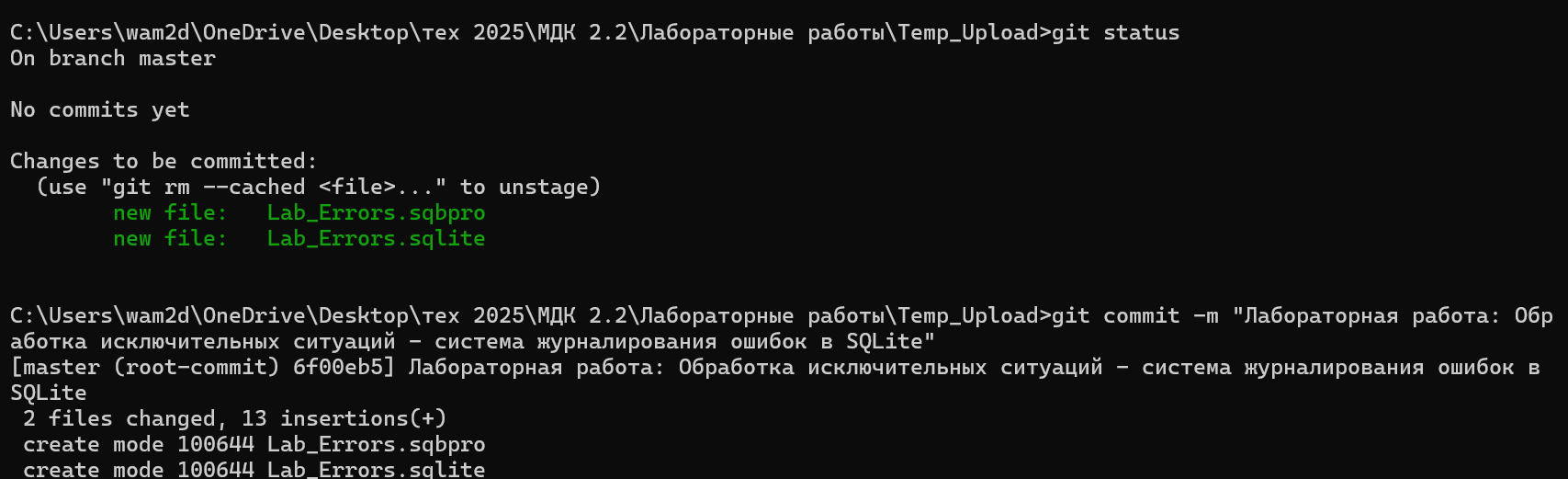


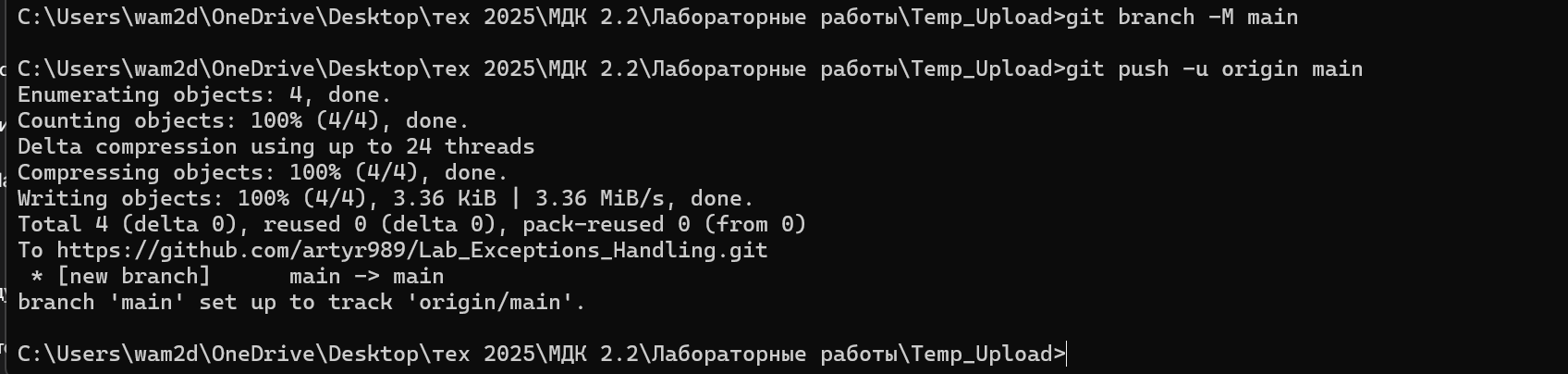


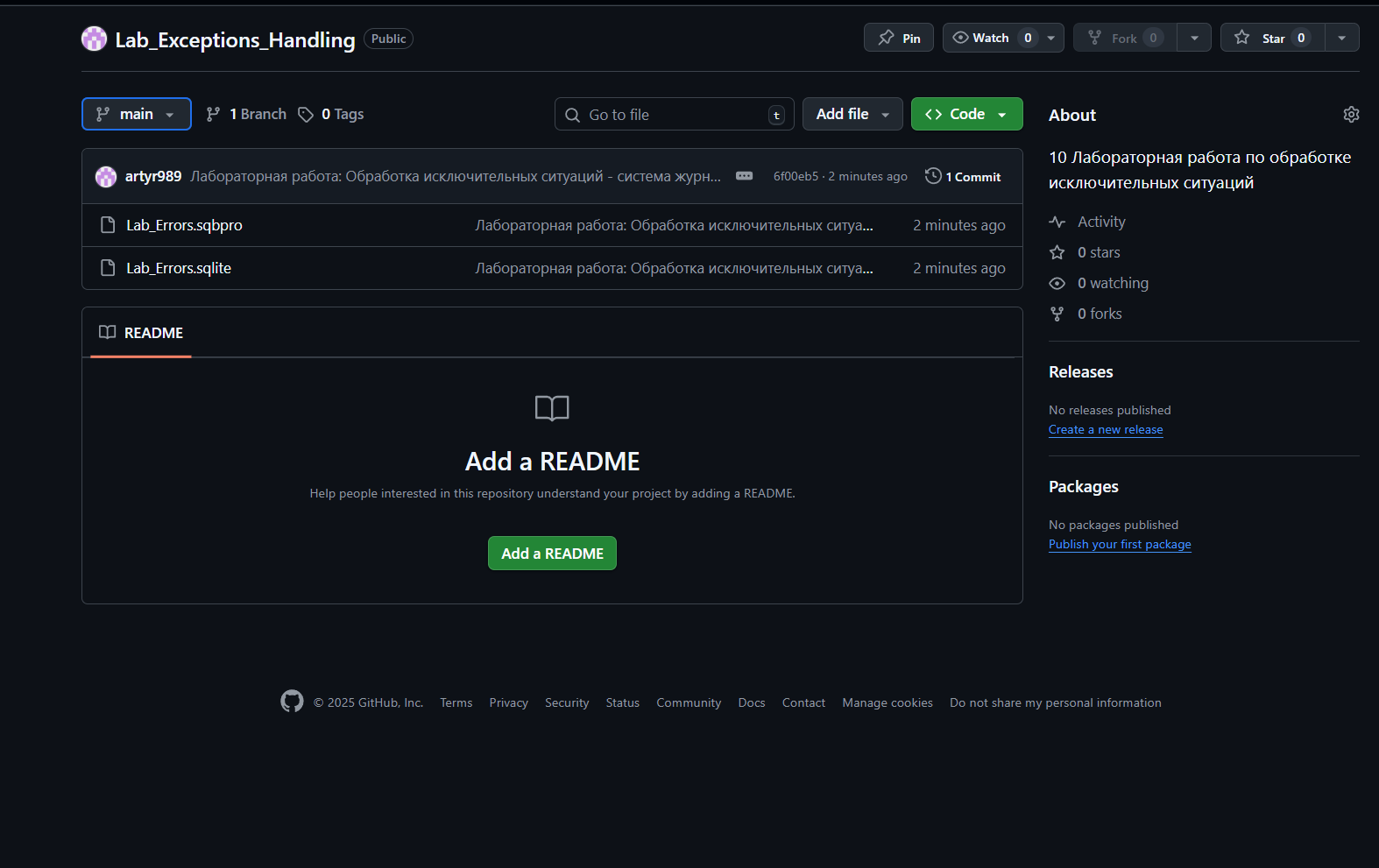


Шаг 12 **СОХРАНЕНИЕ В GIT**

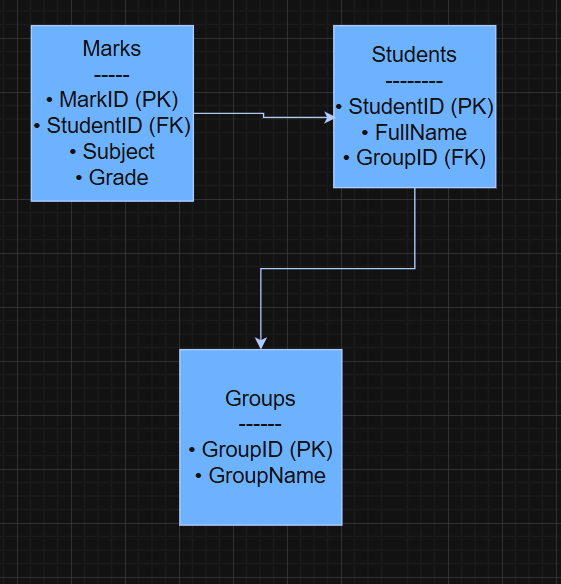




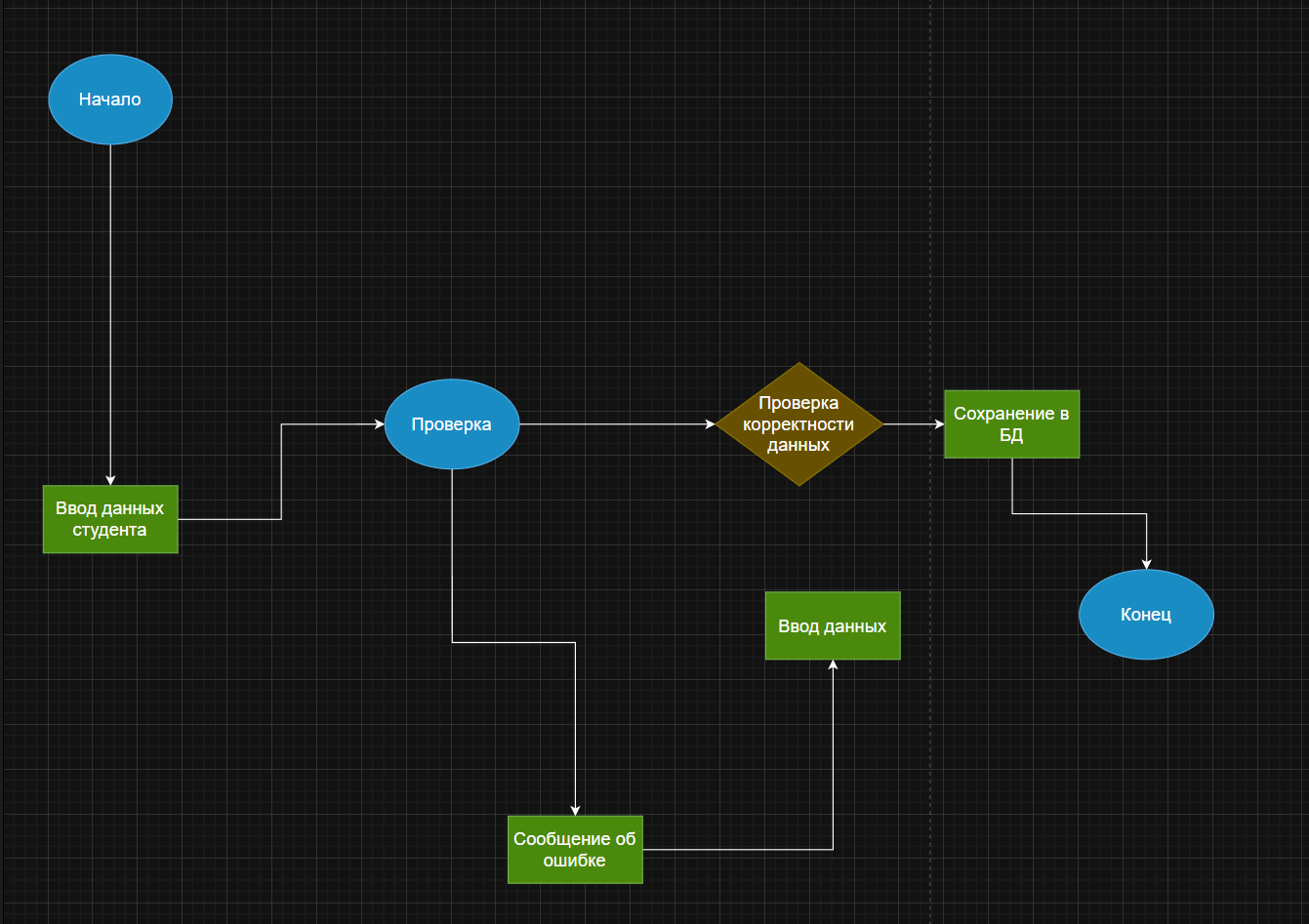




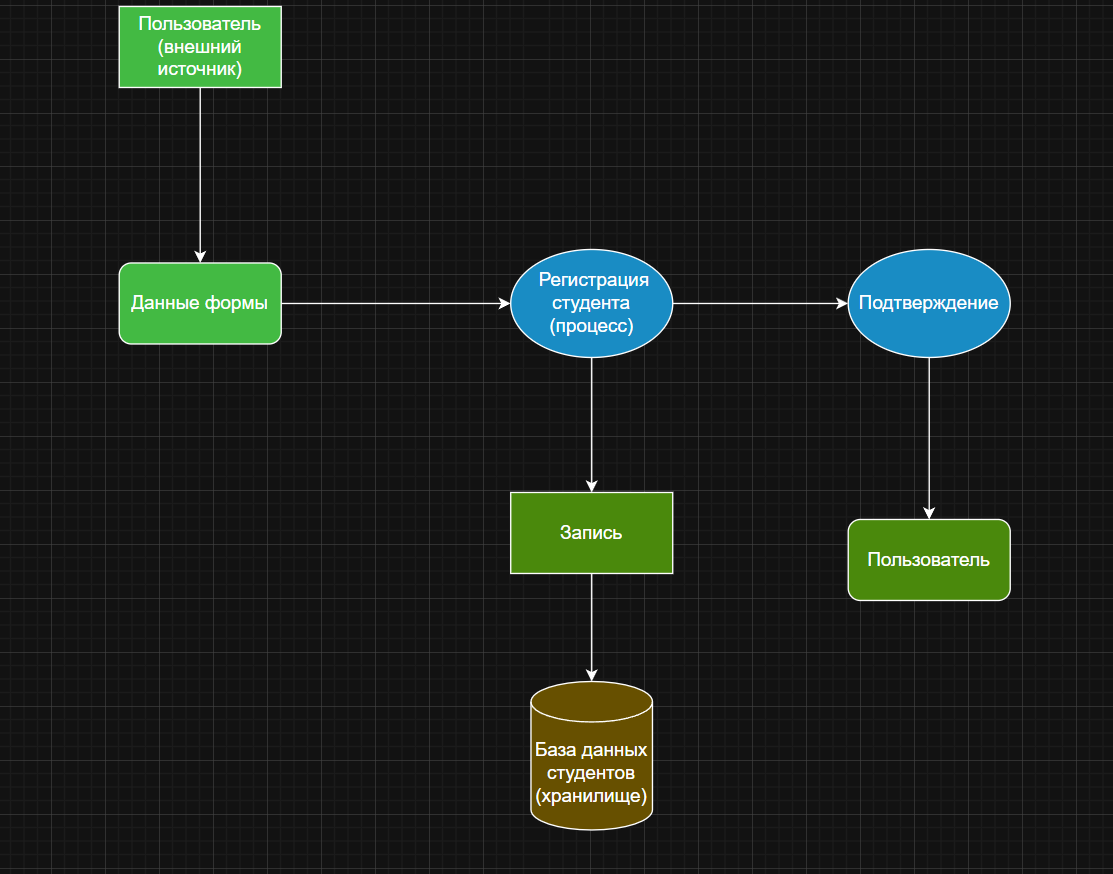
**Этап 2. Построение ER-диаграммы (схемы данных)**



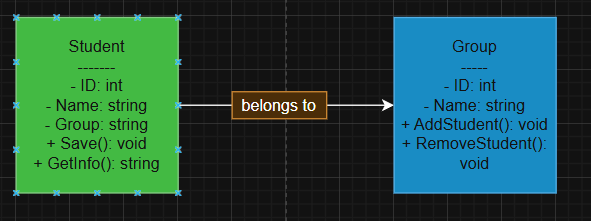
**ЭТАП 3: БЛОК-СХЕМА ОБРАБОТКИ ИСКЛЮЧЕНИЙ**

****

**ЭТАП 4: DFD-ДИАГРАММА (ПОТОКИ ДАННЫХ)**

****

**ЭТАП 5: UML-ДИАГРАММА КЛАССОВ**

****

Ссылка на мой репозиторий https://github.com/artyr989/Lab\_Exceptions\_Handling

## Контрольные вопросы (для защиты)

## 1. Что такое исключение и в чём отличие от логической ошибки?

## Исключение - это событие во время выполнения программы, которое нарушает нормальный поток выполнения (например, деление на ноль, отсутствие файла, разрыв соединения с БД). Исключения возникают в runtime и могут быть обработаны механизмами обработки ошибок.

## Отличие от логической ошибки:

## Исключение - программа не может выполнить операцию (техническая ошибка)

## Логическая ошибка - программа работает, но выдает неправильный результат (ошибка в алгоритме)

## Пример: Если программа пытается открыть несуществующий файл - это исключение. Если программа считает среднее арифметическое, но использует неправильную формулу - это логическая ошибка.

## 2. Методы идентификации сбоев

## Валидация входных данных - проверка формата, диапазонов данных

## Логирование ошибок - запись событий в журнал для диагностики

## Мониторинг и метрики - сбор показателей (количество ошибок, время отклика)

## Автоматические проверки данных - SQL-триггеры и скрипты валидации

## Тестирование - юнит-тесты, интеграционные тесты

## Триггеры оповещений - автоматические уведомления при критических событиях

## На практике я использовал: SQL-триггеры для проверки оценок студентов и автоматического логирования ошибок.

## 3. Для чего нужен ErrorLog? Какие поля обязательны?

## ErrorLog нужен для:

## Централизованного хранения информации об ошибках

## Анализа частоты и типов сбоев

## Отслеживания процесса исправления ошибок

## Статистики и отчетности

## Обязательные поля:

## ErrorID - уникальный идентификатор

## ErrorTime - время возникновения ошибки

## ModuleName - модуль/компонент где произошла ошибка

## ErrorDescription - описание ошибки

## Severity - уровень серьезности

## В моей реализации я добавил также поля для отслеживания статуса исправления: IsResolved, Resolver, ResolvedAt.

## 4. Как автоматически выявлять «сиротские» записи (orphan records) в БД?

## Сиротские записи - это записи в дочерней таблице, которые ссылаются на несуществующие записи в родительской таблице.

## Способы выявления:

SELECT G.ID, G.StudentID

FROM Grades AS G

LEFT JOIN Students AS S ON G.StudentID = S.ID

WHERE S.ID IS NULL;

На практике я реализовал SQL-триггер, который автоматически проверяет при добавлении новой оценки, существует ли студент, и записывает ошибку в ErrorLog если студента нет.

**5. Как реализовать автоматическое создание задачи на исправление критической ошибки?**

Через SQL-триггер:

CREATE TRIGGER create\_bugreport\_for\_critical

AFTER INSERT ON ErrorLog

FOR EACH ROW

WHEN NEW.Severity = 'Critical'

BEGIN

INSERT INTO BugReport (ErrorID, Title, Description, Status, AssignedTo)

VALUES (NEW.ErrorID,

'Критическая ошибка: ' || NEW.ErrorType,

NEW.ErrorDescription,

'Open',

'Администратор');

END;

Алгоритм:

* Ошибка попадает в ErrorLog
* Триггер проверяет Severity = 'Critical'
* Автоматически создается запись в BugReport
* Задача назначается ответственному

**6. Почему важно классифицировать ошибки по Severity?**

**Причины важности классификации:**

* Приоритизация исправлений - критические ошибки исправляются в первую очередь
* Эффективное распределение ресурсов - команда знает какие ошибки требуют срочного вмешательства
* Автоматизация реакции - система может сама решать как реагировать на разные типы ошибок
* Статистический анализ - понимание общей стабильности системы

**Уровни Severity которые я использовал:**

* **Info** - информационные сообщения
* **Warning** - предупреждения
* **Error** - ошибки, требующие исправления
* **Critical** - критические ошибки, требующие немедленного вмешательства

**7. Какие диаграммы вы построили и зачем каждая из них нужна?**

**1. ER-диаграмма** - показывает структуру базы данных, связи между таблицами, первичные и внешние ключи. Нужна для понимания архитектуры хранения данных.

**2. DFD (Data Flow Diagram)** - отображает потоки данных между процессами, хранилищами и внешними сущностями. Помогает понять как информация перемещается в системе.

**3. Блок-схема обработчика исключений** - визуализирует алгоритм обработки ошибок, условия и последовательность действий. Полезна для разработки и отладки кода.

**Практическая ценность:** Эти диаграммы помогли мне спроектировать систему, понять взаимодействие компонентов и реализовать корректную обработку ошибок в SQLite.