**Теоретико-множественные операции**

* **✅**Декартово произведение наборов записей

Декартово произведение в SQL представляет собой операцию, при которой каждая запись из одного набора комбинируется с каждой записью из другого набора. Результатом является новый набор, содержащий все возможные комбинации записей из обоих исходных наборов.

Пример использования:

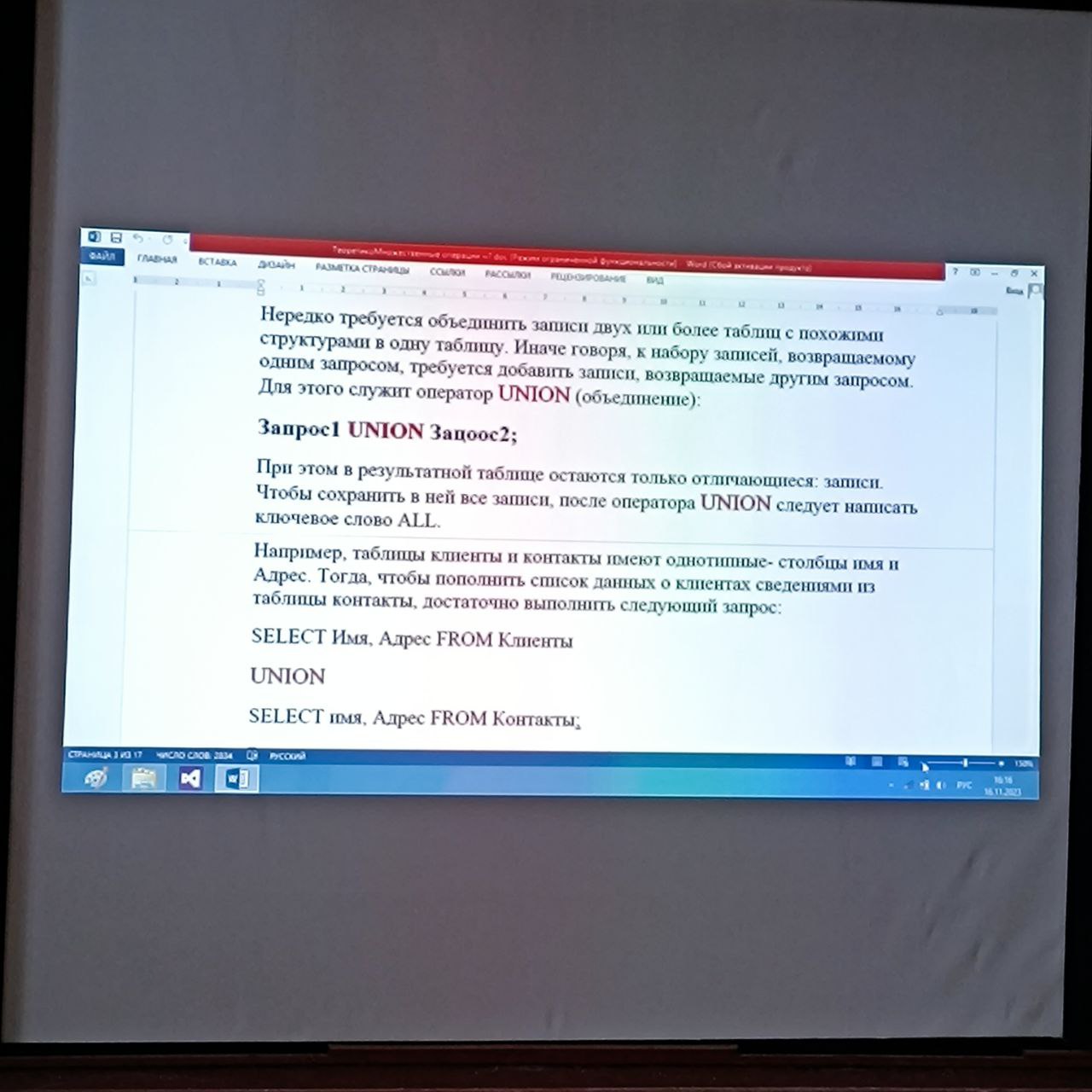
Предположим, у нас есть две таблицы - "Таблица1" и "Таблица2", и мы хотим получить декартово произведение их записей. Мы можем использовать следующий SQL-запрос:

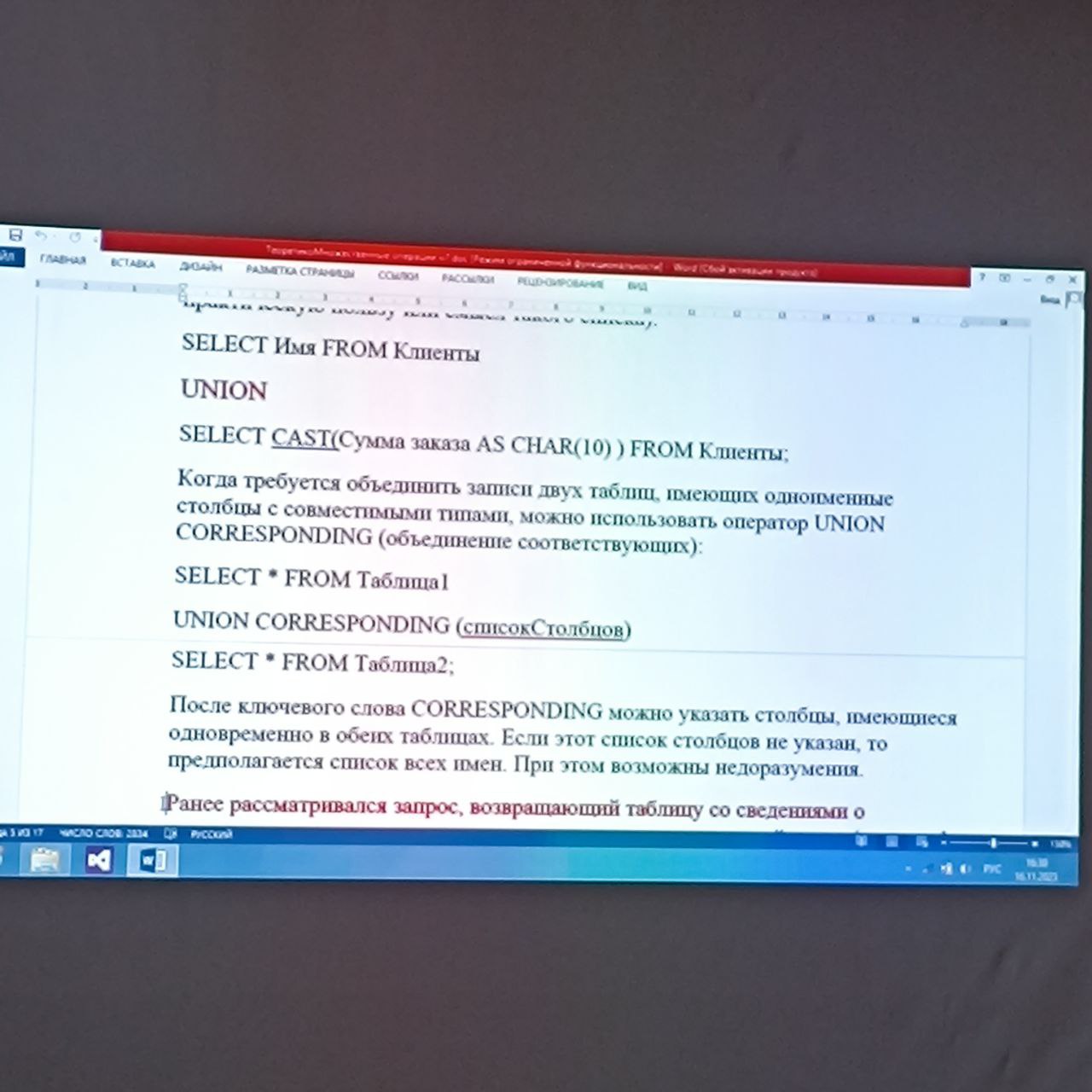
SELECT \*

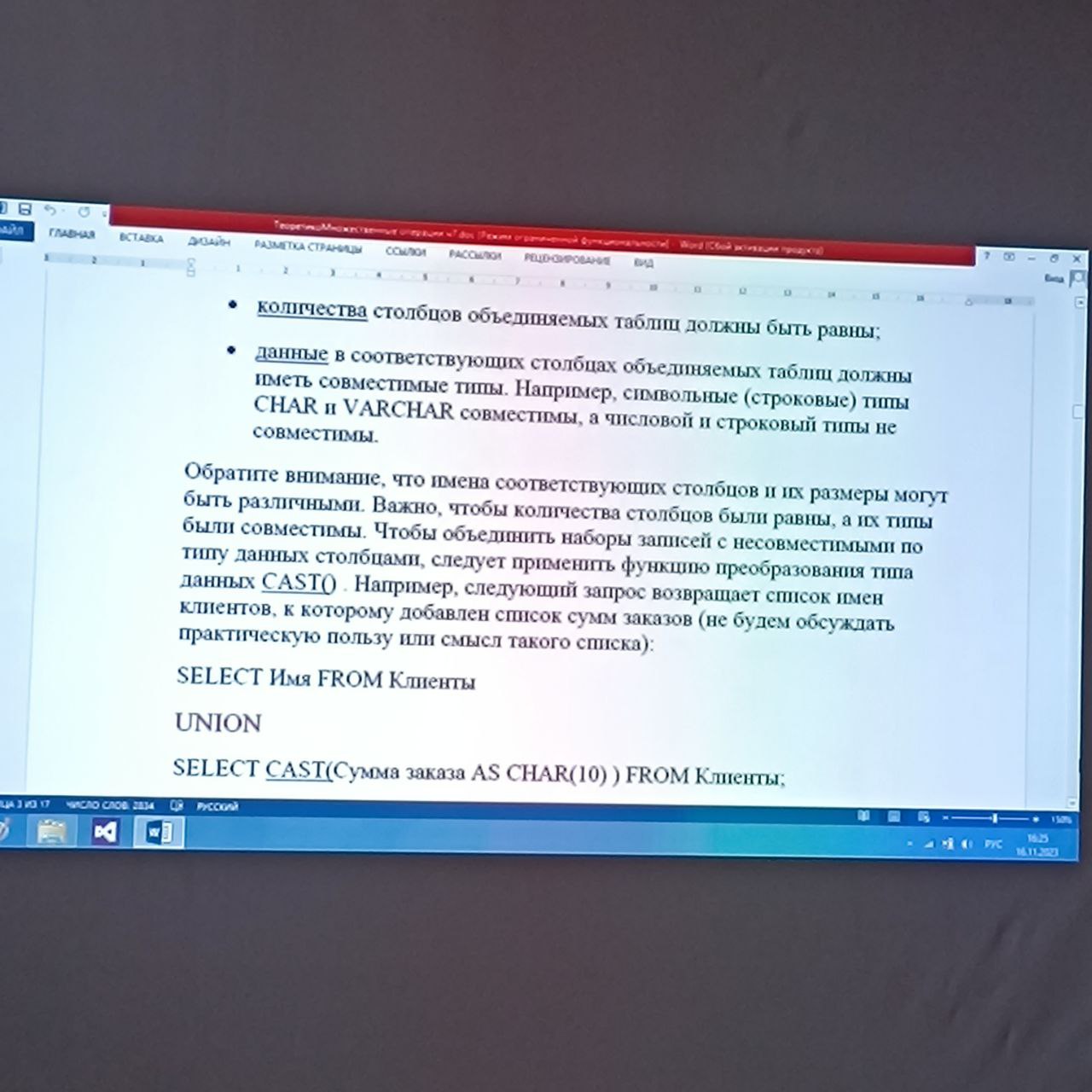
FROM Таблица1, Таблица2;

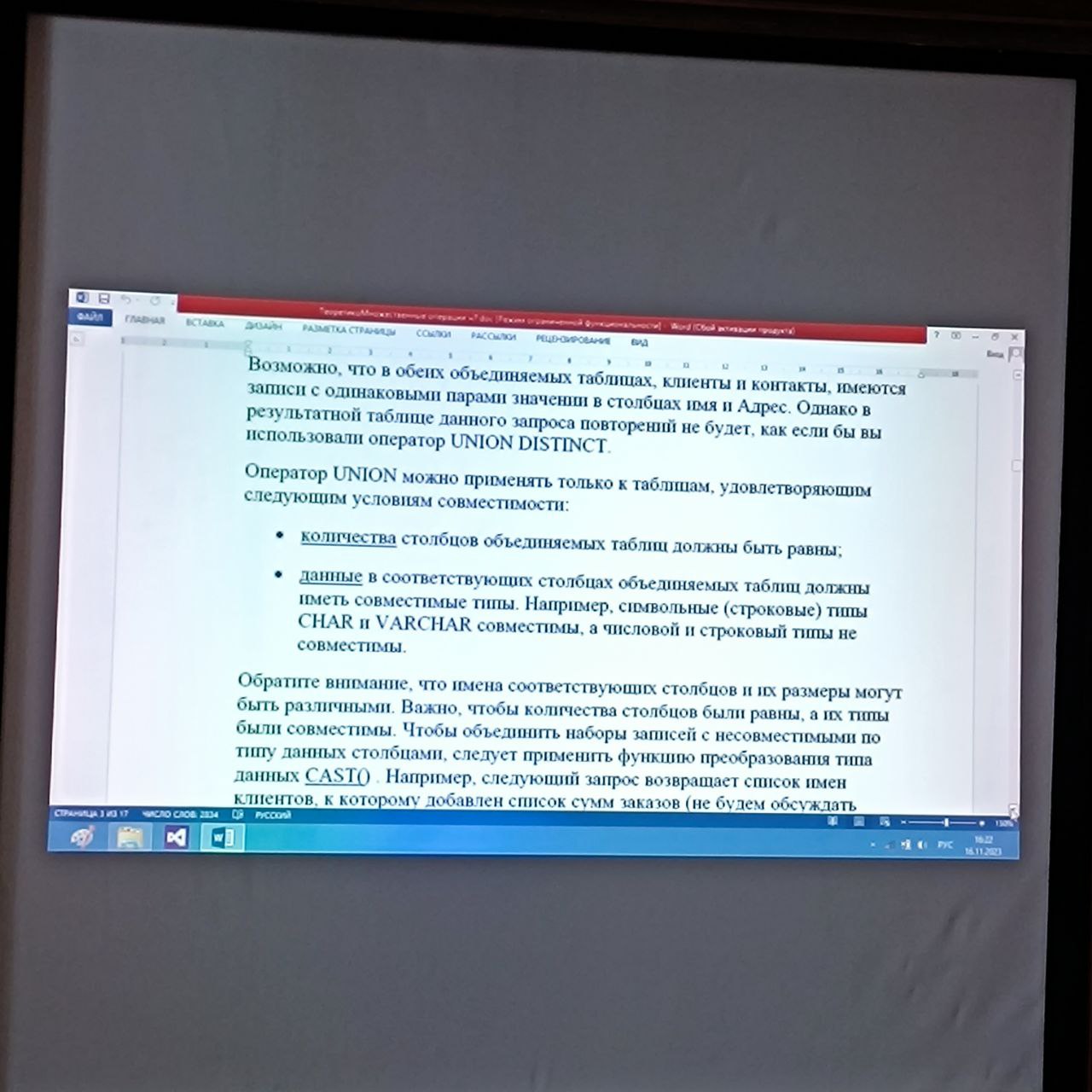
Этот запрос вернет результат, в котором каждая запись из "Таблица1" будет комбинироваться со всеми записями из "Таблица2", создавая полный набор комбинаций записей.

* **✅**Объединение наборов записей (UNION)

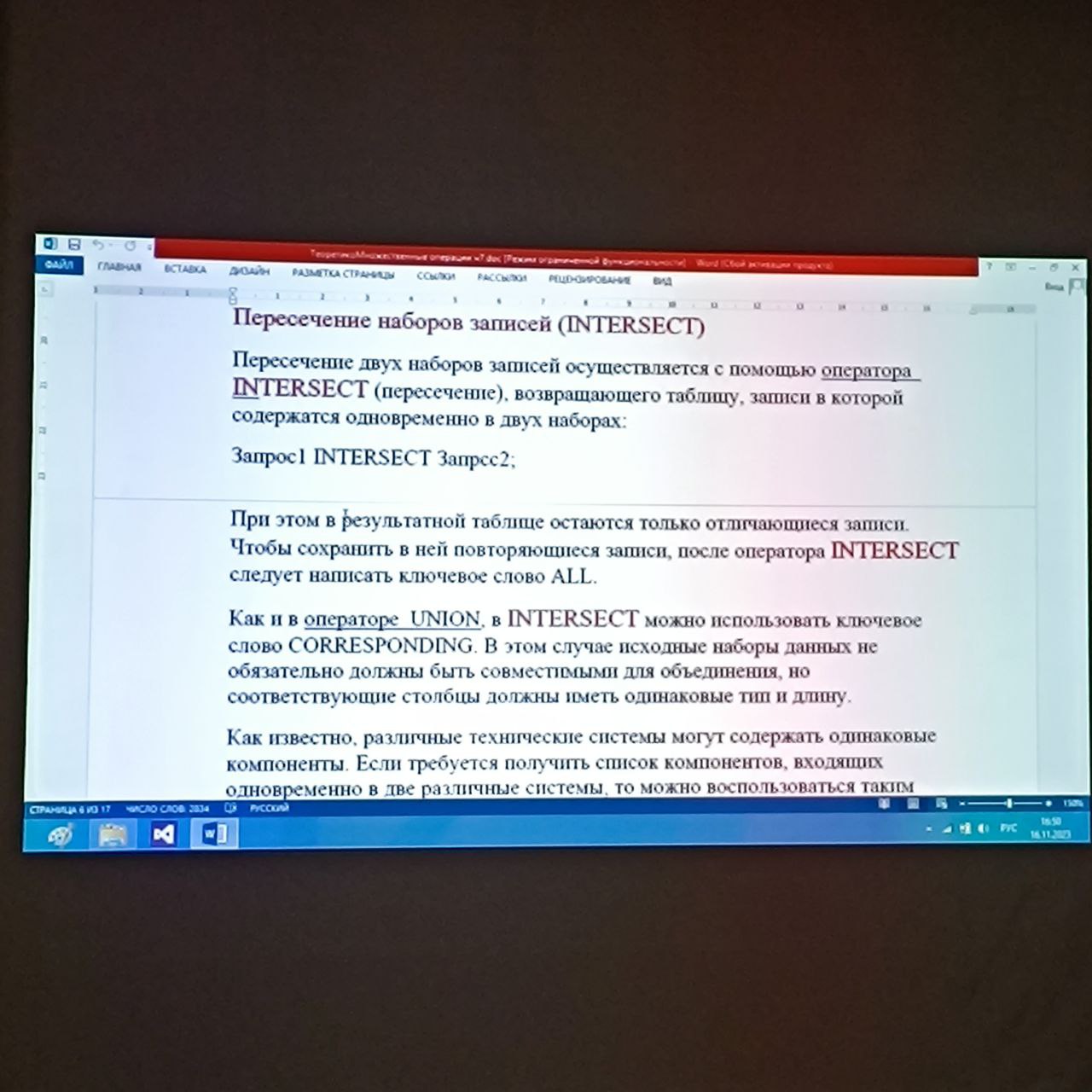
****

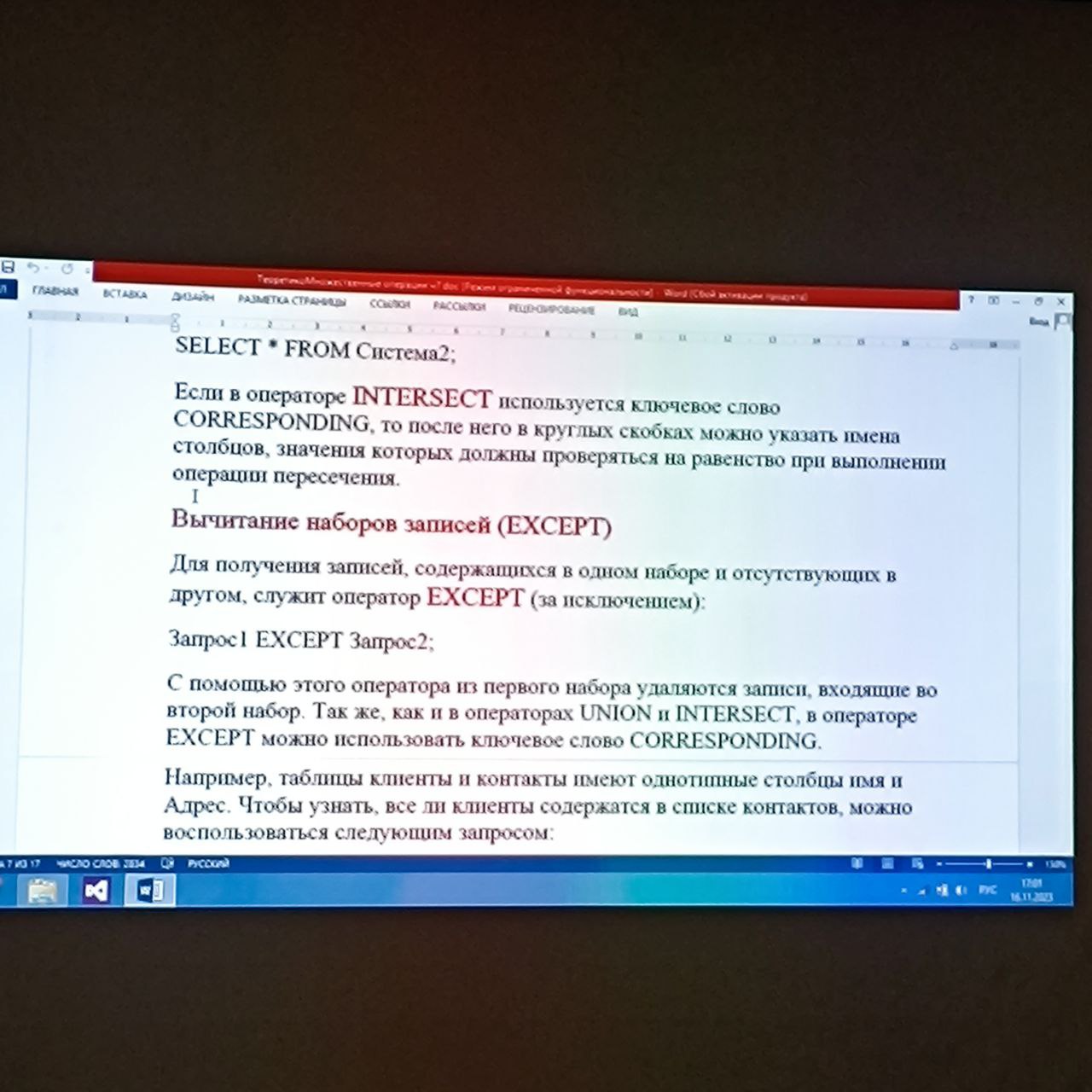


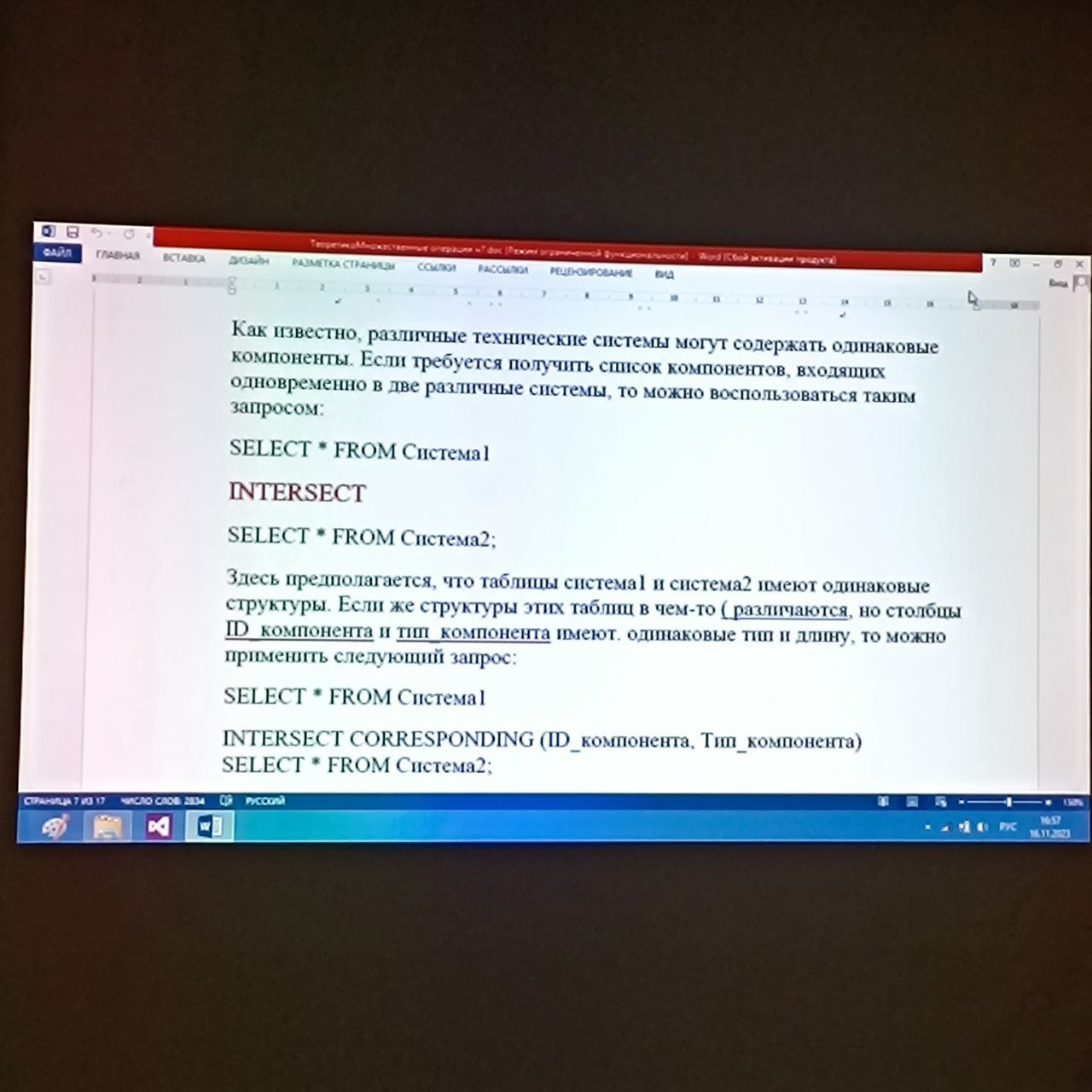




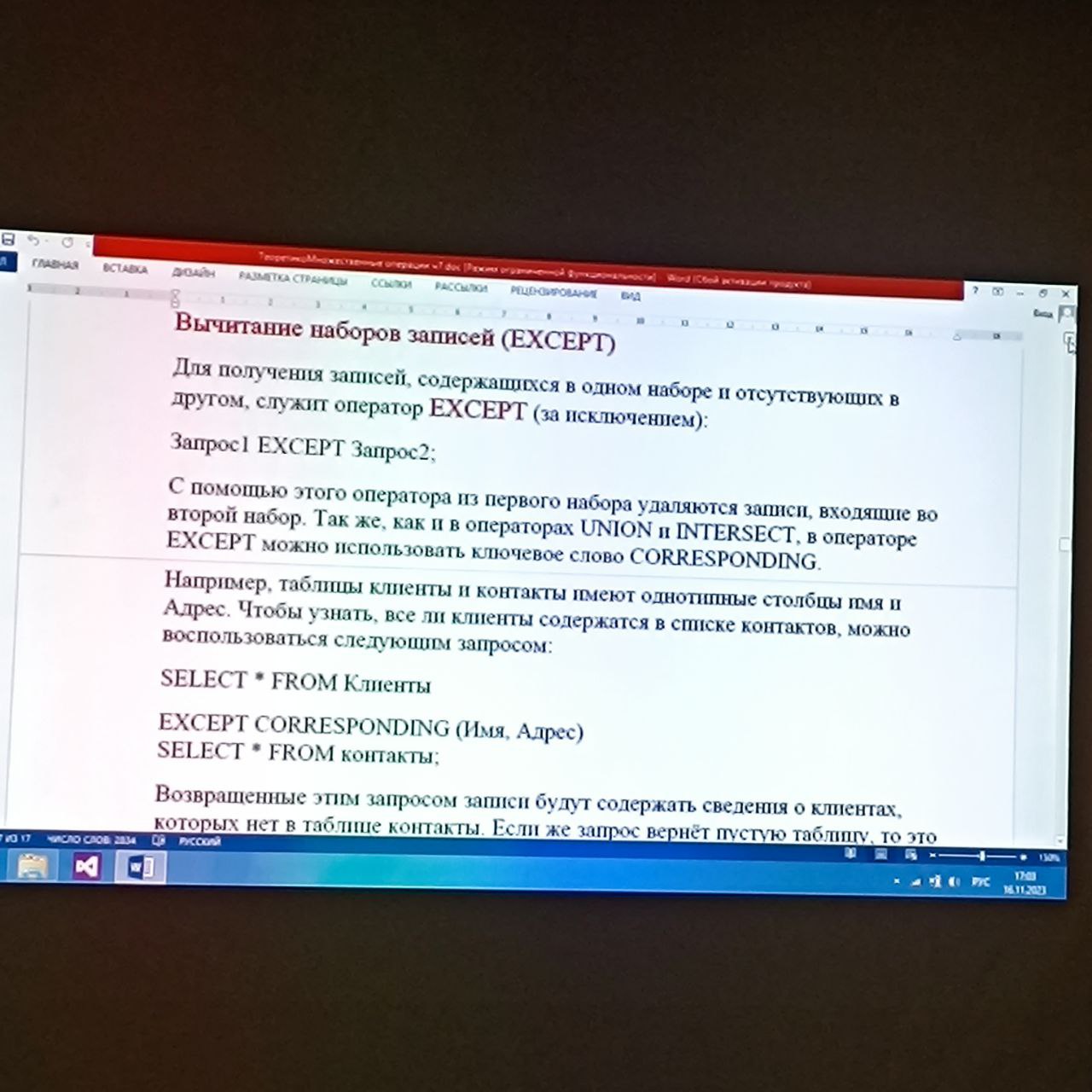
* **✅**Пересечение наборов записей (INТЕRSЕСТ)







* **✅**Вычитание наборов записей (EXCEPT)



**Операции соединения**

* **✅**Естественное соединение (NATURAL JOIN)

Объяснение: Естественное соединение объединяет таблицы на основе всех столбцов с одинаковыми именами. В результате все столбцы с одинаковыми именами объединяются в один столбец.

Пример использования:

SELECT \* FROM таблица1 NATURAL JOIN таблица2;

В этом примере, если обе таблицы содержат одинаковый столбец, результат будет включать его только один раз.

* **✅**Условное соединение ( JOIN ... ON)

Объяснение: Условное соединение позволяет объединять таблицы на основе заданного условия, указанного в операторе ON.

Пример использования:

SELECT \* FROM Табл1 JOIN Табл2 ON

Табл1.поле = Табл2. поле;

В этом запросе, строки объединяются только тогда, когда значения в столбце "поле" совпадают.

* **✅**Соединение по именам столбцов (JOIN... USING)

Объяснение: Этот тип соединения использует один или несколько общих столбцов для объединения таблиц. Столбцы перечисляются в операторе USING.

Пример использования:

SELECT \* FROM Табл1 JOIN Табл2 USING (поле);

В данном случае, результат включит только один столбец "поле".

* **✅**Внешние соединения

Внешние соединения (Outer Joins) представляют собой метод объединения данных из двух или более таблиц с сохранением строк, которые не имеют соответствующих значений в объединяемой таблице. В отличие от внутренних соединений, которые возвращают только те строки, для которых есть соответствие в обеих таблицах, внешние соединения позволяют включать все строки из одной таблицы и соответствующие им строки из другой таблицы, даже если соответствие отсутствует.

* **✅**Левое соединение (LEFT OUTER JOIN)

Объяснение: Левое соединение возвращает все строки из левой таблицы и соответствующие строки из правой таблицы. Если нет соответствия, возвращаются NULL значения для столбцов из правой таблицы.

Пример использования:

SELECT \* FROM sotrudniki LEFT OUTER JOIN transport

ON sotrudniki.car\_id = transport.car\_id;

* **✅**Правое соединение (RIGHT OUTER JOIN)

Объяснение: Правое соединение возвращает все строки из правой таблицы и соответствующие строки из левой таблицы. Если нет соответствия, возвращаются NULL значения для столбцов из левой таблицы.

Пример использования:

SELECT \* FROM sotrudniki

RIGHT OUTER JOIN transport ON sotrudniki.car\_id = transport.car\_id;

В этом запросе возвращаются все машины и соответствующие сотрудники. Если у машины нет хозяйна, то для соответствующих столбцов сотрудников будут NULL значения.

* **✅**Полное соединение (FULL JOIN)

Объяснение: Полное соединение возвращает все строки из обеих таблиц. Если нет соответствия, возвращаются NULL значения для соответствующих столбцов.

Пример использования:

SELECT \* FROM sotrudniki FULL JOIN transport

ON sotrudniki.car\_id = transport.car\_id;

Этот запрос вернет всех сотрудников и все машины, при этом если у сотрудника нет машины или у машины нет сотрудника, соответствующие значения будут NULL.

* **✅**Объединение-соединение (UNION JOIN)

Объяснение: Объединение-соединение используется для объединения результатов UNION с использованием JOIN-условия.

Пример использования:

SELECT \* FROM табл1 UNION JOIN табл2 USING (поле);

Этот запрос объединяет результаты двух таблиц с использованием оператора UNION и соединяет их по общему столбцу "поле".

* **✅**Рекурсивные запросы

Объяснение: Рекурсивные запросы позволяют выполнять запросы, которые ссылаются на самих себя. Это часто используется для работы с иерархическими структурами данных.

Пример использования:

WITH RECURSIVE EmployeeHierarchy AS (

SELECT employee\_id, manager\_id FROM employees WHERE manager\_id IS NULL

UNION

SELECT e.employee\_id, e.manager\_id FROM employees e

JOIN EmployeeHierarchy eh ON e.manager\_id = eh.employee\_id

)

SELECT \* FROM EmployeeHierarchy;

**✅**Добавление новых записей

Объяснение: Эта операция позволяет вам вставить новые данные в таблицу базы данных.

Пример:

INSERT INTO users (id, name, email) VALUES (1, 'Miroslav', qwe@gmail.com);

Этот запрос вставляет новую запись в таблицу "users" с указанными полями.

**✅**Удаление записей

Объяснение: Эта операция удаляет одну или несколько записей из таблицы базы данных.

Пример:

DELETE FROM users WHERE id = 1;

Этот запрос удаляет запись из таблицы " users " с указанным идентификатором пользователя.

**✅**Изменение данных

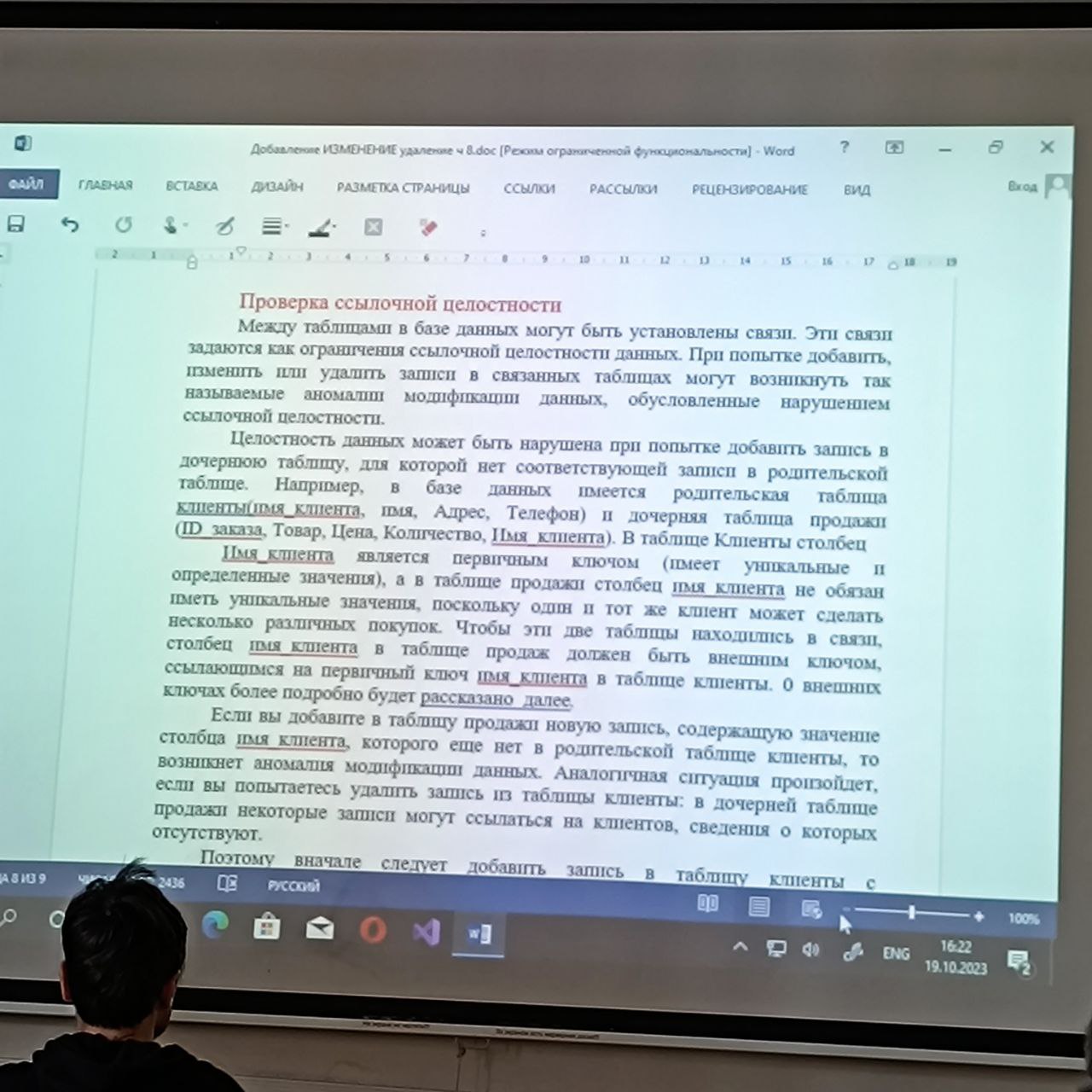
Объяснение: Эта операция обновляет существующие данные в таблице базы данных.

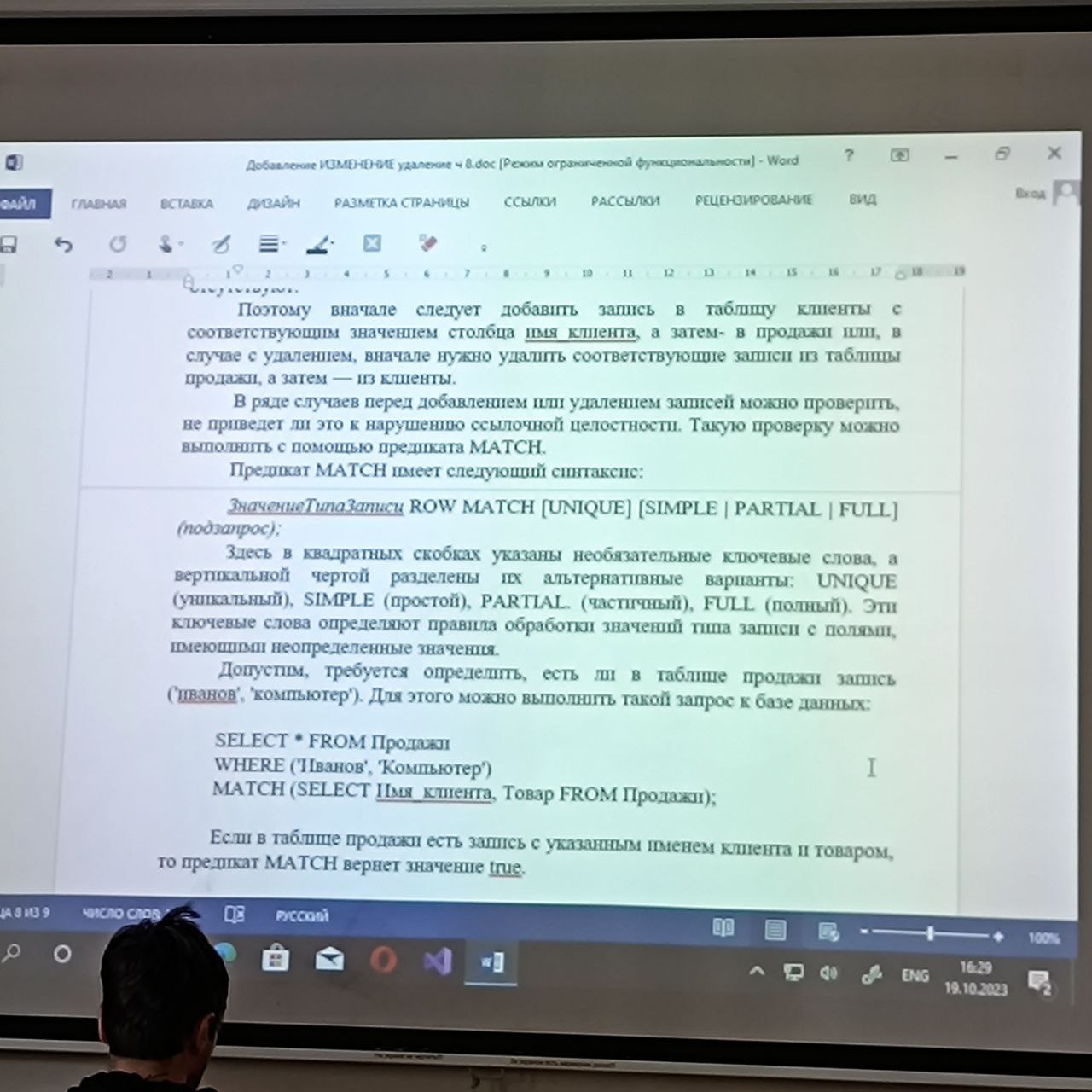
Пример:

UPDATE postavshiki SET flag = 0 WHERE city =Ростов-на-Дону;

Этот запрос изменяет флаг (логическое удаление) поставщика в соответствующей таблице с указанным названием города.

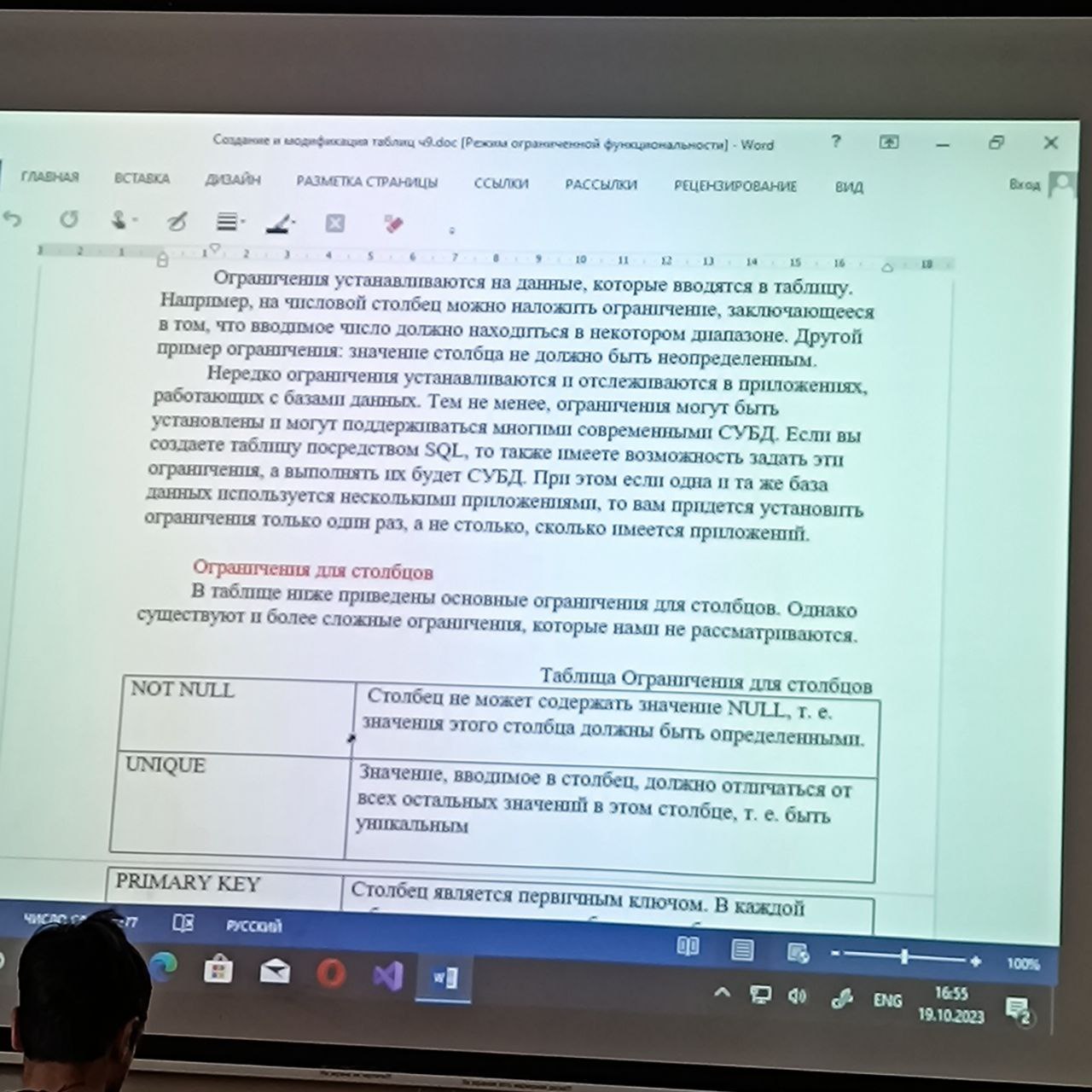
**✅**Проверка ссылочной целостности

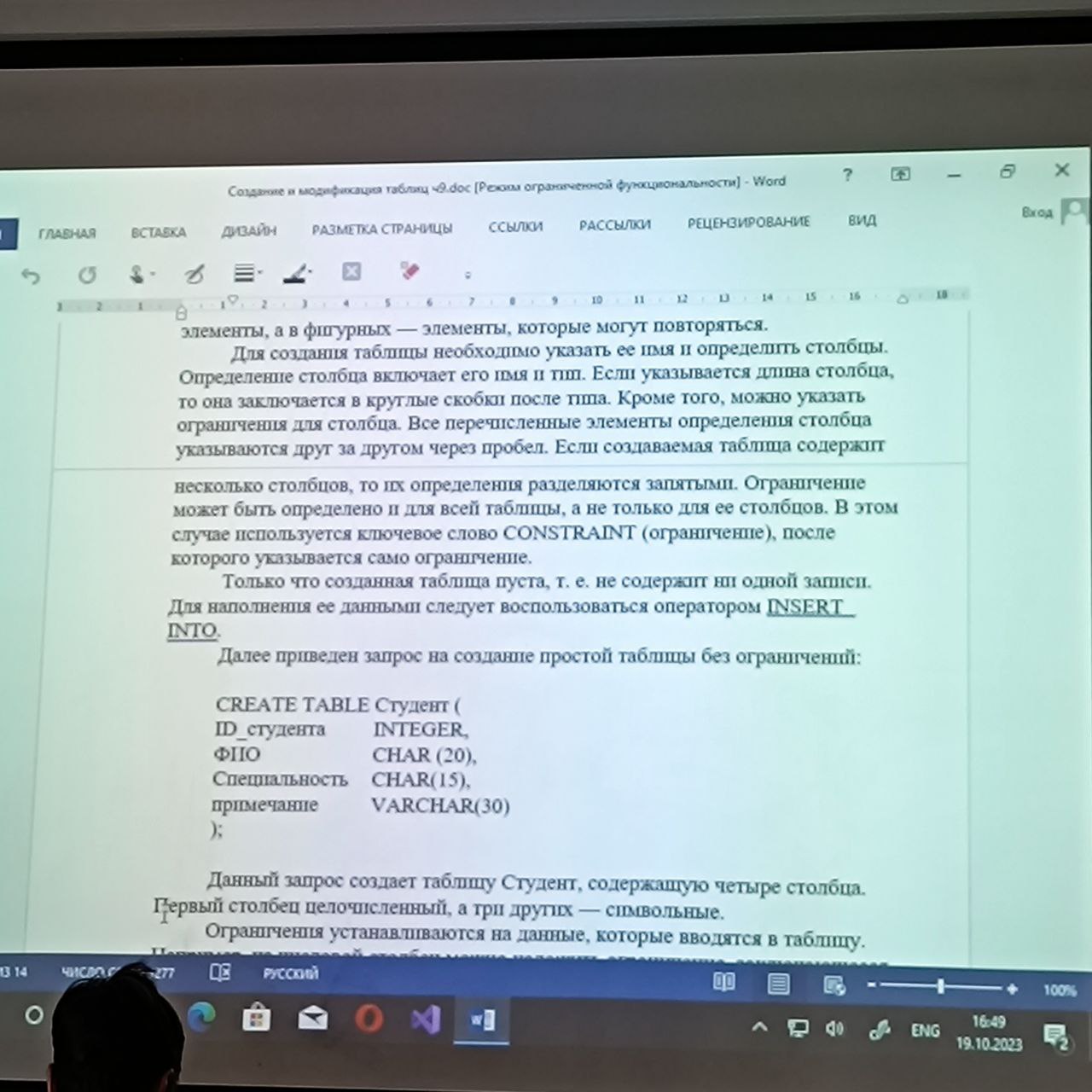


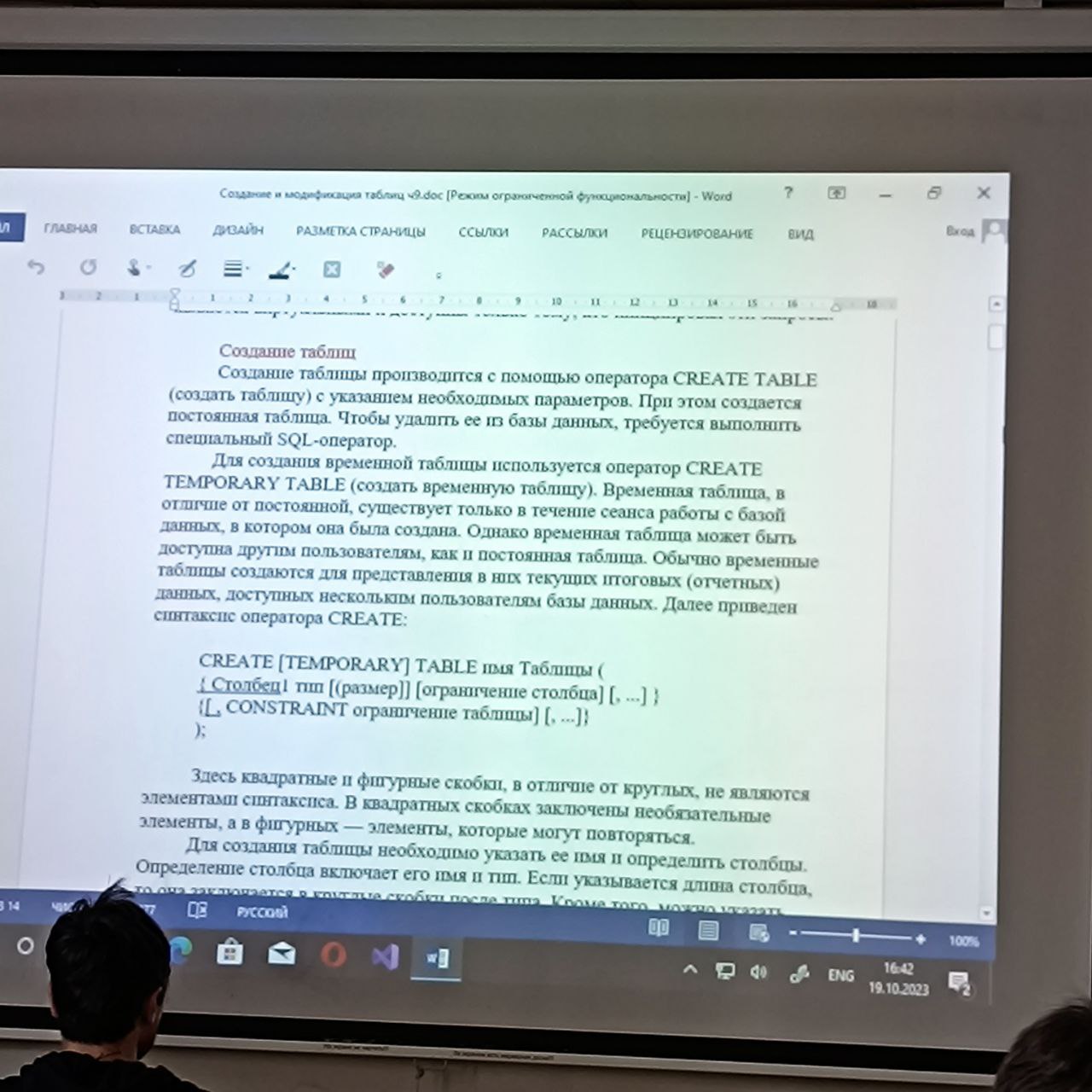


**✅**Создание и модификация таблиц

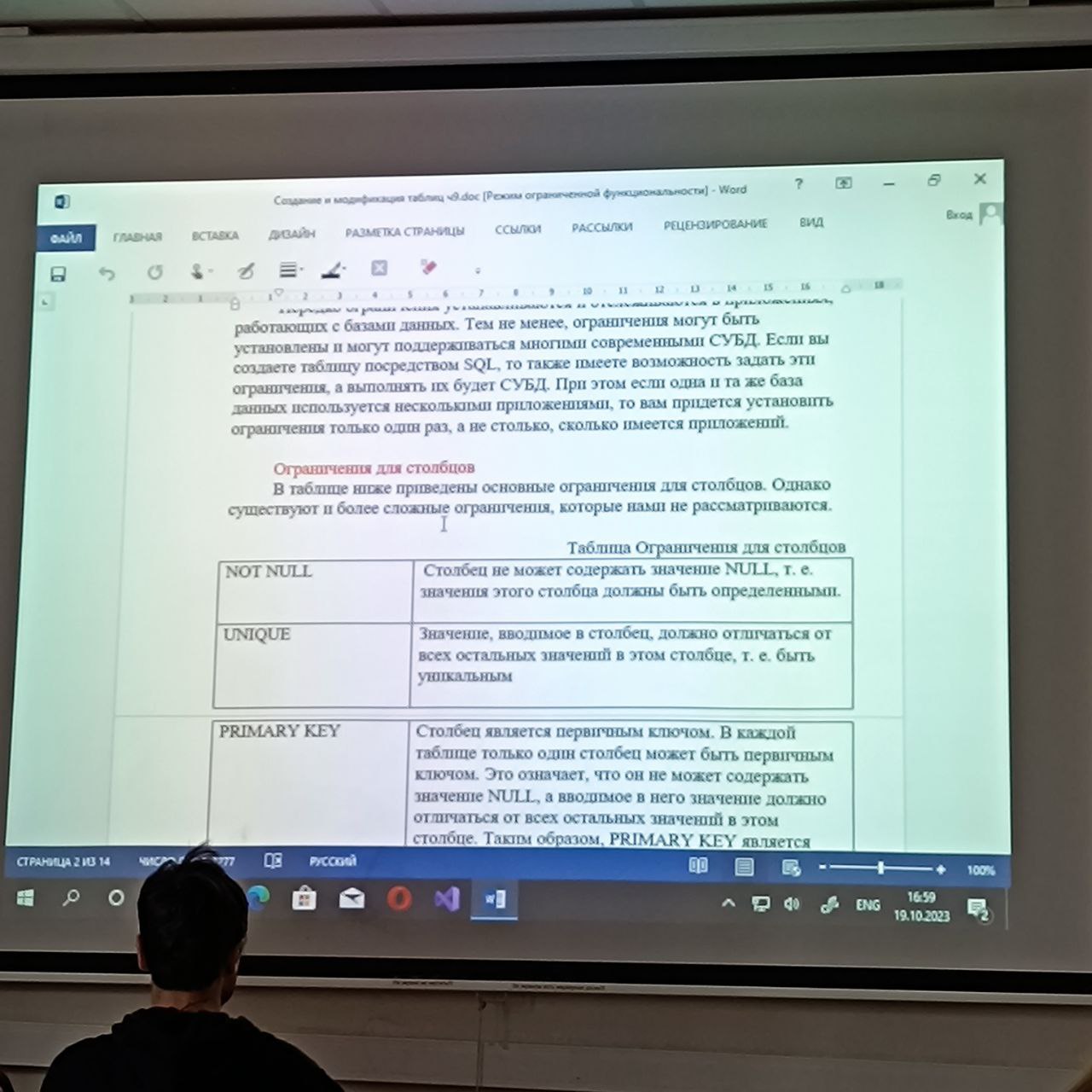
****

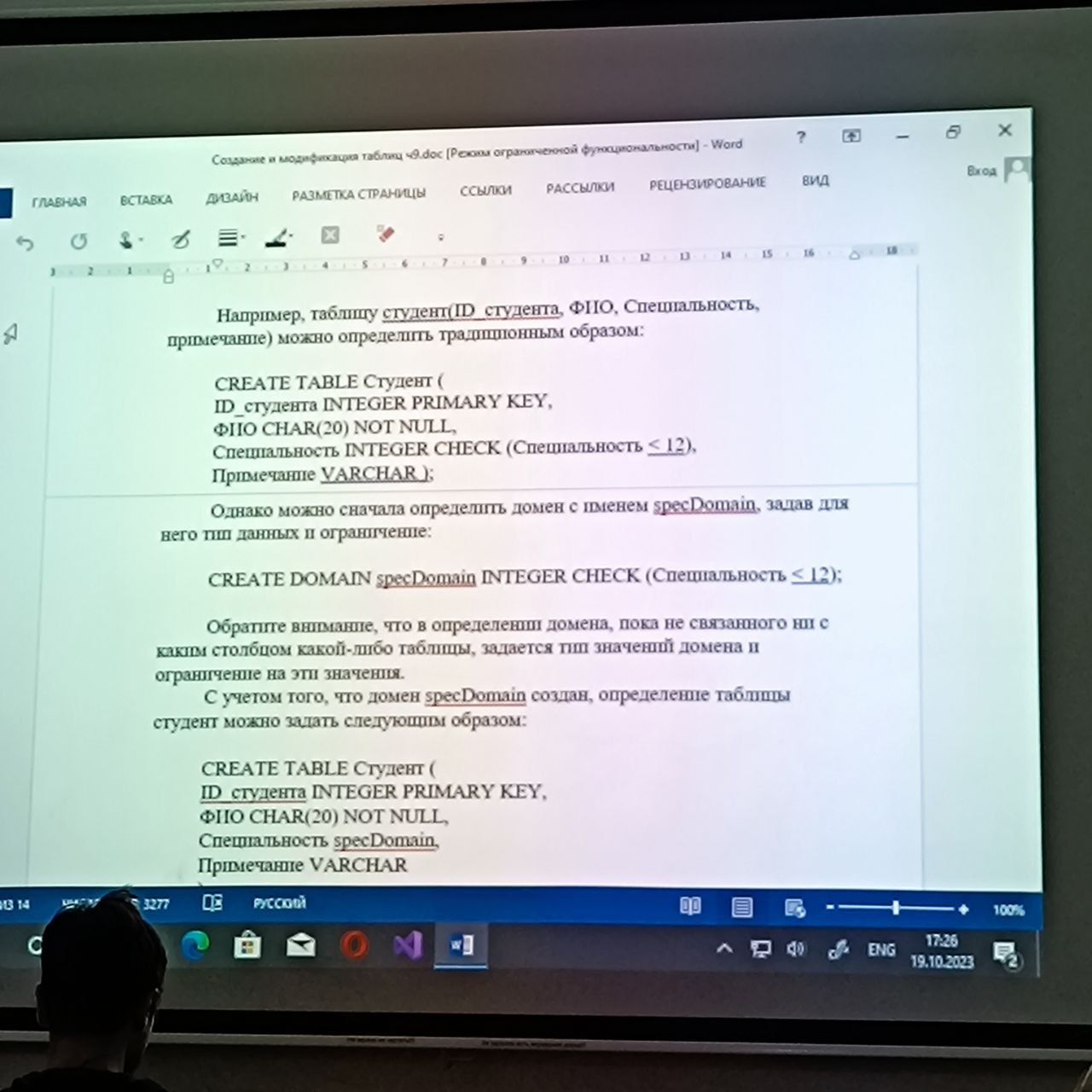


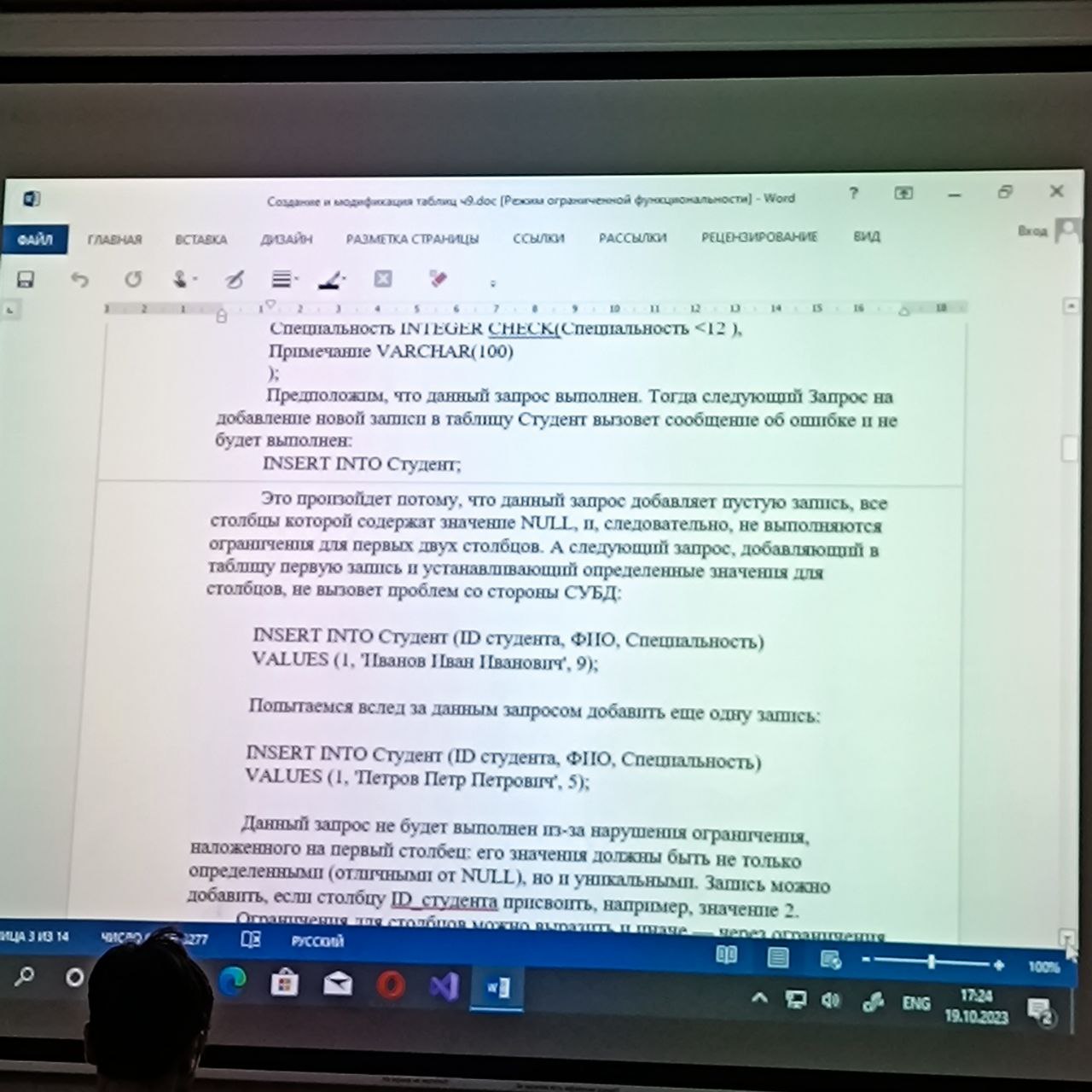


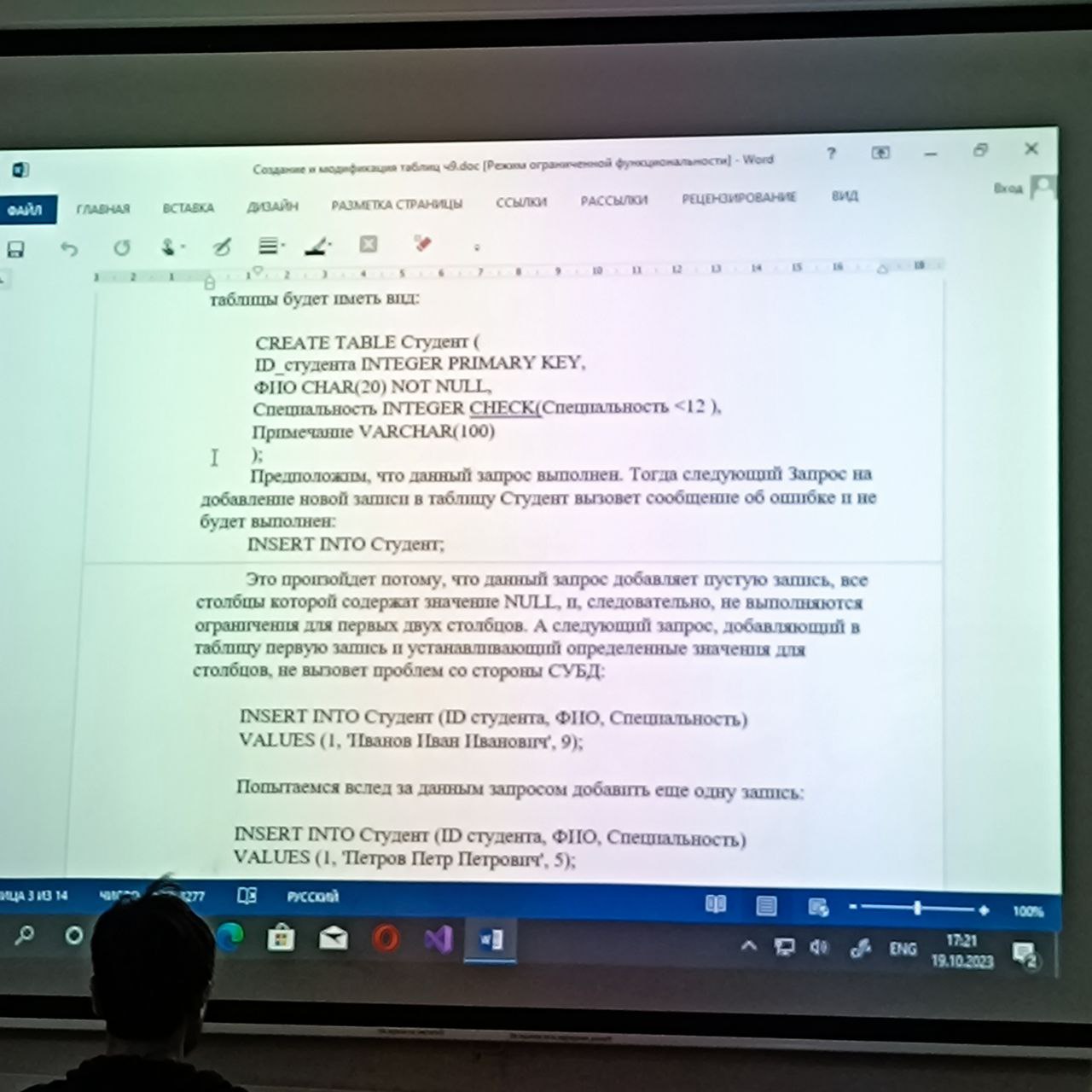


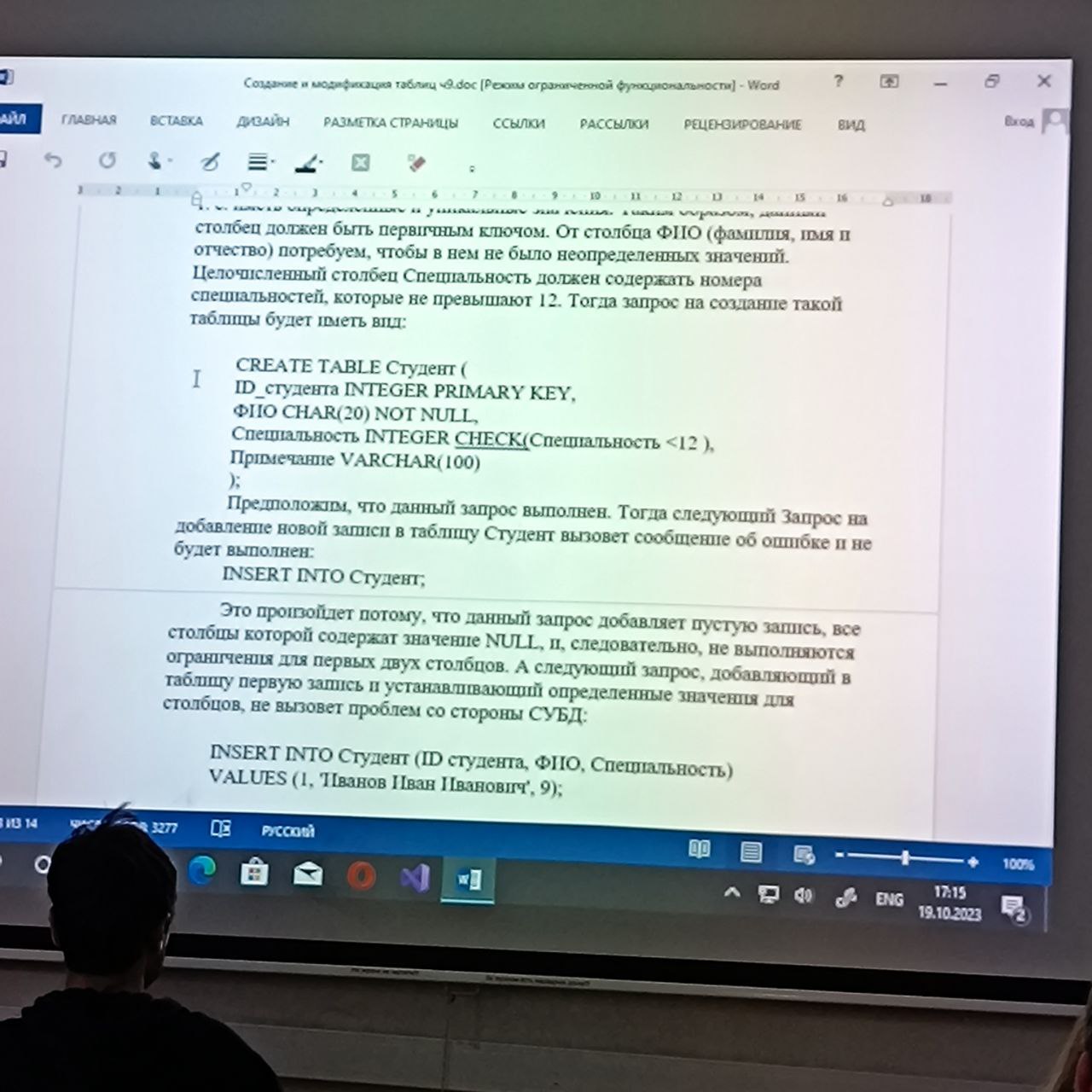
* **✅**Ограничения для столбцов

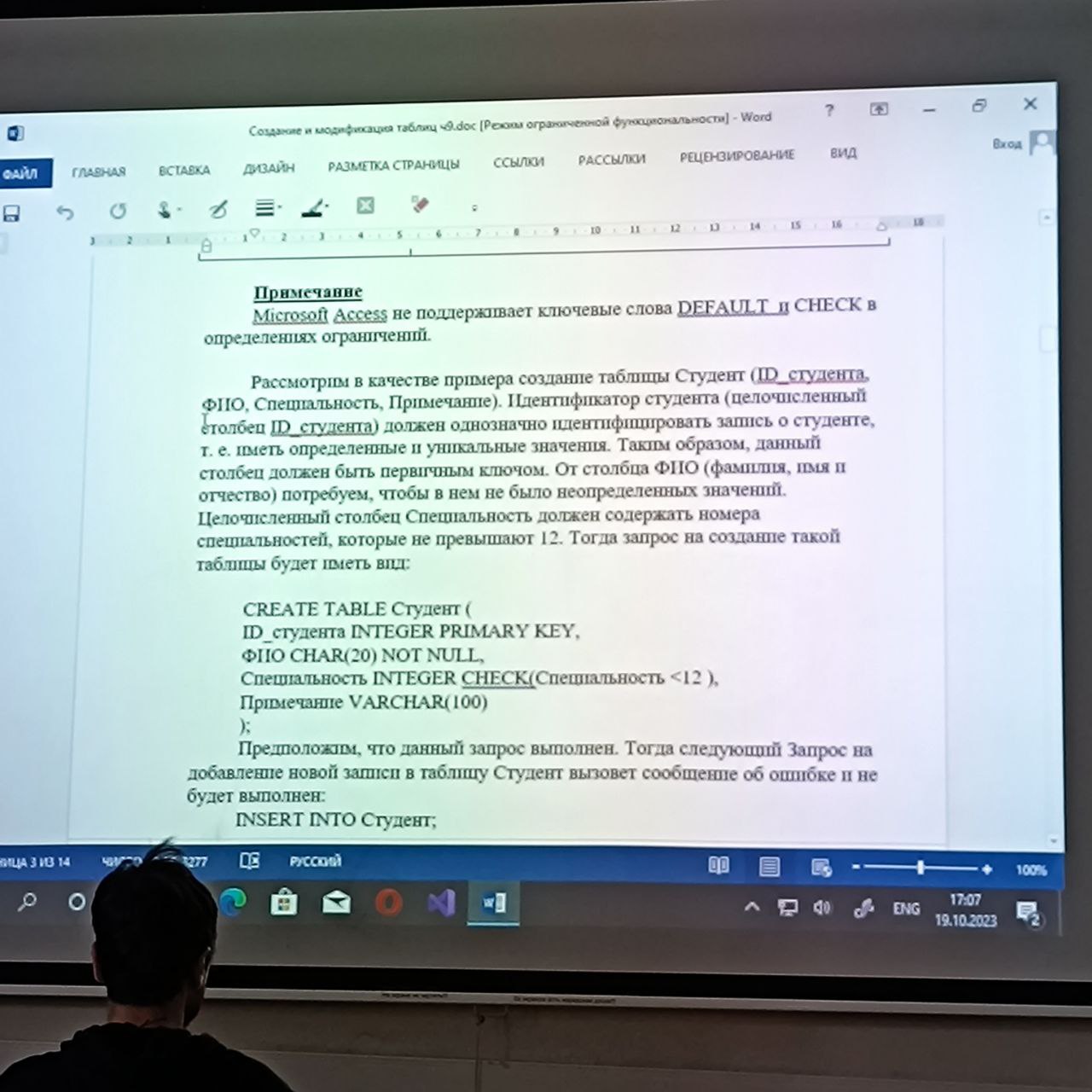
****

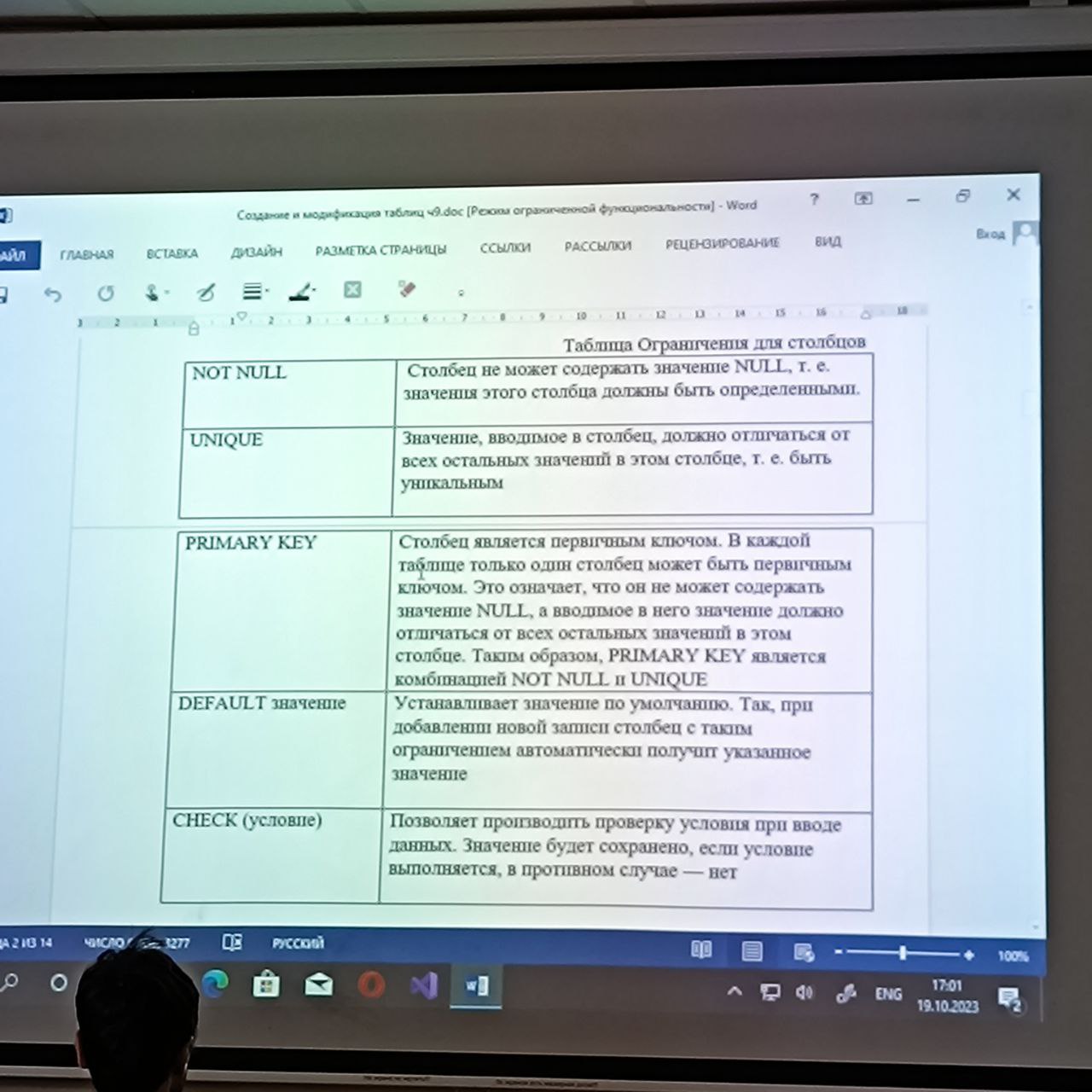




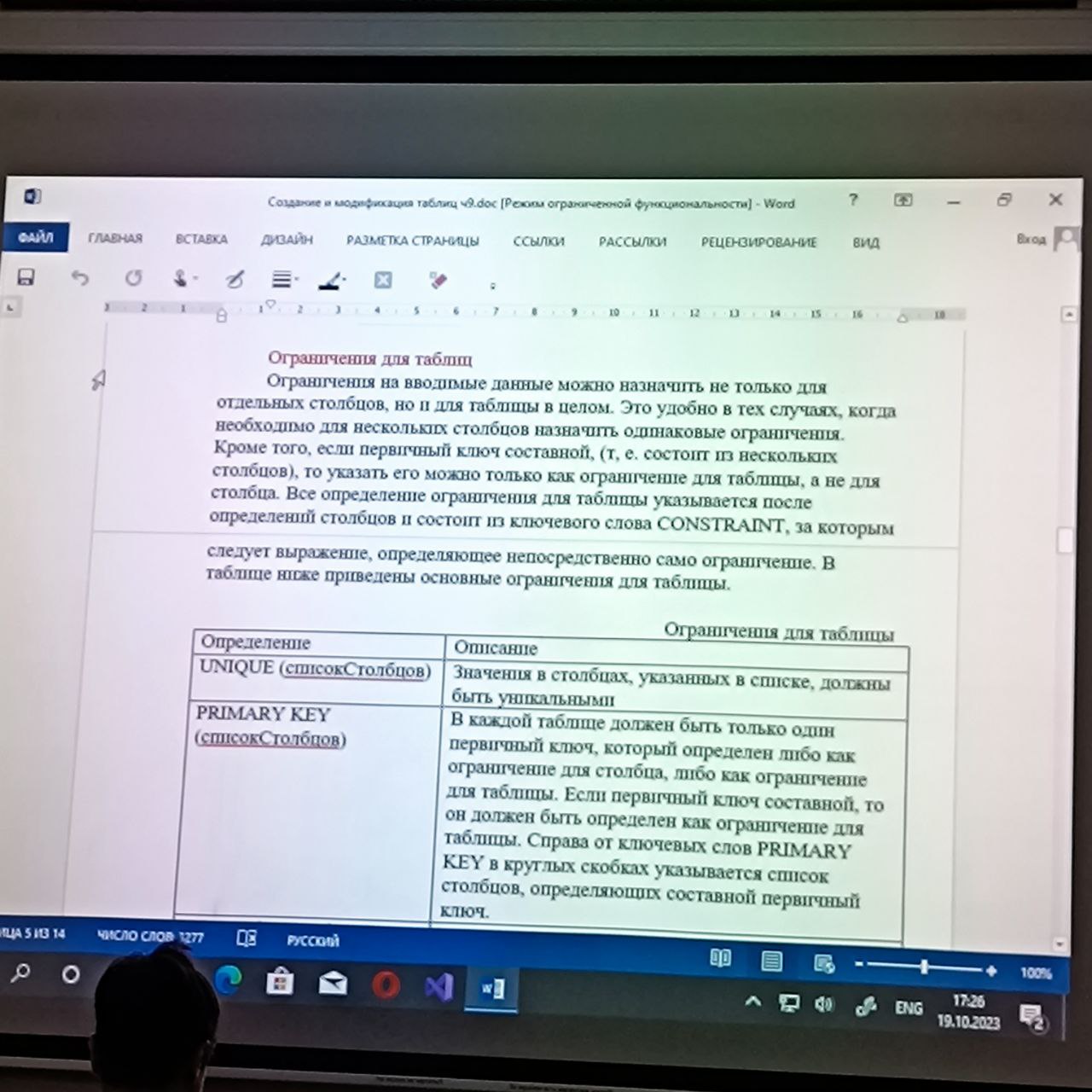


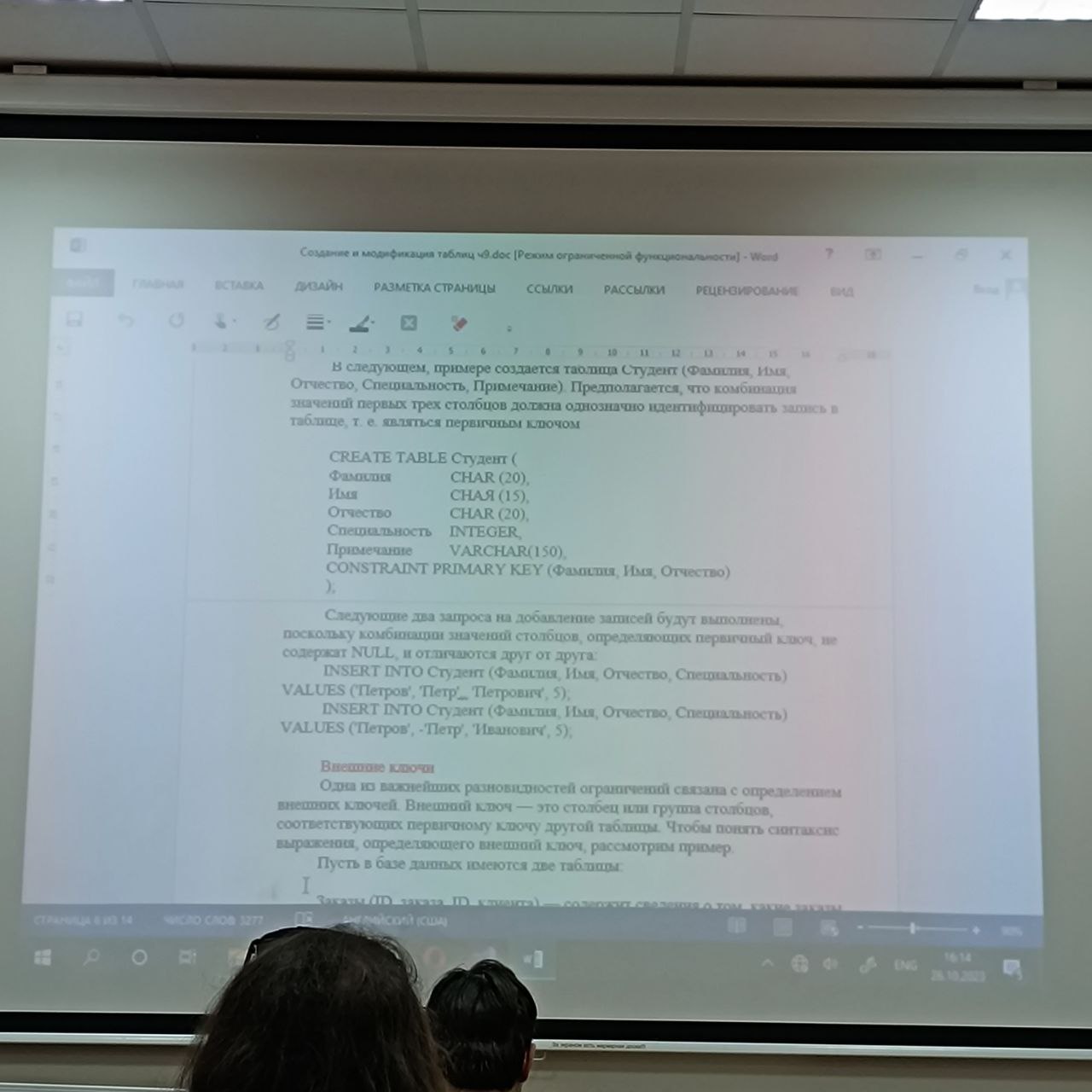


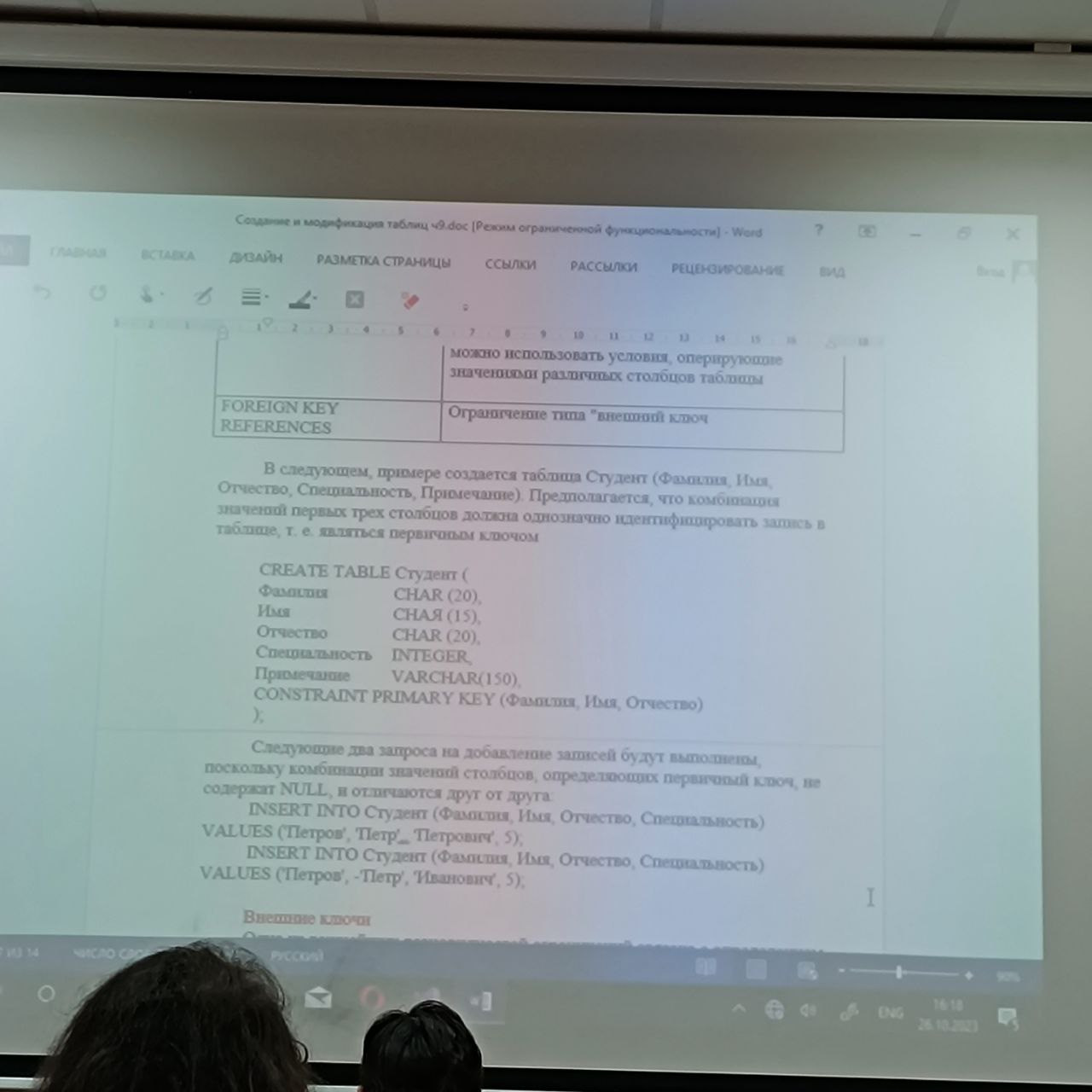




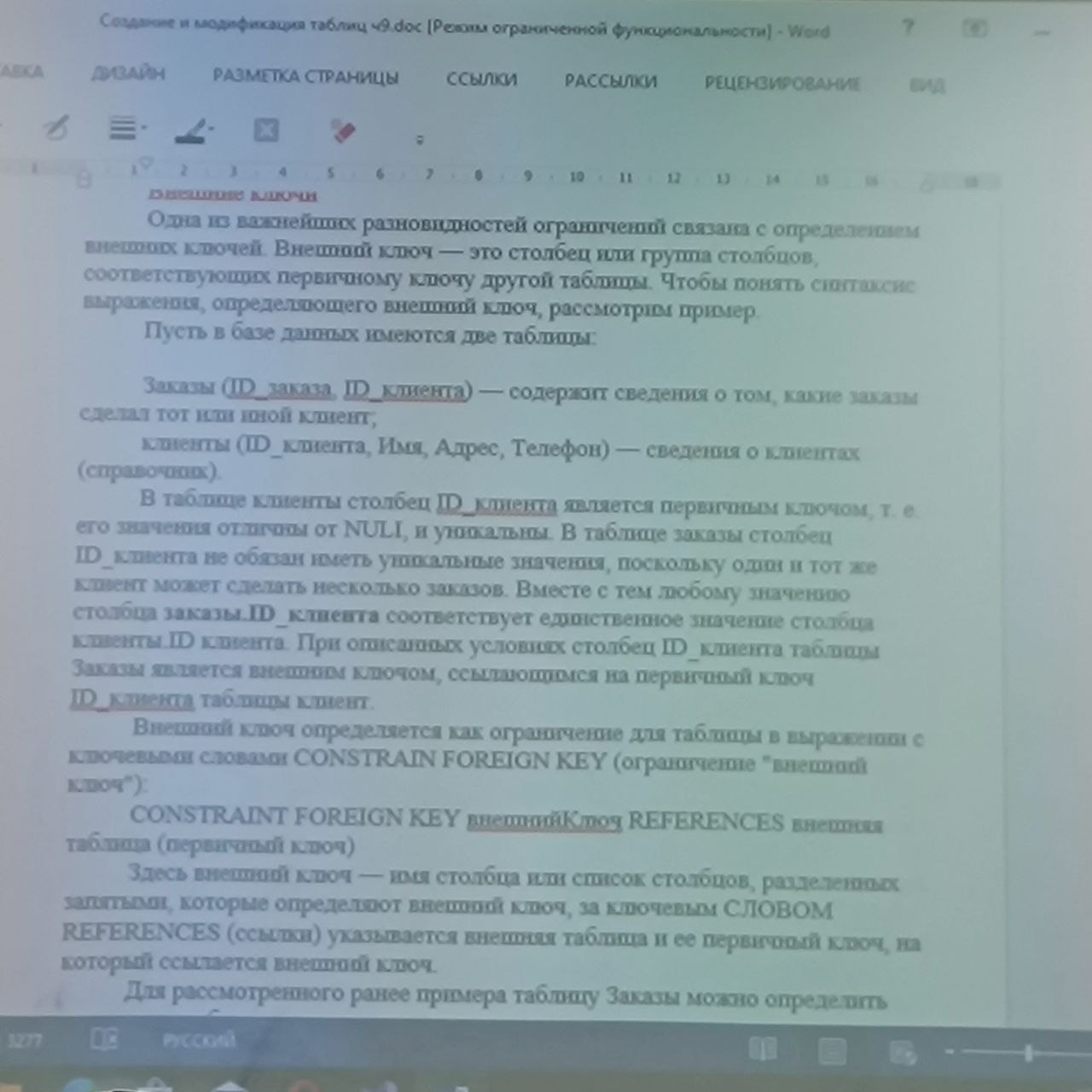
* **✅**Ограничения для таблиц

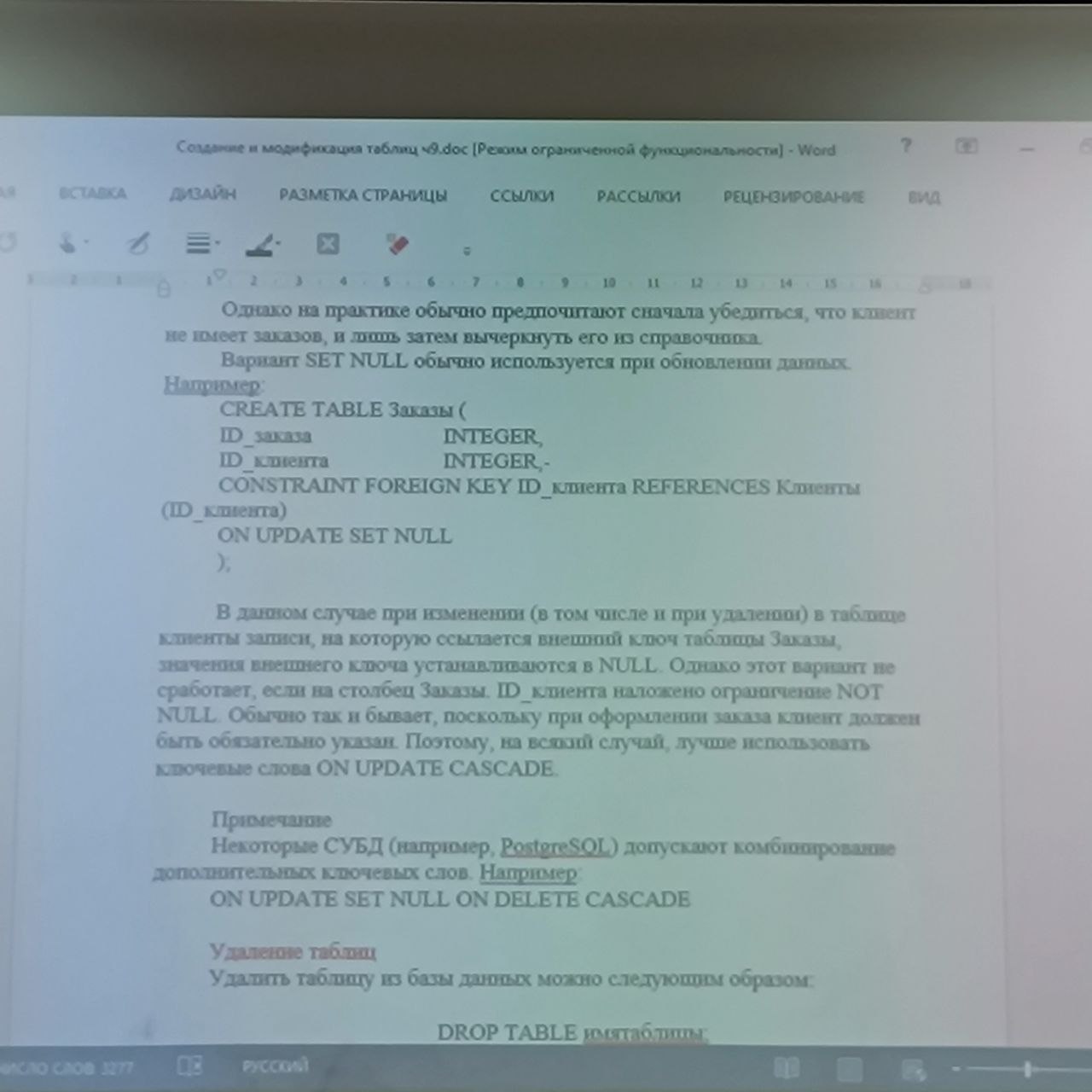
****

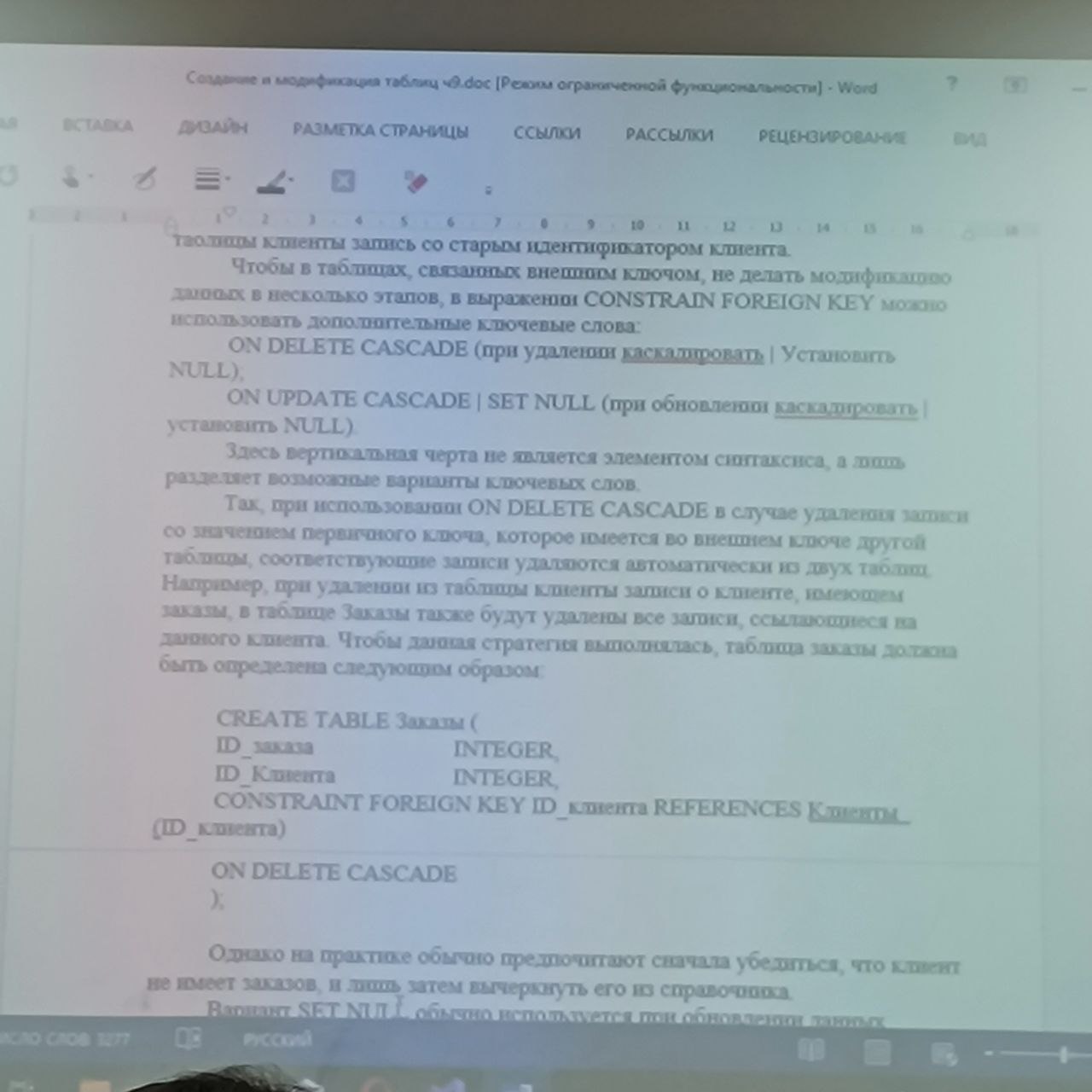


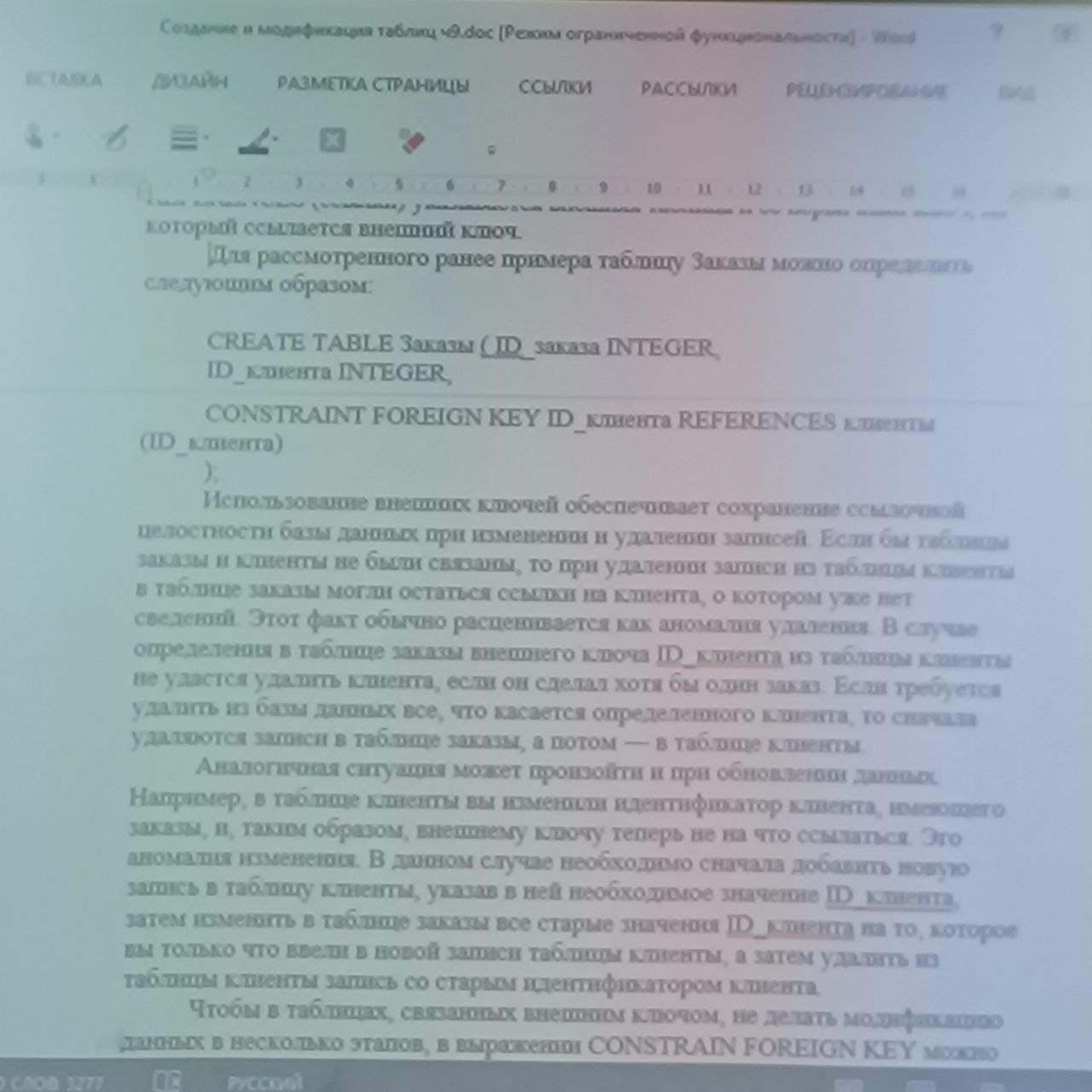
****

**✅**Внешние ключи

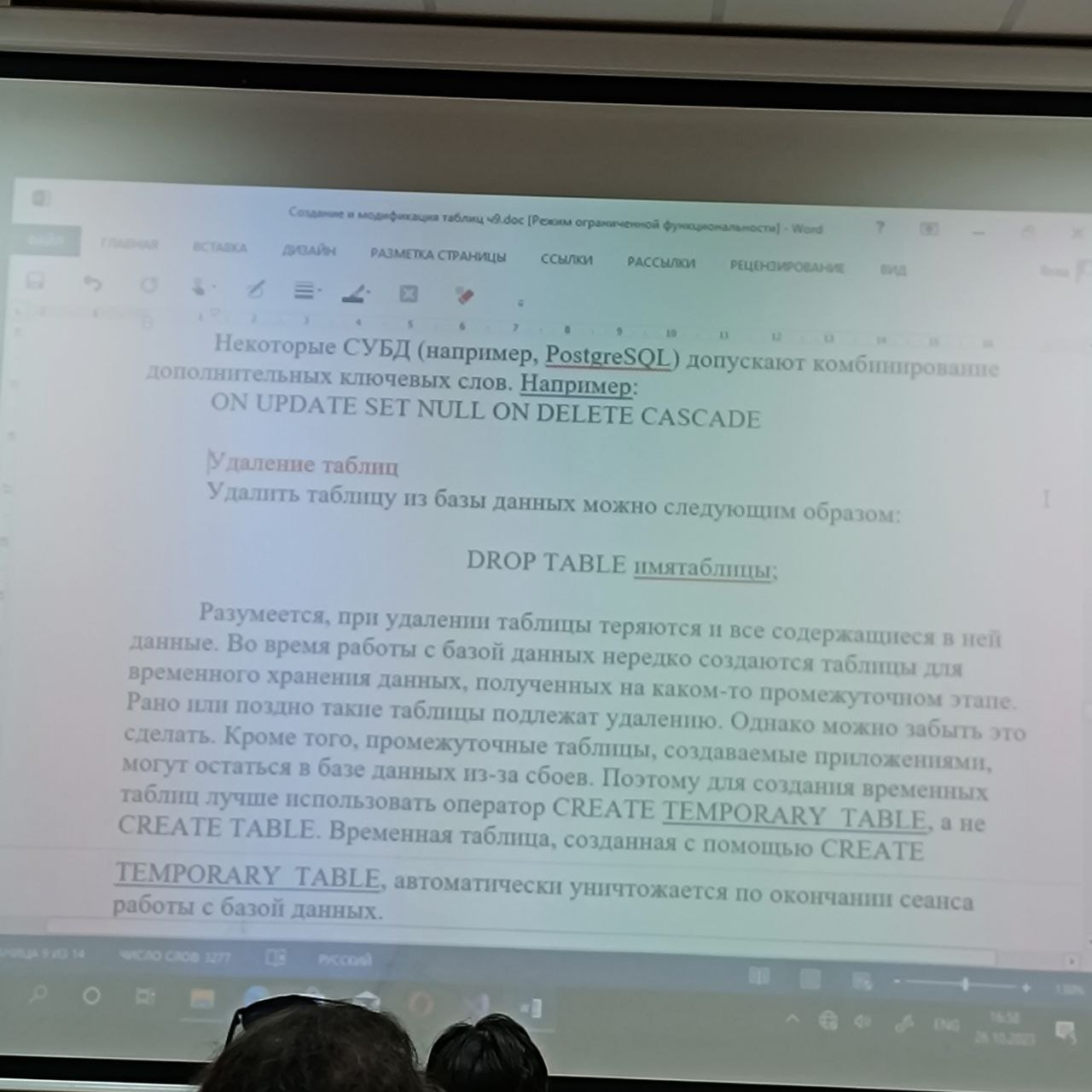
****



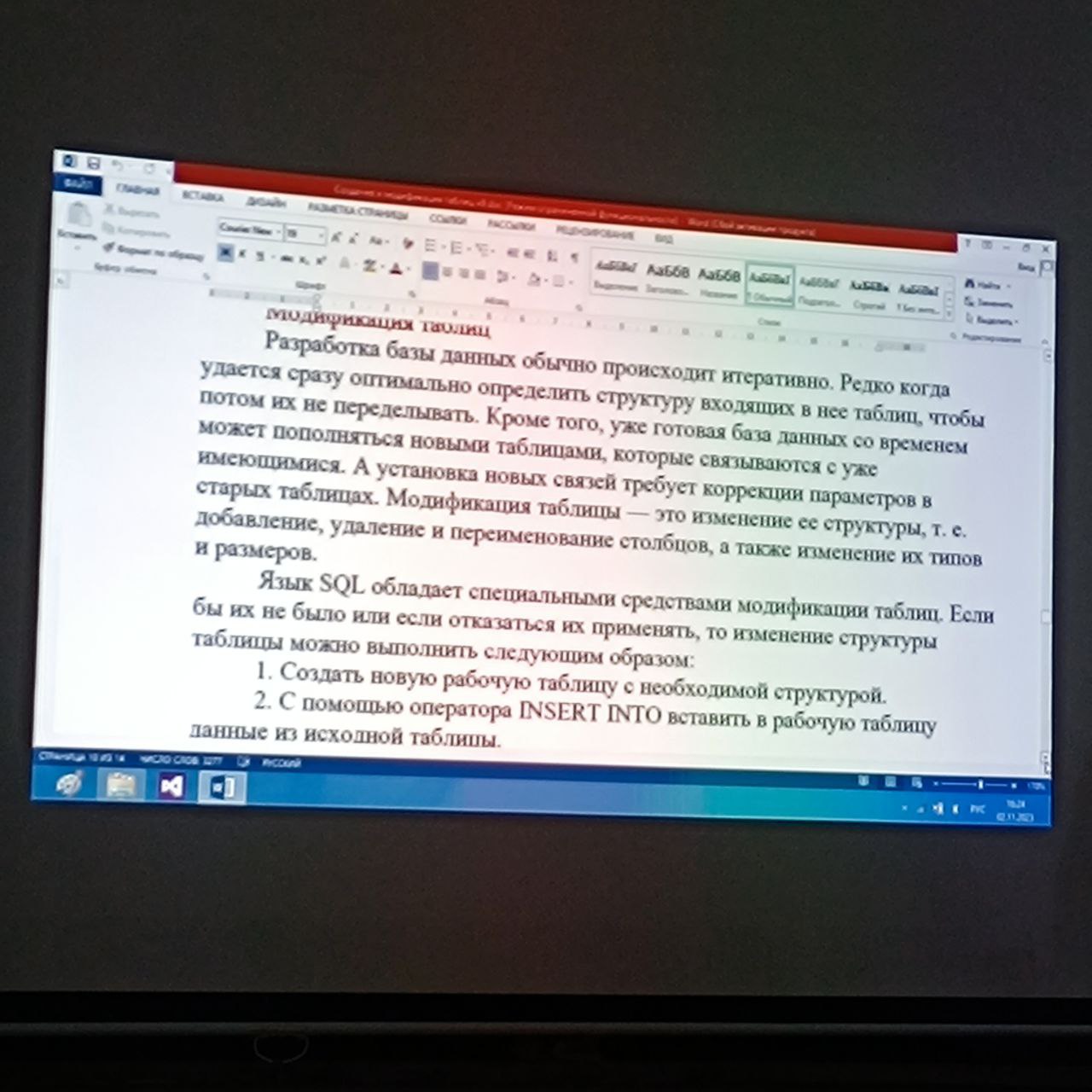


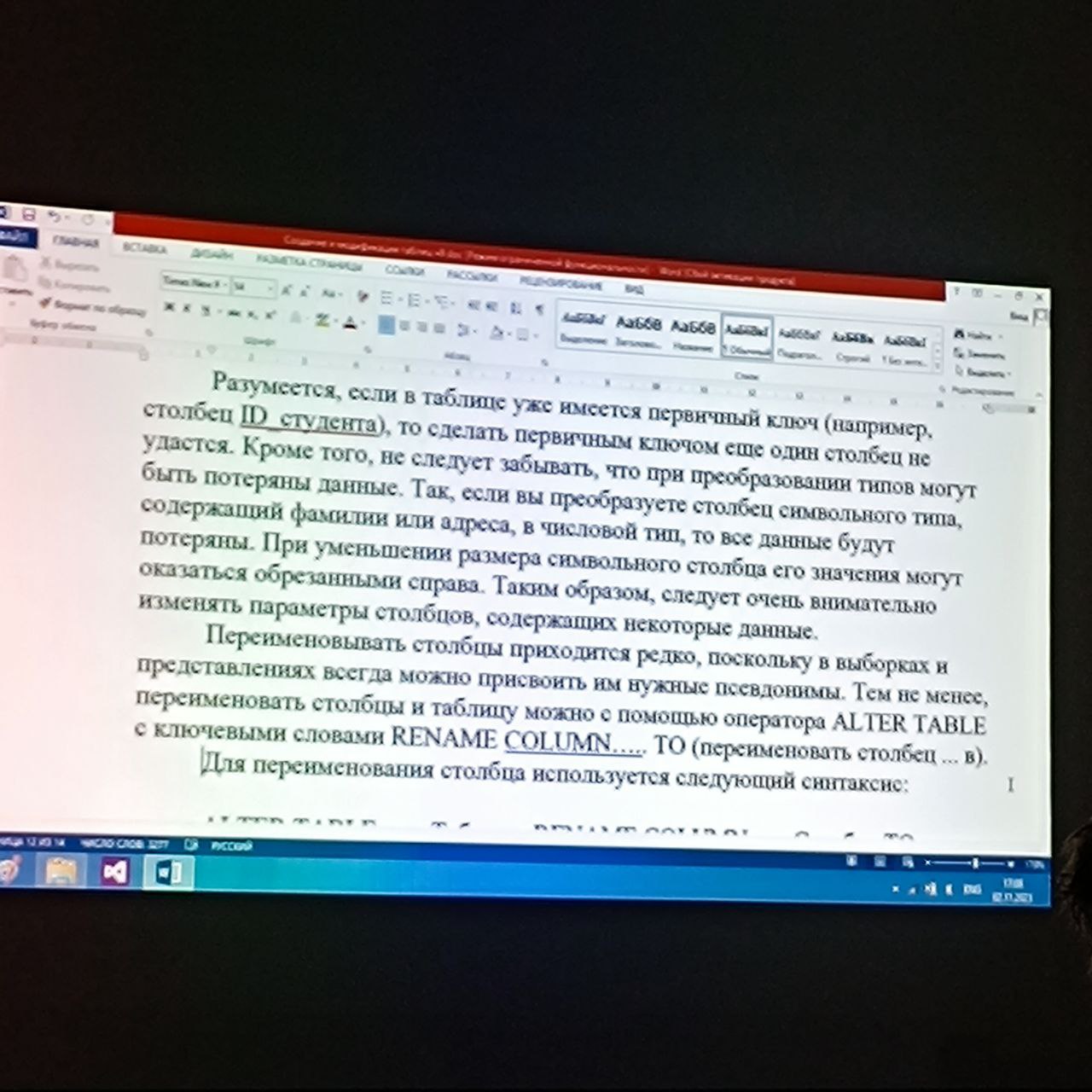


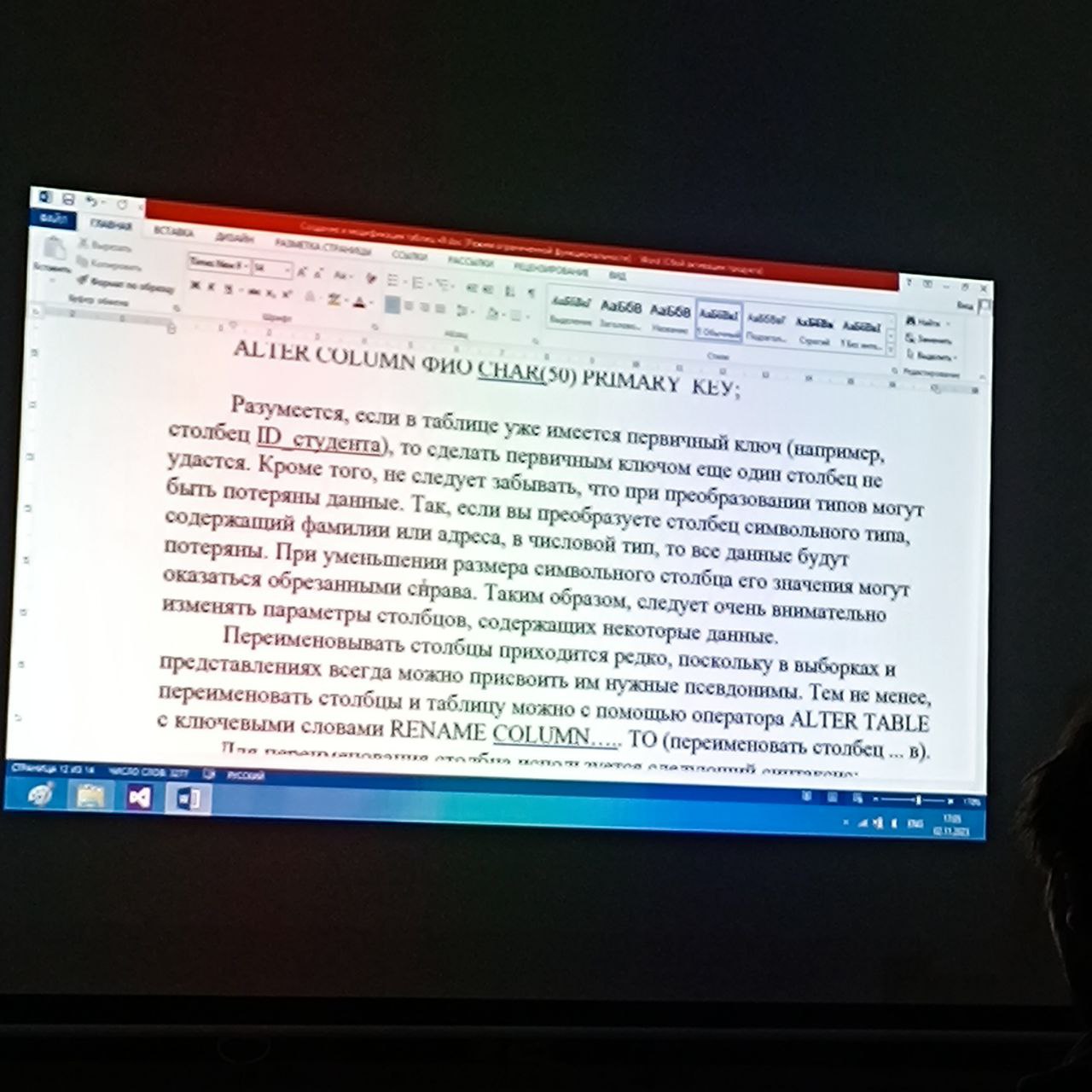
**✅**Удаление таблиц

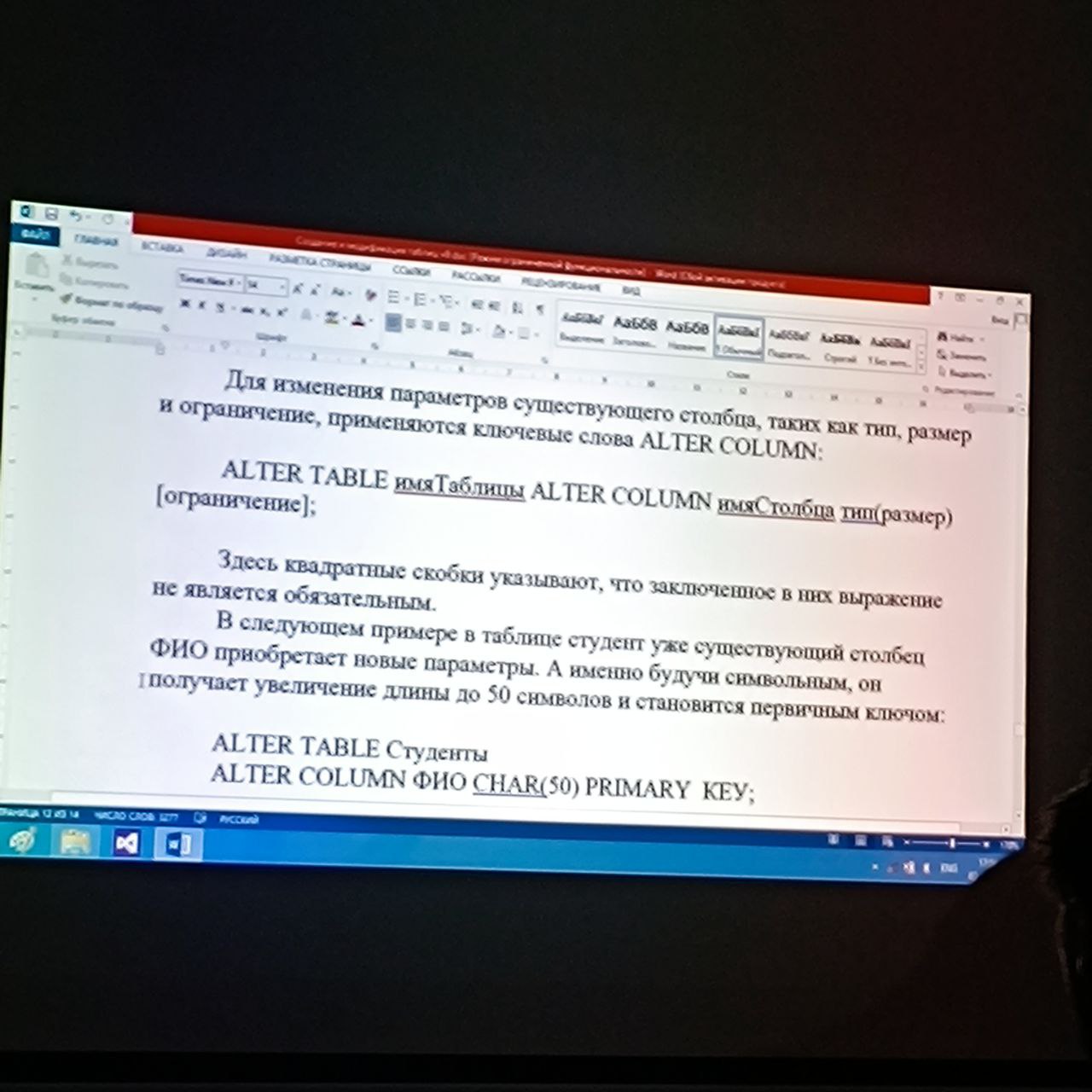
****

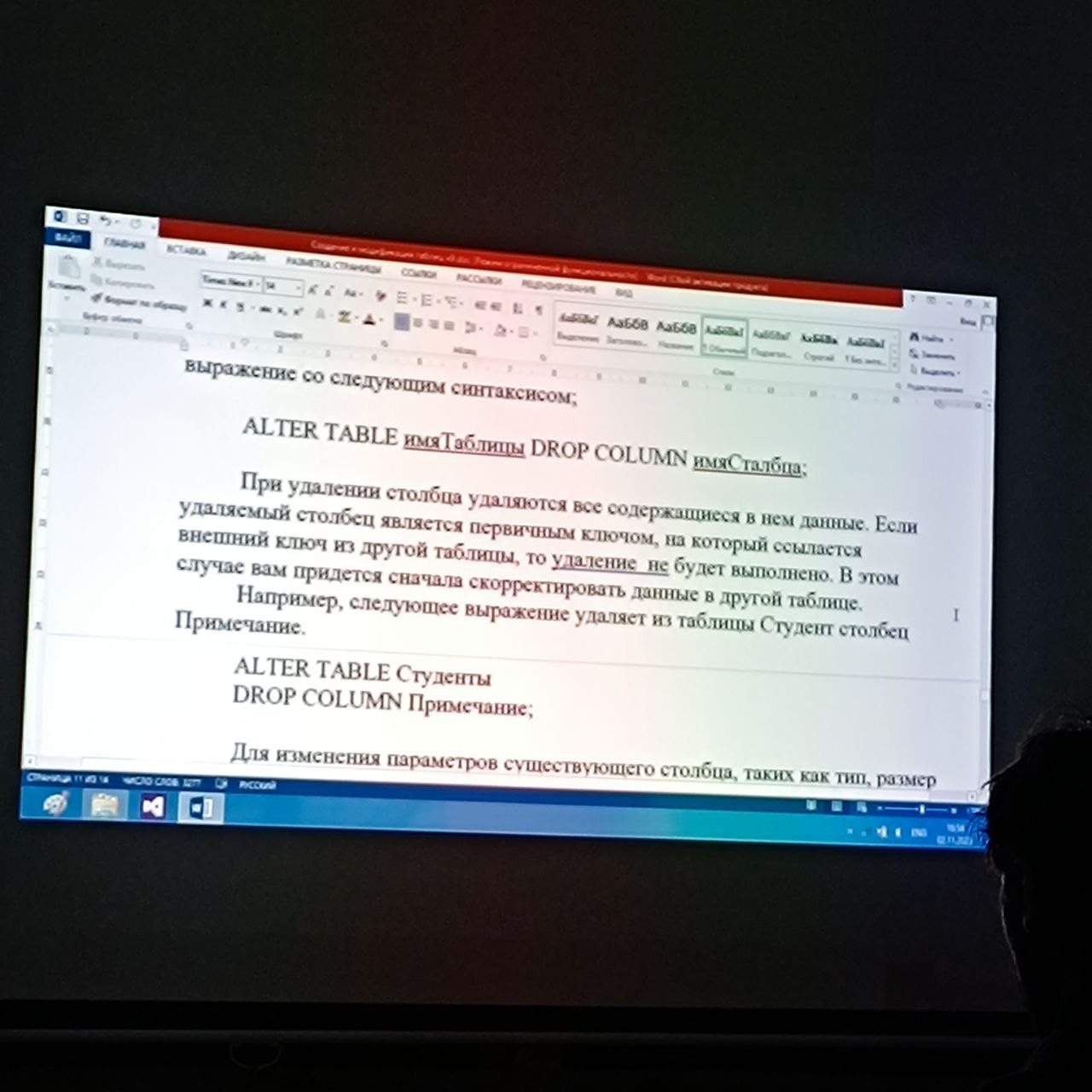
**✅**Модификация таблиц

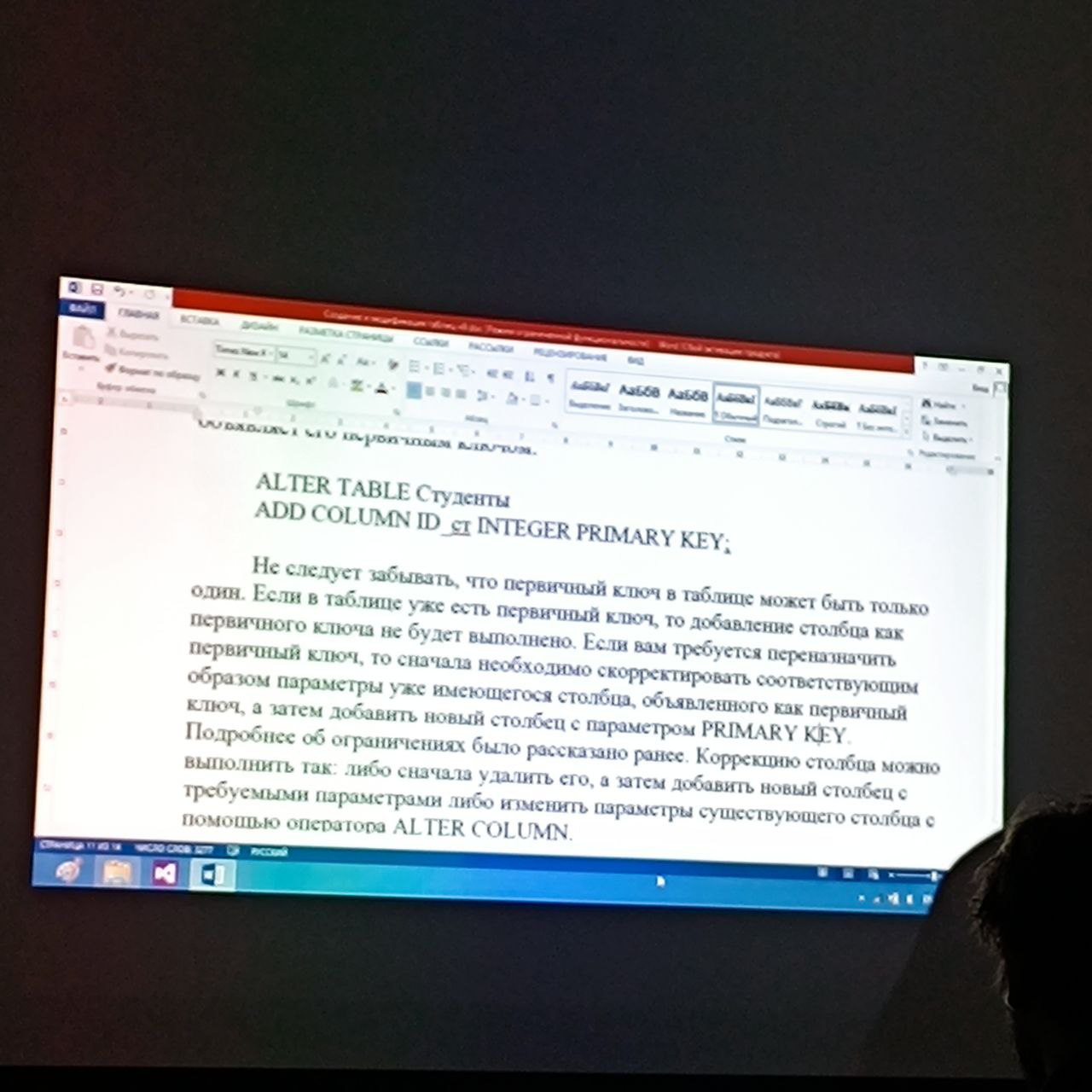
****

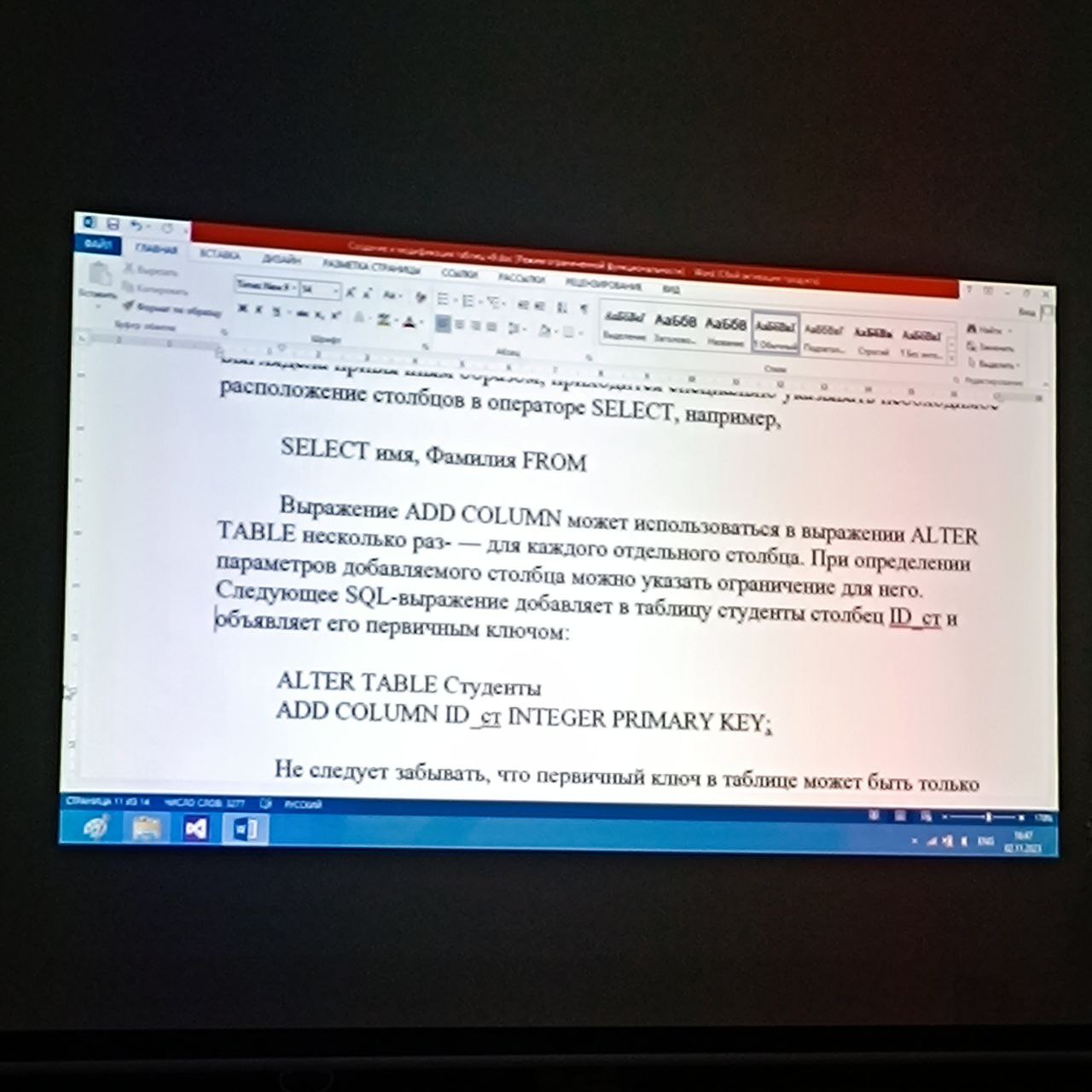


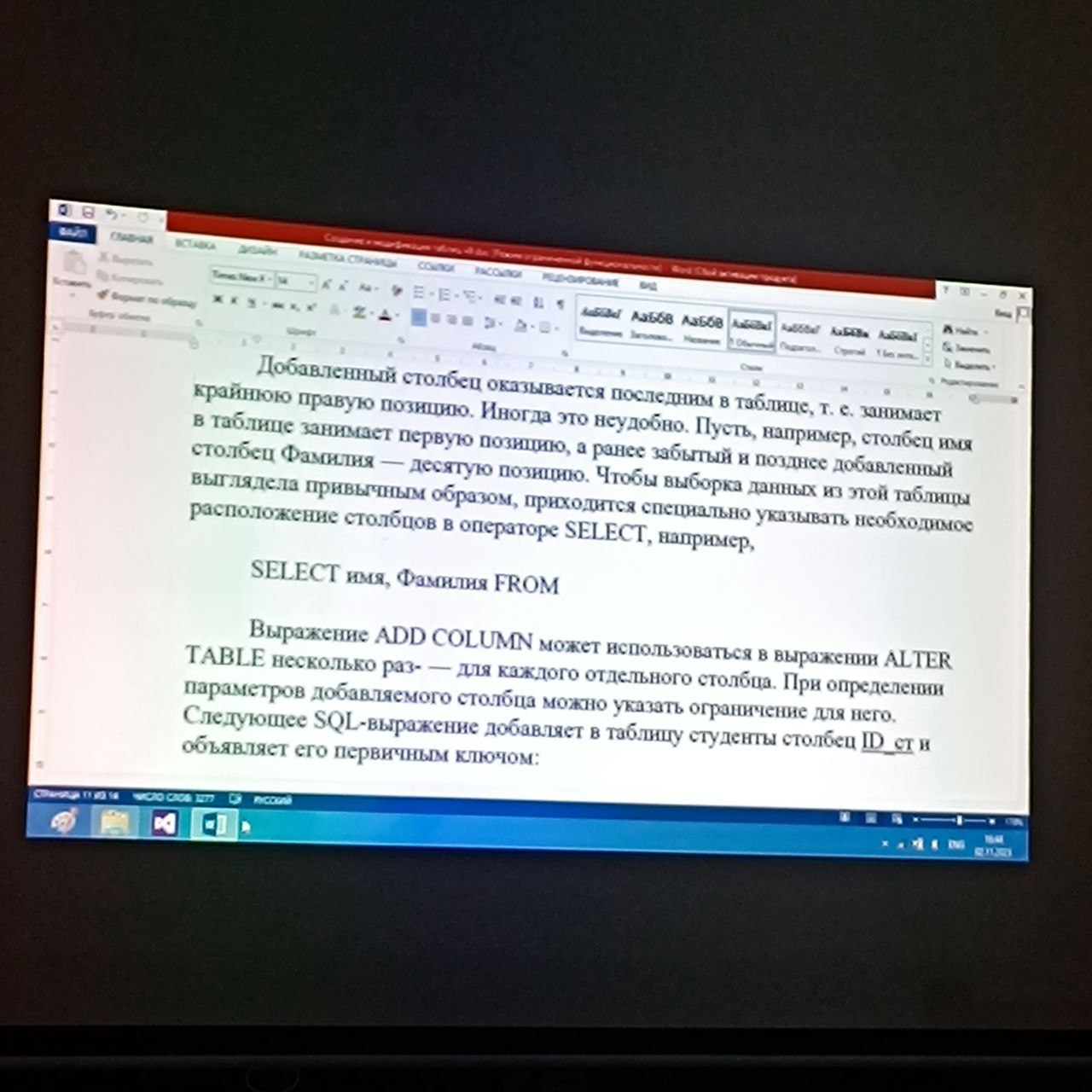


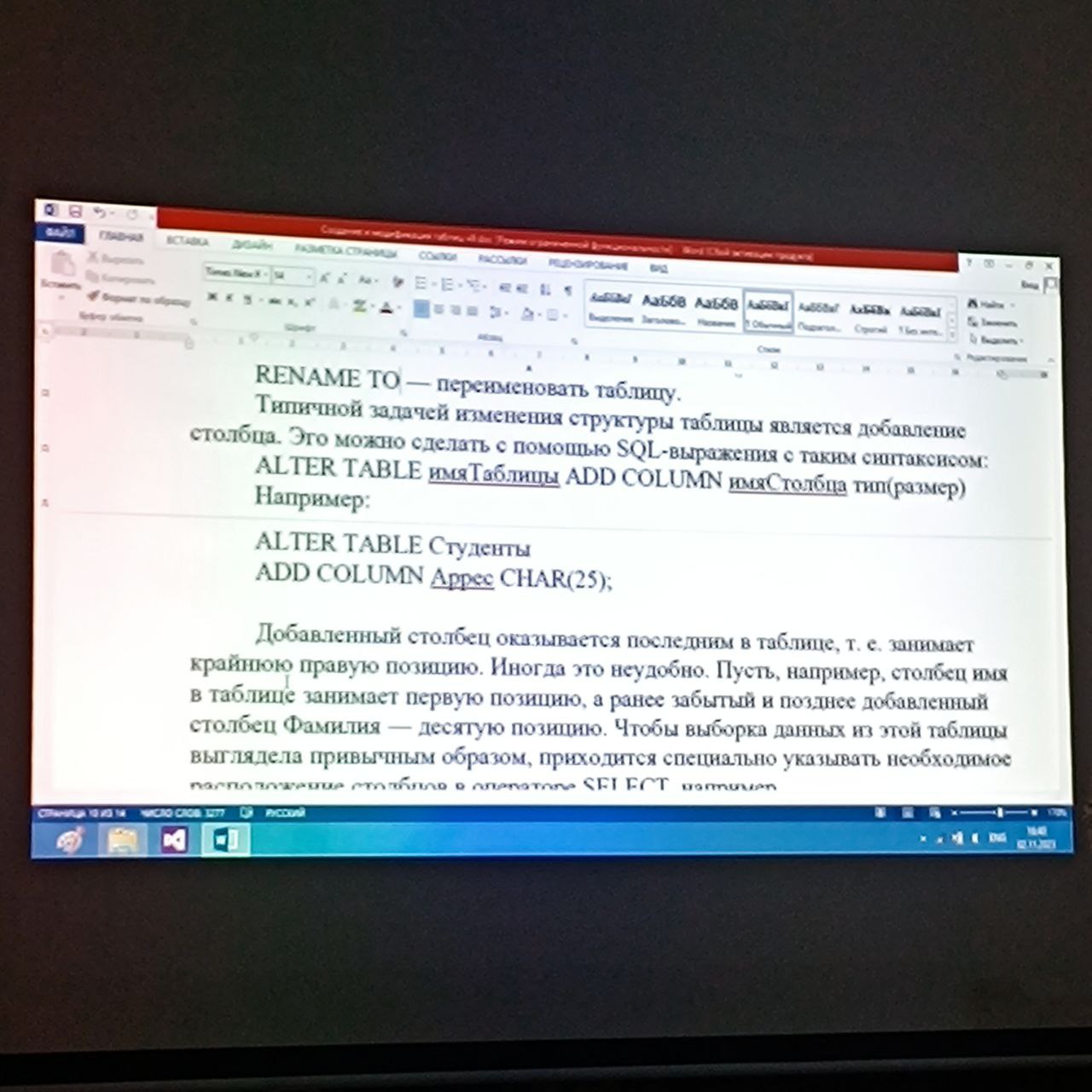


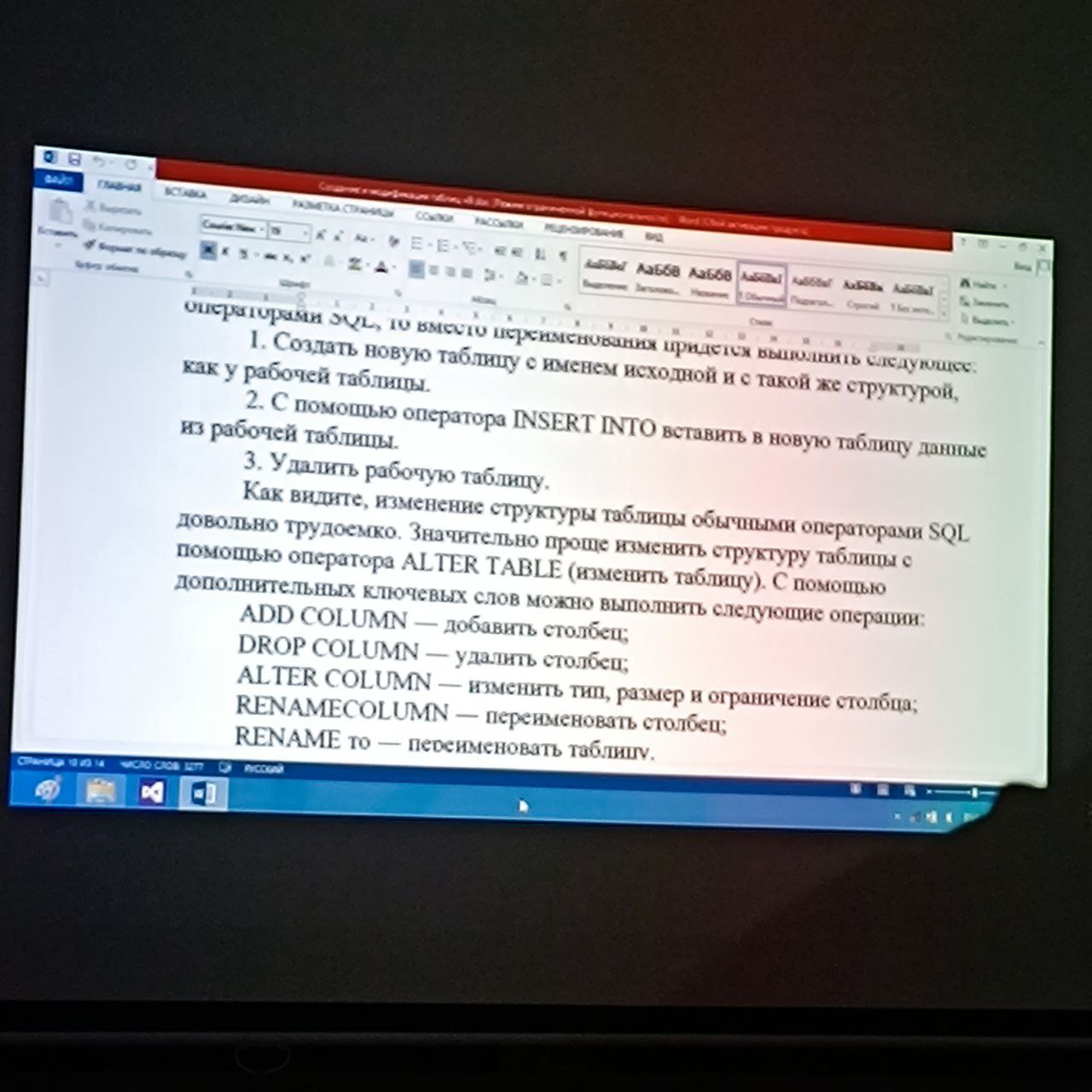


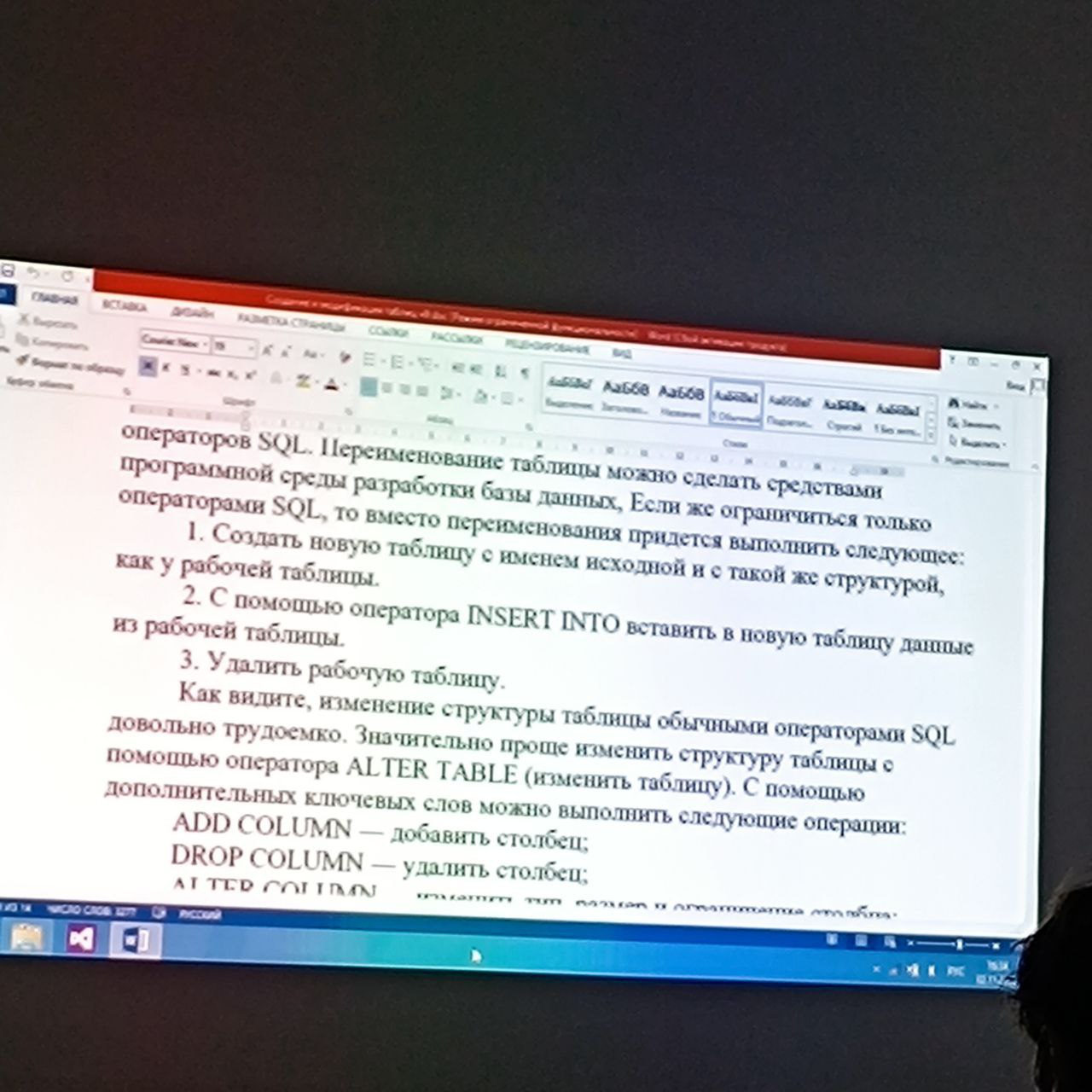


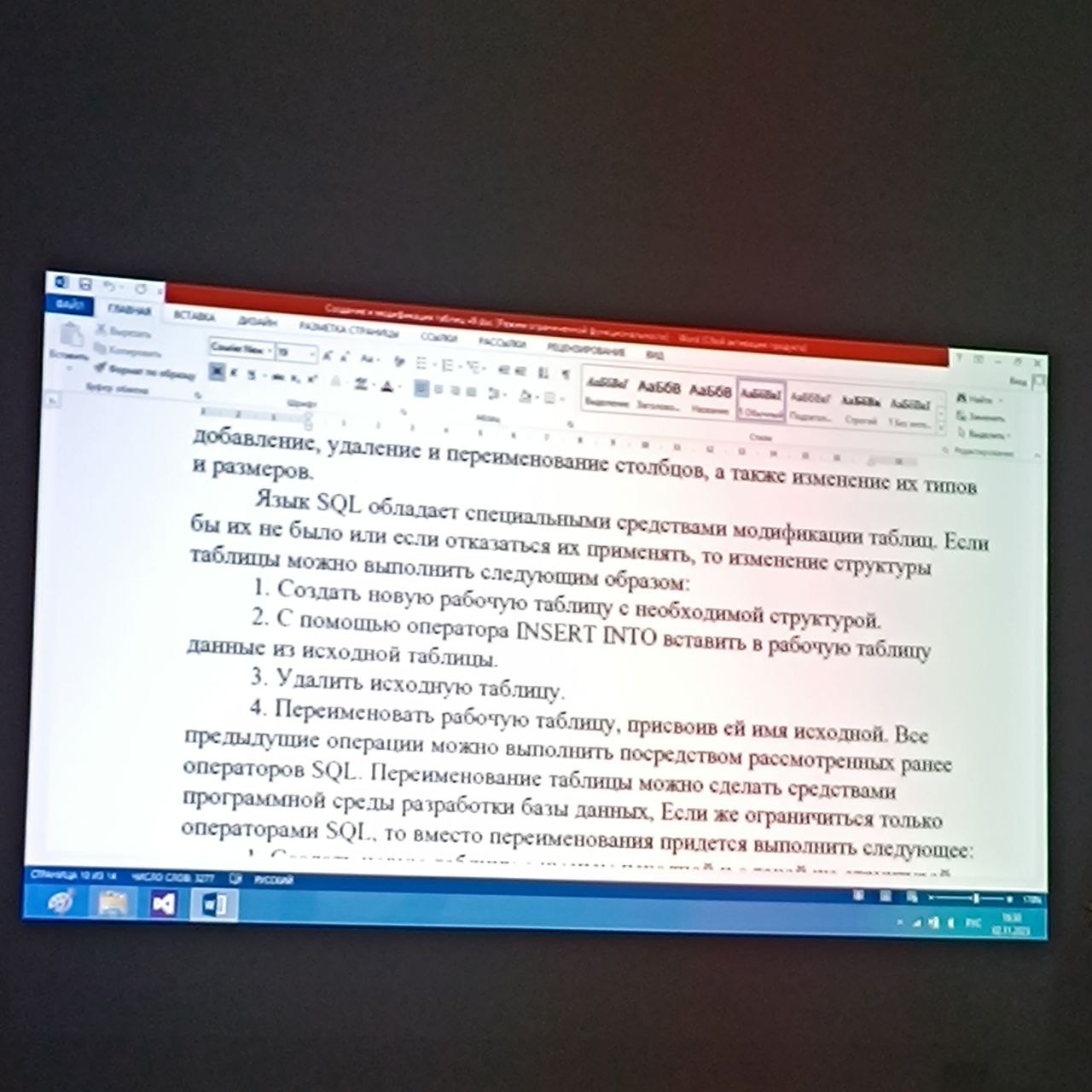












**✅Что такое представление**

Создание представлений

Объяснение: Представление в SQL - это виртуальная таблица, которая основана на результатах выполнения запроса. Оно не содержит фактических данных, а предоставляет удобный способ структурирования и повторного использования сложных запросов.

Пример использования:

CREATE VIEW EmployeeView AS

SELECT EmployeeID, FirstName, LastName, Department

FROM Employees

WHERE Department = 'IT';

**✅**Представления удаляются из базы данных так же, как и таблицы:

Удаление представлений:

Объяснение: Представления можно удалить из базы данных с использованием оператора DROP VIEW. Это освободит ресурсы и удалит определение представления, но не затронет фактических данных в базе.

Пример использования:

DROP VIEW EmployeeView;

**✅**Изменение данных в представлениях

Объяснение: Обычно представления являются только для чтения, и изменения данных напрямую в них невозможны. Однако, можно изменить данные через представление, если выполнены определенные условия, такие как отсутствие выражения DISTINCT, GROUP BY и т.д.

Пример использования:

UPDATE EmployeeView

SET Department = 'HR'

WHERE EmployeeID = 101;

**✅Транзакции**

Как устроена транзакция   
Транзакция в базах данных представляет собой логическую единицу работы, состоящую из одной или нескольких операций базы данных. Транзакции обеспечивают целостность данных и надежность выполнения операций: они либо выполняются полностью, либо откатываются (отменяются) при возникновении ошибки.

Пример использования:

BEGIN TRANSACTION;

-- Ваши SQL-операции

COMMIT; -- Завершение транзакции, если все операции выполнены успешно

ROLLBACK; -- Отмена транзакции при возникновении ошибки

Определение параметров транзакции

Параметры транзакции включают в себя такие аспекты, как изоляция, управление блокировками, уровень журнализации и другие. Они определяют, как транзакция будет взаимодействовать с другими транзакциями и как будет обеспечиваться ее целостность.

**✅**Уровни изоляции транзакций   
Уровень изоляции определяет степень видимости изменений, внесенных одной транзакцией, для других транзакций. В стандарте SQL определены четыре уровня изоляции: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ и SERIALIZABLE.

**✅**Неподтвержденное чтение

Неподтвержденное чтение позволяет одной транзакции читать нефиксированные изменения, внесенные другой транзакцией. Это наименее строгий уровень изоляции.

**✅**Подтвержденное чтение

Подтвержденное чтение позволяет транзакции читать только те изменения, которые были подтверждены (зафиксированы) другими транзакциями.

**✅**Повторяющееся чтение

Повторяющееся чтение гарантирует, что при повторных чтениях одних и тех же данных в рамках транзакции результат останется неизменным, независимо от изменений, внесенных другими транзакциями.

**✅**Последовательное выполнение

Последовательное выполнение обеспечивает максимальную изоляцию, гарантируя, что транзакции выполняются последовательно, без возможности конфликтов и несогласованных изменений.

**✅**Субтранзакции

Субтранзакции – это вложенные транзакции внутри других транзакций. Они позволяют более гибко управлять областью видимости изменений и отката операций.

Пример использования:

BEGIN TRANSACTION; -- Начало основной транзакции

-- Ваши SQL-операции

SAVEPOINT subtransaction; -- Создание субтранзакции

-- Ваши SQL-операции в субтранзакции

ROLLBACK TO subtransaction; -- Откат к точке сохранения в случае ошибки

COMMIT; -- Завершение основной транзакции