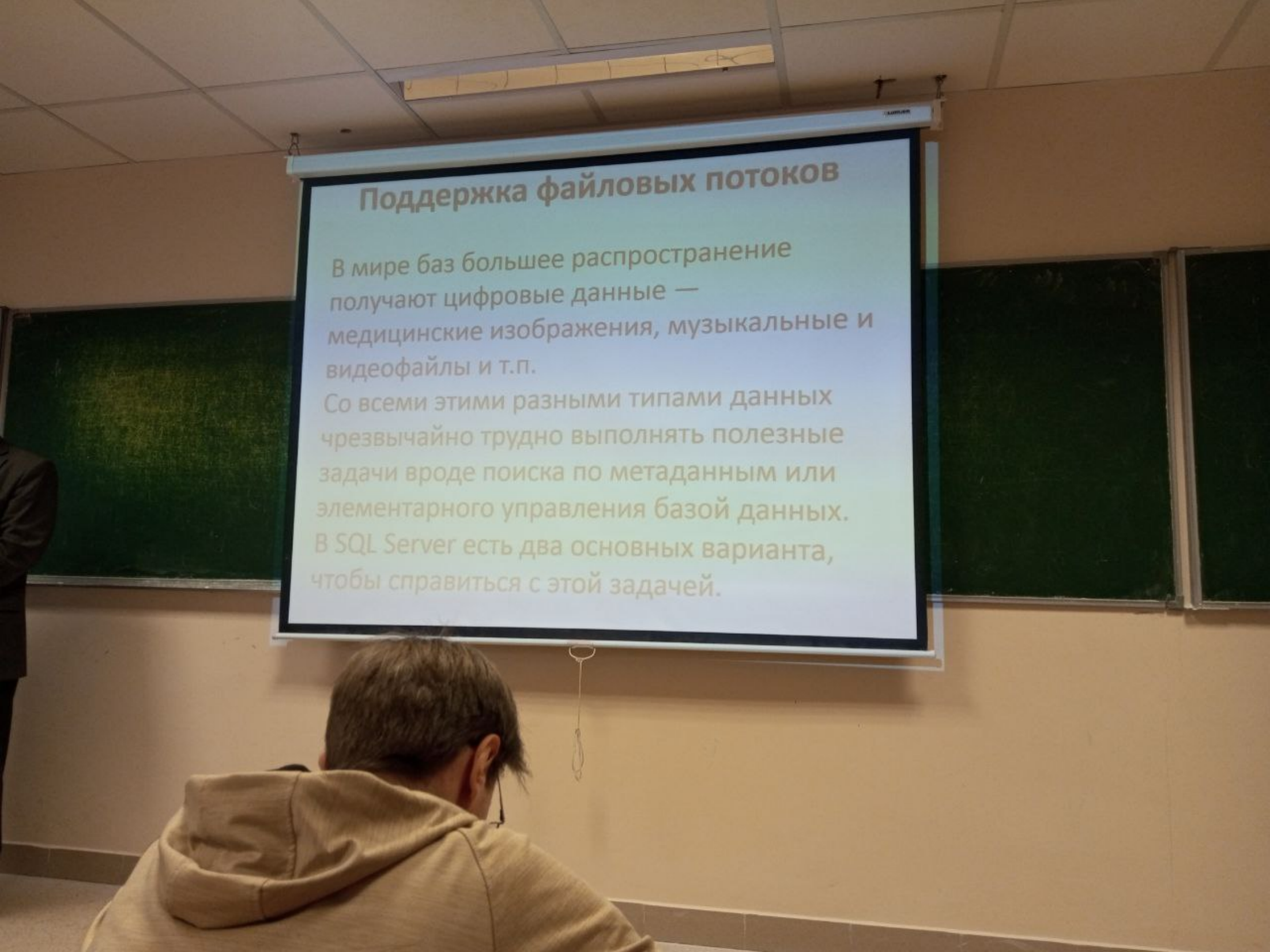
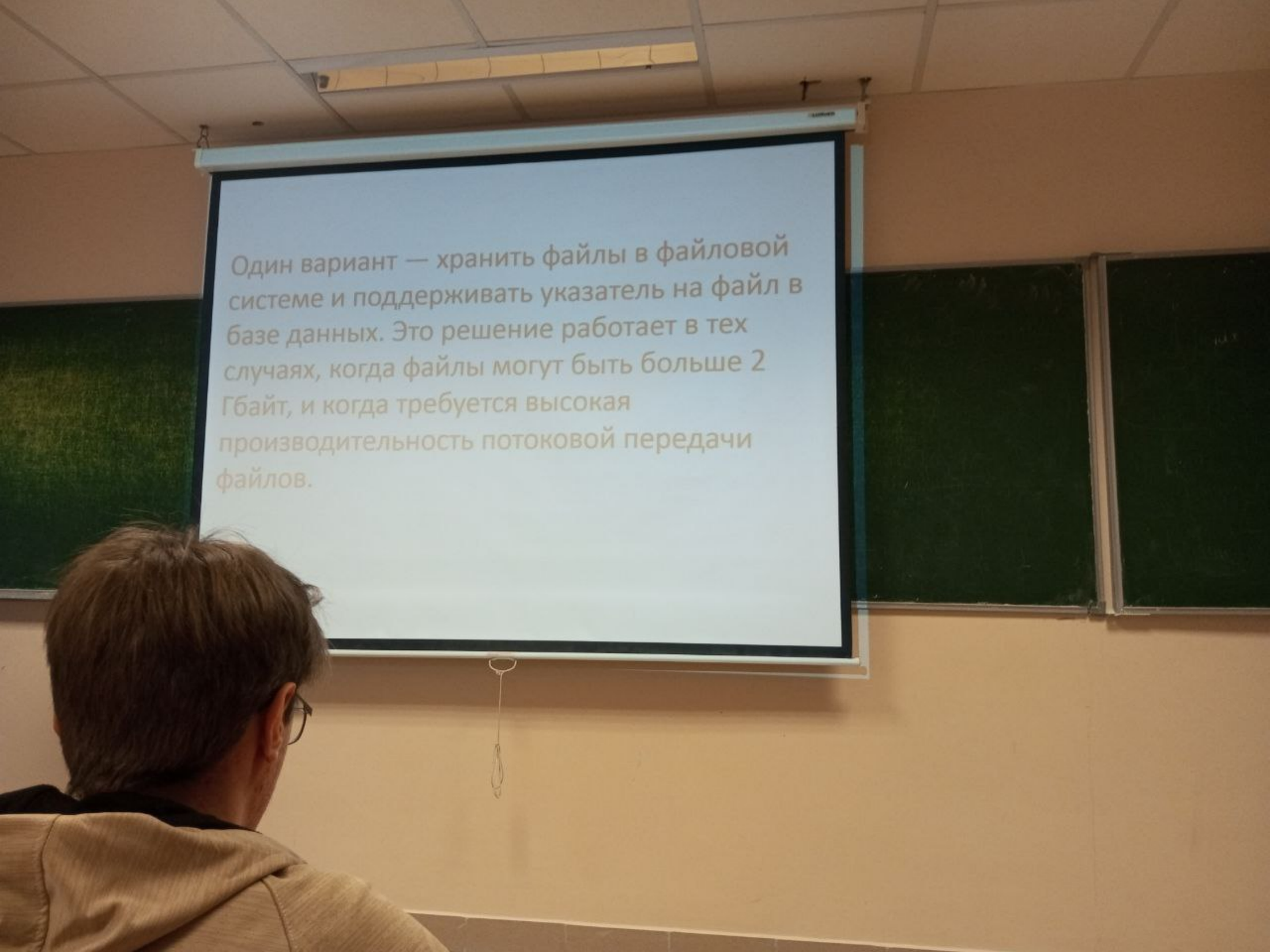
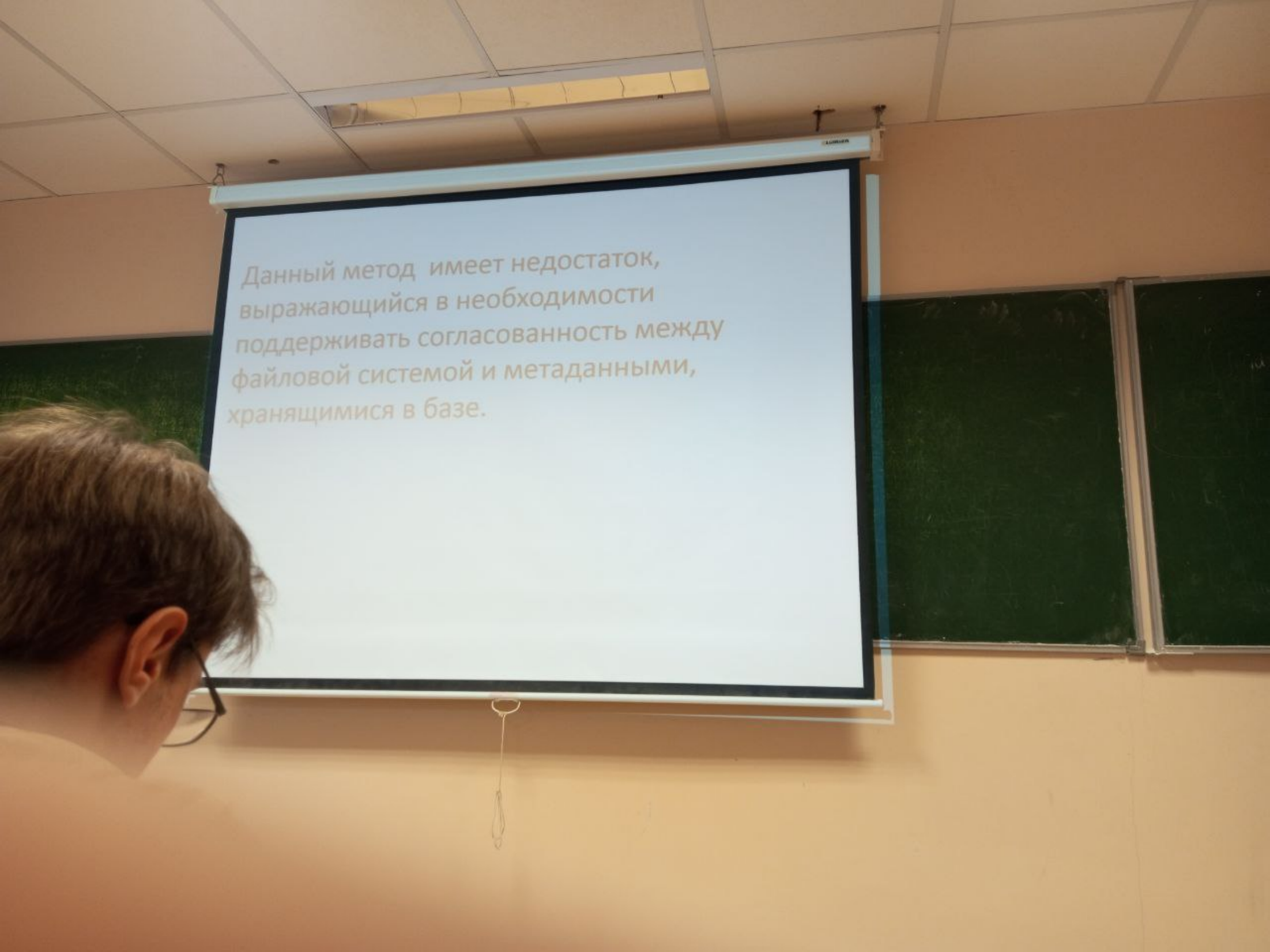
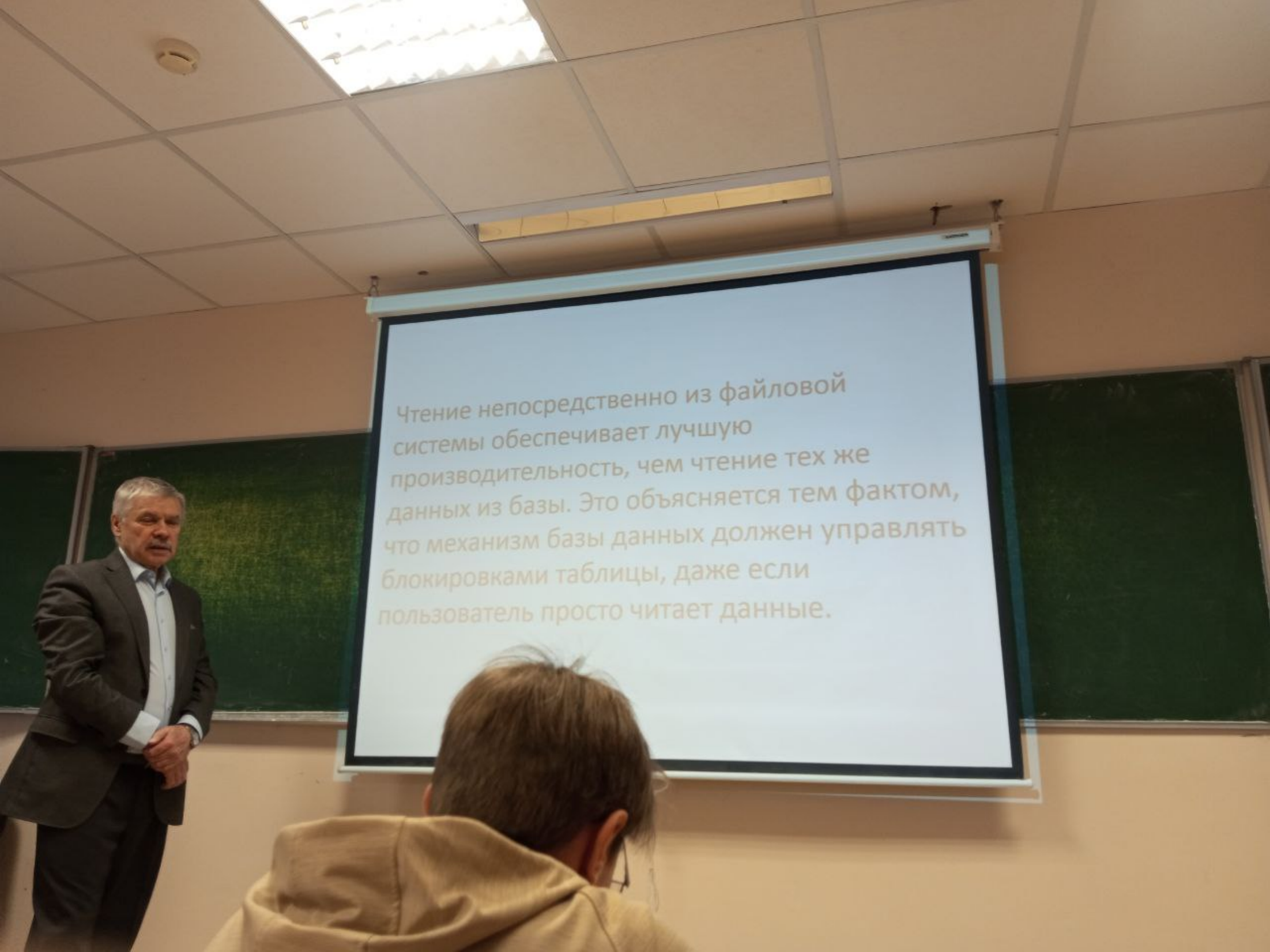
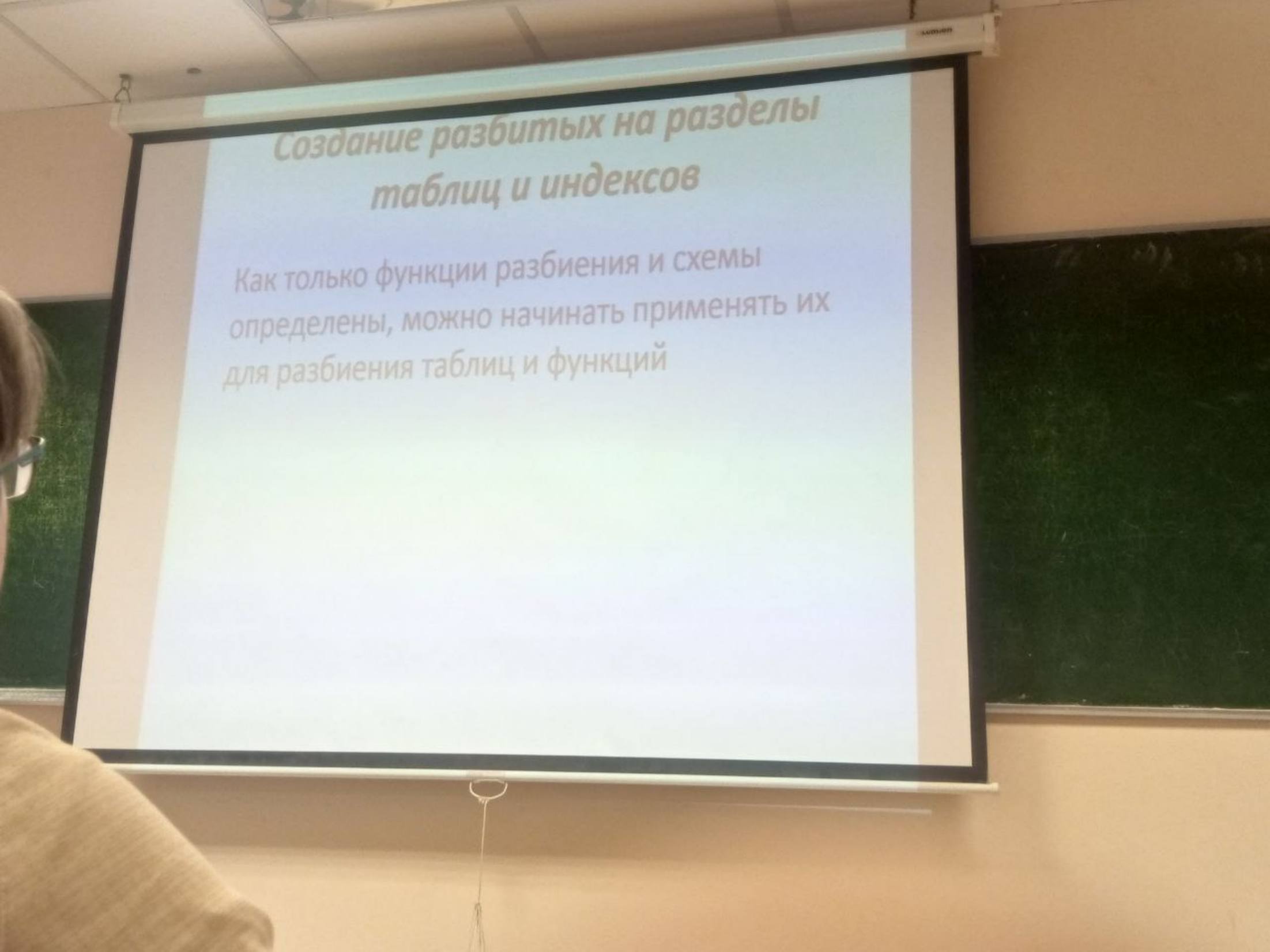
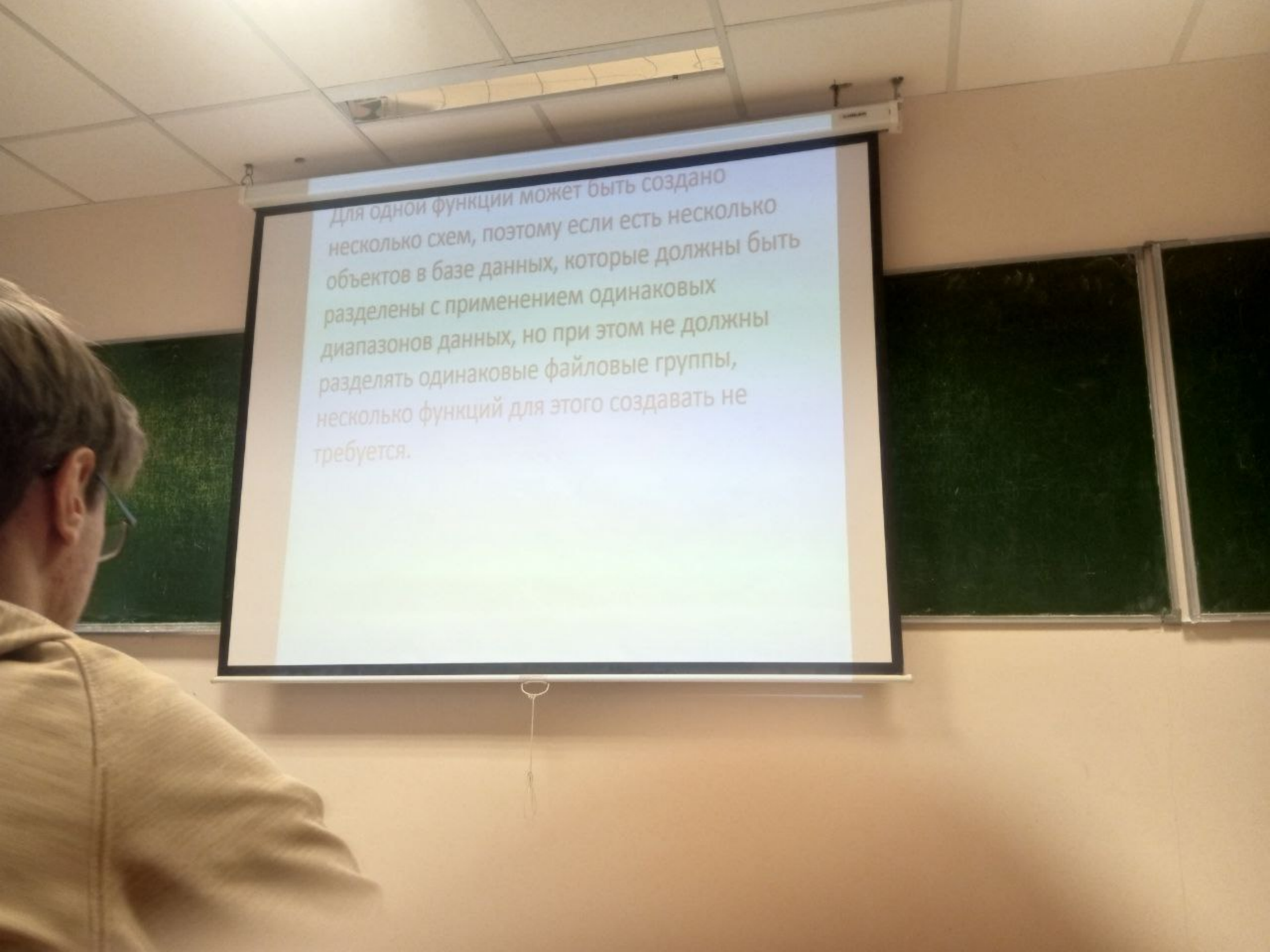

Поддержка файловых потоков
В мире баз большее распространение получают цифровые данные — медицинские изображения, музыкальные и видеофайлы и т.п.
Со всеми этими разными типами данных чрезвычайно трудно выполнять полезные задачи вроде поиска по метаданным или элементарного управления базой данных.
В SQL Server есть два основных варианта, чтобы справиться с этой задачей.


Один вариант — хранить файлы в файловой системе и поддерживать указатель на файл в базе данных. Это решение работает в тех случаях, когда файлы могут быть больше 2 Гбайт, и когда требуется высокая производительность потоковой передачи файлов.

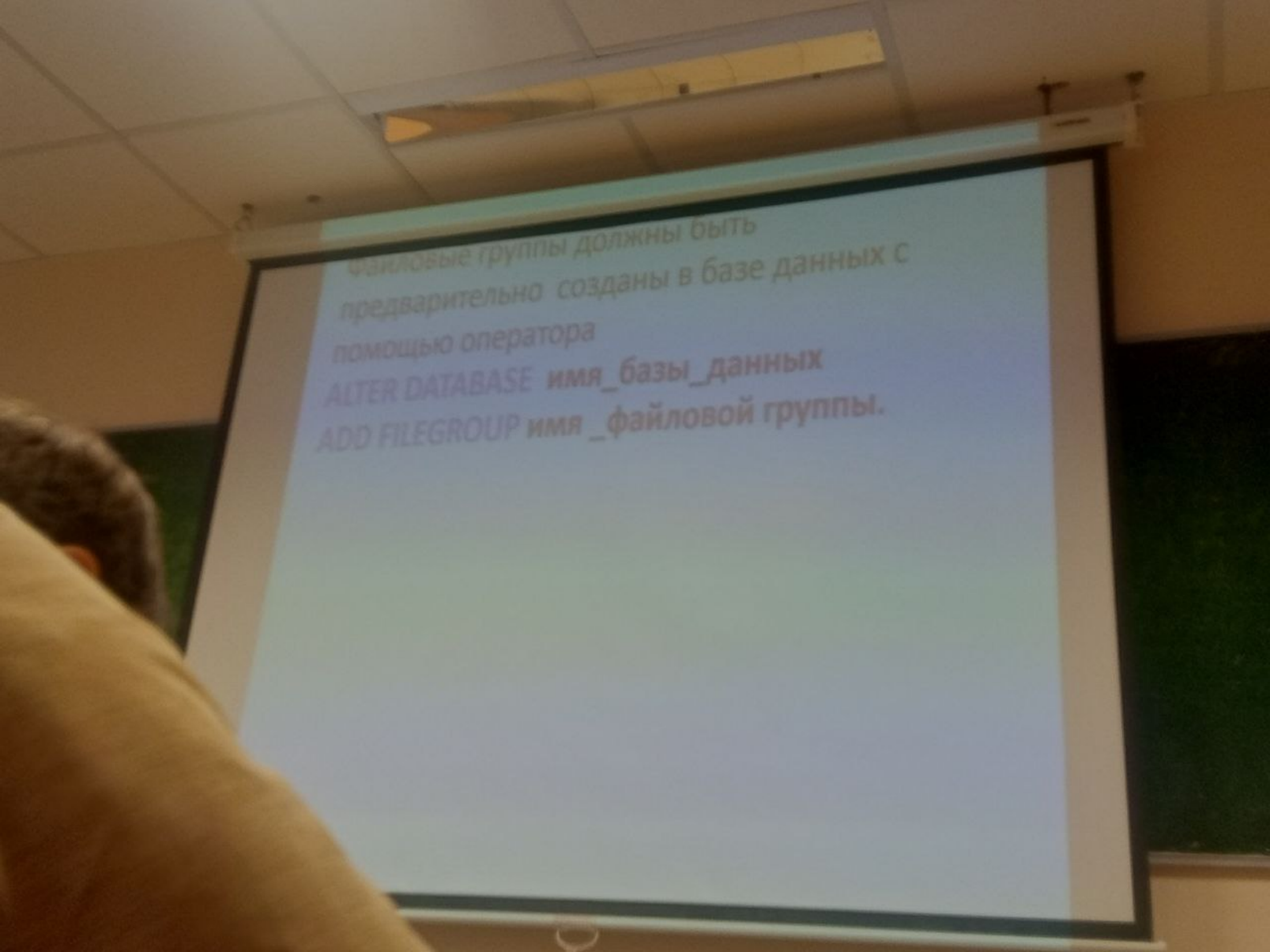

Данный метод имеет недостаток,
выражающийся в необходимости
поддерживать согласованность между
файловой системой и метаданными,
хранящимися в базе.

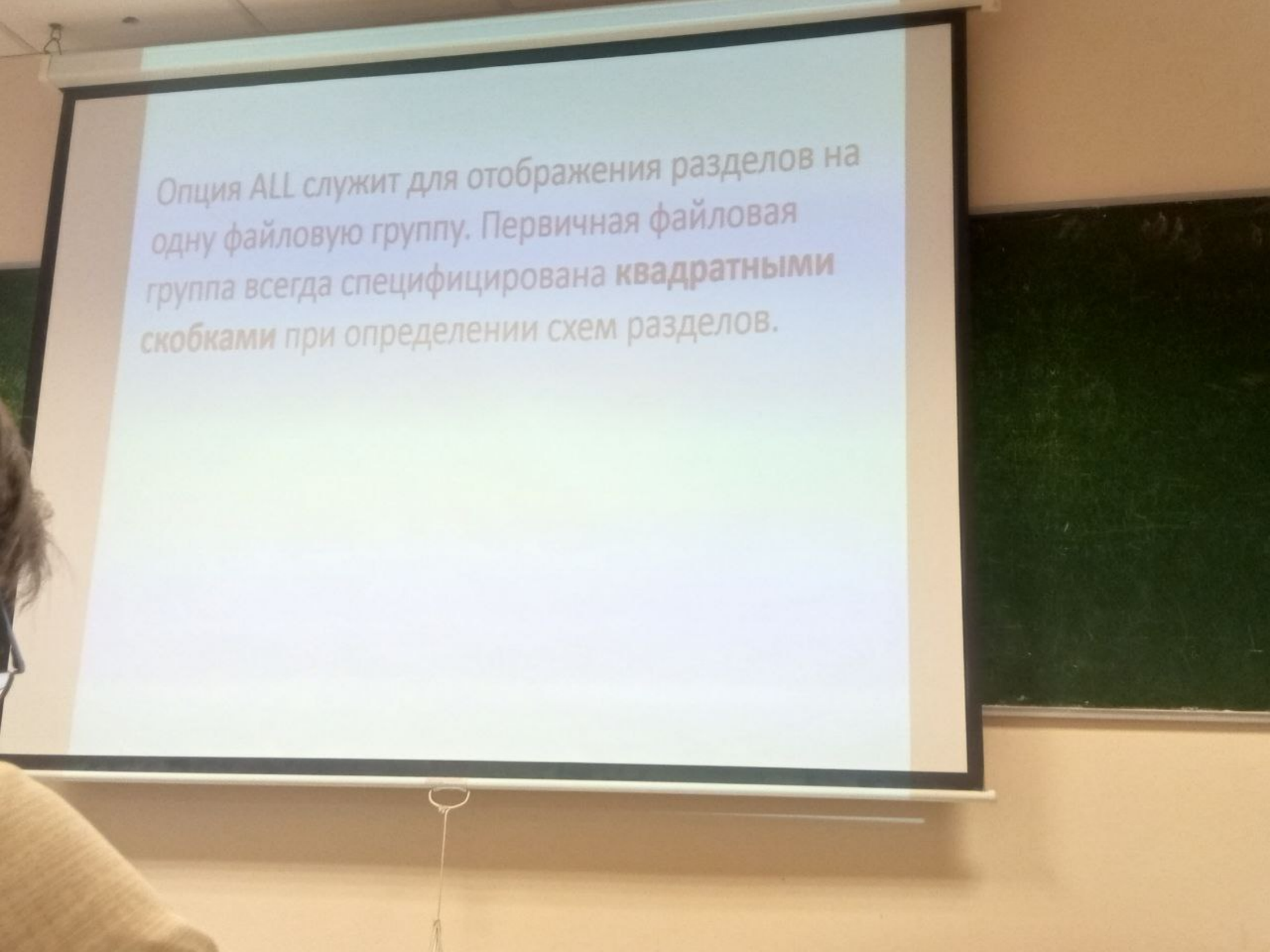

Чтение непосредственно из файловой системы обеспечивает лучшую производительность, чем чтение тех же данных из базы. Это объясняется тем фактом, что механизм базы данных должен управлять блокировками таблицы, даже если пользователь просто читает данные.


Создание разбитых на разделы таблиц и индексов
Как только функции разбиения и схемы определены, можно начинать применять их для разбиения таблиц и функций


Для одной функции может быть создано несколько схем, поэтому если есть несколько объектов в базе данных, которые должны быть разделены с применением одинаковых диапазонов данных, но при этом не должны разделять одинаковые файловые группы, несколько функций для этого создавать не требуется.

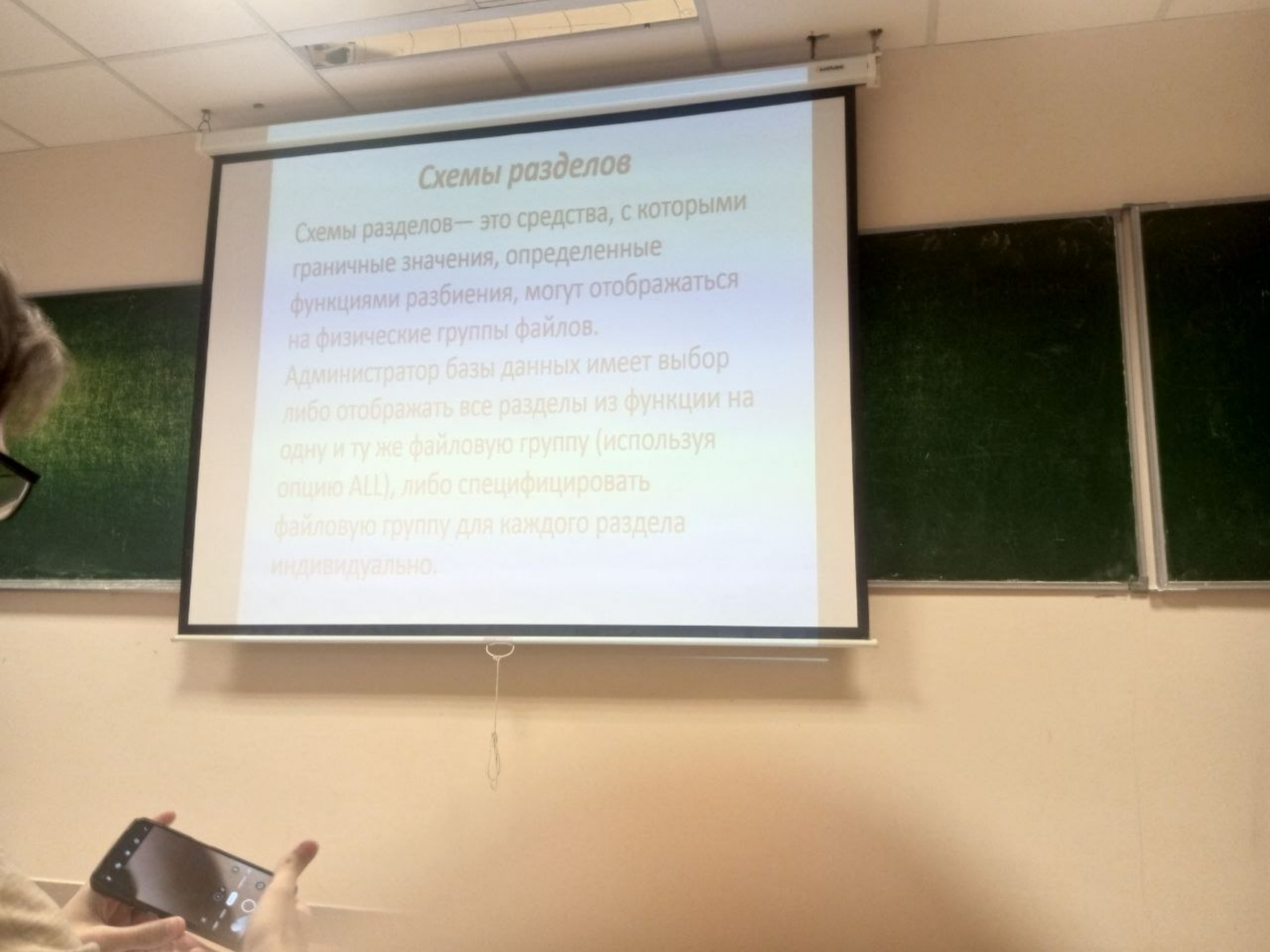
Для одной функции может быть создано несколько схем, поэтому если есть несколько объектов в базе данных, которые должны быть разделены с применением одинаковых диапазонов данных, но при этом не должны разделять одинаковые файловые группы, несколько функций для этого создавать не требуется.


Файловые группы должны быть
предварительно созданы в базе данных с
помощью оператора
ALTER DATABASE имя_базы_данных
ADD FILEGROUP имя _файловой группы.


Опция ALL служит для отображения разделов на одну файловую группу. Первичная файловая группа всегда специфицирована **квадратными скобками** при определении схем разделов.

Базовый синтаксис для создания схемы разделов выглядит так:

```
CREATE PARTITION SCHEME имя_схемы_раздела AS
PARTITION имя_функции_разбиения
[ ALL ] TO ( { имя_файловой_группы | [PRIMARY] }
[ ,...n] )[ ; ]
```


Схемы разделов
Схемы разделов— это средства, с которыми граничные значения, определенные функциями разбиения, могут отображаться на физические группы файлов.
Администратор базы данных имеет выбор либо отображать все разделы из функции на одну и ту же файловую группу (используя опцию ALL), либо специфицировать файловую группу для каждого раздела индивидуально.

Диапазоны разделов не могут быть спроектированы для ограничения допустимых входных значений в заданных пределах. Значения меньше нижней границы попадают в самый нижний раздел. Значения, выходящие за верхнюю границу, помещаются в автоматически генерируемый раздел для значений, выходящих за эту верхнюю границу.

Для функции LEFT каждый раздел будет определен как содержащий все значения, меньшие или равные его верхнему пределу.

Для функции RIGHT каждый раздел определяется как содержащий все значения, меньшие его верхнего предела, само граничное значение попадает в следующий раздел.

Функции разбиения должны принимать ***единственный*** *параметр (столбец) опре-деленного типа данных. Функция определяется в терминах диапазонов, а указатели* **LEFT** *или* **RIGHT** *контролируют местоположение действительного граничного значения.*

Базовый синтаксис создания функции разбиения выглядит так:

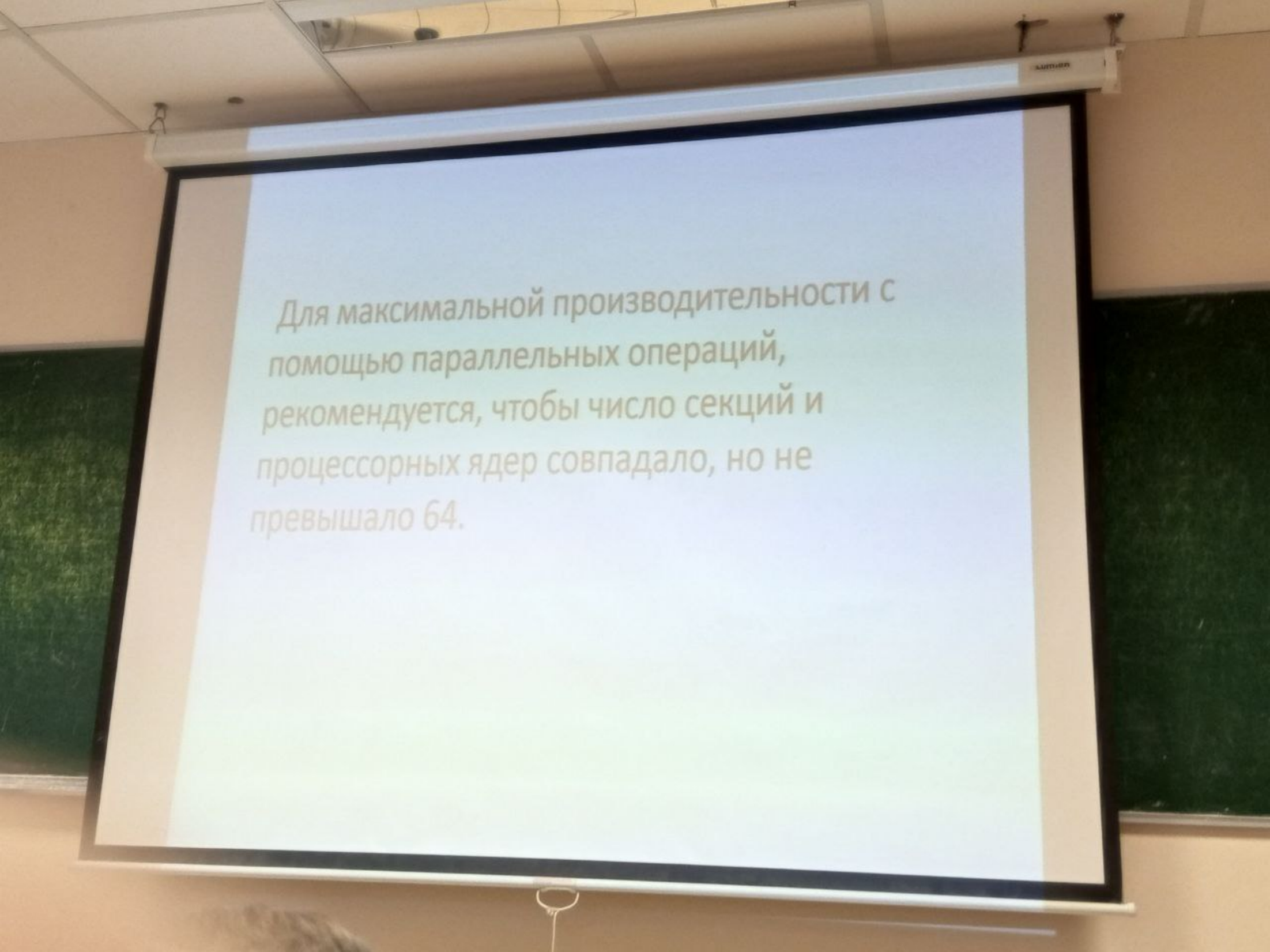
```
CREATE PARTITION FUNCTION имя_функции_разбиения
(тип_входного_параметра)
AS RANGE [ LEFT | RIGHT ]
FOR VALUES ( [ граничное_значение [ ,...n ] ] )[ ; ]
```

Функции разбиения

Функции разбиения — это средства, с помощью которых администратор базы данных может контролировать диапазоны данных, используемые для установки граничных значений разделов. Эти функции отображают разделы на основе типа данных и диапазонов значений этого типа, но сами по себе они ничего не разбивают. Функции разбиения являются повторно используемыми.

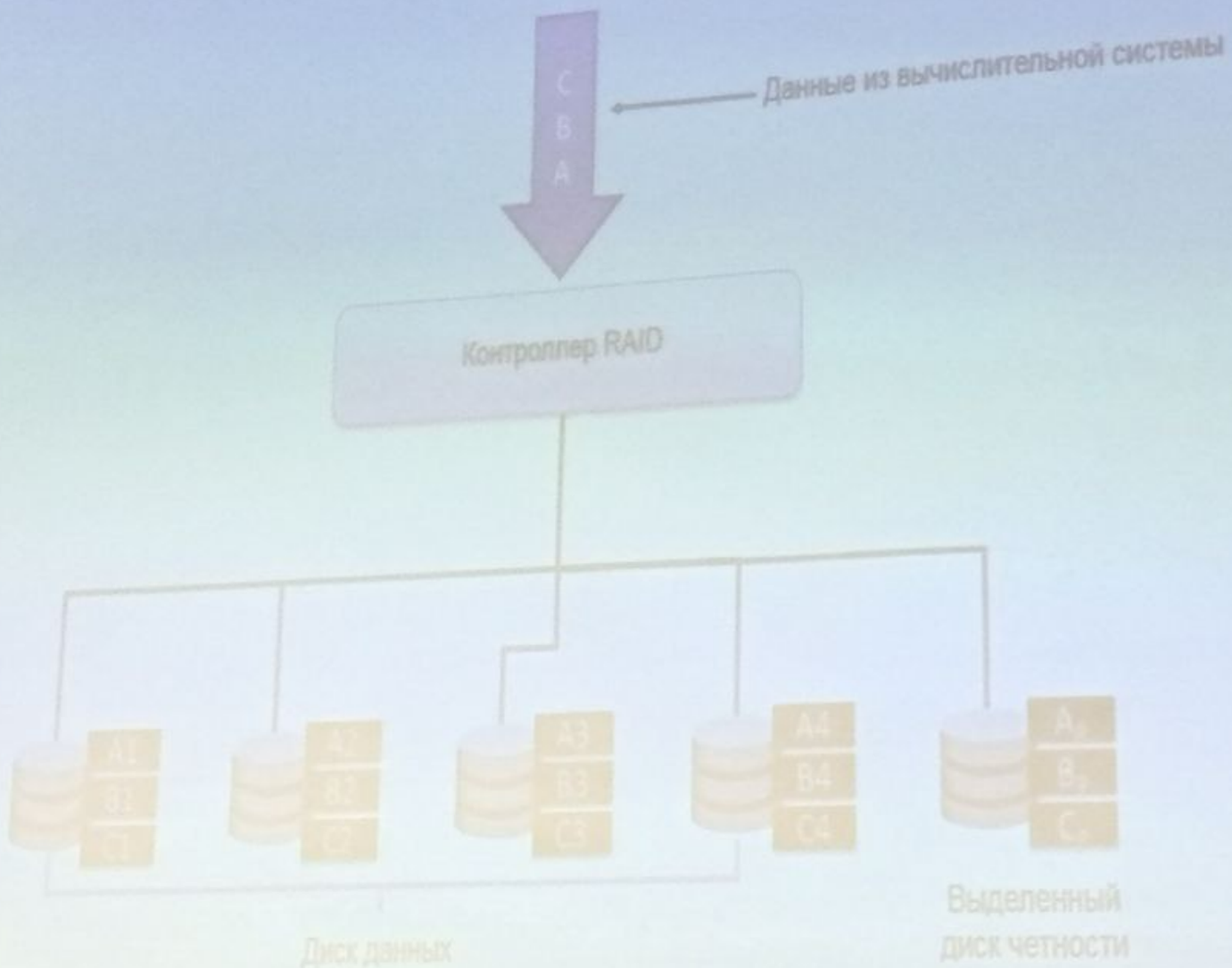
При большом количестве используемых секций рекомендуется использовать ОЗУ не менее 16 ГБ. Если у системы недостаточно памяти, возможен сбой инструкций языка обработки данных (DML), инструкций языка описания данных (DDL) и других операций из-за нехватки памяти.

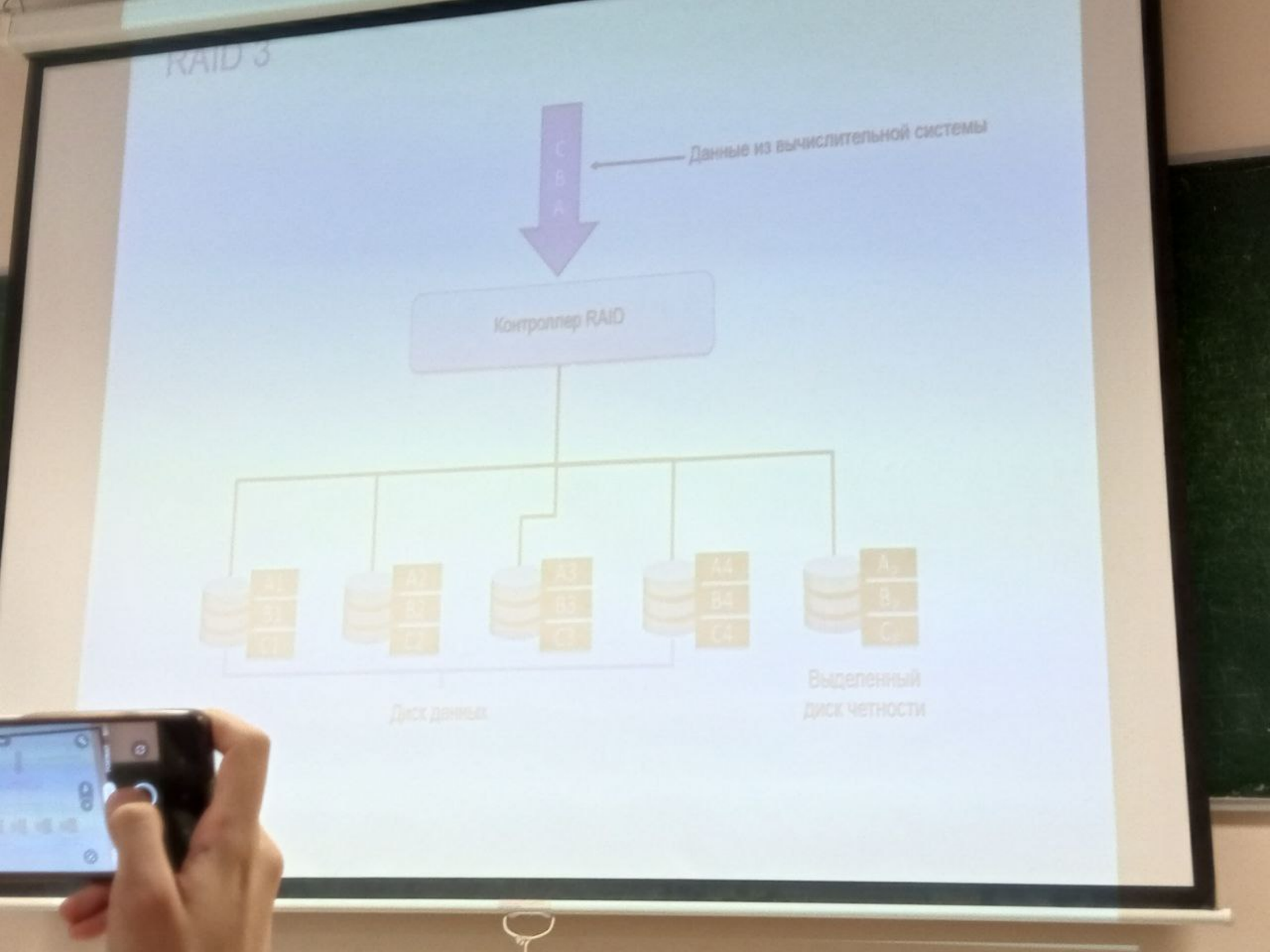
При большом количестве используемых секций рекомендуется использовать ОЗУ не менее 16 ГБ. Если у системы недостаточно памяти, возможен сбой инструкций языка обработки данных (DML), инструкций языка описания данных (DDL) и других операций из-за нехватки памяти.

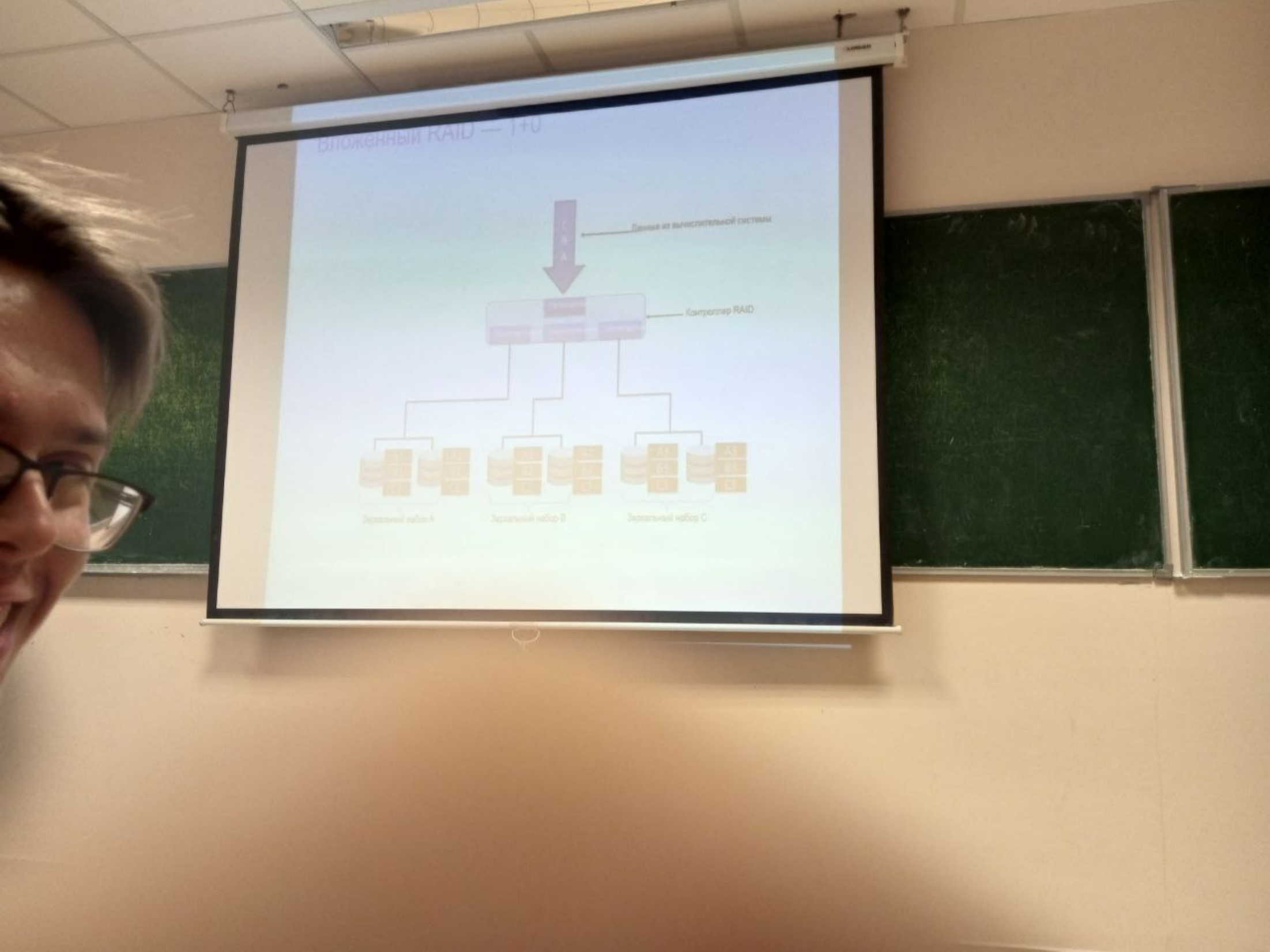

Для максимальной производительности с помощью параллельных операций, рекомендуется, чтобы число секций и процессорных ядер совпадало, но не превышало 64.

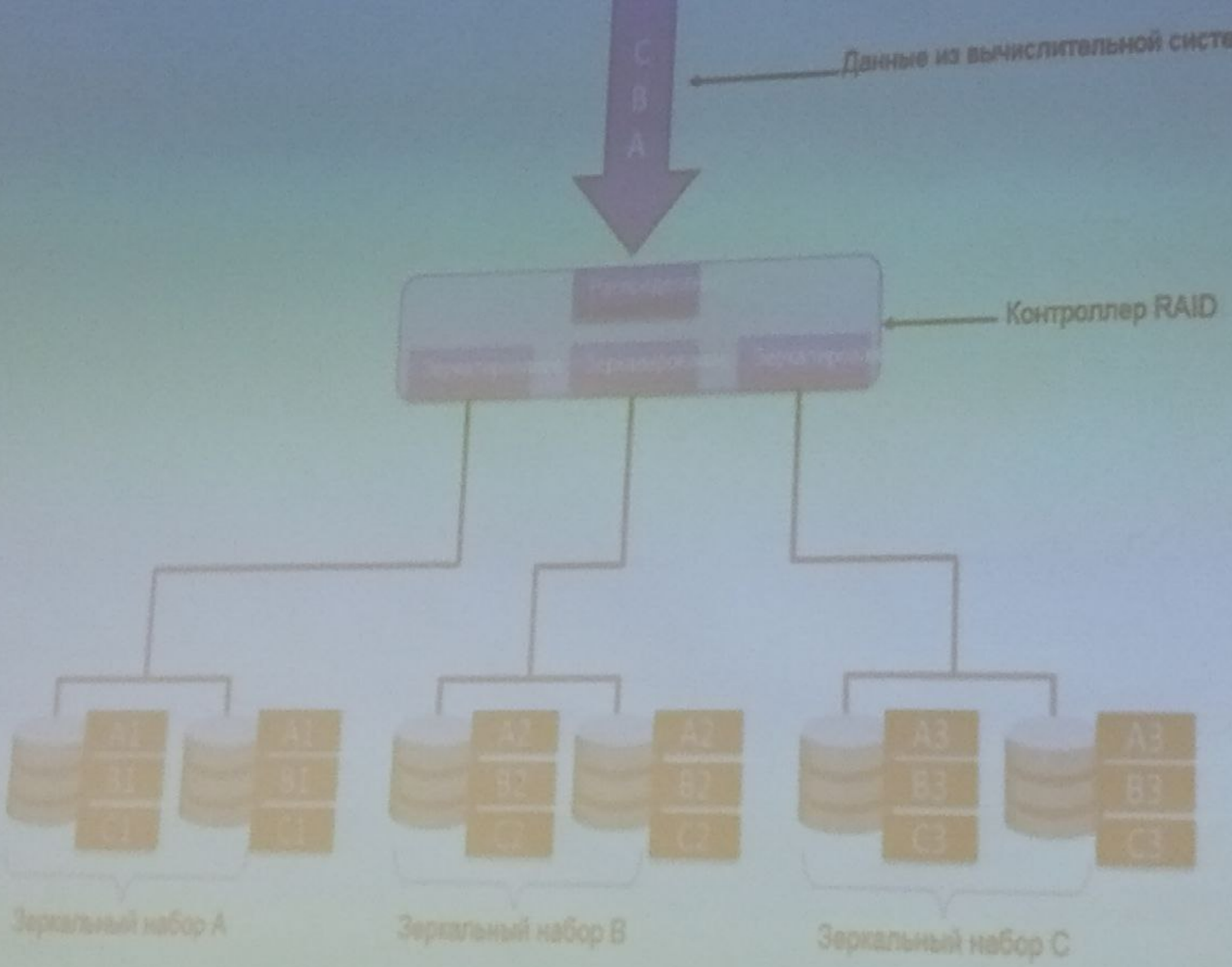
Секционирование помогает повысить производительность, применяя блокировки на уровне секций, а не всей таблицы. Это может уменьшить количество конфликтов блокировок для таблицы.

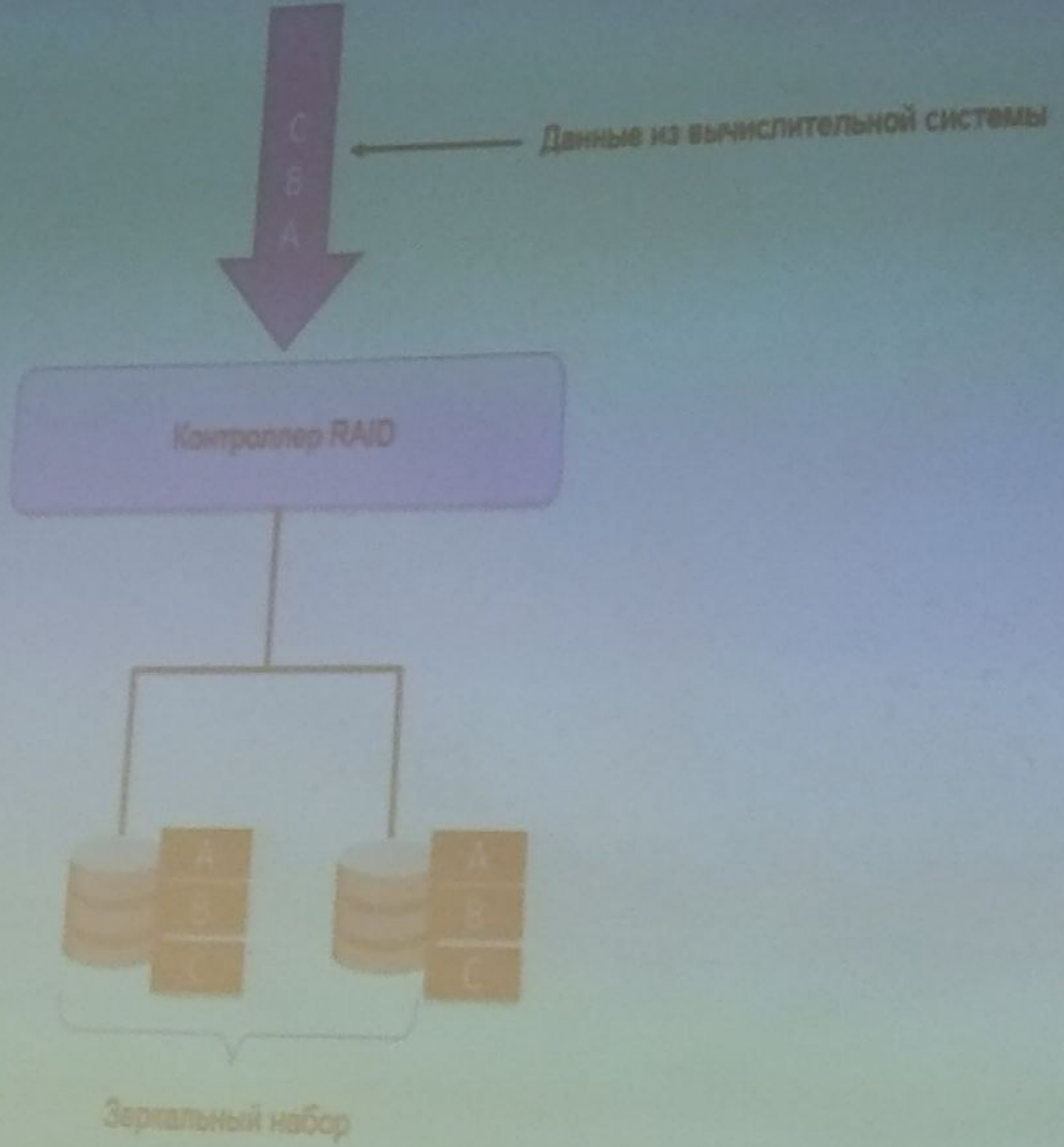
RAID 3

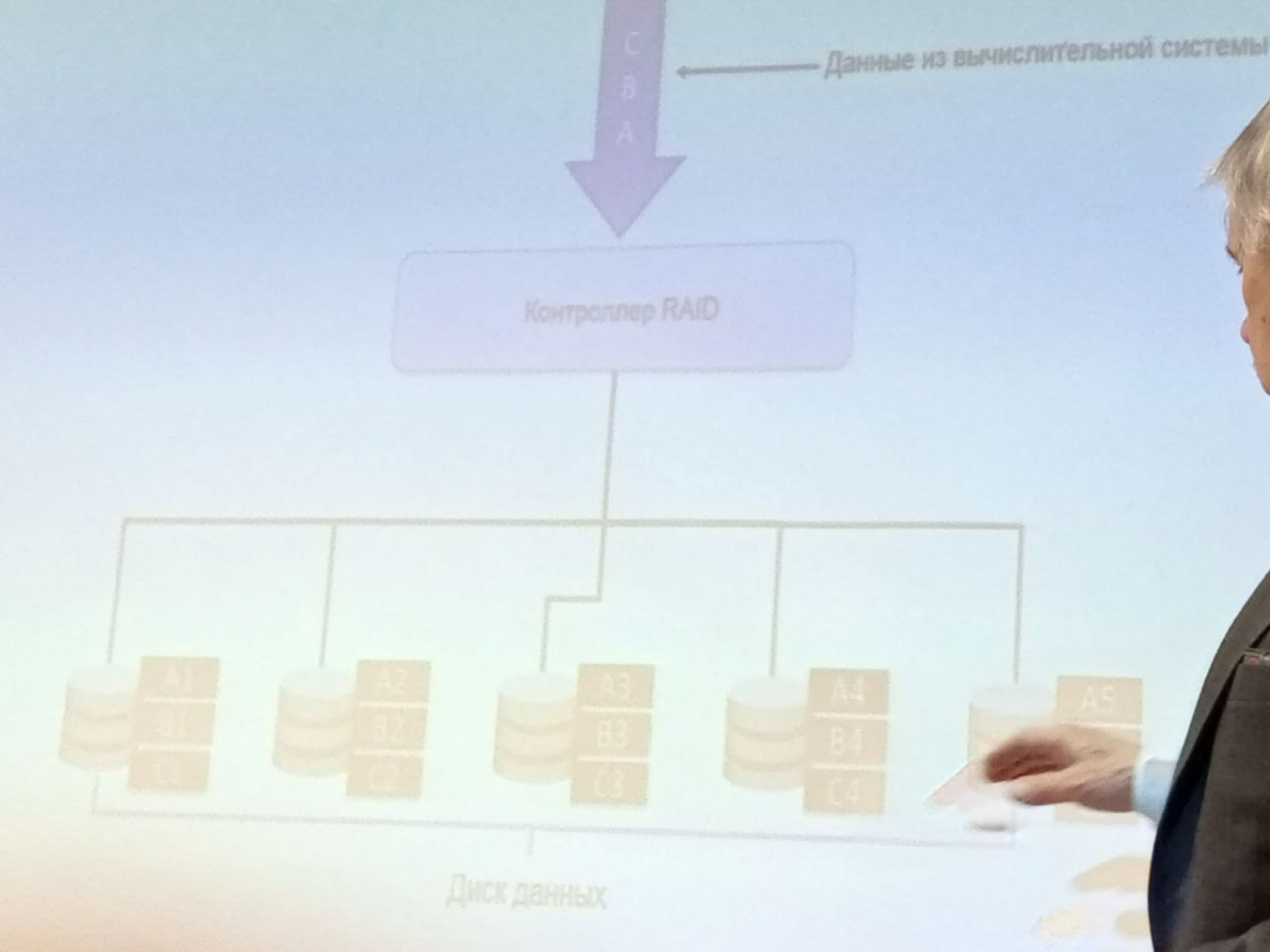

C
B
A
Данные из вычислительной системы
Контроллер RAID
A1
B1
C1
A2
B2
C2
A3
B3
C3
A4
B4
C4
A
B
C
Диск данных
Выделенный
диск четности


RAID 3
Данные из вычислительной системы
Контроллер RAID
Диск данных
Выделенный
диск четности


Вложенный RAID — 1+0
Данные из вычислительной системы
Контроллер RAID


C
B
A
Данные из вычислительной системы
Контроллер RAID
A1
B1
C1
A1
B1
C1
A2
B2
C2
A2
B2
C2
A3
B3
C3
A3
B3
C3
Зеркальный набор A
Зеркальный набор B
Зеркальный набор C

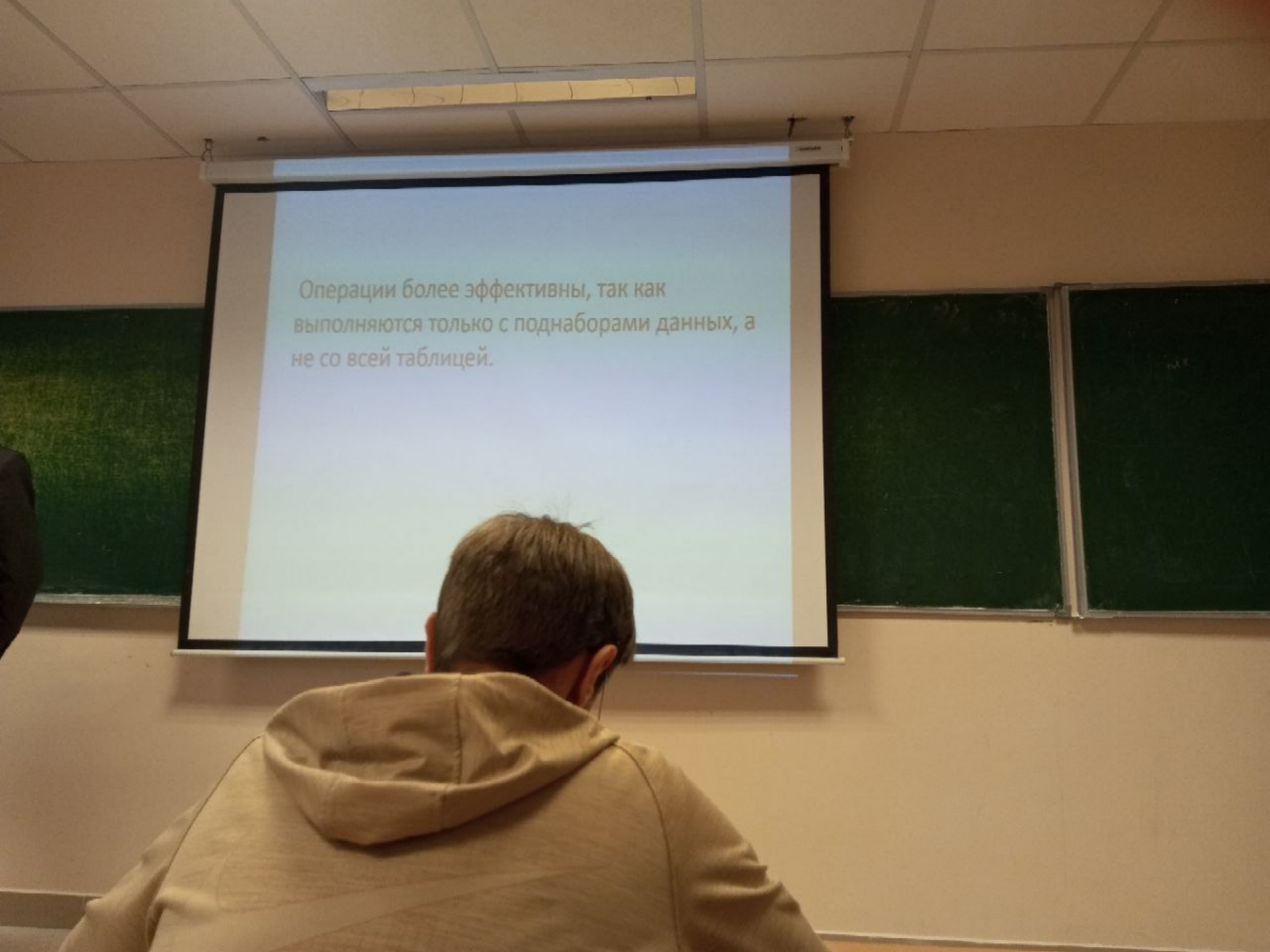

C
B
A
Данные из вычислительной системы
Контроллер RAID
A
B
C
A
B
C
Зеркальный набор

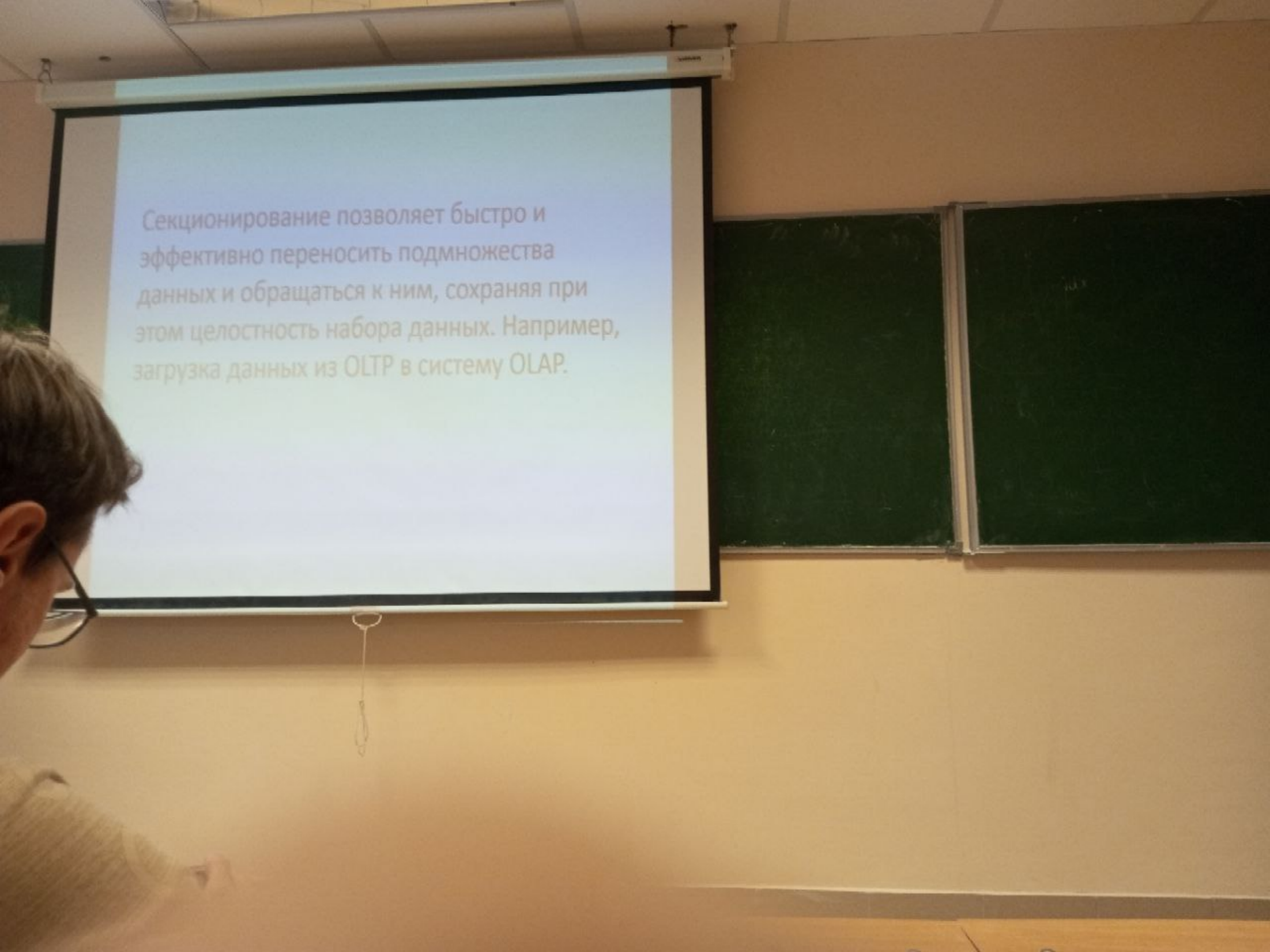

C
B
A
Данные из вычислительной системы
Контроллер RAID
A1
B1
C1
A2
B2
C2
A3
B3
C3
A4
B4
C4
A5
Диск данных

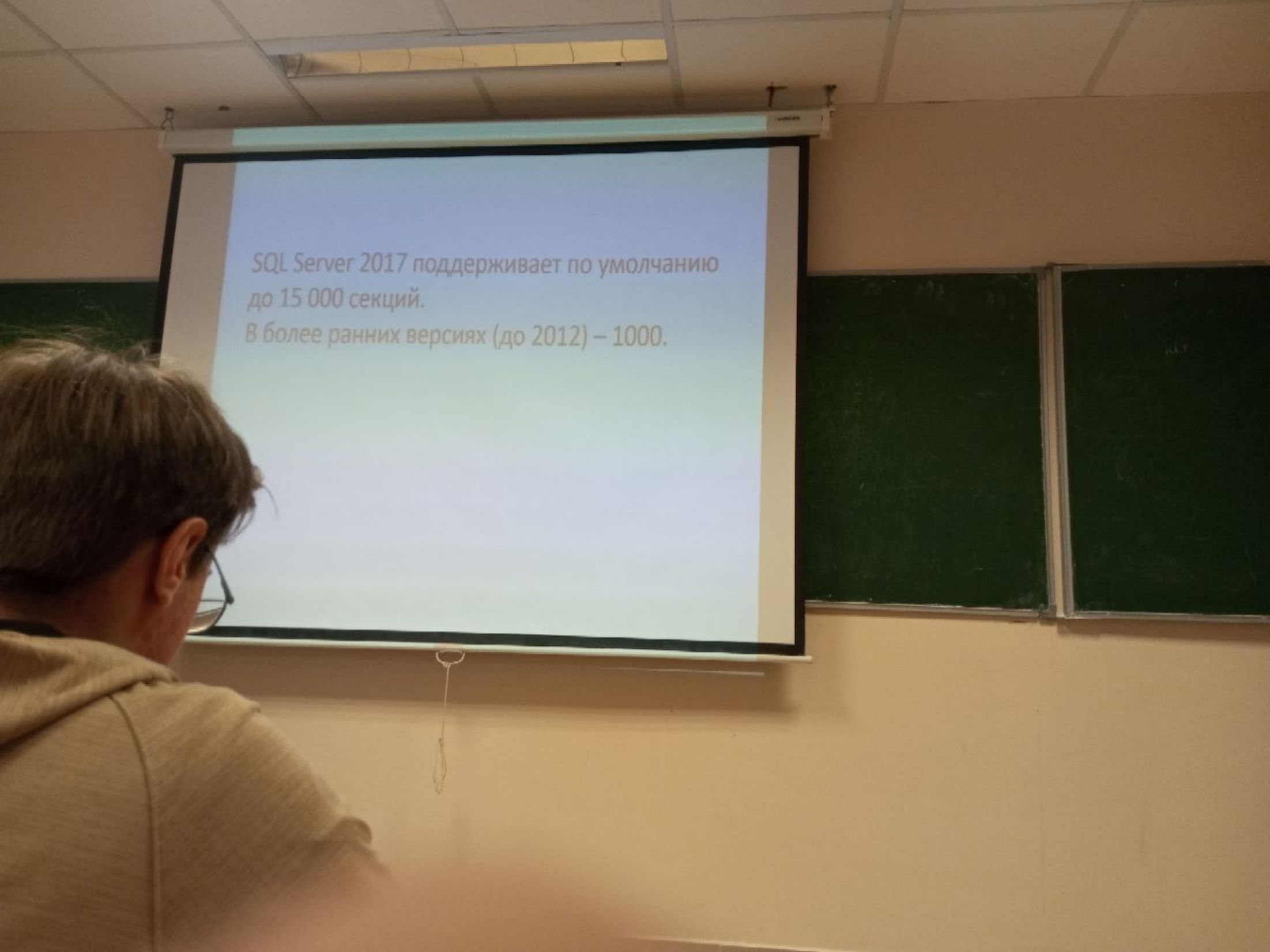
Уровни RAID

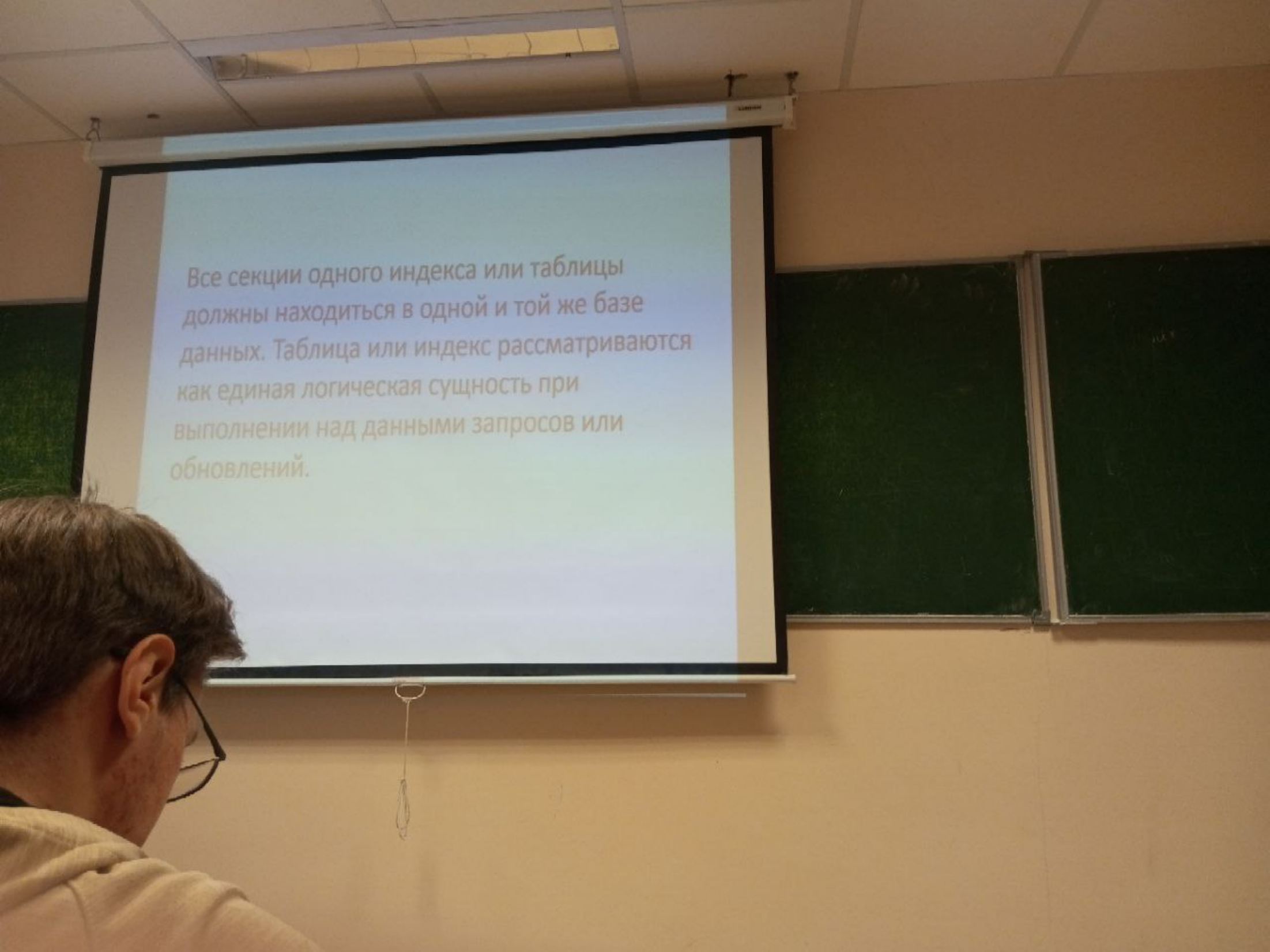
- Часто используемые уровни RAID:
  - RAID 0 — распределенный набор без отказоустойчивости
  - RAID 1 — зеркалирование диска
  - RAID 1 + 0 — вложенный RAID
  - RAID 3 — распределенный набор с параллельным доступом и выделенным диском четности
  - RAID 5 — распределенный набор с независимым доступом к диску и распределенной четностью
  - RAID 6 — распределенный набор с независимым доступом к диску и двойной распределенной четностью

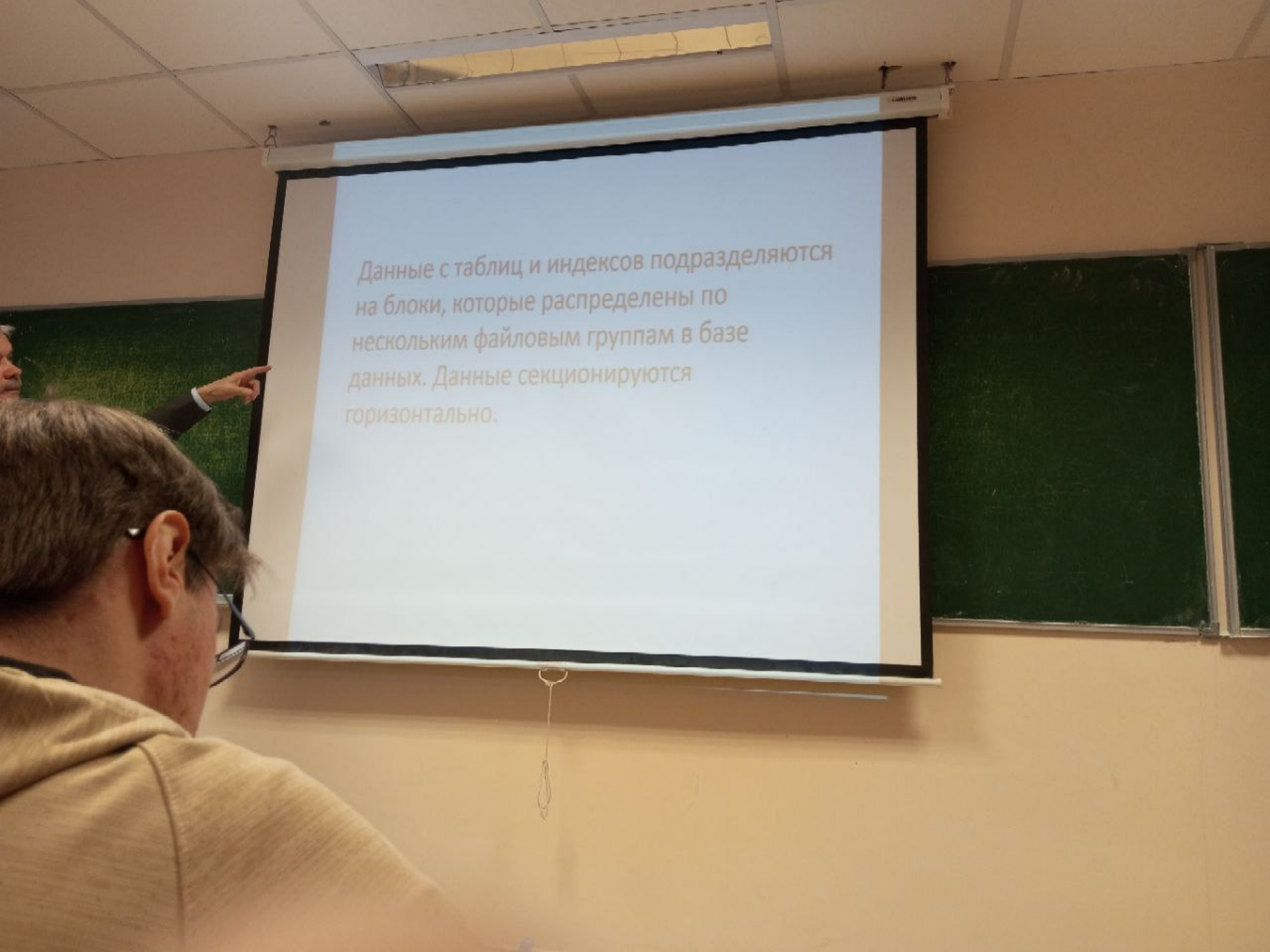

SQL Server может одновременно обращаться только к одному диску. Для ускорения сортировки данных рекомендуется распределить файлы данных в секциях по нескольким жестким дискам, создав RAID. Таким образом, можно одновременно осуществлять доступ ко всем жестким дискам каждой секции.

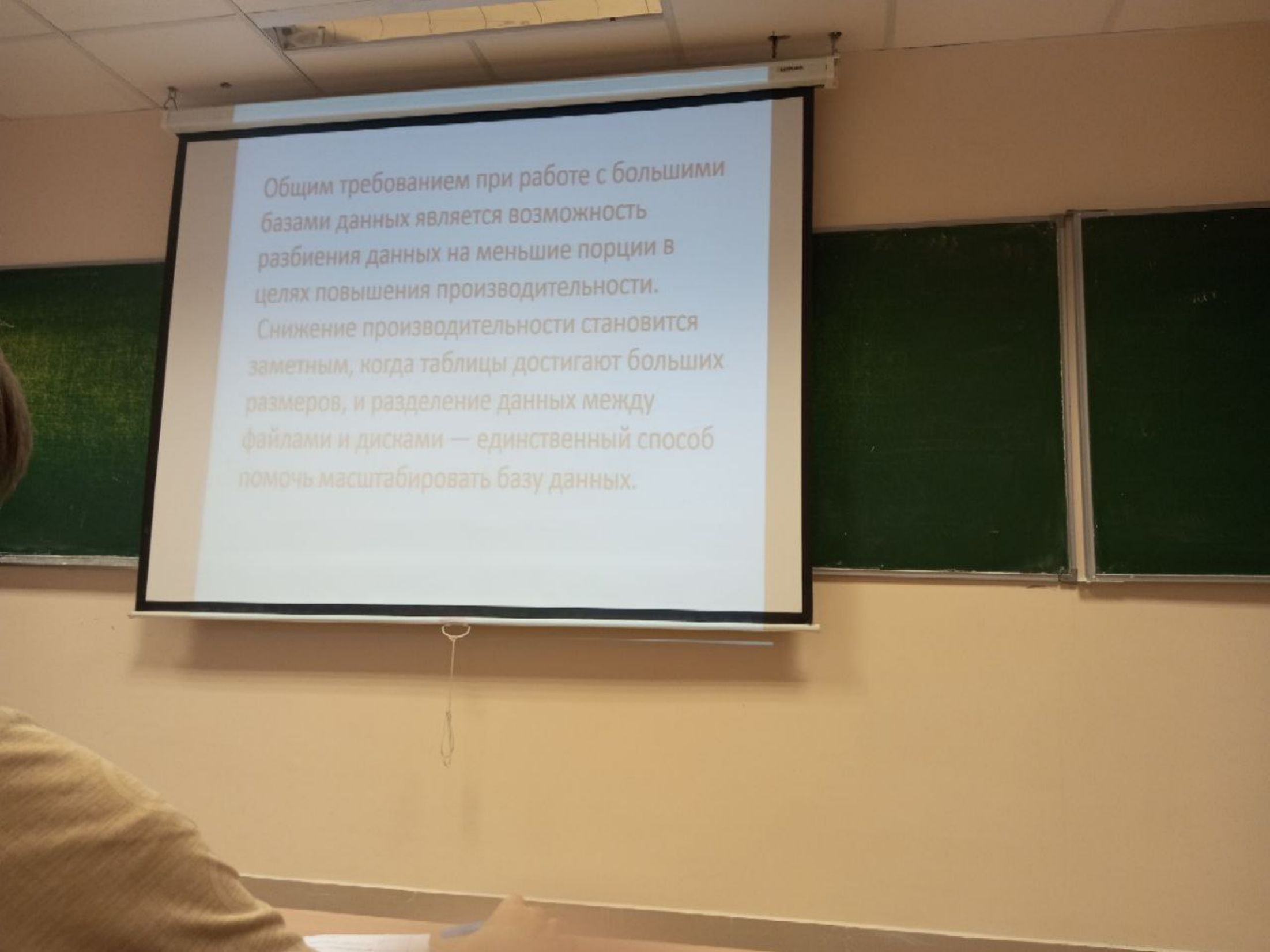

Операции более эффективны, так как выполняются только с поднаборами данных, а не со всей таблицей.

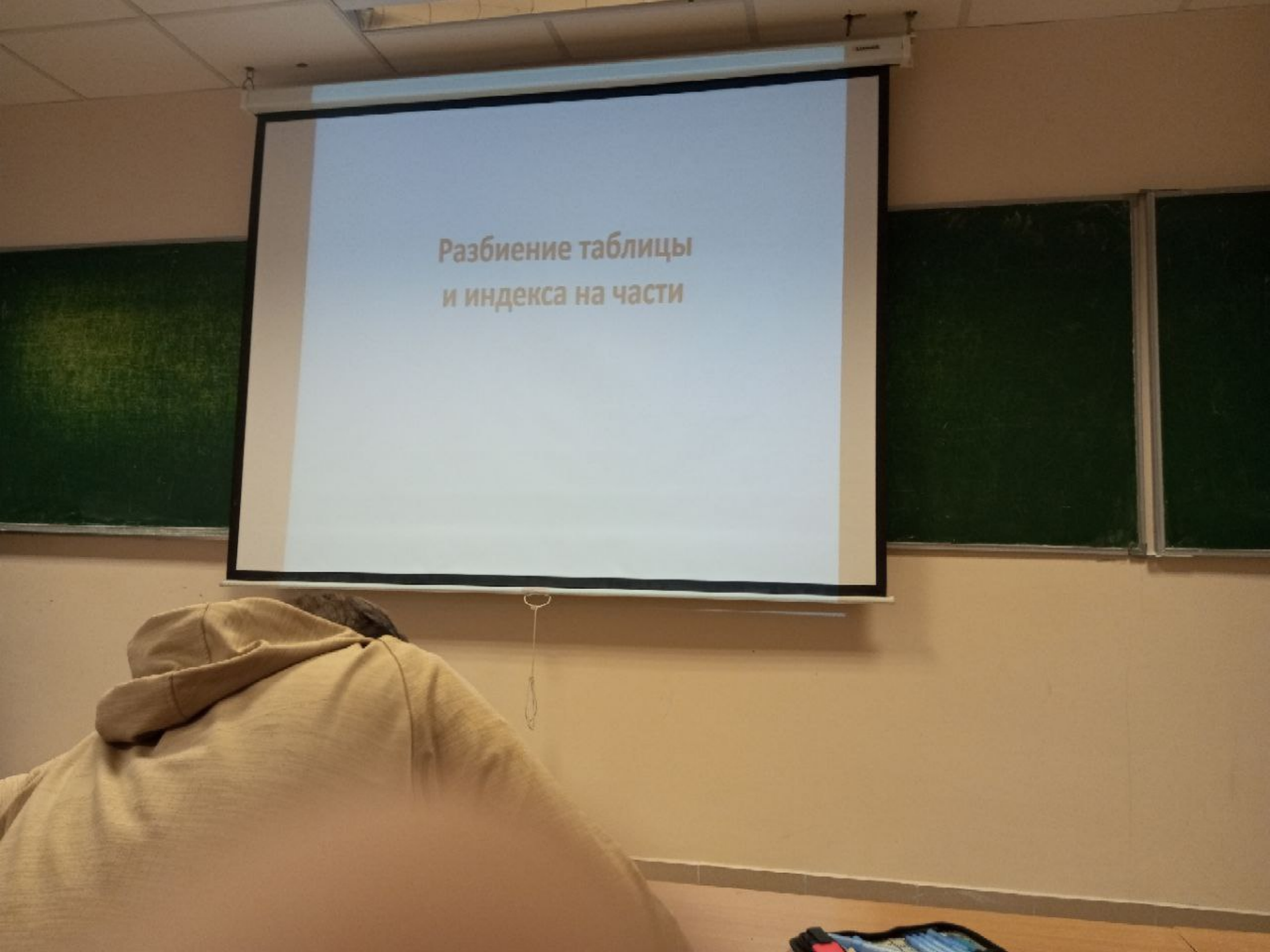

Секционирование позволяет быстро и эффективно переносить подмножества данных и обращаться к ним, сохраняя при этом целостность набора данных. Например, загрузка данных из OLTP в систему OLAP.

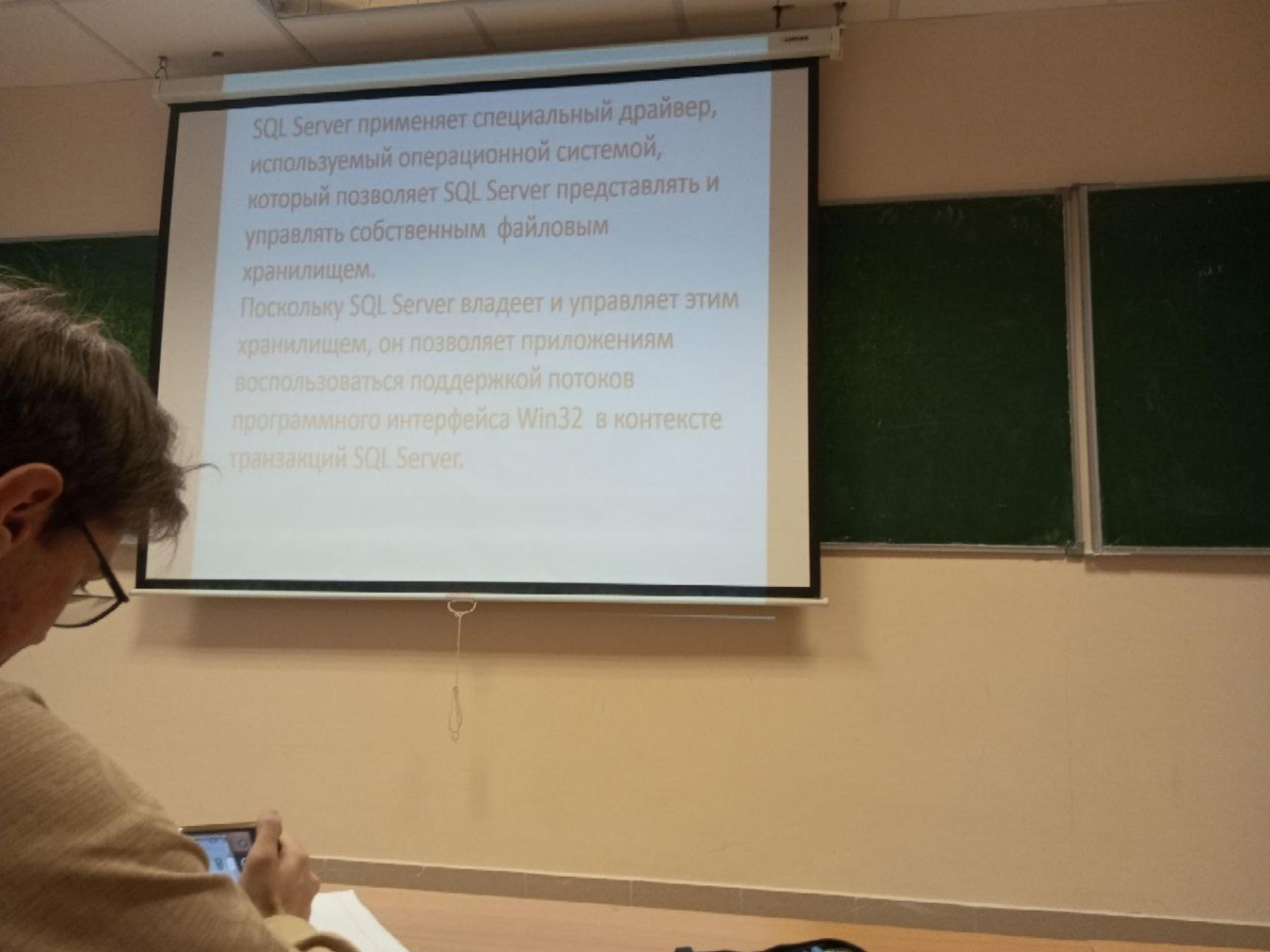

SQL Server 2017 поддерживает по умолчанию
до 15 000 секций.
В более ранних версиях (до 2012) – 1000.

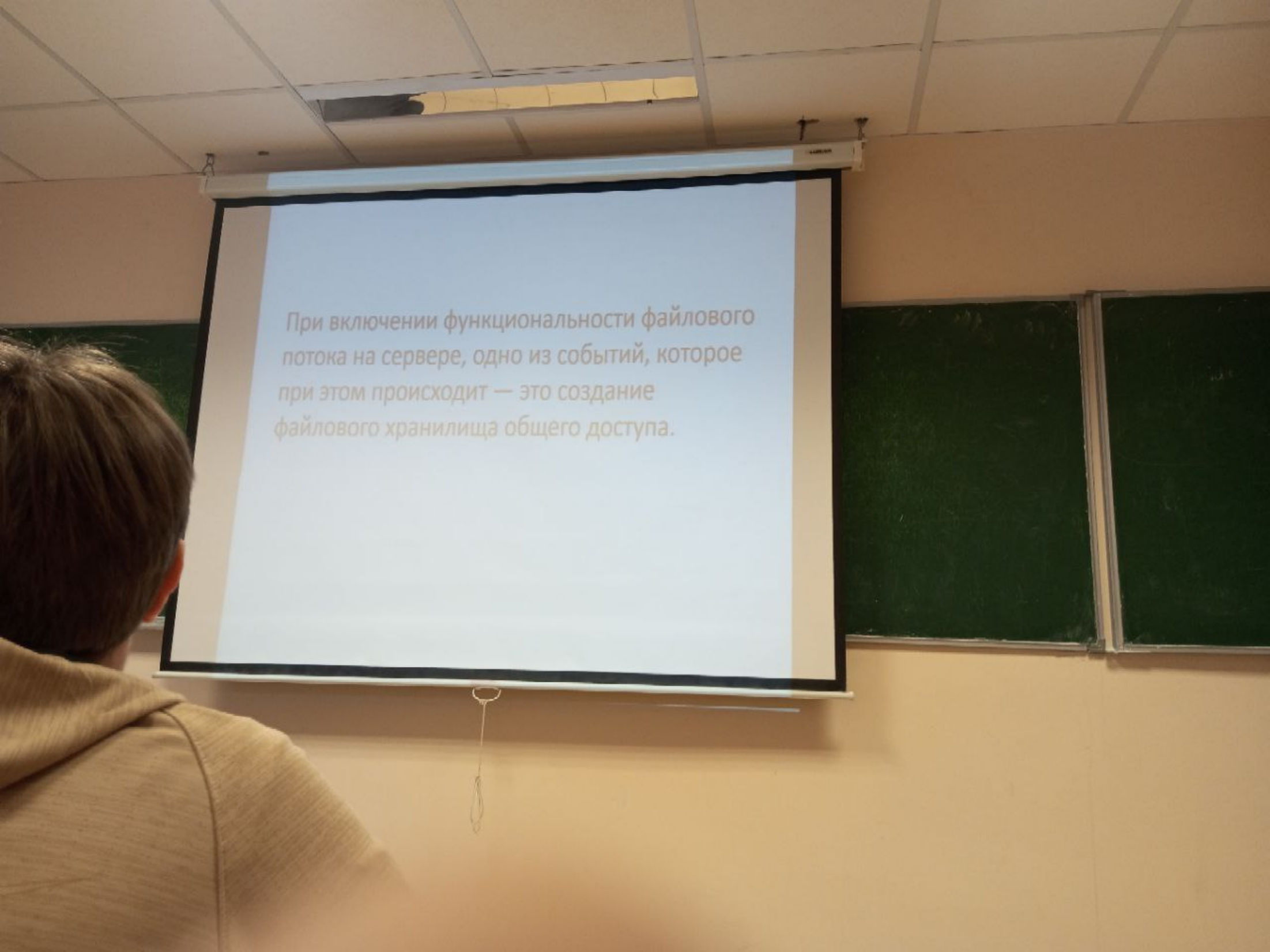

Все секции одного индекса или таблицы должны находиться в одной и той же базе данных. Таблица или индекс рассматриваются как единая логическая сущность при выполнении над данными запросов или обновлений.

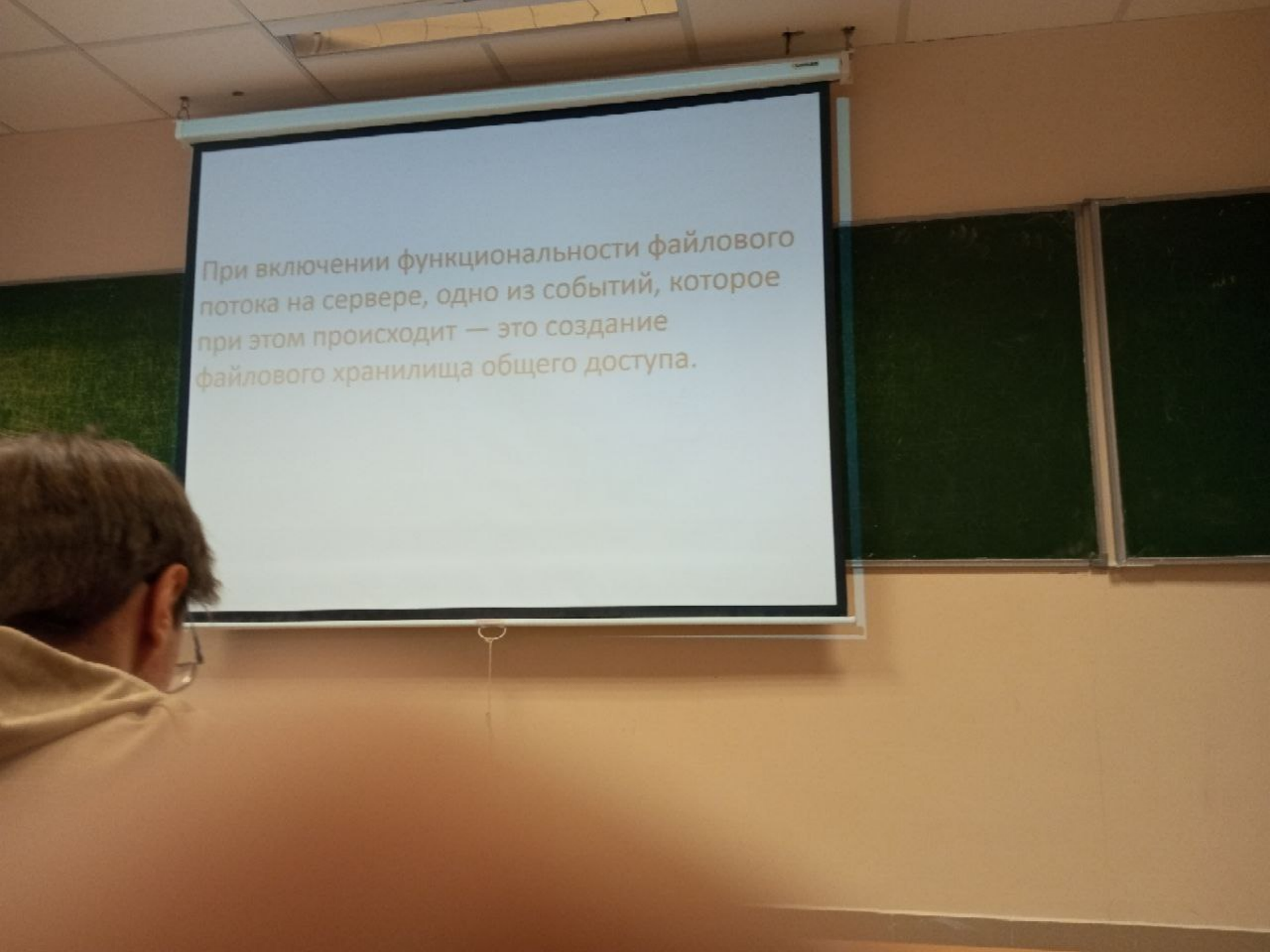

Данные с таблиц и индексов подразделяются на блоки, которые распределены по нескольким файловым группам в базе данных. Данные секционируются горизонтально.

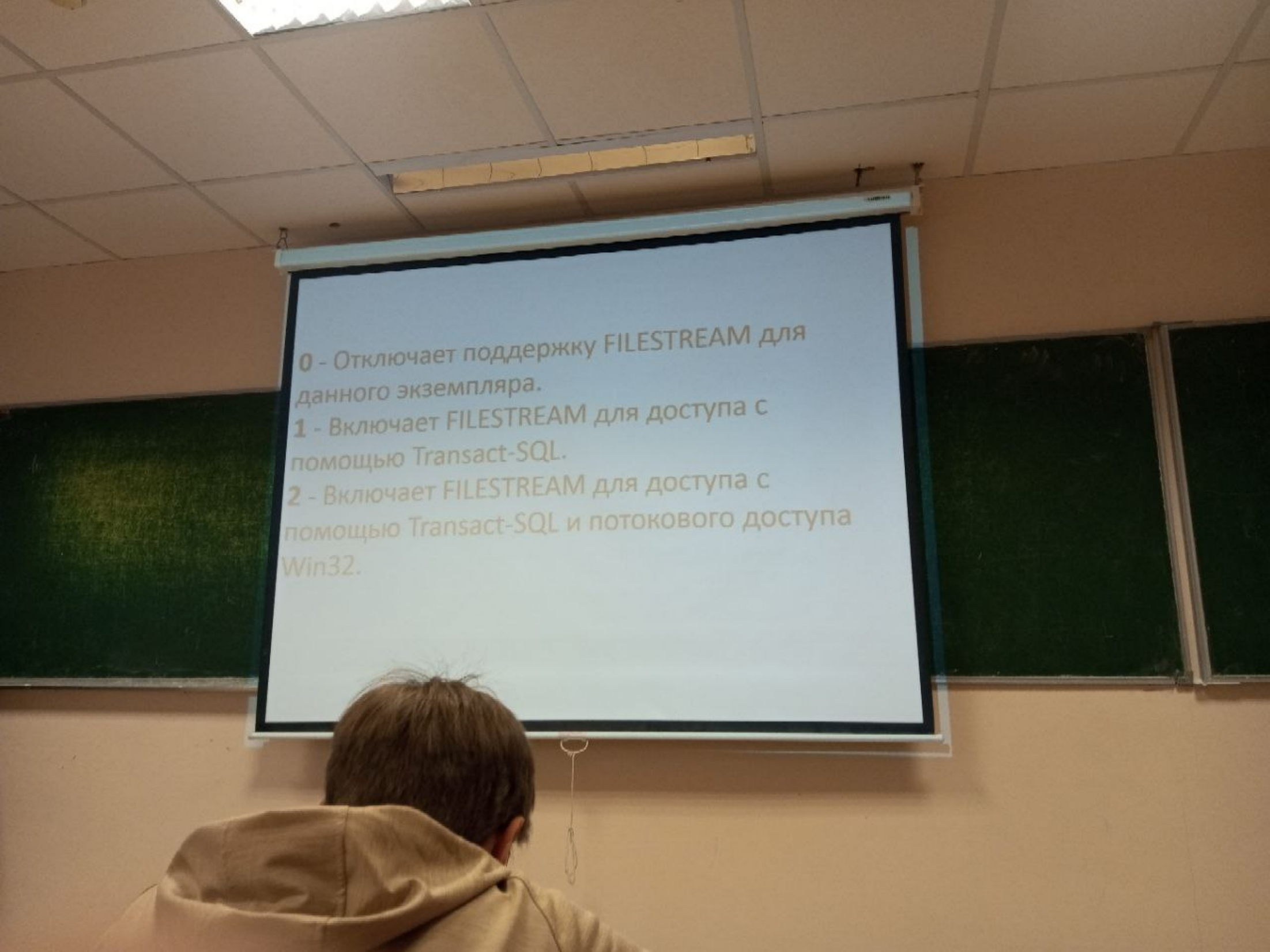

Общим требованием при работе с большими базами данных является возможность разбиения данных на меньшие порции в целях повышения производительности. Снижение производительности становится заметным, когда таблицы достигают больших размеров, и разделение данных между файлами и дисками — единственный способ помочь масштабировать базу данных.

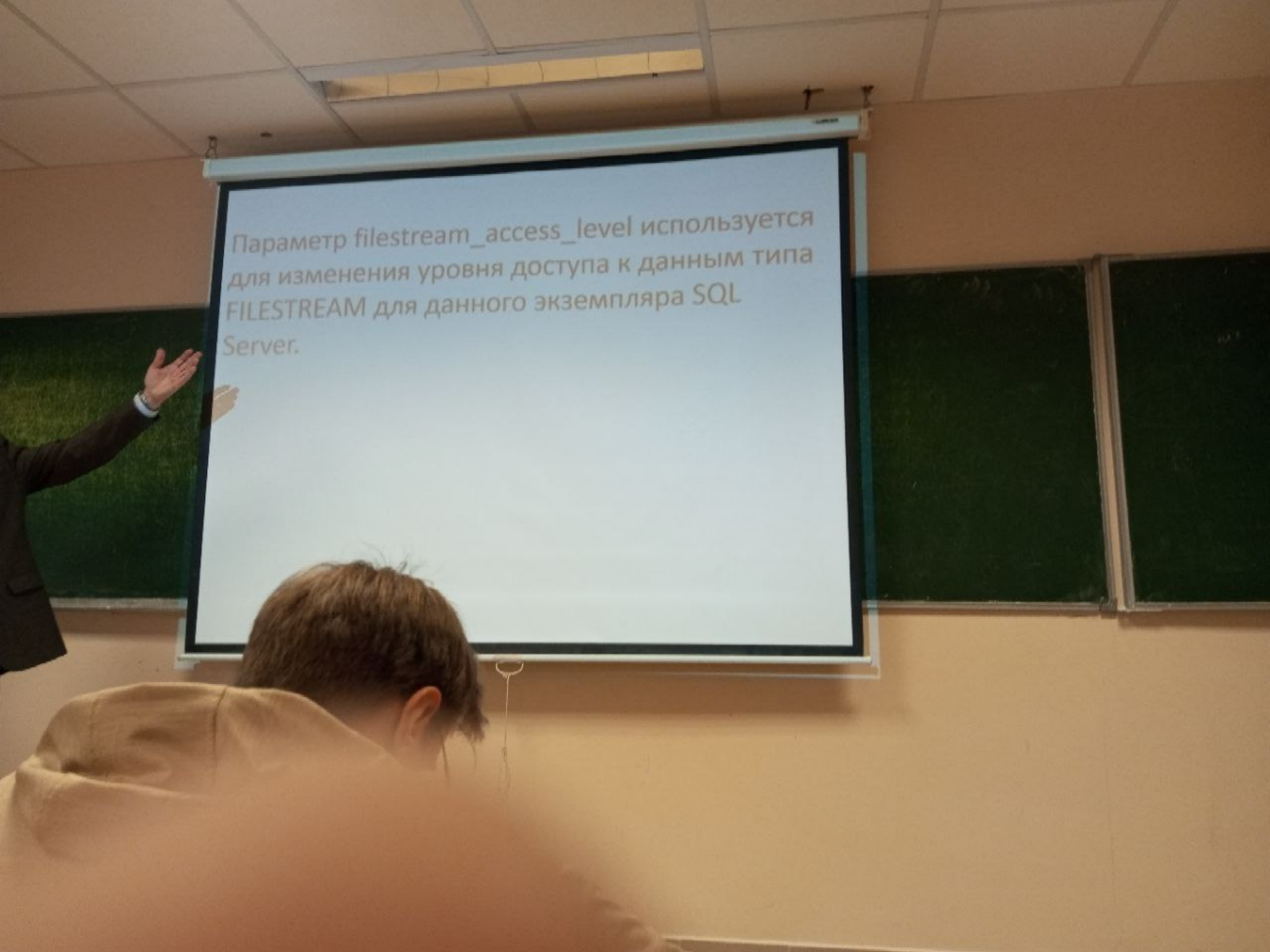

Разбиение таблицы
и индекса на части

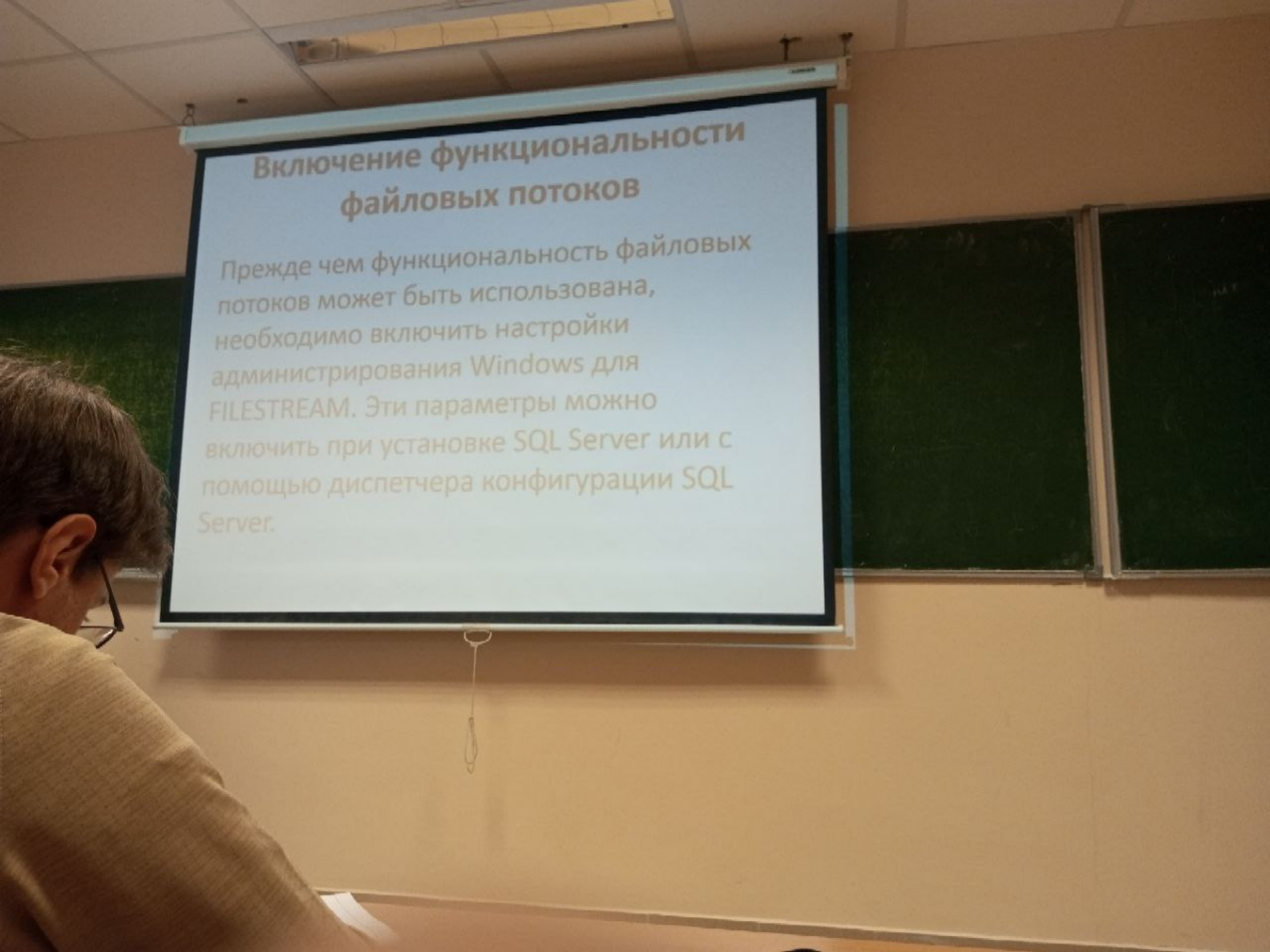

SQL Server применяет специальный драйвер, используемый операционной системой, который позволяет SQL Server представлять и управлять собственным файловым хранилищем.
Поскольку SQL Server владеет и управляет этим хранилищем, он позволяет приложениям воспользоваться поддержкой потоков программного интерфейса Win32 в контексте транзакций SQL Server.

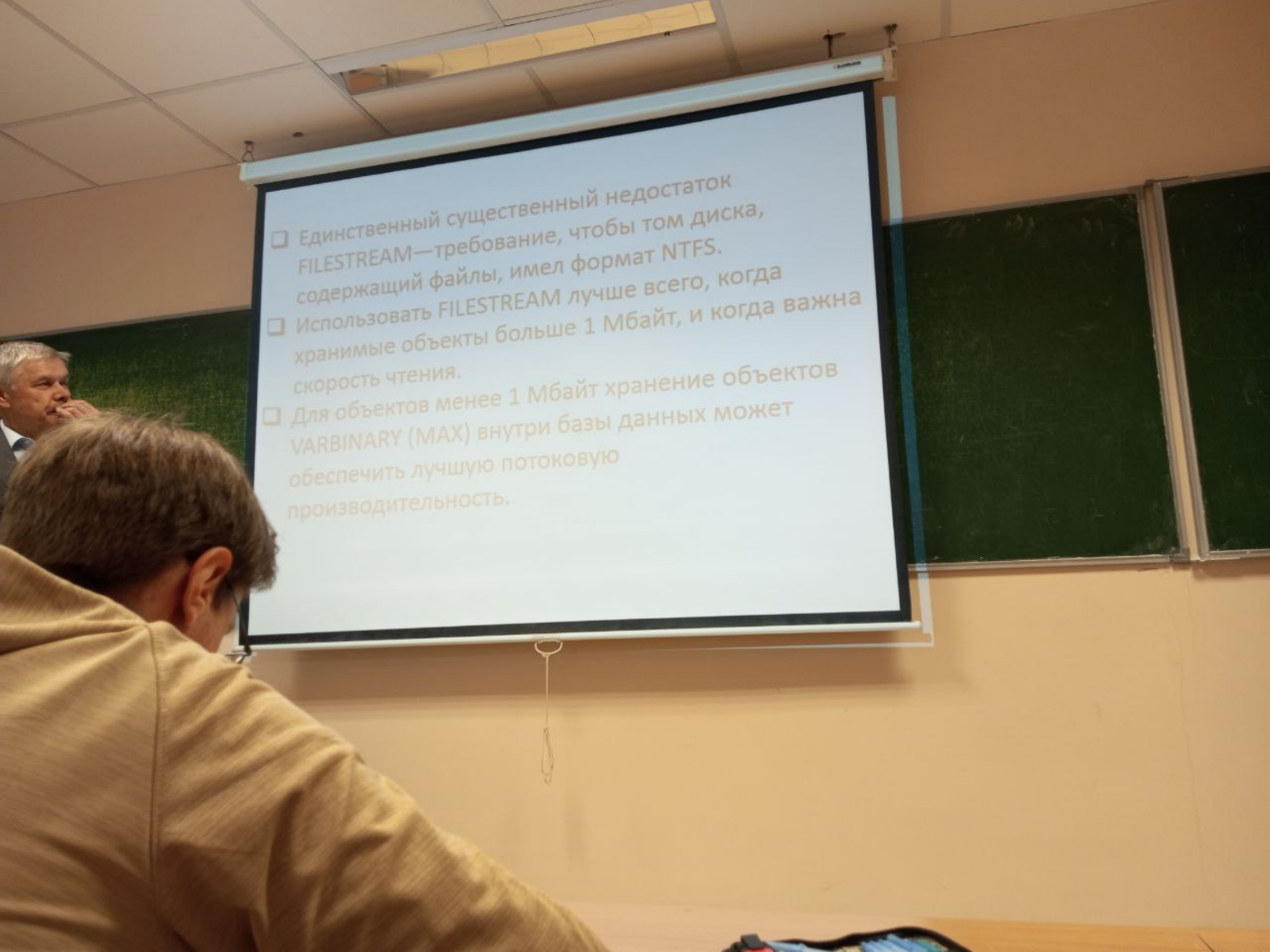

При включении функциональности файлового потока на сервере, одно из событий, которое при этом происходит — это создание файлового хранилища общего доступа.


При включении функциональности файлового потока на сервере, одно из событий, которое при этом происходит — это создание файлового хранилища общего доступа.


0 - Отключает поддержку FILESTREAM для данного экземпляра.
1 - Включает FILESTREAM для доступа с помощью Transact-SQL.
2 - Включает FILESTREAM для доступа с помощью Transact-SQL и потокового доступа Win32.


Параметр filestream_access_level используется для изменения уровня доступа к данным типа FILESTREAM для данного экземпляра SQL Server.


Включение функциональности файловых потоков
Прежде чем функциональность файловых потоков может быть использована, необходимо включить настройки администрирования Windows для FILESTREAM. Эти параметры можно включить при установке SQL Server или с помощью диспетчера конфигурации SQL Server.


Единственный существенный недостаток FILESTREAM—требование, чтобы том диска, содержащий файлы, имел формат NTFS.
Использовать FILESTREAM лучше всего, когда хранимые объекты больше 1 Мбайт, и когда важна скорость чтения.
Для объектов менее 1 Мбайт хранение объектов VARBINARY (MAX) внутри базы данных может обеспечить лучшую потоковую производительность.