Оглавление

[1. Введение 4](#_Toc121085774)

[1.1. Базы данных в общем 4](#_Toc121085775)

[1.1.1. Реляционные базы данных 5](#_Toc121085776)

[1.1.2. Резидентные базы данных 6](#_Toc121085777)

[1.1.3. Поисковые базы данных 6](#_Toc121085778)

[1.1.4. Базы данных с широкими столбцами 7](#_Toc121085779)

[1.1.5. Столбчатые базы данных 8](#_Toc121085780)

[1.1.6. Документоориентированные базы данных 8](#_Toc121085781)

[1.1.7. Графовые базы данных 9](#_Toc121085782)

[2. Реализация база данных с использованием Excel 9](#_Toc121085783)

[2.1. немного про фреймворк Aspose.Cells 9](#_Toc121085784)

[2.2. Программная реализация базы данных 10](#_Toc121085785)

[2.2.1. Создание файла с расширением xls 10](#_Toc121085786)

[2.2.2. Подключение Aspose.Cells к программе 11](#_Toc121085787)

[2.2.3. Создание группы в excel 12](#_Toc121085788)

[2.2.4. Удаление группы 15](#_Toc121085789)

[2.2.5. Cписок групп и потоков 16](#_Toc121085790)

[2.2.6. Добавление студента в группу 18](#_Toc121085791)

[2.2.7. Состав группы / потока 20](#_Toc121085792)

[2.2.8. Удаление студента из группы 21](#_Toc121085793)

[2.2.9. Создание потока 22](#_Toc121085794)

[2.2.10. Сортировка по ФИО или по группе 24](#_Toc121085795)

[2.2.11. сохранение изменений в файл 25](#_Toc121085796)

[3. Заключение 25](#_Toc121085797)

[Литература 27](#_Toc121085798)

# 1. Введение

До появления компьютеров учет студентов ввелся через учетную книгу, туда записывались только потупившие студенты, отчисленные старые студенты, количество академических долгов и замечаний. Вся эта работа требовала очень много времени, пока найдешь, пока откроешь, найдешь нужную фамилию, а потом оказывается, что он в другой книги, или ты неправильно запомнил фамилию и его в книгах нет вообще. Всё изменилось, когда под конец 20 века началось бурное развития ЭВМ и компьютеры были распространены не только среди ученых, но и среди обычных людей. Так начали появляться программы для упрощение некоторых задач, а то и вовсе полностью автоматизировать процесс. К примеру, начали появляться первые версии электронных таблиц, которые в процессе улучшения, стали программой для СУБД (системой управления базами данных), что значительно упростило процесс учета студентов в ВУЗах, как в плане времени, так и в плане денег на новую бумагу.

## 1.1. Базы данных в общем

В базах данных (БД) содержится упорядоченная информация, которой удобно пользоваться. Они делятся на разные типы — чтобы выбрать нужный, важно учесть, какие именно данные будут там храниться и по какому принципу будет удобнее всего работать с ними.

В целом нельзя сказать, что какие-то БД лучше других, — просто каждая из них подходит для решения каких-то определённых задач. Есть базы данных с открытым кодом, с возможностью масштабирования и с другими преимуществами. Лучше выбирать такие БД, которые вы сможете использовать именно так, как они задуманы.

Существует 7 основных типов баз данных: реляционные, резидентные, поисковые, с широкими столбцами, столбчатые, документоориентированные, графовые.

### 1.1.1. Реляционные базы данных

Примеры — MySQL, Oracle DB, PostgreSQL. Это самый популярный тип БД, в которых информация хранится в виде таблиц. В строках находится описание каждого отдельного свойства объекта, а столбцы нужны для извлечения определённых свойств из строки. Таблицы могут быть взаимосвязаны.

Реляционная модель проста, но позволяет выполнить множество разных задач. Ею удобно пользоваться, если нужно связать элементы данных между собой и безопасно и надёжно управлять ими. Такие таблицы можно создать для хранения и обработки телефонных номеров пациентов, логинов и паролей пользователей, для отслеживания товарных запасов. При этом БД обеспечивает целостность данных в различных экземплярах базы в одно и то же время.

В реляционных БД есть поддержка SQL, а также индексация, которая позволяет быстрее находить нужные данные. Особый плюс таких баз — нормализация данных: они делятся на разные таблицы, поэтому исключены повторяющиеся или пустые ячейки. Транзакции реляционных БД соответствуют ACID — набору свойств, который гарантирует их надёжную обработку. Из минусов баз можно отметить относительно низкую скорость доступа к данным, плохую поддержку неструктурированных данных, сложность масштабирования и образование большого количества таблиц, из-за чего бывает трудно понять структуру данных.

Наша база данных студентов будет реализована по принципу реляционных баз данных

### 1.1.2. Резидентные базы данных

Примеры — Redis, Apache Ignite, Tarantool. Сведения хранятся в оперативной памяти. Данные обрабатываются быстро, поэтому резидентные БД популярны там, где нужно обеспечить максимально короткое время отклика. Они помогают управлять телекоммуникационным оборудованием, проводить торги в онлайн-режиме или Real-Time обслуживание. Базы in-memory поддерживают и быстрое написание, и быстрое чтение. В основном они работают с записями «ключ-значение», но также могут работать со столбцами.

Чтобы при неожиданной перезагрузке не потерять данные, нужно сделать запись с предварительным журналированием на энергонезависимом устройстве. Это можно отнести к минусам базы in-memory — приходится вкладываться в дорогостоящие инфраструктурные решения, чтобы обеспечить бесперебойное питание. Также нужно постоянно копировать информацию на твёрдые носители. Ещё один недостаток БД — дорогое масштабирование.

### 1.1.3. Поисковые базы данных

Пример — Elastic. Этот тип БД нужен для получения сведений через фильтр. Искать можно по любому введённому значению, в том числе по отдельным словам. Можно пользоваться полнотекстовым поиском. Поисковые базы данных хорошо масштабируются и удобны для хранения журналов, объёмных текстовых значений.

Можно использовать поисковые БД для мониторинга оптимизации цен, обнаружения ошибок в приложении по бронированию билетов и решения множества других задач. В базе могут хранится миллиарды документов. Поиск осуществляется быстро. Минусы системы — плохая аналитическая поддержка и ограниченная возможность применения БД (можно использовать только для пакетных вставок).

### 1.1.4. Базы данных с широкими столбцами

Примеры — Cassandra, Google BigTable, HBase. БД с широкими столбцами могут запрашивать большие объёмы данных быстрее, чем обычные реляционные. Сведения хранятся в виде записей «ключ-значение» на жёстком диске или твёрдотельном накопителе. Базы данных с широкими столбцами позволяют выполнять быструю запись построчно и быстрое чтение по ключу.

БД хорошо масштабируются и подходят для организации магазинных каталогов, механизмов обнаружения мошенничества. Их удобно использовать для управления огромными объёмами информации на множестве общих серверов в распределённой системе. Недостатками базы данных считается то, что она работает в формате «ключ-значение» и не имеет поддержки аналитики.

### 1.1.5. Столбчатые базы данных

Примеры — Clickhouse, Vertica. В БД такого типа данные хранятся в столбцах, а не в строках. Получение доступа к содержимому осуществляется без помощи ключей. При использовании столбчатых баз данных используют пакетную вставку, чтобы можно было готовить информацию для быстрого чтения по столбцам. В столбчатых БД есть поддержка аналитики и возможность удобного масштабирования.

Такие базы данных используют там, где нужно запрашивать информацию по определёным столбцам, — в системах розничных продаж и финансовых транзакций. Основный минус у БД только один: она подходит только для пакетных вставок.

### 1.1.6. Документоориентированные базы данных

Примеры — CouchDB, Couchbase, MongoDB. Если в реляционных БД для извлечения данных нужно объединять таблицы, то в этих базах отлично хранится несвязанная информация в больших объёмах. Они поддерживают JSON. Для любого ключа можно создать сложное значение и сразу включить всю структуру данных в одну запись. Выборка по запросу может содержать части множества документов без их полной загрузки в оперативную память.

В документоориентированных базах нет привязки к схеме. Они подходят для OLTP и поддерживают сложные типы. Такие БД предпочитают использовать в системах управления содержимым, для поиска документов, в издательском деле. Три недостатка базы данных — отсутствие хорошей аналитической поддержки и поддержки транзакций, а также сложности с масштабированием.

### 1.1.7. Графовые базы данных

Примеры — OrientDB, Neo4j. Данные хранятся в виде графов, то есть моделей с узлами и связями. Они достаточно гибкие, с логичной структурой. Узлы служат для хранения сущностей данных, а рёбра — для хранения взаимосвязей между сущностями, которыми можно управлять.

Графовые БД применяют для решения задач в биоинформатике, а также для моделирования социальных сетей, чтобы хранить взаимосвязанную информацию о людях. Базы данных такого типа плохо поддаются масштабированию, а второй их недостаток — необходимость использовать особый язык запросов SPARQL, который отличается от SQL.

# 2. Реализация база данных с использованием Excel

В данной курсовой работе я буду использовать язык С#/.NET c использованием фреймворком Aspose.Cells для доступа к Excel таблицам.

## 2.1. немного про фреймворк Aspose.Cells

Aspose.Cells for .NET - это API для программного взаимодействия с таблицами Excel, позволяющий разработчикам программного обеспечения манипулировать и конвертировать файлы с таблицами из разработанных ими приложений. Являясь комбинацией из различных API и элементов управления GUI - Aspose.Cells for .NET ускоряет программирование и переход с Microsoft Excel.  
 Этот API для таблиц Excel предоставляет разработчикам набор мощных инструментов для осуществления как простых операций по конвертации файлов, так и для более сложных задач. Разработчики получают контроль над отображением страницы, форматированием, графиками и формулами. Они могут читать и записывать файлы с таблицами и выгружать их в разнообразные графические, текстовые и форматы файлов PDF.  
 Aspose.Cells for .NET - быстр и надежен. Он бережет ваше время и усилия, если сравнивать с самостоятельной разработкой решения по манипуляции таблицами или с автоматизацией от Microsoft.

2.2. Программная реализация базы данных

### 2.2.1. Создание файла с расширением xls

Для создания файла в языке C# нужно объявить конструктор вида FileStream дабы объявить File.Open(), в метод Open() записываем название файла типа String, а вторым параметром условия открытия файла, существует 6 форматов открытия файла, такие как:

- Append - если файл существует, то текст добавляется в конец файла. Если файла нет, то он создается. Файл открывается только для записи.

- Create - создается новый файл. Если такой файл уже существует, то он перезаписывается

- CreateNew - создается новый файл. Если такой файл уже существует, то приложение выбрасывает ошибку

- Open - открывает файл. Если файл не существует, выбрасывается исключение  
 -OpenOrCreate - если файл существует, он открывается, если нет - создается новый

- Truncate - если файл существует, то он перезаписывается. Файл открывается только для записи.

В нашем случае мы будем использовать формат OpenOrCreate, так как нам надо именно проверить существование этого файла и в случае если его нет создать этот файл, а если есть, то идем дальше по программе.

В коде это выглядит так:

 Листинг 2.1 – Создание файла «Students.xls»

Как видно в 1-ом листинге, мы создаем файл Students.xls, поскольку класс FileStream использует потоки, то его работу надо закрывать, а иначе при повторном применении или в случае некорректной работе с потоком могут появляться непонятные ошибки, следовательно, пишем на следующей строчке закрытие потока FileStream.

### 2.2.2. Подключение Aspose.Cells к программе

Теперь, используя фреймворк Aspose.Cells, подключим наш новосозданный xls файл к программе, в нашем случае это «Students.xls».



Листинг 2.2 – Инициализация xls файла

Здесь мы объявляем класс Workbook из Aspose.Cells в который мы записали наш файл, теперь мы можем начать работать с Excel.

### 2.2.3. Создание группы в excel

В концепте данной курсовой работе я решил, что название листа будут обозначать к какой группе или потоку принадлежат студенты. В Aspose.Cells есть специальная команда для добавление нового листа в excel документ.



Листинг 2.3 – Создание нового листа с названием группы

Но если человек ошибся в названии группы, то эта группа и добавится в файл, что нужно в последствии удалять его. Да бы такого не было сделаем проверку на запись группы. В качестве стандарта возьмем принцип записи у

ВУЗа РТУ МИРЭА, а именно первая буква — это название института, вторая – название направления, третья – уровень образования направления (бакалавриат/специалитет/магистратура), четвертая – форма обучения (очно/заочно/дистанционно), потом через дефис идут порядок группы в потоке и год зачисления. Пример, группа ККСО-06-20: К – институт Кибернетики, К- Компьютерная безопасность, С – специалитет, О – очная форма обучения, 06 – 6 группа по счёту в потоке, 20 – группа зачислена в 20 году. Следовательно, в С# эта проверку можно сделать так:

 Листинг 2.4 – Функция проверки группы

Поскольку тип String мы можем разбить на массив из символов типов char, то можем проверить каждый символ по отдельности, как это представлено в листинге выше, теперь используем нашу функцию в основной части кода.



Листинг 2.5 – Проверка записи группы

Теперь группу будут писать верно, в исходнике, который лежит на гитхабе (ссылка во вкладке литература), я использовал систему switch case, для создание консольного меню, чтобы после ошибки записи программа не сбрасывалась и не пришлось пользователю обратно нажимать на меню, я добавил оператор goto для перехода в начало записи группы, прошлая неправильная запись в переменную GroopName перезапишется на другую запись, в исходнике это выглядит так:



Листинг 2.6 – Полная проверка название группы и её добавление в лист

Помимо неправильной записи группы, встает ещё один вопрос, а не создана это группа вообще, к счастью, в Aspose.Cells этот случай учтен и в случает повторного добавление одного и того же листа выдаст исключение Aspose.Cells.CellsException, следовательно, нам надо это просто обработать через конструкцию try catch.



Листинг 2.7 – Проверка на существование группы

В листинге 2.6 этот момент тоже прописан.

### 2.2.4. Удаление группы

Бывает момент, что группа остается пустой, либо студенты объединяются с другой группой или другие случаи, когда группа становится не нужной, то её надо удалить. В нашем случае надо удалить лист из Excel файла, в Aspose.Cells есть соответствующий метод RemoveAt() в классе Worksheets.



Листинг 2.8 – удаление листа из excel

Как и в случае с добавлением группы, нам надо учесть его правильное написание и сделать проверку на его существование, т.е. добавить блок с функцией на проверку и с try catch, для проверки на наличие этой группы, как сделано в исходнике.



Листинг 2.9 – удаление группы из excel

Но в отличии от добавления листа в файл, метод RemoveAt() не имеет встроенного исключения на существовании группы, поэтому мы использовали строчку в которой это есть, но при этом никак не влияет на процесс удаления группы.



Листинг 2.10 – та самая строчка с проверкой на существование группы

Данная строчка будет использоваться везде где нужно проверить на существование группы.

### 2.2.5. Cписок групп и потоков

Здесь все просто. Поскольку класс Workseets имеет в себе конструктор массива, то достаточно использовать цикл for.



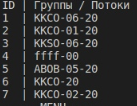
Листинг 2.11 – Вывод списка групп/потоков в консоль

К сожалению, фреймворк платный и требует подписки от 10 долларов, поэтому после каждого сохранения будет создавать лист с надписью «Evaluation Warning» и дабы это вывод не засорялся десятками этими сообщений мы добавили в цикл обход от таких листов и другой счетчик количества групп и потоков. Ещё один момент, это переменная countSheet, она автоматически считывает сколько всего листов в текущем excel файле.



Листинг 2.12 – присваивание значения countSheet

Так же для легкового чтения вывода мы сделали таблицу в консоли, выглядит вывод так:



### 2.2.6. Добавление студента в группу

Теперь самая важная часть всей курсовой работы. После добавления группы нам надо добавить туда студентов. Студенты в мире разные в плане из ФИО, но их объединяет одно, в ФИО нет цифр, следовательно, надо сделать такую проверку:



Листинг 2.13 – Функция проверки имени на цифры

Принцип тот же что и с функцией проверки групп (см. листинг 2.4), проверяется каждое слово, кроме пробелов, на наличие цифр.  
 Теперь мы должны задать в какой группе мы будем писать студентов и с какой ячейки начнем. Мы решили использовать столбик «А». Так же надо учесть, что студентов группе может меньше 30 человек, следовательно, надо сделать выход из цикла, а также учесть момент, где в группе добавляется ещё один или несколько студентов. 

Листинг 2.14 – добавление студентов в группу

Поскольку мы теперь работаем с ячейками электронной таблицы, надо вызвать соответствующую переменную типа Cell. Мы решили, что в группе могут быть максимум 30 студентов, поэтому мы ограничили цикл до 30 итераций. Затем сразу проверяем наличие студентов в группе, если такие есть, то мы прыгаем на итерацию где ячейка с ФИО студента пуста и начинаем оттуда. Теперь записываем наших студентов в столбец «А», это происходит в моменте где cell присваивает значение wb.Worksheets[GroopName].Cells[j,0], вот часть с Cells и есть наша текущая ячейка, где j – номер строчки, а 0 – номер стобца, в данном случае это столбец «А» в excel, и при помощи метода PutValue() вписываем ФИО студента в данную ячейку. Как я уже говорил выше, студентов может быть и меньше 30, мы решили, что если пользователь сделает лишний enter, то программа решит, что студентов на данную группу нет и вернет вас в меню. Строчка с методом DeleteBlankRows() нужна дабы удалить пустую строчку от ненаписанного студента, а иначе при показе и всех студентов у вас высветится пустая строка.

### 2.2.7. Состав группы / потока

Принцип такой же, как в пункте 2.2.5. только мы работаем не с листами, а с ячейками.

Листинг 2.15 – Состав группы и потока

Здесь в роли ограничителя цикла выступает количество заполненных строк в листе. В данном цикле рассматривается два случая, когда в листе больше 30 строк и меньше 30, или рассматривают список потока или группы соответственно. В отличии от групп, поток имеет ещё одну заполненную колонку, где вписаны кто к какой группе принадлежит студент.

### 2.2.8. Удаление студента из группы

В любом хорошем и не очень ВУЗе, всегда есть те, кто не хочет учится или не может, поэтому такие студенты идут отчислятся, став свободными людьми.  
 Удаление происходит в 2 этапа. Первый этап, вы должны посмотреть в пункт 2.2.7. и узнать номер у соответствующего студента. Второй этап, вписать этот номер в строчку, где просят этот номер, дабы удалить студента.  
 Для этого мы используем специальную функцию в классе Cells это метод DeleteRow(). В данный метод мы записываем номер нашего студента, но поскольку отчет строк начинается с нуля, а номера студентов с единицы, то мы вычитаем из номера эту единицу. В итоге мы получаем посередине пустую строку, если нам надо от этого избавится просто прописываем метод DeleteBlankRows(), которая удаляет пустые строки и смещает наверх не пустые строки.



Листинг 2.16 – удаления студента через его номер в списке

### 2.2.9. Создание потока

Приступ к интересной части этой работы. Поток состоит из студентов всех групп одного года вступления. В отличии от добавления студентов в группы, поток авто генерируется после того, как зададим его название и год, как это происходит. Сначала вводим название потока, к примеру поток ККСО или КРБО, а потом его год, это все конечно проверяется на корректность написания. Далее происходит поиск тех групп у кого совпадают с названием и годом потока, т.е. мы задали ККСО-20, программа будет искать те группы, у которых в названии начинается ККСО и заканчивается 20 годом. Когда программа найдет совпадение, то копирует ячейки группы в новый лист с названием потока, и продолжает дальше искать совпадение, и записывает дальше с того места где оканчивается запись прошлой группы, а также во вторую колонку пишет название группы к которой принадлежат студенты.



Листинг 2.17 – Создание листа потока

Как видно листингом выше название потока делается из 2 переменных, так сделано для упрощения поиска совпадения в дальнейшем. Здесь я хотел бы объяснится за строчки ToUpper() в листингах кода, это нужно ибо обычно записываются название в верхнем регистре, и да бы не было случая, где было маленькая буква а где большая, делать сразу все буквы в верхнем регистре.

Листинг 2.18 – Объединение групп в поток

Как я говорил ранее нам нужны определённые группы, которые начинаются и заканчиваются так же, как и название потока, следовательно, при помощи встроенных методов StartWith() и EndsWith() мы найдем такие соответствующие группы. Что делаем далее в столбец «А» мы записываем ФИО студентов, а в столбец «B» записываем группу этих студентов на каждой их строчке, после того как запишем первую группу мы запоминаем к на какой строчке мы остановились через переменную addcell и продолжаем с нее, когда найдем новую группу и потом также фиксируем строку. В итоге мы получим лист со всеми студентами из одного потока без пустых строчек.

### 2.2.10. Сортировка по ФИО или по группе

В Aspose.Cells есть встроенная сортировка вот только её надо настроить.



Листинг 2.19 – сортировка по ФИО

Сначала мы задаем ключ для DataSorted, этот ключ является какая колонка является ключевой, для листинга 2.19 ключевой колонкой является колонка «А», значит сортировать будет по ней, а каким образом сортировать это скажет строчка где SortOrder.Ascending, это значит что сортироватся от меньшего к большему, т.е. для ФИО сортировка будет идти по закону алфавита. Для того чтобы сортировать по группа нам достаточно изменить строчку с Key1



### 2.2.11. сохранение изменений в файл

Сохранение в файла в Aspose.Cells выглядит так:



Листинг 2.20 – Сохранение изменений

# 3. Заключение

В заключении хочу сказать, что это был прекрасный опыт работы с фреймворком. Благодаря нему я больше понял, как работают базы данных на примитивном уровне. В целом можно сказать что excel это тоже, в своем роде, СУБД, но только функциональнее за счет формул и разнообразных графиков, а фреймворк позволил сделать процесс заполнение таблиц удобнее и быстрее.

Литература

1. Исходник основного кода на GitHub (<https://github.com/reinbowARA/DataBaseStudents/tree/main> )

2. Документация по Aspose.Cells (<https://docs.aspose.com/cells/net/>)

3. Сайт о программировании METANIT.COM раздел про С# (<https://metanit.com/sharp/>)