Программа

Оглавление

[Принципы 3](#_Toc100080747)

[Установка приложения 3](#_Toc100080748)

[При каждом запуске программы 3](#_Toc100080749)

[Каждое изменение пользователя ведёт к обновлению базы данных и из неё обновление внутренней модели 3](#_Toc100080750)

[Все виды задач организовывать с помощью одного класса 3](#_Toc100080751)

[Структура базы данных, её проверки и отчистка 4](#_Toc100080752)

[Удаление, выполнение, отмена и т. д. проектов и задач тянет удаление, выполнение и отмену всех входящих сущностей 4](#_Toc100080753)

[Id тип string 4](#_Toc100080754)

[Использование хранимых процедур 5](#_Toc100080755)

[Сущности дел на сегодня зависят от просто дел 5](#_Toc100080756)

[Пересечение сфер и любых задач 5](#_Toc100080757)

[Группы сфер 6](#_Toc100080758)

[To do 6](#_Toc100080759)

[Функционал 6](#_Toc100080760)

[Технология 6](#_Toc100080761)

[Долгий ящик 8](#_Toc100080762)

[Учесть 8](#_Toc100080763)

[Функционал 8](#_Toc100080764)

[Проекты 8](#_Toc100080765)

[Создание 8](#_Toc100080766)

[Удаление 8](#_Toc100080767)

[Выполнение 9](#_Toc100080768)

[Задачи 9](#_Toc100080769)

[Создание 9](#_Toc100080770)

[Удаление 9](#_Toc100080771)

[Выполнение 9](#_Toc100080772)

[Подзадачи 9](#_Toc100080773)

[Планирование на сегодня 9](#_Toc100080774)

[Графическое представление 9](#_Toc100080775)

[При нажатии Enter во время редактирования название задачи и всё сохраняется и обновляется 10](#_Toc100080776)

[Принцип помидора, вроде 10](#_Toc100080777)

[Задачи на сегодня 10](#_Toc100080778)

[Дневник 10](#_Toc100080779)

[Цель 10](#_Toc100080780)

[Функционал 10](#_Toc100080781)

[Принципы 10](#_Toc100080782)

[Периодические задачи 11](#_Toc100080783)

[Список сфер 11](#_Toc100080784)

[Потребности 11](#_Toc100080785)

[Аналитика 11](#_Toc100080786)

[Люди 11](#_Toc100080787)

[Главная последовательность 11](#_Toc100080788)

# Принципы

## Установка приложения

Создать базу данных c именем «Имя программы.db» с таблицами tasks, hierarchy, planner. В таблицу tasks добавить первую строку с именем «root», индексом = 1, датой создания, описанием = «корневой проект».

## При каждом запуске программы

Проверять наличие базы данных.

## Каждое изменение пользователя ведёт к обновлению базы данных и из неё обновление внутренней модели

Можно для снижения вычислительных издержек менять модель изнутри без обращения к базе данных. Но это плохо. Присутствует дублирующий компонент. Вычислительные издержки не так велики в абсолютном выражении, чтобы вводить дополнительных механизмы. Пусть параметр записывается в базу данных и абсолютно вся информация снова загружается в модель из базы данных. Даже при обновлении модели из базы данных только где произошли изменения реализуется именно этот принцип.

Плюсы

* Снижается сложность работы программы
* Отсутствует дублирующий элемент
* Снижается трудоёмкость
* Отсутствует практическая потребность в этом. На моём количестве данных всё обновляется быстро. Незаметно для глаза.

Минусы

* Постоянная загрузка всей модели при каждом внесении изменении пользователем

Выводы

* Останавливаемся на загрузке из базы данных
* В том числе и при обновлении только части модели

## Все виды задач организовывать с помощью одного класса

Не выделять отдельный класс для Subtask, MyTask и Project. Всё организовать с помощью одного класса.

Преимущества

* Количество кода сильно сократиться
* Всё будет унифицировано
* При изменении процедур взаимодействия с базой данных все усовершенствования будут автоматически вносится во все классы
* и т.д.
* А ещё закладывается потенциал реализовать задачи бесконечной вложенности. Хоть концептуально я выступаю против этого.

Недостатки

* В каждой подзадачи будет пустое поле – список под-подзадач
  + Ну и что? Пускай.

## Структура базы данных, её проверки и отчистка

* В таблице tasks всегда должна быть первая строка root с индексом =1, датой создания, описанием = «корневой проект»
* В таблице иерархии отслеживать наличие повторов. Задача 1 принадлежит задаче 2.
* Root не может быть добавлен в список задач на сегодня

Структура базы данных и правильность её параметров в модели:

* Таблицы
* Из поля
* Хранимые процедуры
* Имена параметров хранимых процедур

Проверять на

* Все параметры в модели присутствуют в базе данных
* В базе данных нет параметров не отражённых в модели
* Все параметры в модели имеют правильные имена и другие свойства
* Т. е. параметры базы данных полностью соответствуют параметрам модели

При назначении задачи на сегодня проверять добавлена ли она уже. Если добавлена не добавлять.

* Задач на сегодня присвоена только один раз
* Задача присвоена проекту только один раз
* Подзадача присвоена задаче только один раз
* Соответствие размера текстовых полей в
  + Таблицах
  + Хранимых процедурах
  + Обычных запросах
  + Например, nvarchar(30) – имеется в виду «30»

Имена полей всех таблиц не повторяют встроенные названия типов. Date поле нельзя использовать, поскольку есть тип date. Один раз столкнулся.

## Удаление, выполнение, отмена и т. д. проектов и задач тянет удаление, выполнение и отмену всех входящих сущностей

При удалении проекта – удаляются все задачи и подзадачи.

При удалении задачи удаляются все подзадачи.

Для этого функция удаления задачи предусматривает удаление всех входящих подзадач.

То же самое для отметки «выполнено» у проекта и задачи. А также «отменено» и, возможно, другие аналогичные функции.

## Id тип string

Удобно. В базе данных он int. Во всех операциях в программе можно использовать string. При взаимодействии с базой данных только string.

Из базы данных получаю его в строковом виде (из DataTable). Внутри программы во всех операциях могу использовать и int, и string без разницы. При отправке в базу данных только string.

Таким образом, сэкономлю свои ресурсы на типе.

## Использование хранимых процедур

Существует несколько вариантов работы с базой данных:

* Запрос к базе данных
* Использование хранимых процедур
* Entity framework

Пока я рассмотрел первые два варианта.

И остановился на хранимых процедурах.

Преимущества

* Они уже скомпилированы
* Передаётся меньше информации
* При использовании хранимой процедуры требуется перебрать все возвращаемые строки в то время, как при запросе к базе данных сразу возвращается таблица. Однако потом таблица всё равно разбирается в цикле. Получается отсутствует дополнительный этап – получения таблицы. А также исключаются все действия с таблицей: инициализация DataSet, DataTable и их хранение и/или уничтожение.

Недостатки

* ?

## Сущности дел на сегодня зависят от просто дел

Не нужно организовывать дополнительный опрос базы данных для заполнения структур, отвечающих за хранение и одобрения всей информации на сегодня – все проекты на сегодня, задачи и подзадачи.

Они являются структурами, ссылающимися на модель.

Преимущества

* В результате имеем только одну сущность в одном экземпляре. Всё остальное – ссылки на эти сущности.
* Обновление происходит быстрее. Достаточно просто пробежаться по иерархическим спискам задач и составить новый. Вместо того, чтобы формировать заново все новые сущности.

Недостатки

* ?

## Пересечение сфер и любых задач

Некоторые проекты внутри разных сфер пересекаются и могут быть вписаны как в одну, так и в другую. На практике куда боль подходит, туда и следует отправлять. Но лучше добавить функционал – и туда и туда, чтобы было удобнее искать и работать.

Поэтому следует сделать функционал позволяющий включать задачу как подзадачу в несколько разных задач. И установить флажок уникальное поле у child в таблице иерархий в положение выключено.

## Группы сфер

Исследование показало существование 4-х групп сфер у меня. Следует использовать это.

Выделять цветом в списке дели в аналитике.

На специальной странице со сферами показывать группы и как они удовлетворены или выполнены.

# To do

## Функционал

* Возможность включать одну задачу в несколько сфер. Чтобы быстрее ориентироваться. Подумал – она должна относиться к этому проекту, взглянул и она там. В следующий раз подумал – эта же задача должна относиться к другому проекту. Взглянул – и она там.
  + Установить флажок уникальное поле у child в таблице иерархий в положение выключено
  + Отразить изменения в текстах формирующих таблицы в базе данных
* Периодические задачи
* Для дневника перенести сюда мои исследования как его вести
* База данных в интернете
* Продумать добавление проекта на сегодня
  + Либо все проекты отмечаются, либо только содержащие задачи на сегодня
  + Думаю, сделать и то и другое и менять в настройках
  + При этом поэкспериментировать как лучше
* Чтобы отмечалось, когда закрыл большое количество вкладок. Как дополнительное подчёркивание выполнения дела.
* Подзадачи
* Обновление всего на F5. И кнопка обновить всё. Либо лучше всего сделать авто обновление.

## Технология

* На пятницу
  + Добавить функционал
    - Добавление задачи
    - Удаление задачи
      * Проверить удаление на всех уровнях
    - Отметка о выполнении
      * Проверить выполнение на всех уровнях
    - Снятие выполнения
      * Проверить выполнение на всех уровнях
    - Добавление задачи на сегодня
      * Проверить добавление на всех уровнях
      * Сделать несколько вариантов
    - Удаление задачи из сегодня
      * Согласовать с принципами
    - Изменение названия
    - Изменение описания
  + Красота
    - Сделать более-менее красивое соотношение элементов интерфейса
    - Добавить поле задачи на сегодня
  + Организация
    - Переорганизовать дистрибутив на GitLab
* В таблице на сегодня отслеживать наличие повторов. Задача 1 уже добавлена.
* Пока проводится первый проход для установки значений – добавлено на сегодня.
  + Однако, согласно моим рассуждениям, требуется если подзадача настроена на сегодня, то добавлять и задачу, и проект на сегодня.
  + Поэтому сделать и обратный проход.
* Сделать классы для хранения имён параметров у хранимых процедур
* Сделать хранимые процедуры. Сначала в общем виде, потом для проектов, где id =1 всегда, это обстоятельство должно ускорить процедуру. Хотя стоит ли это делать? Не универсальность.
  + Создание новой задачи любого уровня
    - Отдельно создание нового проекта
  + Удаление задачи
    - Колеблюсь – сделать удаление всех входящих на уровне хранимых процедур или на уровне модели. Интересно как там с рекурсивными методами в sql? Если удастся организовать рекурсию – лучше сделать на уровне хранимых процедур.
    - Не имеет смысл отдельно делать процедуру для удаления проекта
  + Получение списка задач
    - Получение списка проектов
  + Назначение задачи на сегодня
  + Переименование задачи
    - Требуется часто
  + Простановка задачи выполненной
    - Требуется часто
  + Проверки базы данных
    - Из пункта принципов
* При создании задачи проверять существование надзадачи
* При создании задачи проверять присвоена ли она уже проекту. И подзадача задаче.
  + По идее не должно быть, поскольку id выдаётся сразу же после создания задачи.
* Добавить поле: дата создания задачи
* При создании задачи автоматом прописывать дату её создания
* Убрать поле: source
  + И вообще обновить поля
* Метод записи даты завершения работает. Однако, при запуске всё обновляется и автоматом перезаписывается. Исправить.
* Сделать, чтобы при записи даты выполнения задачи все подзадачи, которые не были выполнены либо отменялись, либо проставлялась даты их выполнения
  + Предлагать пользователю выбрать
* Сделать id типом string
* Все типы задач оформить в виде единого класса
* При создании новой задачи ищем последнюю строку в таблице задач для получения присвоенной ей id. Однако теоретически (на практике нет) возможно, что будет создана другая задача перед получением номера и вернётся другой номер. В связи с этим следует посмотреть может можно сделать хранимую процедуру, которая будет возвращать id созданной задачи. Или есть какое-то другое решение.
* Может вынести функцию добавить задачу в класс Менеджера
* При удалении проектов, задач и подзадач выводить подтверждающий вопрос – удалить проект/задачу/подзадачу?
* Сделать чтобы обновлялась только та часть модели (подгружалась из базы данных), в которой произошли изменения.
  + Если изменилась подзадача – изменения в подзадаче и задаче её содержащей. В задаче – в задаче и проекте. В проекте – в проекте и списке проектов.
* Вынести все взаимодействия с базой данных в один класс – получить все проекты. Получить задачи на сегодня. И так далее.
* Не настроена установка и первичная загрузка приложения. Согласно принципам установки приложения.
* Сделать структуру статическую или статический класс для названий всех таблиц в базе данных и вместо строкового представления названий таблиц в базе данных использовать поля структуры.
  + Это кроме прочего сделает простым проверку полей таблиц
* То же самое сделать с полями всех таблиц. И вместо строкового представления полей во всех функциях подставлять поля.
  + Это кроме прочего сделает простым проверку полей таблиц.

## Долгий ящик

# Учесть

* Вроде при чтении всё у меня перезаписывается дополнительно. Может это связано с тем, что я устанавливаю значения в поля класса задач и других классов и поскольку они установились, это вызывает сигнал на запись. Как только поменялись значения – я записываю их в базу данных.
  + Это получается при изменении каждого свойства выполняется execute в менеджере и всё загружается и перезаписывается ещё раз. Да уж.
* Мне немного не нравится, что в свойствах объект MyTask используется строковое представление поля. Может следует переделать на какое-то символьное
  + Может какой-нибудь параметр nameof имя поля
* При удалении строк задач проработать что будет если будет отсутствовать строка с информацией о задаче в таблицах иерархии и таблице задач
* Сделать функцию проверки наличия записей и соответствия таблиц иерархии и задач
* При удалении задачи, наверное, не стоит её удалять из планера – можно будет потом посмотреть статистику
* Если отжать кнопку «выполнено», то дата выполнения из базы данных не удалится
* Названия проектов могут быть одинаковыми. Тогда часть логики взаимодействия с базой данных ломается. Следует это учесть и обновить код.
  + Кроме прочего, может ли DataSet содержать таблицы с одинаковыми названиями?

# Функционал

## Проекты

### Создание

* создаётся строка в таблице задач
* создаётся строка в таблице иерархий

### Удаление

* удаляются все задачи, входящие в проект
  + поскольку вызывается команда удалить задачу, она автоматом удаляет все подзадачи
* удаляется задача в таблице задач
* удаляется строка в таблице иерархий

### Выполнение

* отмечаются выполненными все подзадачи всех задач
* отмечаются выполненными все задачи в него входящие
* отмечается выполненным проект

## Задачи

### Создание

* создаётся строка в таблице задач
* создаётся строка в таблице иерархий

### Удаление

* удаляются все подзадачи
* удаляется задача в таблице задач
* удаляется строка в таблице иерархии

### Выполнение

* отмечаются выполненными все подзадачи
* отмечается выполненной задача

## Подзадачи

Задачи содержат подзадачи.

Это удобно по следующим причинам (пока две причины)

* Иногда бывает удобно более подробное деление
  + Купить матрац
    - Найти в интернете удобных продавцов
    - Сходить полежать на нескольких матрацах
    - Отследить наличие. Заказать.
    - Получить доставку. Поднять. Установить матрац
* В других случаях бывает удобно отметить прогресс без какого-либо явного контрастного смысла между подзадачами.
  + Сегодня выполнял дело такое-то. Много сделал, но не всё. Ещё осталось много. Но сделал это, это и это.
  + Потом смотреть статистику прогресса.

### Планирование на сегодня

Думаю, правильно при назначении подзадачи на сегодня так же автоматом назначать и саму задачу на сегодня.

### Графическое представление

В общем списке дел.

Показываются все задачи в свёрнутом виде.

Имеется возможность развернуть подзадачи нажатием кнопки.

Хотя может сделать раскрытие списка подзадач те, которые были раскрыты при закрытии программы.

В делах на сегодня.

Думаю, лучше показывать задачу целиком – не только выводить подзадачу, но и всю задачу. Так более понятен масштаб и путь достижения задачи. При показе только подзадачи не виден масштаб проблемы.

Показываются в свёрнутом виде (изначально).

Если подзадача была запланирована на сегодня, то может сделать вынос этой подзадачи (этих подзадач) на плитку задачи.

## При нажатии Enter во время редактирования название задачи и всё сохраняется и обновляется

При нажатии Enter во время редактирования название задачи и всё сохраняется и обновляется

## Принцип помидора, вроде

20 минут работаешь потом проходишься.

Или как правильно? Изучить.

## Задачи на сегодня

Добавить в список задач на сегодня

Удалить из списка задач на сегодня

## Дневник

### Цель

* Сохранить воспоминания
* Улучшить что-то в себе

### Функционал

* Напоминание заполнить дневник
* Напоминание прочесть дневник
* Внесение информации
  + Текст
  + Фото
  + Видео
  + Так, чтобы и текст, и фото, и видео могли идти в любом порядке
  + Согласовать со структурой OneNote
* Просмотр информации
  + Это тянет за собой календарь

### Принципы

* На практике я не использую OneNote для хранения фото и видео. Только текст.
  + Я не отправляю на практике фотографии в OneNote ни откуда. Ни с телефона, ни с компьютера.
  + На практике я использую его только как записную книжку
  + В таком случае придётся только экспортировать заголовок записки и его текст

## Периодические задачи

## Список сфер

Может сделать отдельное окно для сфер. Сферы – это проекты в программе. И, думаю в проектах помимо сфер не может быть чего-то ещё. Если появляется какой-то проект, то включить его в одну из сфер.

И можно отображать на сколько они выполнены.

## Потребности

Степень удовлетворения потребностей. Их описание. Инструкции по удовлетворению.

## Аналитика

Основные показатели моей прекрасной жизни.

Можно включить раздел потребности, сферы и другие.

## Люди

Описание людей, окружающих меня. Начиная от формальных вещей – даты рождения, до тонкостей человека.

# Главная последовательность

* Получаем таблицы из базы данных
  + Таблица проектов
  + Таблицы задач для каждого проекта
* Формируем массив объектов проектов и задач
  + Связываем свойства объектов проектов и задач с соответствующими ячейками в таблицах
* Формируем массив элементов на форме, соответствующие спискам проектов и задач
  + Связываем двусторонней связью контролы со свойствами объектов проектов и задач