МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный политехнический университет»

КАФЕДРА ИНФОКОГНИТИВНЫХ ТЕХНОЛОГИЙ

КУРСОВОЙ ПРОЕКТ

по дисциплине

ИНЖЕНЕРНОЕ ПРОЕКТИРОВАНИЕ

Задание № 22

РЕЛЯЦИОННАЯ БАЗА ДАННЫХ «ЗАДВИЖКИ ТРУБОПРОВОДНОЙ АРМАТУРЫ»

Студент: Пересторонин А. М.

Группа: 201-324

Преподаватель: к.т.н. доцент Лянг В.Ф.

Москва – 2022

Задание 22

Разработать реляционную базу данных «Клапаны трубопроводной арматуры» (рис. 22.1 и 22.2) в системе управления базами данных Microsoft Access или на любом языке высокого уровня C#, C++ и т.д. в среде Visual Studio.

База данных должна содержать всю информацию, представленную в справочнике промышленного оборудования на страницах 102, 103, 100 ÷ 112, 134, 135, 140 ÷ 142 (файл «Справочник Пром об 1»).

База данных должна содержать информацию о марке, назначении, условиях эксплуатации, технических характеристиках, методах проектирования запорных клапанов (вентилей), представленных на рис. 22.1 и 22.2. Информация о запорных клапанах может быть расширена, например, за счет чертежей и методов расчета.

База данных должна быть расширена за счет клапанов аналогичных типов.

В БД должен быть обеспечен быстрый и удобный выбор клапанов по ряду параметров: марка, материал, номинальное давление, рабочая среда, температура окружающей среды, тип присоединения к трубопроводу, габаритные и присоединительные размеры, назначению, условиям эксплуатации. Параметры поиска могут быть расширены.

Создать комплекс форм, запросов, отчетов и меню, обеспечивающих удобную работу пользователя с базой данных. Обеспечить возможность дополнять, редактировать, удалять, сортировать, группировать данные, выполнять запросы, генерировать отчеты. Обеспечить защиту базы данных при изменении и добавлении данных.

Разработать полный комплект документов по эксплуатации базы данных в соответствии с Единой системой программной документации (ЕСПД).

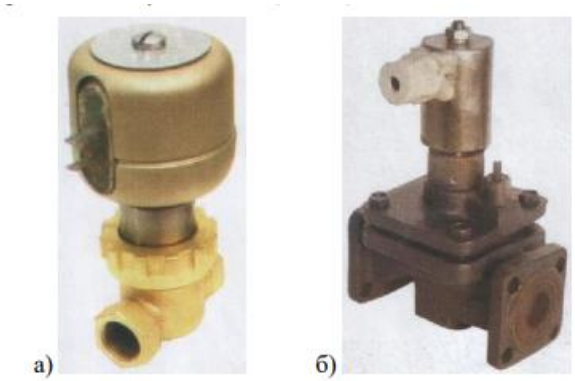


Рис. 22.1. Клапаны: а) — электромагнитный муфтовый латунный 15Б859п (П3.26291-010М); б) — мембранный с электромагнитным приводом фланцевый из ковкого чугуна типа СКМО ≈ ГО1 15кч835р и 15кч835р1

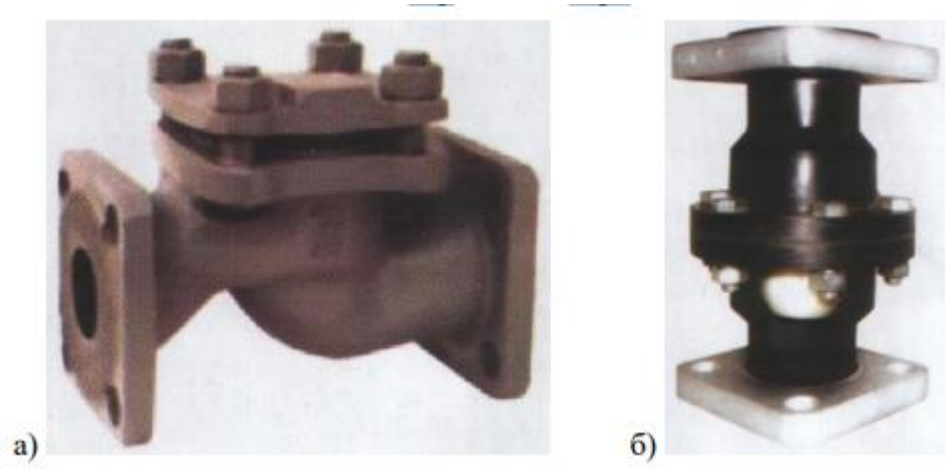


Рис. 27.2. Задвижки с невыдвижным шпинделем фланцевые чугунные: а) — с обрезиненным клином 30ч39р (МЗВГ); б) — параллельная 30ч3бр (МТР)

АННОТАЦИЯ

Трушин В.А. Реляционная база данных «Задвижки трубопроводной арматуры».

Кафедра Инфокогнитивных технологий, 2022 г. Пояснительная записка – 36 стр.; иллюстративные материалы – 25 плакатов; презентационная графика – 7 слайдов.

Реляционная база данных «Задвижки трубопроводной арматуры», содержит информацию о габаритах задвижки, химическом составе всех её деталей, физических и технологических свойствах. Функционирует в системе СУБД «Supabase», к которой можно подключаться с помощью веб-сайта. Предназначена для использования в автоматизированных системах проектирования во многих отраслях промышленности, в том числе на любых технологических и транспортных трубопроводах в системах жилищно-коммунального хозяйства, газо- и водоснабжения, нефтепроводах, объектах энергетики и многих других.

Изложены сведения о системах автоматизированного проектирования, а также о месте данного программного продукта в структуре САПР.

Описан процесс проектирования базы данных: постановка задачи, основные этапы проектирования, анализ данных, создание таблиц, запросов, форм с помощью высокоуровневого языка программирования JavaScript, а также языка разметки HTML.

Разработан полный комплект документации по эксплуатации базы данных в соответствии с ЕСПД: описание программного продукта, руководства системного программиста, администратора базы данных, пользователя.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью проекта является разработка реляционной базы данных «Задвижки трубопроводной » в системе виртуальных баз данных «Supabase». Визуальная часть реализована с использованием основного набора (стек) инструментов Web-проектирования. Серверная часть реализована на Node JS и выложена на хостинг «heroku» в связи отсутствия как таковой поддержки у «mospolytech».

Основные задачи:

1. Анализ данных таблиц.
2. Создание таблиц, запросов форм.
3. Создание панели инструментов и меню.
4. Обеспечение поиск используя несколько основных поисковых алгоритмов.
5. Автоматизация работы приложения с помощью алгоритмического языка JavaScript.

I. Постановка задачи

Выполняемая работа предназначена для описания имеющихся задвижек. В данной курсовой работе необходимо разработать реляционную базу данных, формирующую у пользователя представление о задвижках и их характеристиках. Внедрить развитую поисковую систему с плавающей точностью. Сделать визуализацию в виде приложения кроссплатформенного типа.

II. Метод решения задачи

Поставленные задачи решаются методами СУБД. Для обработки результатов запроса и вывода информации будет использоваться язык JavaScript. Для работоспособности базы данных потребуется аккаунт в СУБД «Supabase». Поиск будет производится прямыми запросами в базу данных или поиск по угловому коэффициенту вектора запроса.

Supabase — это альтернатива Firebase с открытым исходным кодом. Позволяет начать свой проект с базы данных, аутентификации, мгновенных API, подписок в реальном времени и хранилища. Она полностью бесплатна, обладает мощным, хорошо документированным и простым API.

Как заявлено на официальном сайте, проекты «Supabase» представляют из себя базы данных на основе «PostgreSQL».

Помимо «Supabase» существуют и другие базы данных, не использованные в проекте, но которые могут стать альтернативой в случае невозможности использовать Supabase:

- MySQL – это реляционная база данных с открытым исходным кодом, которая работает на различных платформах, таких как Windows, Linux, Mac OS и т. д. MySQL может использоваться для упакованного программного обеспечения, а также для критически важных для бизнеса систем и крупных веб-сайтов.
- PostgreSQL – это система управления базами данных с открытым исходным кодом корпоративного класса. Он поддерживает как SQL для реляционных, так и JSON для нереляционных запросов. Он поддерживается опытным сообществом разработчиков, которые внесли огромный вклад в создание высоконадежного программного обеспечения для управления базами данных. PostgreSQL позволяет создавать собственные типы данных и диапазон методов запросов. Вы можете запустить процедуру хранения на разных языках программирования.

- MongoDB – это документно-ориентированная база данных NoSQL, используемая для хранения больших объемов данных. Это база данных, которая появилась в середине 2000-х годов. Он подпадает под категорию базы данных NoSQL. MongoDB позволяет вам проверить документ. Он не подходит для приложений, имеющих сложные транзакции.
- OrientDB – это многомодельная база данных NoSQL с открытым исходным кодом, которая помогает организациям раскрыть возможности графических баз данных без развертывания нескольких систем для обработки других типов данных. Это поможет вам повысить производительность и безопасность при поддержке масштабируемости. OrientDB имеет возможность выполнять репликацию с несколькими мастерами, совместно использовать данные с использованием кластеров и автоматизировать распределенные запросы и транзакции.

III. Реализация задачи

Ниже описан процесс создания проекта – описание, проектирование, создание и настройка базы данных (глава 3.1), создание сервера, способного принимать запросы от людей и обращаться к базе данных (глава 3.2), а также отзывчивого интерфейса для пользователя (глава 3.3).

3.1 Реализация базы данных

В данной главе рассказаны и показаны типы данных и их подробная реализация.

3.1.1 Определение информационных потребностей базы данных

Первый этап проектирования заключается в определении информационных потребностей базы данных. Он включает в себя поиск информации и просмотр подобных баз, чтобы понять, сформулировать и задокументировать требования. Анализ методических материалов с целью нахождения зависимостей параметров.

3.1.2 Построение семантической модели

Семантическая модель (концептуальная модель, инфологическая модель) — модель предметной области, предназначенная для представления семантики предметной области на самом высоком уровне абстракции. Это означает, что устранена или минимизирована необходимость использовать понятия «низкого уровня», связанные со спецификой физического представления и хранения данных.

Данный этап проектирования базы данных включает в себя анализ объектов изделия и постановка образа базы данных в графическом виде.

Выделим следующие этапы:

1. Идентификация основных параметров изделия исходя из атласа.

База данных предназначена для ускорения и оптимизации поиска информации об изделии. Структурированной системе хранения и добавления информации исключая избыток.

2. Анализ параметров и зависимостей изделия с дальнейшей построением семантической модели базы данных.

Сущность представляет тип объектов, которые должны храниться в базе данных. Каждая таблица в базе данных должна представлять одну сущность. Как правило, сущности соответствуют объектам из реального мира. У каждой сущности определяют набор атрибутов.

Необходимо различить такие понятия, как тип сущности и экземпляр сущности.

Понятие тип сущности относится к набору однородных личностей, предметов, событий или идей, выступающих как целое. Сущности бывают трех классов; стержневые, ассоциативные, характеристические, а также подкласс ассоциативных сущностей — обозначения.

Связь — ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации базы данных - это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между ними определенные связи. Наличие такого множества связей и определяет сложность инфологических моделей.

Атрибут - наименованная характеристика сущности. Её наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым также для сущностей различного типа. Атрибуты используются для определения того, какая информация должна быть собрана о сущности.

Схема, представленная на рис. 1, показывает связь между основными таблицами.

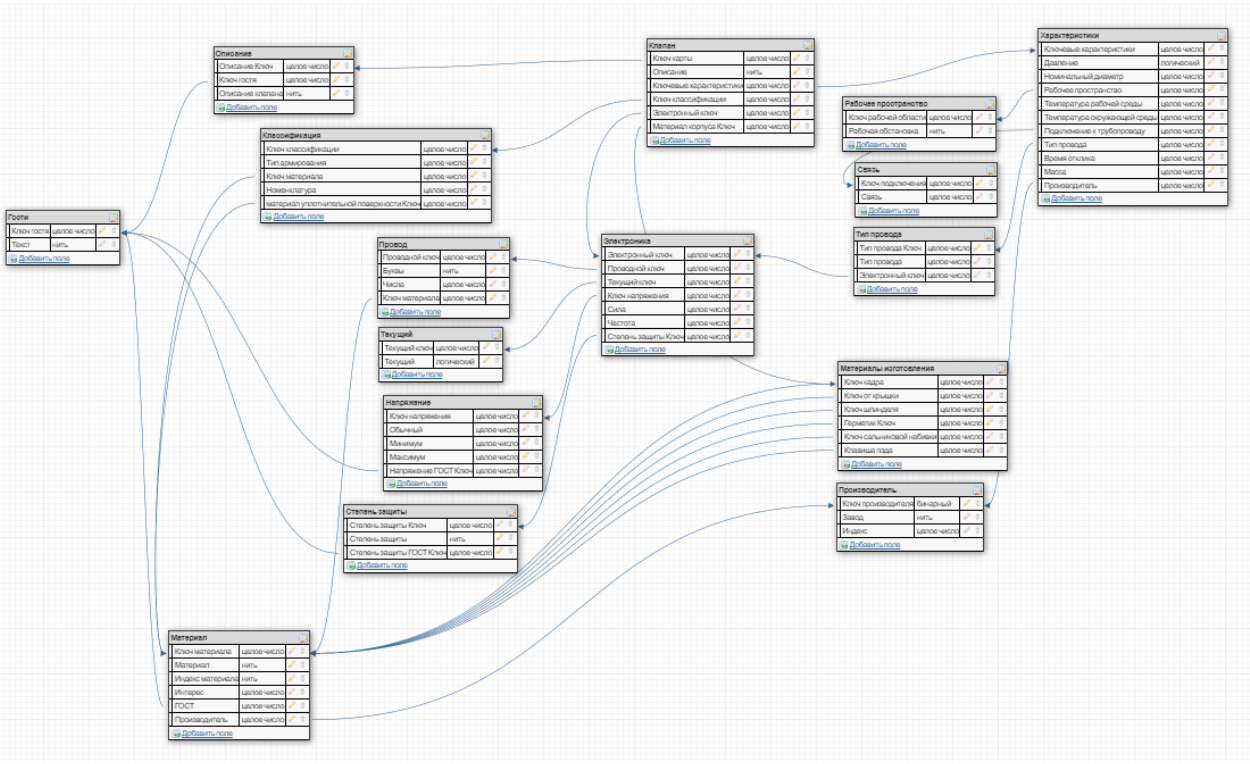


Рисунок 1 – Инфологическая модель базы данных «Задвижки трубопроводной арматуры»

3.1.3 Установление соответствий

Третий этап проектирования базы данных заключается в установлении соответствия между сущностями и характеристиками предметной области, отношениями и атрибутами в контексте выбранной СУБД. Для каждой таблицы необходимо подставить собственные атрибуты (параметры) исходя их реальных зависимостей семантического изделия.

Даталогическая модель — набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи. Преобразование концептуальной модели в логическую модель, как правило, осуществляется по формальным правилам.

Процедура проектирования даталогической модели состоит из следующих этапов:

1. Представить каждый стержень (независимую сущность) таблицей базы данных (базовой таблицей) и специфицировать первичный ключ этой базовой таблицы.

2. Представить каждое обозначение, которое не рассматривалось в предыдущем пункте, как базовую таблицу с внешним ключом, идентифицирующим обозначаемую сущность. Специфицировать, связанные с каждым таким внешним ключом ограничения.

3. Представить каждое свойство как поле в базовой таблице, представляющей сущность, которая непосредственно описывается этим свойством.

Сущности «Технические характеристики задвижки трубопроводной арматуры» будет соответствовать отношение, содержащее следующие поля:

- Код (*)
- Ключ;
- Описание;
- Ключевые характеристики;
- Классификация;
- Температура работы;
- и т.д.

3.1.4 Определение первичных ключей

На четвертом этапе проектирования базы данных определяются атрибуты, которые уникальным образом идентифицируют каждый объект. Это необходимо для того, чтобы система могла получить любую единичную строку таблицы.

В таблицах базы данных «Клапан» поле «Ключ карты» является первичным. В каждой таблице есть ключевое поле «Условное обозначение ключа».

3.1.5 Обеспечение целостности данных

Пятый этап предполагает создания требований и правил который будут создавать и поддерживать целостность базы данных. В серверной СУБД такие системы выполняются скриптами, а в пользовательских СУБД поддерживаются защитами добавления проверок на стороне промежуточного ПО.

Целостность базы данных — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется ограничением целостности.

Целостность базы данных складывается из двух элементов:

- Целостности данных
- Ссылочной целостности.

Для обеспечения целостности данных требуется, чтобы все первичные ключи были уникальными в пределах одной таблицы, а для поддержания ссылочной целостности необходимо, чтобы всем значениям внешних ключей соответствовали значения первичных ключей базовых таблиц.

Процесс нормализации позволяет обеспечить целостность данных и ссылочную целостность, а система управления базами обязана поддерживать эту целостность при вводе данных.

Отказ от обеспечения целостности базы данных может привести к появлению неправильных данных, а в худшем случае, к полному разрушению всей базы данных.

3.1.6. Нормализация и установление связей

На шестом этапе проектирования устанавливаются связи между объектами (таблицами и столбцами) и производится нормализация таблиц.

Основные типы связей, применяемых в реляционных базах данных, это:

- «Один-к-одному»
- «Один-ко-многим»

Связь «один-к-одному» — это связь между информацией из двух таблиц, когда каждая запись используется в каждой таблице только один раз. Например, связь типа "один-к-одному" может использоваться между сотрудниками и их служебными автомобилями.

Связь «один-ко-многим» — Это наиболее часто встречаемый тип связей. В этом типе связей несколько строк из дочерний таблицы зависят от одной строки в родительской таблице. Например, в одном блоге может быть несколько статей. В этом случае таблица блогов является родительской, а таблица статей - дочерней.

После того, как установлены связи между таблицами, необходимо провести нормализацию логических структур к базе данных.

3.1.7 Нормализация таблиц

Нормализацией логических структур называется представление логических структур в виде отношений.

Нормализацией также называется формальная процедура, в ходе которой атрибуты данных группируются в таблицы, а таблицы группируются в базы данных. Задачами нормализации являются:

- исключение повторяющейся информации в таблицах;
- создание структуры, в которой предусмотрена возможность ее будущих изменений;
- создание структуры, в которой влияние структурных изменений на приложения, использующие данные этой базы данных, сведено к минимуму.

Зачастую нормализация производится по 3NF , при этом четверная нормальная форма встречается редко, а последующие больше являются теоретическими.

Нормализация выполняется поэтапно.

3.1.7.1 Первая нормальная форма (1NF)

Переменная отношения находится в первой нормальной форме тогда и только тогда, когда в любом допустимом значении этой переменной каждый кортеж отношения содержит только одно значение для каждого из атрибутов.

В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение.

Что же касается различных таблиц, то они могут не быть правильными представлениями отношений и, соответственно, могут не находиться в 1НФ. В соответствии с определением Кристофера Дейта для такого случая таблица нормализована (эквивалентно — находится в первой нормальной форме) тогда и только тогда, когда она является прямым и верным представлением некоторого отношения. Конкретнее, рассматриваемая таблица должна удовлетворять следующим пяти условиям:

1. Нет упорядочивания строк сверху вниз (другими словами, порядок строк не несет в себе никакой информации).

2. Нет упорядочивания столбцов слева направо (другими словами, порядок столбцов не несет в себе никакой информации).

3. Нет повторяющихся строк.

4. Каждое пересечение строки и столбца содержит ровно одно значение из соответствующего домена (и больше ничего).

5. Все столбцы являются обычными.

«Обычность» всех столбцов таблицы означает, что в таблице нет «скрытых» компонентов, которые могут быть доступны только в вызове некоторого специального оператора взамен ссылок на имена регулярных столбцов, или которые приводят к побочным эффектам для строк или таблиц при вызове стандартных операторов. Таким образом, например, строки не имеют идентификаторов кроме обычных значений потенциальных ключей (без скрытых «идентификаторов строк» или «идентификаторов объектов»). Они также не имеют скрытых временных меток.

3.1.7.2 Вторая нормальная форма (2NF)

Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый неключевой атрибут неприводимо зависит от (каждого) её потенциального ключа.

Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость. Для неприводимой функциональной зависимости часто используется эквивалентное понятие «полная функциональная зависимость».

Если потенциальный ключ является простым, то есть состоит из единственного атрибута, то любая функциональная зависимость от него является неприводимой (полной). Если потенциальный ключ является составным, то, согласно определению второй нормальной формы, в отношении не должно быть неключевых атрибутов, зависящих от части составного потенциального ключа.

3.1.7.3 Третья нормальная форма (3NF)

Переменная отношения R находится в 3NF тогда и только тогда, когда выполняются следующие условия:

- R находится во второй нормальной форме.

- ни один неключевой атрибут R не находится в транзитивной функциональной зависимости от потенциального ключа R.

Пояснения к определению:

Неключевой атрибут отношения R — это атрибут, который не принадлежит ни одному из потенциальных ключей R.

Функциональная зависимость множества атрибутов Z от множества атрибутов X (записывается $X \to Z$, произносится «икс определяет зет») является транзитивной, если существует такое множество атрибутов Y, что $X \to Y$ и $Y \to Z$. При этом ни одно из множеств X, Y и Z не является подмножеством другого, то есть функциональные зависимости $X \to Z$, $X \to Y$ и $Y \to Z$ не являются тривиальными, а также отсутствует функциональная зависимость $Y \to X$.

Определение 3NF, эквивалентное определению Кодда, но по-другому сформулированное, дал Карло Заниоло в 1982 году. Согласно ему, переменная отношения находится в 3NF тогда и только тогда, когда для каждой из её функциональных зависимостей $X \to A$ выполняется хотя бы одно из следующих условий:

- X содержит A (то есть $X \to A$ — тривиальная функциональная зависимость)
- X — суперключ
- A — ключевой атрибут (то есть A входит в состав потенциального ключа).

Определение Заниоло чётко определяет разницу между 3NF и более строгой нормальной формой Бойса-Кодда (НФБК): НФБК исключает третье условие («A — ключевой атрибут»).

3.1.8 Базы данных. Математические модели, структура, определение

3.1.8.1 Иерархическая база данных, структура иерархических баз данных

Основными информационными единицами в иерархической модели данных являются сегмент и поле. Поле данных определяется как наименьшая неделимая единица данных, доступная пользователю. Для сегмента определяются тип сегмента и экземпляр сегмента. Экземпляр сегмента образуется из конкретных значений полей данных. Тип сегмента — это поименованная совокупность входящих в него типов полей данных.

Как и сетевая, иерархическая модель данных базируется на графовой форме построения данных, и на концептуальном уровне она является просто частным случаем сетевой модели данных. В иерархической модели данных вершине графа соответствует тип сегмента или просто сегмент, а дугам — типы связей предок — потомок. В иерархических структурах сегмент — потомок должен иметь в точности одного предка.

Иерархическая модель представляет собой связный неориентированный граф древовидной структуры, объединяющий сегменты. Иерархическая БД состоит из упорядоченного набора деревьев.

В рамках иерархической модели выделяют языковые средства описания данных (ЯОД) и средства манипулирования данными (ЯМД). Каждая физическая база описывается набором операторов, обусловливающих как её логическую структуру, так и структуру

хранения БД. При этом способ доступа устанавливает способ организации взаимосвязи физических записей.

Определены следующие способы доступа:

- иерархически последовательный;
- иерархически индексно-последовательный;
- иерархически прямой;
- иерархически индексно-прямой;
- индексный.

Помимо задания имени БД и способа доступа описания должны содержать определения типов сегментов, составляющих БД, в соответствии с иерархией, начиная с корневого сегмента. Каждая физическая БД содержит только один корневой сегмент, но в системе может быть несколько физических БД.

Среди операторов манипулирования данными можно выделить операторы поиска данных, операторы поиска данных с возможностью модификации, операторы модификации данных. Набор операций манипулирования данными в иерархической БД невелик, но вполне достаточен.

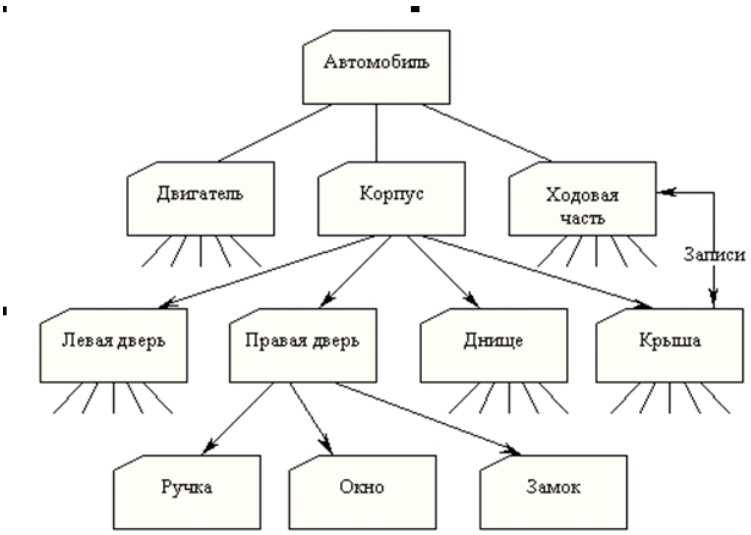


Рисунок 2 – Структура иерархической базы данных.

На рисунке можно увидеть структуру иерархической базы данных, в самом верху находится родитель или корневой элемент, ниже находятся дочерние элементы, элементы, находящиеся на одном уровне называются братьями, ну или соседними элементами. Соответственно чем ниже уровень элемента, тем вложенность этого элемента больше.

3.1.8.2 Сетевая база данных, структура сетевых баз данных

Основное достоинство сетевой модели – это высокая эффективность затрат памяти и оперативность. Недостаток – сложность и жесткость схемы базы, а также сложность понимания. Кроме того, в этой модели ослаблен контроль целостности, так как в ней допускается устанавливать произвольные связи между записями.

Сравнивая иерархические и сетевые базы данных, можно сказать следующее. В целом иерархические и сетевые модели обеспечивают достаточно быстрый доступ к данным. Но поскольку в сетевых базах основная структура представления информации имеет форму сети,в которой каждая вершина (узел) может иметь связь с любой другой, то данные в сетевой базе более равноправны, чем в иерархической, так как доступ к информации может быть осуществлен, начиная с любого узла.

Однако следует отметить жесткость организации данных в иерархических и сетевых моделях. Доступ к информации осуществляется только в соответствии со связями, определенными при проектировании структуры конкретной базы данных. Базы данных с такими моделями сложно реорганизовывать. Недостатком этих моделей является и сложность механизма доступа к данным, а также необходимость на физическом уровне четко определять связи данных. А поскольку каждый элемент данных должен содержать ссылки на некоторые другие элементы, то для этого требуются значительные ресурсы памяти ЭВМ. Кроме того, для таких моделей характерна сложность реализации систем управления базами данных.

Графовые (иерархические и сетевые) модели реализованы в качестве моделей данных в системах управления базами данных, работающих на больших ЭВМ. Для персональных компьютеров больше распространены реляционные базы данных, хотя имеются и системы управления базами данных, поддерживающих сетевую модель.

Для большей наглядности и понимания структуры сетевых баз данных обратите внимание на рисунке 3:

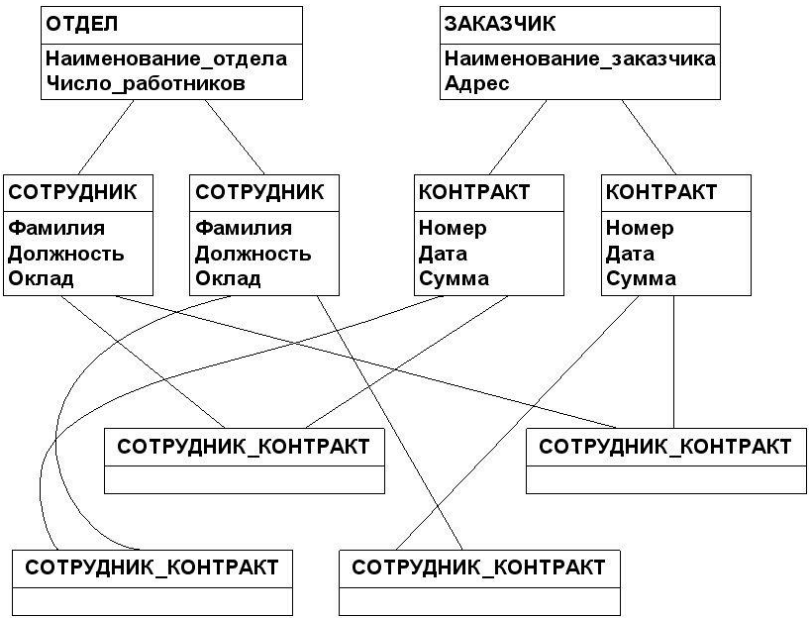


Рисунок 3 – Структура сетевых баз данных

Стоит заметить, что сетевые базы данных обладают примерно теми же характеристиками, что и иерархические базы данных.

3.1.8.3 Реляционные базы данных, структура реляционных баз данных

Реляционные базы данных основаны на реляционной модели — интуитивно понятном, наглядном табличном способе представления данных. Каждая строка,

содержащая в таблице такой базы данных, представляет собой запись с уникальным идентификатором, который называют ключом. Столбцы таблицы имеют атрибуты данных, а каждая запись обычно содержит значение для каждого атрибута, что дает возможность легко устанавливать взаимосвязь между элементами данных. Реляционная модель подразумевает логическую структуру данных: таблицы, представления и индексы. Логическая структура отличается от физической структуры хранения. Такое разделение дает возможность администраторам управлять физической системой хранения, не меняя данных, содержащихся в логической структуре. Например, изменение имени файла базы данных не повлияет на хранящиеся в нем таблицы.

Разделение между физическим и логическим уровнем распространяется в том числе на операции, которые представляют собой четко определенные действия с данными и структурами базы данных. Логические операции дают возможность приложениям определять требования к необходимому содержанию, в то время как физические операции определяют способ доступа к данным и выполнения задачи.

Чтобы обеспечить точность и доступность данных, в реляционных базах должны соблюдаться определенные правила целостности. Например, в правилах целостности можно запретить использование дубликатов строк в таблицах, чтобы устранить вероятность попадания неправильной информации в базу данных.

3.2 Реализация системы запросов

Запрос – обращение к базе данных с требованием вернуть определённые данные, или совершить определённое действие. Это может быть добавление записей, их изменение, удаление, а также запросы на получение списка записей (как всех, так и некоторых).

Существующее решение «Supabase» предлагает набор простых и удобных инструкций для языка JavaScript для отправки запросов. Каждая такая инструкция будет рассмотрена ниже.

Инженерный проект представляет из себя веб-сервер, работающий на определённом хосте и на определённом порту. Он реализован с помощью системы NodeJS и Express – универсальных решений на языке JavaScript, предоставляющих возможность быстро создать сервер, способный принимать различные запросы от клиента.

Следует заметить, что модули для NodeJS не приложены к пояснительной записке ввиду их большого размера и большого количества малых файлов, поэтому программист должен установить их самостоятельно непосредственно на сервере.

От клиента сервер будет лишь ожидать запросы (инструкции будут даны для клиента ниже, в главе 3.3). От системного администратора требуется иметь возможность разместить проект на сервере, поддерживающем NodeJS. Дальнейшие инструкции – опираясь на рисунок 4 и далее.

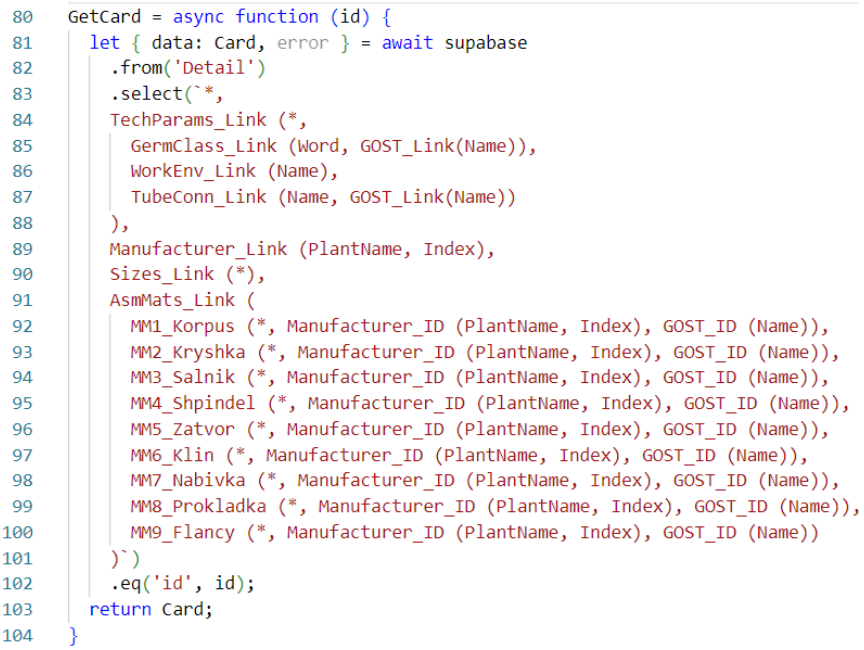
```
1  const express = require('express');
2  const app = express();
3
4  app.use('/main', express.static(__dirname + '/main'));
5
6  app.listen(3000, () => {
7    console.log('Сервер запущен по адресу http://localhost:3000');
8  });
9
10 app.get('/', (req, res) => {
11   res.sendFile(__dirname + '\\main\\index.html');
12 });
13
14 app.get('/card/:id', (req, res) => {
15   if (req.params.id == 'new')
16     res.sendFile(__dirname + '\\main\\new.html');
17   else
18     res.sendFile(__dirname + '\\main\\index_card.html');
19 });
```

Рисунок 4 – Часть реализации запросов

Программист импортирует Express для последующей реализации системы запросов. Затем он определяет папку со статическим контентом (иконки, доп. данные; строка 4). Он же запускает сервер на выбранном порту (строки 6-8; в данном случае выбран порт 3000).

Самый первый реализованный запрос – GET-запрос. Это запрос от пользователя, который пока что просто хочет загрузить главную страницу, где содержится список всех записей. Реализация главной странице будет указана ниже.

На рисунке 5 показана часть реализации сервера. Каждый раз, когда пользователь переходит на страницу карточки, он посылает простой GET-запрос на сервер. Сервер обращается к базе данных с запросом, показанным на рисунке 5, и возвращает всю информацию, которая есть, чтобы отобразить её.



```
80  GetCard = async function (id) {
81    let { data: Card, error } = await supabase
82      .from('Detail')
83      .select(`*,
84      TechParams_Link (*,
85        GermClass_Link (Word, GOST_Link(Name)),
86        WorkEnv_Link (Name),
87        TubeConn_Link (Name, GOST_Link(Name))
88      ),
89      Manufacturer_Link (PlantName, Index),
90      Sizes_Link (*),
91      AsmMats_Link (
92        MM1_Korpus (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
93        MM2_Kryshka (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
94        MM3_Salnik (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
95        MM4_Shpindel (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
96        MM5_Zatvor (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
97        MM6_Klin (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
98        MM7_Nabivka (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
99        MM8_Prokladka (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
100       MM9_Flancy (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name))
101     )`)
102     .eq('id', id);
103   return Card;
104 }
```

Рисунок 5 – Запрос на карточку

На рисунке 6 представлена обработка запросов, связанных с показом страницы. Например, префикс «/find» означает, что когда пользователь вбивает этот адрес в поиск, то

сервер обязан перенаправить его на страницу поиска карточки. Таким же образом работают и все остальные запросы.

В некоторых случаях сервер посылает запрос к базе данных (например, выполняя функцию «AddCard»). В таком случае сервер возвращает не веб-страницу, а данные, такие как статус выполнения (он скрыт от пользователя, это техническая информация), и данные карточки.

```
app.get('/find', (req, res) => {
  res.sendFile(__dirname + '\\main\\find.html');
});

app.get('/card_edit/:id', (req, res) => {
  res.sendFile(__dirname + '\\main\\edit.html');
});

app.get('/all_data', (req, res) => {
  let r = GetAllTheData();
  r.then(res_main => { res.send(JSON.stringify(res_main)) });
});

app.get('/card_fe/:id', (req, res) => {
  let r = GetCardForEdit(req.params.id);
  r.then(res_main => res.send(`{"data":` + JSON.stringify(res_main) + "}"));
});

const jsonParser = express.json();
app.post('/card/:id', jsonParser, (req, res) => {
  if (req.params.id == 'new') {
    let r = AddCard(req.body);
    r.then(res_main => res.send(`{"status":"0"}`));
  }
  else {
    if (req.params.id == 'find') {
      //console.log(req.body);
      let r = FindAllCards(req.body);
      r.then(res_main => res.send(`{"data":` + JSON.stringify(res_main) + "}"));
    }
    else {
      let r = GetCard(req.params.id);
      r.then(res_main => res.send(`{"data":` + JSON.stringify(res_main) + "}"));
    }
  }
});
```

Рисунок 6 – Запросы пользователя

На рисунке 7 показан сам процесс установки соединения с базой данных. Сервер загружает для себя модуль «supabase», и затем подключается к ней с помощью URL-адреса и API-ключа. API-ключ – длинная строка, обеспечивающая доступ к базе данных, является коммерческой тайной для организацией. Здесь он полностью не показан.

```
app.put('/card/:id', jsonParser, (req, res) => {
  let r = ChangeCard(req.body);
  r.then(res_main => res.send(`{"status":"0"}`));
});

app.delete('/card/:id', (req, res) => {
  let r = DeleteCard(req.params.id);
  r.then(res_main => res.send(`{"status":"0"}`));
});

supb = require('@supabase/supabase-js');

const supabaseUrl = 'https://yiyqsrpsalvozpvjsfxm.supabase.co'
const supabaseKey = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJp
const supabase = supb.createClient(supabaseUrl, supabaseKey)
```

Рисунок 7 – подключение к базе данных

3.2.1 Запрос на все записи

Когда пользователь открывает главную страницу, она автоматически посылает запрос на сервер с требованием предоставить краткую информацию обо всех карточках. Как работает сервер, было указано выше.

Когда данные возвращаются к пользователю, система автоматически формирует список, добавляя в код веб-страницы отдельные участки, составляющие карточку, на которой показан текст, описание и ссылка на карточку. Процесс показан на рисунке 8.

```
<main class="wrapper">
    <div class="wrap" id="CardsList">
    </div>
    <script>
        let list = document.getElementById("CardsList");
        let r = fetch('http://localhost:3000/all');
        let rr = r.then((res) => { console.log(res); return res.json(); });
        rr.then((data) => {
            console.log(data);
            console.log(data.data.length);
            for (let i = 0; i < data.data.length; i++)
                list.insertAdjacentHTML('beforeend',
                    `
        <div class="card">
            <div class="card-pic-wrap">
                <img src="/main/Cards/1.png" alt="A leafy plant">
            </div>
            <div class="card-content">
                <h3>` + data.data[i].Label + `</h3>
                <p class="truncate-text">` + data.data[i].Desc + `</p>
                <p><a href="http://localhost:3000/card/` + data.data[i].id + `">Посмотреть</a></p>
            </div>
        </div>
                    `
                );
        });
    </script>
</main>
```

Рисунок 8 – Создание списка карточек

На рисунке 9 показан запрос к базе данных на получение информации. «Card» – возвращаемая пользователю структура со списком всех карточек.

```
GetCards = async function () {
  let { data: Card, error } = await supabase
    .from('Detail')
    .select(`id, Label, Desc`)
    .eq('Type', 0);
  return Card;
}

Nullor = function (str){
  return str == "" ? null : str;
}
```

Рисунок 9 – Запрос на все карточки

3.2.2 Запрос на добавление записи

На странице пользователя есть поля для ввода данных. Каждое из них обладает уникальным идентификатором (как на рис. 10). Когда пользователь нажимает кнопку «Добавить», формируется запрос, в который добавляются все данные (в том числе из тех полей, которые пользователь решил проигнорировать). Этот запрос посылается на сервер (рис. 11). Сервер уже обращается к базе данных (рис. 12), добавляя в неё все данные.

```
</tr>
<tr>
    <td class="text-left">Набивка</td>
    <td class="text-left">
        <select id="MM7" name="MM7">
            <option disabled value="null" selected>Выберите материал</option>
        </select>
    </td>
</tr>
<tr>
    <td class="text-left">Прокладка</td>
    <td class="text-left">
        <select id="MM8" name="MM8">
            <option disabled value="null" selected>Выберите материал</option>
        </select>
    </td>
</tr>
<tr>
    <td class="text-left">Фланцы</td>
    <td class="text-left">
        <select id="MM9" name="MM9">
            <option disabled value="null" selected>Выберите материал</option>
        </select>
    </td>
</tr>
```

Рисунок 10 – Пример полей для выбора данных

```
    Tmax: document.getElementById("Tmax").value,
    Germ: document.getElementById("Germ").value,
    Work: document.getElementById("Work").value,
    Conn: document.getElementById("Conn").value,
    DN: document.getElementById("DN").value,
    D: document.getElementById("D").value,
    D1: document.getElementById("D1").value,
    D2: document.getElementById("D2").value,
    n: document.getElementById("n").value,
    d: document.getElementById("d").value,
    D0: document.getElementById("D0").value,
    L: document.getElementById("L").value,
    H: document.getElementById("H").value,
    H1: document.getElementById("H1").value,
    MM1: document.getElementById("MM1").value,
    MM2: document.getElementById("MM2").value,
    MM3: document.getElementById("MM3").value,
    MM4: document.getElementById("MM4").value,
    MM5: document.getElementById("MM5").value,
    MM6: document.getElementById("MM6").value,
    MM7: document.getElementById("MM7").value,
    MM8: document.getElementById("MM8").value,
    MM9: document.getElementById("MM9").value
});
let request = new XMLHttpRequest();
request.open("post", 'http://localhost:3000/card/new', true);
request.setRequestHeader("Content-Type", "application/json");
request.addEventListener("load", function () {
    location.href = "/";
});
request.send(user);
```

Рисунок 11 – Отправка данных

Причём данные сначала добавляются в дополнительные таблицы, и лишь потом – в основную. Это связано с тем, что база данных самостоятельно выбирает первичный ключ, а он становится известен лишь после добавления записи. В основной таблице есть ссылки, в которых нужны значения ключей, поэтому основная таблица заполняется последней. Это хорошо видно на рисунке 12.

```
251     console.log("ответ 2: " + JSON.stringify(resp2) + "\n");
252     let resp3 = await supabase.from('AssembleMats').insert([{
253       MM1_Korpus: (params.MM1 != "null" ? params.MM1 : null),
254       MM2_Kryshka: (params.MM2 != "null" ? params.MM2 : null),
255       MM3_Salnik: (params.MM3 != "null" ? params.MM3 : null),
256       MM4_Shpindel: (params.MM4 != "null" ? params.MM4 : null),
257       MM5_Zatvor: (params.MM5 != "null" ? params.MM5 : null),
258       MM6_Klin: (params.MM6 != "null" ? params.MM6 : null),
259       MM7_Nabivka: (params.MM7 != "null" ? params.MM7 : null),
260       MM8_Prokladka: (params.MM8 != "null" ? params.MM8 : null),
261       MM9_Flancy: (params.MM9 != "null" ? params.MM9 : null),
262     }]);
263     console.log("ответ 3: " + JSON.stringify(resp3) + "\n");
264     await supabase.from('Detail').insert([{
265       Label: Nullor(params.NameL),
266       Desc: Nullor(params.Desc),
267       OKP: Nullor(params.OKP),
268       Manufacturer_Link: Nullor(params.Manufact),
269       mass: Nullor(params.Mass),
270       TechParams_Link: resp1.data != null ? resp1.data[0].id : null,
271       Sizes_Link: resp2.data != null ? resp2.data[0].id : null,
272       AsmMats_Link: resp3.data != null ? resp3.data[0].id : null,
273       Type: 0
274     }])
275   }
```

Рисунок 12 – Добавление в базу данных

Важное замечание. Когда формируется страница поиска/добавления/редактирования, на сервер посылается запрос (рис. 13) с требованием предоставить всю информацию о ГОСТах, средах выполнения, типах соединения, материалах, и прочем. Эта информация будет занесена в выпадающие списки.

```
let r_all = fetch('http://localhost:3000/all_data', {
    method: 'GET'
});
let rr_all = r_all.then((res) => { console.log(res); return res.json(); });
rr_all.then((data_all) => {
    console.log(data_all);
    for (let i = 0; i < data_all.germ.length; i++) {
        document.getElementById("Germ").insertAdjacentHTML('beforeend', `<option value="` + d
    }
    for (let i = 0; i < data_all.work.length; i++) {
        document.getElementById("Work").insertAdjacentHTML('beforeend', `<option value="` + d
    }
    for (let i = 0; i < data_all.tube.length; i++) {
        document.getElementById("Conn").insertAdjacentHTML('beforeend', `<option value="` + d
    }
    for (let i = 0; i < data_all.manuf.length; i++) {
        document.getElementById("Manufact").insertAdjacentHTML('beforeend', `<option value="`
    }
    for (let i = 0; i < data_all.mats.length; i++) {
        for (let j = 1; j < 10; j++) {
            document.getElementById("MM" + j).insertAdjacentHTML('beforeend', `<option value=
        }
    }
```

Рисунок 13 – Получение информации

На рисунке 14 показан сам запрос в базу данных, и формирование ответа для пользователя.

```
GetAllTheData = async function () {
  let { data: GermClasses, error } = await supabase
    .from('GermClasses')
    .select('*, GOST_Link(Name)');
  let { data: Mats, error2 } = await supabase
    .from('Materials')
    .select('*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)');
  let { data: WorkEnv, error3 } = await supabase
    .from('WorkEnviroments')
    .select('*');
  let { data: TubeConn, error4 } = await supabase
    .from('TubeConn')
    .select('*, GOST_Link(Name)');
  let { data: Manuf, error5 } = await supabase
    .from('Manufacturer')
    .select('*');
  return { germ: GermClasses, mats: Mats, work: WorkEnv, tube: TubeConn, manuf: Manuf };
}
```

Рисунок 14 – Сбор статичных данных

3.2.3 Запрос на изменение записи

Похожим (на добавление и просмотр) образом работает изменение, но есть различия. На рисунке 15 показан запрос к базе данных. В отличие от просмотра карточки, здесь возвращаются лишь первичные ключи, без каких-либо имён. Имена возвращаются в качестве информации, что было показано на рисунке 14.

```
GetCardForEdit = async function (id) {
  let { data: Card, error } = await supabase
    .from('Detail')
    .select(`*,
    TechParams_Link (*),
    Manufacturer_Link,
    Sizes_Link (*),
    AsmMats_Link (
      MM1_Korpus,
      MM2_Kryshka,
      MM3_Salnik,
      MM4_Shpindel,
      MM5_Zatvor,
      MM6_Klin,
      MM7_Nabivka,
      MM8_Prokladka,
      MM9_Flancy
    )`)
    .eq('id', id);
  return Card;
}
```

Рисунок 15 – Запрос на изменение карточки

Когда пользователь подтверждает изменение, он точно так же нажимает на «Подтвердить», и точно так же формируется и посылается запрос, как было описано в добавлении. Разница лишь в том, что в данном случае основная таблица заполняется первой, так как она возвращает отчёт, в котором содержатся ссылки на другие таблицы, которые тоже надо изменить, но которые были не доступны без изменения основной.

На рисунке 16 показан сам запрос к базе данных и изменение данных. Функция «Nullor» смотрит, является ли полученный от пользователя параметра пустым, и если да, то переводит пустую строку в значение Null, которое будет правильно интерпретировано базой данных.

```
ChangeCard = async function (params) {
  let resp = await supabase.from('Detail').update({
    Label: Nullor(params.NameL),
    Desc: Nullor(params.Desc),
    OKP: Nullor(params.OKP),
    Manufacturer_Link: Nullor(params.Manufact),
    mass: Nullor(params.Mass)
  }).eq('id', params.id);
  await supabase.from('TechParams').update({
    Pressure: Nullor(params.Prss),
    Tmin: Nullor(params.Tmin),
    Tnorm: Nullor(params.Tnorm),
    Tmax: Nullor(params.Tmax),
    GermClass_Link: Nullor(params.Germ),
    WorkEnv_Link: Nullor(params.Work),
    TubeConn_Link: Nullor(params.Conn)
  }).eq('id', resp.data[0].TechParams_Link);
  await supabase.from('Sizes').update({
    DN: Nullor(params.DN),
    D: Nullor(params.D),
    D1: Nullor(params.D1),
    D2: Nullor(params.D2),
    n: Nullor(params.n),
    d: Nullor(params.d),
    D0: Nullor(params.D0),
    L: Nullor(params.L),
    H: Nullor(params.H),
    H1: Nullor(params.H1)
  }).eq('id', resp.data[0].Sizes_Link);
  await supabase.from('AssembleMats').update({
    MM1_Korpus: (params.MM1 != "null" ? params.MM1 : null),
    MM2_Kryshka: (params.MM2 != "null" ? params.MM2 : null),
    MM3_Salnik: (params.MM3 != "null" ? params.MM3 : null),
    MM4_Shpindel: (params.MM4 != "null" ? params.MM4 : null),
    MM5_Zatvor: (params.MM5 != "null" ? params.MM5 : null),
    MM6_Klin: (params.MM6 != "null" ? params.MM6 : null),
    MM7_Nabivka: (params.MM7 != "null" ? params.MM7 : null),
```

Рисунок 16 – Запрос на изменение

3.2.4 Запрос на удаление записи

Запрос на удаление – самый простой из всех реализованных. Пользователь только посылает на сервер номер записи, которую следует удалить, и система удаляет эту запись, а также все те, которые имеют к ней непосредственное отношение. Команда представлена на рисунке 17.

```
DeleteCard = async function (id) {
  let resp = await supabase.from('Detail').delete().eq('id', id);
  await supabase.from('TechParams').delete().eq('id', resp.data[0].TechParams_Link);
  await supabase.from('Sizes').delete().eq('id', resp.data[0].Sizes_Link);
  await supabase.from('AssembleMats').delete().eq('id', resp.data[0].AsmMats_Link);
}
```

Рисунок 17 – Удаление записи

3.2.5 Выбор записи с помощью фильтра

Когда пользователь ищет детали с помощью фильтров, то на странице клиента подготавливается строка в формате JSON, собирающая все введённые пользователем значения, и отправляет их на сервер (рис. 18).

На самом сервере существует процедура «FindAllCards» (рис. 19), которая создаёт ответ для клиента (тоже в формате JSON). В этом ответе содержится список всех деталей,

которые удовлетворяли запросу клиента. Чтобы отфильтровать базу данных, используется функция «filter» для массивов, удаляющая все неподходящие записи.

```
    DN_1: document.getElementById("DN_1").value,
    D_1: document.getElementById("D_1").value,
    D1_1: document.getElementById("D1_1").value,
    D2_1: document.getElementById("D2_1").value,
    n1: document.getElementById("n1").value,
    d1: document.getElementById("d1").value,
    D0_1: document.getElementById("D0_1").value,
    L1: document.getElementById("L1").value,
    H1: document.getElementById("H1").value,
    H11: document.getElementById("H11").value,
    DN_2: document.getElementById("DN_2").value,
    D_2: document.getElementById("D_2").value,
    D1_2: document.getElementById("D1_2").value,
    D2_2: document.getElementById("D2_2").value,
    n2: document.getElementById("n2").value,
    d2: document.getElementById("d2").value,
    D0_2: document.getElementById("D0_2").value,
    L2: document.getElementById("L2").value,
    H2: document.getElementById("H2").value,
    H12: document.getElementById("H12").value,
    MM1: document.getElementById("MM1").value,
    MM2: document.getElementById("MM2").value,
    MM3: document.getElementById("MM3").value,
    MM4: document.getElementById("MM4").value,
    MM5: document.getElementById("MM5").value,
    MM6: document.getElementById("MM6").value,
    MM7: document.getElementById("MM7").value,
    MM8: document.getElementById("MM8").value,
    MM9: document.getElementById("MM9").value
});
let request = new XMLHttpRequest();
request.open("post", 'http://localhost:3000/card/find', true);
request.setRequestHeader("Content-Type", "application/json");
request.addEventListener("load", function () {
    let data = JSON.parse(request.response);
    console.log(data);
```

Рисунок 18 – Формирование запроса

```
106 FindAllCards = async function (params) {
107   const { data: Result, error } = await supabase
108     .from('Detail')
109     .select(`*,
110   TechParams_Link (*),
111   Manufacturer_Link,
112   Sizes_Link (*),
113   AsmMats_Link (*)
114   `)
115   .eq('Type', 0);
116   let res1 = Result;
117   let nm = params.NameL.toLowerCase();
118   let ds = params.Desc.toLowerCase();
119   console.log(params);
120   if (params.NameL != "") res1 = res1.filter(x => (x.Label.toLowerCase().indexOf(nm) >= 0));
121   console.log(res1);
122   if (params.Desc != "") res1 = res1.filter(x => (x.Desc.toLowerCase().indexOf(ds) >= 0));
123   if (params.OKP != "") res1 = res1.filter(x => x.OKP == params.OKP);
124   if (params.Manufact != "") res1 = res1.filter(x => x.Manufacturer_Link == params.Manufact);
125   if (params.Mass1 != "") res1 = res1.filter(x => x.mass >= params.Mass1);
126   if (params.Mass2 != "") res1 = res1.filter(x => x.mass <= params.Mass2);
127
128   if (params.Prss1 != "") res1 = res1.filter(x => x.TechParams_Link.Pressure >= params.Prss1);
129   if (params.Prss2 != "") res1 = res1.filter(x => x.TechParams_Link.Pressure <= params.Prss2);
130   if (params.Tmin1 != "") res1 = res1.filter(x => x.TechParams_Link.Tmin >= params.Tmin1);
131   if (params.Tmin2 != "") res1 = res1.filter(x => x.TechParams_Link.Tmin <= params.Tmin2);
132   if (params.Tnorm1 != "") res1 = res1.filter(x => x.TechParams_Link.Tnorm >= params.Tnorm1);
133   if (params.Tnorm2 != "") res1 = res1.filter(x => x.TechParams_Link.Tnorm <= params.Tnorm2);
134   if (params.Tmax1 != "") res1 = res1.filter(x => x.TechParams_Link.Tmax >= params.Tmax1);
135   if (params.Tmax2 != "") res1 = res1.filter(x => x.TechParams_Link.Tmax <= params.Tmax2);
```

(parameter) x: any

Рисунок 19 – Фильтрация

3.3 Реализация интерфейса

Интерфейс пользователя был реализован, как веб-сайт. При создании сайта использовался язык разметки HTML, а также язык программирования JavaScript.

Переход между формами основан на ссылках. Пользователь нажимает определённую кнопку, и веб-сайт перемещает его на определённую страницу.

Интерфейс позволяет пользователю более удобно взаимодействовать с базой данных.

3.3.1 Главная форма

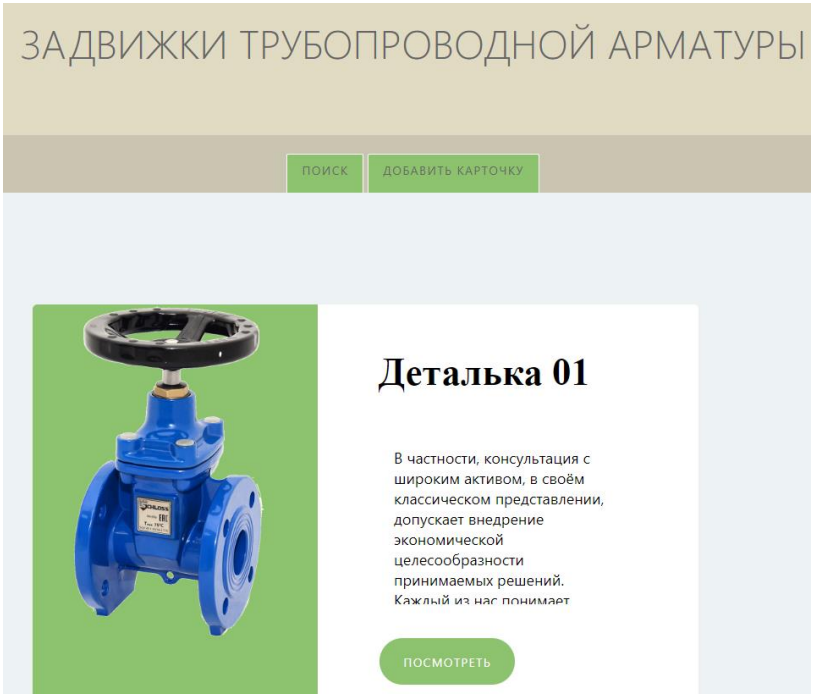


Рисунок 20 – Главная страница

На главной странице сайт помещает список всех элементов, которые существуют в базе данных – без какого-либо фильтра. Каждая такая карточка (на рис. 20) показывает название задвижки, её описание, рисунок, а также кнопку с возможностью перейти на страницу задвижки для получения вообще всей информации. Форма (или страница) просмотра будет рассмотрена в главе 3.3.4, а формы поиска и добавления карточки будут рассмотрены в следующих же главах (3.3.2 и 3.3.3 соответственно).

3.3.2 Форма поиска

На странице поиска у пользователя есть возможность указать все возможные значения, которые понадобятся для поиска. Пользователь может искать по названию или описанию (и тогда система будет искать совпадения по именам), также он может указать диапазон для числовых значений (например, на рисунке 21 видна возможность указать диапазон возможных значений для общей массы детали), и тогда система отфильтрует все найденные элементы. Чтобы запустить поиск, нажмите кнопку «Подтвердить». Кнопка «Назад» переводит пользователя обратно на главную страницу.

Все найденные элементы появятся в самом низу страницы, в точно таком же стиле, в каком они показаны на главной странице.

Для удобного поиска нужных разделов поиска слева есть удобное меню навигации, которое позволяет сразу спуститься к нужным параметрам. Такое меню присутствует на всех страницах, кроме главной.

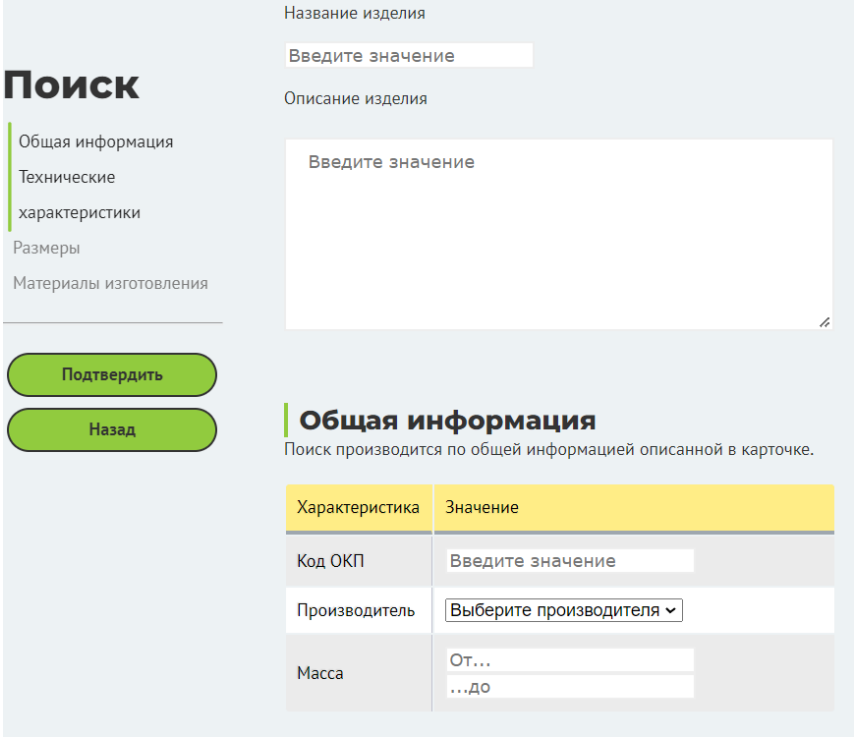


Рисунок 21 – Страница поиска

Система поиска отправляет запрос на сервер. Весь процесс был подробно описан в главе 3.2.5.

3.3.3 Форма добавления

На рисунке 22 показан процесс добавления карточки. Сайт загружает всю информацию, что есть в базе данных, и процесс добавления разблокируется. Пользователь может указать нужные данные, а также выбрать из доступных (например, среды работы, материалы и типы соединений определены заранее, и их нельзя вписать в качестве текста). Числовые значения ограничены системой сайта, их минимальное значение – 0 (не может быть отрицательной массы/габаритов/проч.).

После ввода всех данных, пользователь нажимает на кнопку «Подтвердить», и данные отправляются на сервер. Их сохранение там описано в главе 3.2.2.

Пользователь может оставить некоторые поля пустыми, и тогда при просмотре они будут помечены, как «…», или поле будет пустое. Ответственность за заполнение всех параметров лежит на пользователе.

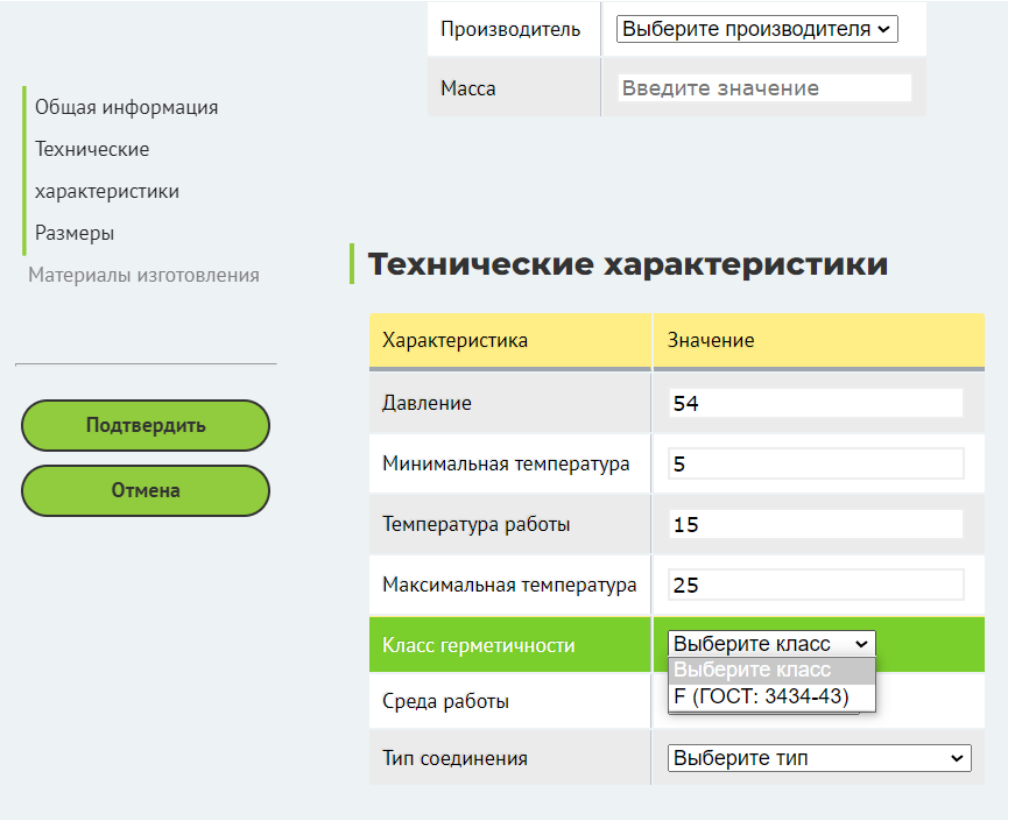


Рисунок 22 – Добавление карточки

3.3.4 Форма просмотра

На странице просмотра (рис. 23), как и на странице поиска и добавления, представлены все категории – общая информация, габариты, материалы и технические характеристики. Пользователь с помощью навигации может быстро к ним перемещаться.

Каждая категория содержит таблицу с данными, которые задавал пользователь. Все эти данные пользователь может изменять по нажатию на кнопку «Редактировать».

Таблицы сопровождаются рисунками. Например, в категории «Размеры» есть взятый из учебного материала габаритный чертёж задвижки, он здесь нужен для помощи пользователю.

Также над таблицами может быть краткая справка. Например, пояснение номинального давления и кода ОКП.

Рисунок 23 – Просмотр карточки

Пользователь может удалить карточку по желанию. В таком случае формируется запрос на удаление, который был описан в главе 3.2.4, а пользователь перемещается на главную страницу.

3.3.5 Форма редактирования

Пользователь имеет возможность редактировать карточку. В таком случае пользователь переходит на страницу редактирования (рис. 24). Она почти полностью повторяет интерфейс страницы обычного просмотра карточки, с той лишь разницей, что вместо текста появляются поля ввода данных (в некоторых случаях – выпадающие списки).

Когда пользователь заканчивает редактирование и решает это сохранить, по нажатию кнопки «Подтвердить» посылается запрос на сервер. Принцип действия этого описан в главе 3.2.3.

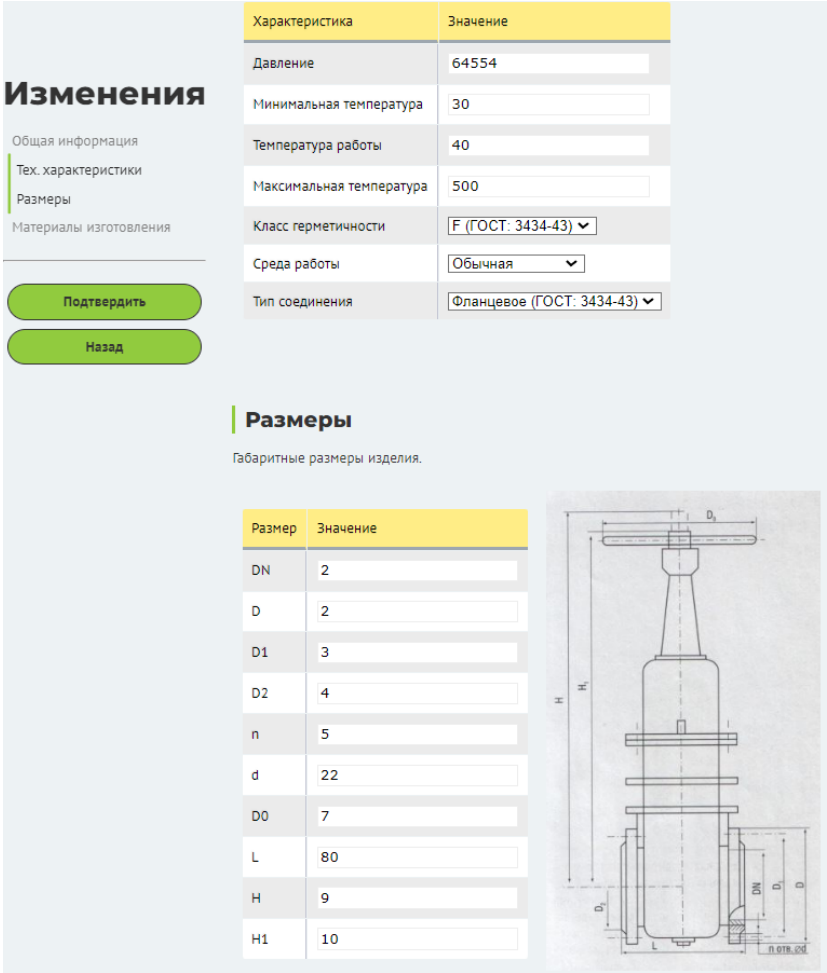


Рисунок 24 – Страница редактирования

3.4 Поиск по векторам

Еще один фильтр является самообучаемым фильтром векторного поиска. Он реализован частично и представляет собой векторизацию изначальных параметров в сжатый вектор и поиск по пространству предложенных векторов. Система является ранней версией поискового аппарата Google опубликованного в Стенфордском университете (журнал).

Система поиска ориентирована на применения адаптивной системы фильтрации. Давайте разберемся, что такое фильтр в базе данных.

Фильтр по выделенному фрагменту, обычный фильтр и поле Фильтр для (Filter For) являются очень простыми способами отбора записей, причем самым простым является фильтр по выделенному фрагменту — он позволяет найти все записи, содержащие определенное значение в выбранном поле. Обычный фильтр используется для отбора записей по значениям нескольких полей. Поле Фильтр для (Filter For) используется, если фокус ввода находится в поле таблицы и нужно ввести конкретное искомое значение или выражение, результат которого будет применяться в качестве условия отбора. Для создания сложных фильтров следует использовать окно расширенного фильтра.

При этом можно заметить что каждый отдельный фильтр при фильтрации откидывает при запросе к базе данных часть значений. Это так называемое дробление базы данных.

Давайте представим что база данных, а конкретно её значения нормализованы. Это обозначает что при фильтрации определенным фильтром функционально дающим выбор из n позиций база данных выдает значения равной соотношению количество n значений фильтра на выбранный диапазон фильтров. Если принят это за аксиому то можно сделать логический вывод, что каждый фильтр имеющий высокое количество выбираемых позиций является фильтром дробящим базу данных сильнее чем остальные фильтры имеющие меньшое количество единиц дробления.

Из вышесказанных слов можно сделать вывод что при сочетании фильтров каждый отдельный будет дробить результат на части равной его единицы фильтрации. Тем самым если встает вопрос об отключении фильтров в определённом порядке необходимо выбирать фильтр с наибольшей значением единицы фильтрации. Этот процесс позволит увеличить диапазон фильтраций и тем самым выдаст большое количество элементов отфильтрованной базы данных.

Теперь необходимо доказать необратимый процесс энтропии систем базы данных при добавлении новых карточек со временем. Для этого рассмотрим понятие энтропии в физическом понимании.

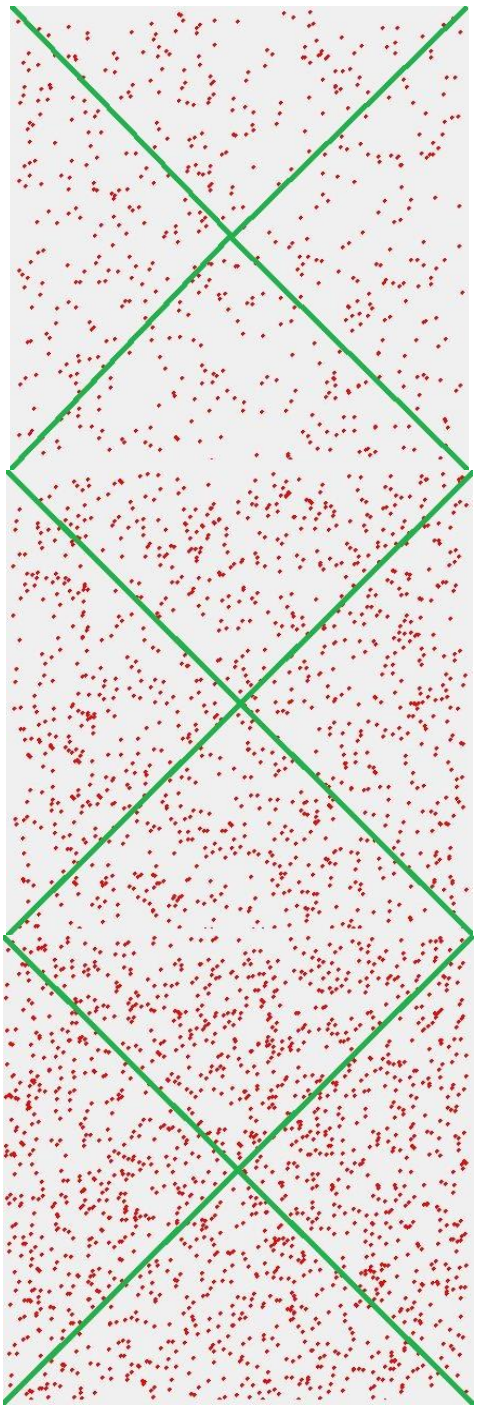
Энтропи́я широко используемый в естественных и точных науках термин (впервые введён в рамках термодинамики как функция состояния термодинамической системы), обозначающий меру необратимого рассеивания энергии или бесполезности энергии (потому что не всю энергию системы можно использовать для превращения в какую-нибудь полезную работу). Для понятия энтропии в данном разделе физики используют название термодинамическая энтропия; термодинамическая энтропия обычно применяется для описания равновесных (обратимых) процессов.

В статистической физике энтропия характеризует вероятность осуществления какого-либо макроскопического состояния. Кроме физики, термин широко употребляется в математике: теории информации и математической статистике. В этих областях знания энтропия определяется статистически и называется статистической или информационной энтропией. Данное определение энтропии известно также как энтропия Шеннона (в математике) и энтропия Больцмана—Гиббса (в физике).

Хотя понятия термодинамической и информационной энтропии вводятся в рамках различных формализмов, они имеют общий физический смысл — логарифм числа доступных микросостояний системы. Взаимосвязь этих понятий впервые установил Людвиг Больцман. В неравновесных (необратимых) процессах энтропия также служит мерой близости состояния системы к равновесному: чем больше энтропия, тем ближе система к равновесию (в состоянии термодинамического равновесия энтропия системы максимальна).

Тем самым можно сделать вывод что при случайном выпадении каточек количество значений подходящим под фильтры будет стремится к конечному числу равным количество карточек деленом на количества значений фильтраций конкретного фильтра. Это дает нам сделать вывод что предыдущие суждения о веденом коэффициенте каждого фильтра верно.

Примером этого может служить тестовая программа а конкретнее её визуализация в виде нормального случайного распределения и представления фильтрации конкретных чесоткой (зон ответственности) фильтров. Это представлена в таблице 1.



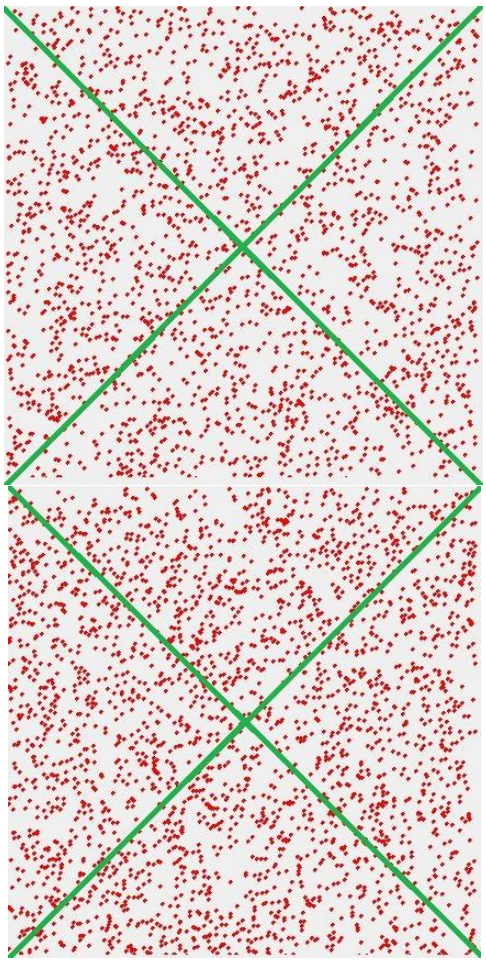


Таблица 1

3.4.1 Векторный поиск

Векторный поиск основан на концепции поиска ближайшего угла между вектором искомого и вектором находящегося в базе данных.

Для начало необходимо понять что каждый элемент имеет равное количество искомых объектов (параметров) однако для поиска этого недостаточно. Это связано с тем что вектора искомого образа представлены в виде разряженных векторов для исправления этого требуется уплотнить вектора. Для понимания того что представляют разряженные вектора представлена краткая вырезка.

Разре́женный масси́в — абстрактное представление обычного массива, в котором данные представлены не непрерывно, а фрагментарно; большинство элементов его принимают одно и то же значение (значение по умолчанию, обычно 0 или null). Причём хранение большого числа нулей в массиве неэффективно как для хранения, так и для обработки массива.

В разреженном массиве возможен доступ к неопределенным элементам. В этом случае массив вернет некоторое значение по-умолчанию.

Простейшая реализация этого массива выделяет место под весь массив, но когда значений, отличных от значений по умолчанию, мало, такая реализация неэффективна. К

этому массиву не применяются функции для работы с обычными массивами в тех случаях, когда о разреженности известно заранее (например, при блочном хранении данных).

Для сведения этого массива или вектора (зависит от типа хранения данных и интерпретации) потребуется авто-энкодер сводящий значения к вектору меньшего измерения. Для этого воспользуемся механизмом умножения матриц последовательно друг на друга с случайным расположением весов изначальных параметров.

Автоэнкодеры — это нейронные сети прямого распространения, которые восстанавливают входной сигнал на выходе. Внутри у них имеется скрытый слой, который представляет собой код, описывающий модель. Автоэнкодеры конструируются таким образом, чтобы не иметь возможность точно скопировать вход на выходе. Обычно их ограничивают в размерности кода (он меньше, чем размерность сигнала) или штрафуют за активации в коде. Входной сигнал восстанавливается с ошибками из-за потерь при кодировании, но, чтобы их минимизировать, сеть вынуждена учиться отбирать наиболее важные признаки. Это представлено на рисунке 25.

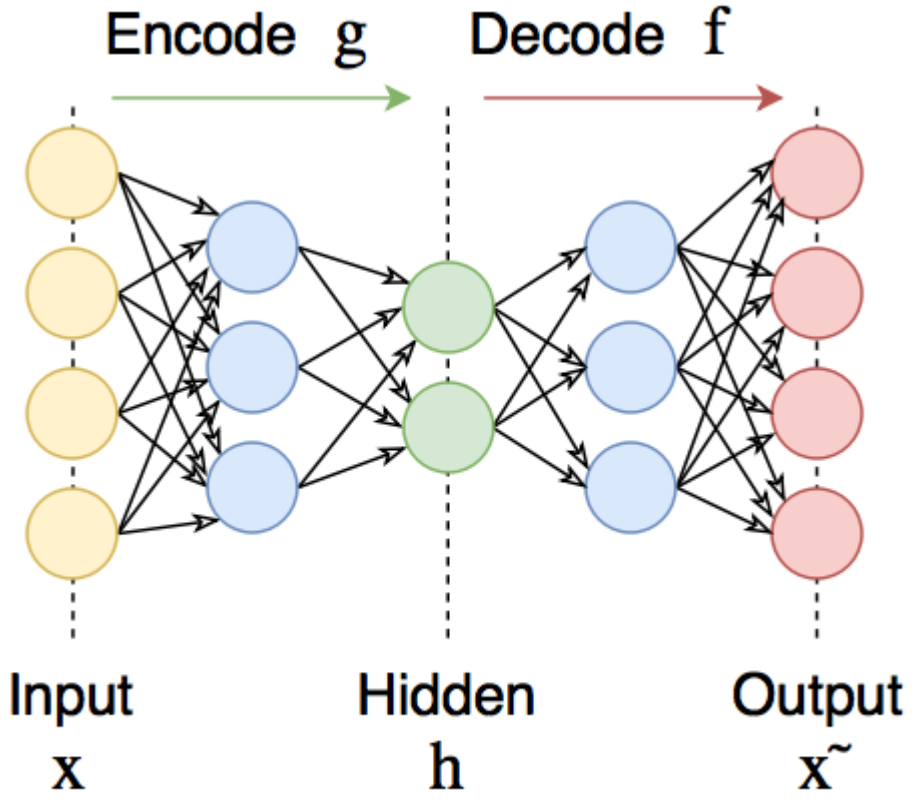


Рисунок 25 – Автоэнкодер

Из этого мы получаем механизм сравнивания векторов высокой плотности. Однако в дальнейшем хоть и каждое значение будет сформировано по каждой группе понадобится способ корректировки весов для смежной векторизации и более точного нахождения искомой точки вхождения в сравниваемое пространство.

Для этого применяется до обучения весов матриц так как известен конкретно элемент нашего влечения, но в связи с длительностью обучения и необходимости длительного выбора этот процесс может затянутся значительно на долгое время.

ЗАКЛЮЧЕНИЕ

В ходе работы была разработана реляционная база данных в системе сайтов вариации объектов. База содержит информацию о задвижках, их характеристиках и размерах. Данные можно фильтровать по параметрам, которые может выбрать пользователь. Обеспечена защита данных, путем создания защиты данных представителем услуг хранения размещения. Каждый участник может добавлять и удалять каточки. Производить их редактирования и просматривать основную информацию.

Проект можно продолжать развивать. Например, добавить новые характеристики, или запретить удаление и редактирование для некоторых пользователей.

СПИСОК ЛИТЕРАТУРЫ

- А.Ю. Лезин, «Справочник промышленного оборудования» №4(7), 2005. – 74-77 с., 117-122 с.

Информация

- Виды соединения трубопроводной арматуры: [Электронный ресурс]. URL: https://www.armprof.ru/articles/vidy-soedineniy-truboprovodnoy-armatury/ (Дата обращения: 01.07.2022).

Нормы и классы герметичности: [Электронный ресурс]. URL: https://gazovik-gaz.ru/spravochnik/armatura/normyi-i-klassyi-germetichnosti.html (Дата обращения: 01.07.2022).

- Материалы металлических конструкций: [Электронный ресурс]. URL: https://megalektsii.ru/s26177t2.html (Дата обращения: 01.07.2022).

ГОСТы

- ГОСТ 5762-2002, Арматура трубопроводная. Задвижки на номинальное давление: [Электронный ресурс]. URL: https://docs.cntd.ru/document/1200031271 (Дата обращения: 01.07.2022).

«Supabase»

- Создание и настройка базы данных: [Электронный ресурс]. URL: https://supabase.com/docs/reference/javascript/installing (Дата обращения: 01.07.2022).
- Подключение: [Электронный ресурс]. URL: https://supabase.com/docs/reference/javascript/initializing (Дата обращения: 01.07.2022).
- Фильтрация: [Электронный ресурс]. URL: https://supabase.com/docs/reference/javascript/eq (Дата обращения: 01.07.2022).
- Выборка: [Электронный ресурс]. URL: https://supabase.com/docs/reference/javascript/select (Дата обращения: 01.07.2022).
- Вставка: [Электронный ресурс]. URL: https://supabase.com/docs/reference/javascript/insert (Дата обращения: 01.07.2022).
- Изменение: [Электронный ресурс]. URL: https://supabase.com/docs/reference/javascript/update (Дата обращения: 01.07.2022).
- Удаление: [Электронный ресурс]. URL: https://supabase.com/docs/reference/javascript/delete (Дата обращения: 01.07.2022).

ПРИЛОЖЕНИЯ

Листинг программного кода

```
const express = require('express');
const app = express();

app.use('/main', express.static(__dirname + '/main'));

const PORT = process.env.PORT || 3000;

let N0;
N0 = news(35, 2);

let N1;
N1 = news(15, 2);

let N2;
N2 = news(7, 2);

let N3;
N3 = news(3, 2);

let IDL;
IDL = news(3, 1);

let k = 0.5;

let W01;
W01 = news(35, 14);

let W12;
W12 = news(15, 6);

let W23;
W23 = news(7, 3);

N0[34][0] = 1;
N1[14][0] = 1;
N2[6][0] = 1;


app.listen(PORT, () => {
  console.log('Сервер запущен, ' + PORT);
});

app.get('/', (req, res) => {
  res.sendFile(__dirname + '/main/index.html');
});

app.get('/card/:id', (req, res) => {
  if (req.params.id == 'new')
    res.sendFile(__dirname + '/main/new.html');
  else
    res.sendFile(__dirname + '/main/index_card.html');
});

app.get('/find', (req, res) => {
  res.sendFile(__dirname + '/main/find.html');
});

app.get('/card_edit/:id', (req, res) => {
  res.sendFile(__dirname + '/main/edit.html');
});

app.get('/all_data', (req, res) => {
  let r = GetAllTheData();
  r.then(res_main => { res.send(JSON.stringify(res_main)) });
});
```

```
app.get('/card_fe/:id', (req, res) => {
  let r = GetCardForEdit(req.params.id);
  r.then(res_main => res.send(`{"data":` + JSON.stringify(res_main) + "}"));
});


function Zeroer(p) {
  if (p == '') return 0;
  if (p == "null") return 0;
  if (p == NaN) return 0;
  if (p == null) return 0;
  if (p == undefined) return 0;
  return p;
}

const jsonParser = express.json();
app.post('/card/:id', jsonParser, (req, res) => {
  if (req.params.id == 'new') {
    let r = AddCard(req.body);
    r.then(res_main => res.send(`{"status":"0"}`));
  }
  else {
    if (req.params.id == 'find') {
      //console.log(req.body);
      let r = FindAllCards(req.body);
      r.then(res_main => {
        if (res_main.length == 0) {
          console.log(req.body);
          let q = [[

            Zeroer(req.body.Manufact),
            Zeroer(req.body.Mass1),
            Zeroer(req.body.Mass2),
            Zeroer(req.body.Prss1),
            Zeroer(req.body.Tmin1),
            Zeroer(req.body.Tnorm1),
            Zeroer(req.body.Tmax1),
            Zeroer(req.body.Prss2),
            Zeroer(req.body.Tmin2),
            Zeroer(req.body.Tnorm2),
            Zeroer(req.body.Tmax2),
            Zeroer(req.body.Germ),
            Zeroer(req.body.Work),
            Zeroer(req.body.Conn),
            Zeroer(req.body.DN_1),
            Zeroer(req.body.D_1),
            Zeroer(req.body.D1_1),
            Zeroer(req.body.D2_1),
            Zeroer(req.body.n1),
            Zeroer(req.body.d1),
            Zeroer(req.body.D0_1),
            Zeroer(req.body.L1),
            Zeroer(req.body.H1),
            Zeroer(req.body.H11),
            Zeroer(req.body.DN_2),
            Zeroer(req.body.D_2),
            Zeroer(req.body.D1_2),
            Zeroer(req.body.D2_2),
            Zeroer(req.body.n2),
            Zeroer(req.body.d2),
            Zeroer(req.body.D0_2),
            Zeroer(req.body.L2),
            Zeroer(req.body.H2),
            Zeroer(req.body.H12),
            Zeroer(req.body.MM1),
            Zeroer(req.body.MM2),
            Zeroer(req.body.MM3),
            Zeroer(req.body.MM4),
```

```
  Zeroer(req.body.MM5),
  Zeroer(req.body.MM6),
  Zeroer(req.body.MM7),
  Zeroer(req.body.MM8),
  Zeroer(req.body.MM9)
]];
let ee = vector(q);
let d = 1000000000000000;
let card = null;

let RRR = GetCards();
RRR.then(RESSS => {



 for (let I = 0; I < RESSS.length; I++) {
  let R2 = GetCard(RESSS[I].id);
  R2.then(RES => {
   console.log("Карточка: " + RES[0].id);
   let ee2 = vector([[

    Zeroer(RES[0].Manufact),
    Zeroer(RES[0].Mass),
    Zeroer(0),
    Zeroer(RES[0].TechParams_Link.Prss),
    Zeroer(RES[0].TechParams_Link.Tmin),
    Zeroer(RES[0].TechParams_Link.Tnorm),
    Zeroer(RES[0].TechParams_Link.Tmax),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(RES[0].TechParams_Link.Germ),
    Zeroer(RES[0].TechParams_Link.Work),
    Zeroer(RES[0].TechParams_Link.Conn),
    Zeroer(RES[0].Sizes_Link.DN),
    Zeroer(RES[0].Sizes_Link.D),
    Zeroer(RES[0].Sizes_Link.D1),
    Zeroer(RES[0].Sizes_Link.D2),
    Zeroer(RES[0].Sizes_Link.n),
    Zeroer(RES[0].Sizes_Link.d),
    Zeroer(RES[0].Sizes_Link.D0),
    Zeroer(RES[0].Sizes_Link.L),
    Zeroer(RES[0].Sizes_Link.H),
    Zeroer(RES[0].Sizes_Link.H1),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(0),
    Zeroer(RES[0].MM1_Korpus),
    Zeroer(RES[0].MM2_Kryshka),
    Zeroer(RES[0].MM3_Salnik),
    Zeroer(RES[0].MM4_Shpindel),
    Zeroer(RES[0].MM5_Zatvor),
    Zeroer(RES[0].MM6_Klin),
    Zeroer(RES[0].MM7_Nabivka),
    Zeroer(RES[0].MM8_Prokladka),
    Zeroer(RES[0].MM9_Flancy)

   ]]);
   console.log("вектор: " + ee2);

   let d2 = Distance(ee, ee2);
```

```
            console.log("дистанция: " + d2);
            console.log("//////////");
            if (d2 < d) {
              d = d2;
              card = RES;
            }
            if (I == RESSS.length - 1) res.send(`{"data":` + JSON.stringify(card) + "}");
          });
        }
        console.log(d);
        //res.send(`{"data":` + JSON.stringify(card) + "}");



      });
    } else
      res.send(`{"data":` + JSON.stringify(res_main) + "}");
  });
 }
 else {
  let r = GetCard(req.params.id);
  r.then(res_main => res.send(`{"data":` + JSON.stringify(res_main) + "}"));
 }
}
});

app.put('/card/:id', jsonParser, (req, res) => {
 let r = ChangeCard(req.body);
 r.then(res_main => res.send(`{"status":"0"}`));
});

app.delete('/card/:id', (req, res) => {
 let r = DeleteCard(req.params.id);
 r.then(res_main => res.send(`{"status":"0"}`));
});

supb = require('@supabase/supabase-js');

const supabaseUrl = 'https://yiyqsrpsalvozpvjsfxm.supabase.co'
const supabaseKey =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InlpeXFzcnBzYWx2b3pwdmpzZnhtIiwic
m9sZSI6ImFub24iLCJpYXQiOjE2NTU3MjM4NDcsImV4cCI6MTk3MTI5OTg0N30.60qE6w84VKX3ci4gfRWTz7wQm-
1j7EYtjYJYE18EzOI'
const supabase = supb.createClient(supabaseUrl, supabaseKey)

app.get('/all', (req, res) => {
 let r = GetCards();
 r.then(resss =>
  res.send(`{"data":` + JSON.stringify(resss) + `}`));
});

GetCard = async function (id) {
 let { data: Card, error } = await supabase
  .from('Detail')
  .select(`*,
  TechParams_Link (*,
   GermClass_Link (Word, GOST_Link(Name)),
   WorkEnv_Link (Name),
   TubeConn_Link (Name, GOST_Link(Name))
  ),
  Manufacturer_Link (PlantName, Index),
  Sizes_Link (*),
  AsmMats_Link (
   MM1_Korpus (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
   MM2_Kryshka (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
   MM3_Salnik (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
   MM4_Shpindel (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
   MM5_Zatvor (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
   MM6_Klin (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
```

```
    MM7_Nabivka (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
    MM8_Prokladka (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)),
    MM9_Flancy (*, Manufacturer_ID (PlantName, Index), GOST_ID (Name))
  )`)
  .eq('id', id);
 return Card;
}

FindAllCards = async function (params) {
 const { data: Result, error } = await supabase
  .from('Detail')
  .select(`*,
 TechParams_Link (*),
 Manufacturer_Link,
 Sizes_Link (*),
 AsmMats_Link (*)
 `);
 let res1 = Result;
 let nm = params.NameL.toLowerCase();
 let ds = params.Desc.toLowerCase();
 console.log(params);
 if (params.NameL != "") res1 = res1.filter(x => (x.Label.toLowerCase().indexOf(nm) >= 0));
 console.log(res1);
 if (params.Desc != "") res1 = res1.filter(x => (x.Desc.toLowerCase().indexOf(ds) >= 0));
 if (params.OKP != "") res1 = res1.filter(x => x.OKP == params.OKP);
 if (params.Manufact != "") res1 = res1.filter(x => x.Manufacturer_Link == params.Manufact);
 if (params.Mass1 != "") res1 = res1.filter(x => x.mass >= params.Mass1);
 if (params.Mass2 != "") res1 = res1.filter(x => x.mass <= params.Mass2);

 if (params.Prss1 != "") res1 = res1.filter(x => x.TechParams_Link.Pressure >= params.Prss1);
 if (params.Prss2 != "") res1 = res1.filter(x => x.TechParams_Link.Pressure <= params.Prss2);
 if (params.Tmin1 != "") res1 = res1.filter(x => x.TechParams_Link.Tmin >= params.Tmin1);
 if (params.Tmin2 != "") res1 = res1.filter(x => x.TechParams_Link.Tmin <= params.Tmin2);
 if (params.Tnorm1 != "") res1 = res1.filter(x => x.TechParams_Link.Tnorm >= params.Tnorm1);
 if (params.Tnorm2 != "") res1 = res1.filter(x => x.TechParams_Link.Tnorm <= params.Tnorm2);
 if (params.Tmax1 != "") res1 = res1.filter(x => x.TechParams_Link.Tmax >= params.Tmax1);
 if (params.Tmax2 != "") res1 = res1.filter(x => x.TechParams_Link.Tmax <= params.Tmax2);

 if (params.Germ != "") res1 = res1.filter(x => x.TechParams_Link.GermClass_Link == params.Germ);
 if (params.Work != "") res1 = res1.filter(x => x.TechParams_Link.WorkEnv_Link == params.Work);
 if (params.Conn != "") res1 = res1.filter(x => x.TechParams_Link.TubeConn_Link == params.Conn);

 if (params.DN_1 != "") res1 = res1.filter(x => x.Sizes_Link.DN >= params.DN_1);
 if (params.DN_2 != "") res1 = res1.filter(x => x.Sizes_Link.DN <= params.DN_2);
 if (params.D_1 != "") res1 = res1.filter(x => x.Sizes_Link.D >= params.D_1);
 if (params.D_2 != "") res1 = res1.filter(x => x.Sizes_Link.D <= params.D_2);
 if (params.D1_1 != "") res1 = res1.filter(x => x.Sizes_Link.D1 >= params.D1_1);
 if (params.D1_2 != "") res1 = res1.filter(x => x.Sizes_Link.D1 <= params.D1_2);
 if (params.D2_1 != "") res1 = res1.filter(x => x.Sizes_Link.D2 >= params.D2_1);
 if (params.D2_2 != "") res1 = res1.filter(x => x.Sizes_Link.D2 <= params.D2_2);
 if (params.n1 != "") res1 = res1.filter(x => x.Sizes_Link.n >= params.n1);
 if (params.n2 != "") res1 = res1.filter(x => x.Sizes_Link.n <= params.n2);
 if (params.d1 != "") res1 = res1.filter(x => x.Sizes_Link.d >= params.d1);
 if (params.d2 != "") res1 = res1.filter(x => x.Sizes_Link.d <= params.d2);
 if (params.D0_1 != "") res1 = res1.filter(x => x.Sizes_Link.D0 >= params.D0_1);
 if (params.D0_2 != "") res1 = res1.filter(x => x.Sizes_Link.D0 <= params.D0_2);
 if (params.L1 != "") res1 = res1.filter(x => x.Sizes_Link.L >= params.L1);
 if (params.L2 != "") res1 = res1.filter(x => x.Sizes_Link.L <= params.L2);
 if (params.H1 != "") res1 = res1.filter(x => x.Sizes_Link.H >= params.H1);
 if (params.H2 != "") res1 = res1.filter(x => x.Sizes_Link.H <= params.H2);
 if (params.H11 != "") res1 = res1.filter(x => x.Sizes_Link.H >= params.H11);
 if (params.H12 != "") res1 = res1.filter(x => x.Sizes_Link.H <= params.H12);

 if (params.MM1 != "null") res1 = res1.filter(x => x.AsmMats_Link.MM1_Korpus == params.MM1);
 if (params.MM2 != "null") res1 = res1.filter(x => x.AsmMats_Link.MM2_Kryshka == params.MM2);
 if (params.MM3 != "null") res1 = res1.filter(x => x.AsmMats_Link.MM3_Salnik == params.MM3);
 if (params.MM4 != "null") res1 = res1.filter(x => x.AsmMats_Link.MM4_Shpindel == params.MM4);
 if (params.MM5 != "null") res1 = res1.filter(x => x.AsmMats_Link.MM5_Zatvor == params.MM5);
 if (params.MM6 != "null") res1 = res1.filter(x => x.AsmMats_Link.MM6_Klin == params.MM6);
```

```
  if (params.MM7 != "null") res1 = res1.filter(x => x.AsmMats_Link.MM7_Nabivka == params.MM7);
  if (params.MM8 != "null") res1 = res1.filter(x => x.AsmMats_Link.MM8_Prokladka == params.MM8);
  if (params.MM9 != "null") res1 = res1.filter(x => x.AsmMats_Link.MM9_Flancy == params.MM9);

  return res1;
}

GetCardForEdit = async function (id) {
  let { data: Card, error } = await supabase
    .from('Detail')
    .select(`*,
    TechParams_Link (*),
    Manufacturer_Link,
    Sizes_Link (*),
    AsmMats_Link (
      MM1_Korpus,
      MM2_Kryshka,
      MM3_Salnik,
      MM4_Shpindel,
      MM5_Zatvor,
      MM6_Klin,
      MM7_Nabivka,
      MM8_Prokladka,
      MM9_Flancy
    )`)
    .eq('id', id);
  return Card;
}

GetAllTheData = async function () {
  let { data: GermClasses, error } = await supabase
    .from('GermClasses')
    .select('*, GOST_Link(Name)');
  let { data: Mats, error2 } = await supabase
    .from('Materials')
    .select('*, Manufacturer_ID (PlantName, Index), GOST_ID (Name)');
  let { data: WorkEnv, error3 } = await supabase
    .from('WorkEnviroments')
    .select('*');
  let { data: TubeConn, error4 } = await supabase
    .from('TubeConn')
    .select('*, GOST_Link(Name)');
  let { data: Manuf, error5 } = await supabase
    .from('Manufacturer')
    .select('*');
  return { germ: GermClasses, mats: Mats, work: WorkEnv, tube: TubeConn, manuf: Manuf };
}

GetCards = async function () {
  let { data: Card, error } = await supabase
    .from('Detail')
    .select(`id, Label, Desc`);
  return Card;
}

Nullor = function (str) {
  return str == "" ? null : str;
}

AddCard = async function (params) {
  let resp1 = await supabase.from('TechParams').insert([{
    Pressure: Nullor(params.Prss),
    Tmin: Nullor(params.Tmin),
    Tnorm: Nullor(params.Tnorm),
    Tmax: Nullor(params.Tmax),
    GermClass_Link: Nullor(params.Germ),
    WorkEnv_Link: Nullor(params.Work),
    TubeConn_Link: Nullor(params.Conn)
  }]);
```

```
  console.log("ответ 1: " + JSON.stringify(resp1) + "\n");
  let resp2 = await supabase.from('Sizes').insert([{
    DN: Nullor(params.DN),
    D: Nullor(params.D),
    D1: Nullor(params.D1),
    D2: Nullor(params.D2),
    n: Nullor(params.n),
    d: Nullor(params.d),
    D0: Nullor(params.D0),
    L: Nullor(params.L),
    H: Nullor(params.H),
    H1: Nullor(params.H1)
  }]);
  console.log("ответ 2: " + JSON.stringify(resp2) + "\n");
  let resp3 = await supabase.from('AssembleMats').insert([{
    MM1_Korpus: (params.MM1 != "null" ? params.MM1 : null),
    MM2_Kryshka: (params.MM2 != "null" ? params.MM2 : null),
    MM3_Salnik: (params.MM3 != "null" ? params.MM3 : null),
    MM4_Shpindel: (params.MM4 != "null" ? params.MM4 : null),
    MM5_Zatvor: (params.MM5 != "null" ? params.MM5 : null),
    MM6_Klin: (params.MM6 != "null" ? params.MM6 : null),
    MM7_Nabivka: (params.MM7 != "null" ? params.MM7 : null),
    MM8_Prokladka: (params.MM8 != "null" ? params.MM8 : null),
    MM9_Flancy: (params.MM9 != "null" ? params.MM9 : null),
  }]);
  console.log("ответ 3: " + JSON.stringify(resp3) + "\n");
  await supabase.from('Detail').insert([{
    Label: Nullor(params.NameL),
    Desc: Nullor(params.Desc),
    OKP: Nullor(params.OKP),
    Manufacturer_Link: Nullor(params.Manufact),
    mass: Nullor(params.Mass),
    TechParams_Link: resp1.data != null ? resp1.data[0].id : null,
    Sizes_Link: resp2.data != null ? resp2.data[0].id : null,
    AsmMats_Link: resp3.data != null ? resp3.data[0].id : null,
    Type: 0
  }])
}

ChangeCard = async function (params) {
  let resp = await supabase.from('Detail').update({
    Label: Nullor(params.NameL),
    Desc: Nullor(params.Desc),
    OKP: Nullor(params.OKP),
    Manufacturer_Link: Nullor(params.Manufact),
    mass: Nullor(params.Mass)
  }).eq('id', params.id);
  await supabase.from('TechParams').update({
    Pressure: Nullor(params.Prss),
    Tmin: Nullor(params.Tmin),
    Tnorm: Nullor(params.Tnorm),
    Tmax: Nullor(params.Tmax),
    GermClass_Link: Nullor(params.Germ),
    WorkEnv_Link: Nullor(params.Work),
    TubeConn_Link: Nullor(params.Conn)
  }).eq('id', resp.data[0].TechParams_Link);
  await supabase.from('Sizes').update({
    DN: Nullor(params.DN),
    D: Nullor(params.D),
    D1: Nullor(params.D1),
    D2: Nullor(params.D2),
    n: Nullor(params.n),
    d: Nullor(params.d),
    D0: Nullor(params.D0),
    L: Nullor(params.L),
    H: Nullor(params.H),
    H1: Nullor(params.H1)
  }).eq('id', resp.data[0].Sizes_Link);
  await supabase.from('AssembleMats').update({
```

```
    MM1_Korpus: (params.MM1 != "null" ? params.MM1 : null),
    MM2_Kryshka: (params.MM2 != "null" ? params.MM2 : null),
    MM3_Salnik: (params.MM3 != "null" ? params.MM3 : null),
    MM4_Shpindel: (params.MM4 != "null" ? params.MM4 : null),
    MM5_Zatvor: (params.MM5 != "null" ? params.MM5 : null),
    MM6_Klin: (params.MM6 != "null" ? params.MM6 : null),
    MM7_Nabivka: (params.MM7 != "null" ? params.MM7 : null),
    MM8_Prokladka: (params.MM8 != "null" ? params.MM8 : null),
    MM9_Flancy: (params.MM9 != "null" ? params.MM9 : null),
  }).eq('id', resp.data[0].AsmMats_Link);
}

DeleteCard = async function (id) {
  let resp = await supabase.from('Detail').delete().eq('id', id);
  await supabase.from('TechParams').delete().eq('id', resp.data[0].TechParams_Link);
  await supabase.from('Sizes').delete().eq('id', resp.data[0].Sizes_Link);
  await supabase.from('AssembleMats').delete().eq('id', resp.data[0].AsmMats_Link);
}




function mutiplyMatrices(a, b) {
  if (b.length == a[0].length) {
    let result = [];

    for (let i = 0; i < a.length; i++) {

      result[i] = [];

      for (let j = 0; j < b[0].length; j++) {

        let sum = 0;

        for (let k = 0; k < a[0].length; k++) {
          sum += a[i][k] * b[k][j];
        }
        result[i][j] = sum;
      }
    }
    return result;
  } else return null;
  return null
}
function vector(v) {

  let w = [
    [1, 1, 1, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 110, 1],
    [110, 1, 1, 0, 1, 1, 1, 1, 1, 1, 100000, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 0, 1, 1, 1, 100, 1, 1, 1, 1000, 1, 1, 1],
    [1, 1, 1, 1000, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1000, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1],
    [1, 1, 1000, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 100, 1, 1, 1, 1000, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1000, 1, 1, 1, 1, 100, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 100, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1000, 1, 1, 1, 1, 1, 1, 1000, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1000, 1, 1, 1, 1, 1000, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 10000, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1000, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
```

```
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 0, 1, 10000, 1, 1, 1, 1, 1, 1, 1000, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1000, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1000, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1000, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 100, 1, 1, 1, 1, 1, 1, 1000, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 100, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1000, 1, 1, 1, 1],
    [1, 1, 1, 1000, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1000, 1, 1, 1]
  ]

  let r = [
    [1, 5656, 1, 1, 1, 6543, 1],
    [1, 1, 0, 1, 1, 1, 1],
    [1, 1, 1, 345, 1, 2, 1],
    [1, 2, 1, 1, 1, 1, 1],
    [1, 1, 359, 1, 36, 1, 1],
    [1399, 1, 0, 1, 1, 2, 1],
    [1, 1, 0, 465, 343, 1, 1],
    [1, 343, 0, 1, 1, 1, 1],
    [1, 3, 1, 1, 1, 4, 1],
    [1, 1, 3596, 1, 5445, 3, 1],
    [1, 1, 100, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1],
    [39, 9683, 10, 1, 539569, 1, 3591],
    [1, 1, 1, 1, 75643, 1, 1],
    [1, 1, 1000, 1, 1, 1, 1],
  ]

  let y = [
    [1, 3445, 1],
    [1, 453, 5443],
    [3232, 1, 1],
    [1, 34553, 1],
    [1, 325, 5343],
    [2233, 1, 1],
    [1, 1, 5446],
  ]

  return mutiplyMatrices(mutiplyMatrices(mutiplyMatrices(v, w), r), y);

}

function Distance(x, y) {
  return Math.sqrt(Math.pow(x[0][0] - y[0][0], 2) + Math.pow(x[0][1] - y[0][1], 2) + Math.pow(x[0][2] - y[0][2], 2));
}

function forWards(li, w, lo) {

  let wt = w.length;
  let ht = w[0].length;
  console.log(wt);
  console.log(ht);
  console.log(li);
  console.log(lo);

  for (let y = 0; y < ht; y++) {
```

```
    lo[y, 0] = 0;
    for (let x = 0; x < wt; x++) {
      lo[y, 0] = lo[y, 0] + li[x, 0] * w[x, y];
      console.log(lo[y, 0] + li[x, 0] * w[x, y]);
    }
    lo[y, 0] = 1 / (1 + Math.exp(-1 * lo[y, 0]));
    console.log(1 / (1 + Math.exp(-1 * lo[y, 0])));
  }

  return lo;

} // проход по слою

function fixOutError(idl, n3) {

  let i = 0;

  let wt = idl.length;
  for (let x = 0; x < wt; x++) {
    i = idl[x, 0] - n3[x, 0];
    n3[x, 1] = i;
  }

  return n3;
} // нахождение отклонений

function findError(li, w, lo) {

  let wt = w.length - 1;
  let ht = w[0].length;

  for (let x = 0; x < wt; x++) {
    li[x, 1] = 0;
    for (let y = 0; y < ht; y++) {
      li[x, 1] = li[x, 1] + w[x, y] * lo[y, 1];

    }
    li[x, 1] = li[x, 1] * li[x, 0] * (1 - li[x, 0]);
  }

  return li;
} // нахождение ошибок слоя

function backWards(li, w, lo, k) {

  let wt = w.lengt;
  let ht = w[0].length;

  for (let y = 0; y < ht; y++) {
    for (let x = 0; x < wt; x++) {
      w[x, y] = w[x, y] + k * lo[y, 1] * li[x, 0] * lo[y, 0] * (1 - lo[y, 0]);
    }
  }

  return w;
} // изменение весов

function fillW(w, Q) {

  let wt = w.lengt;
  let ht = w[0].length;

  for (let x = 0; x < wt; x++) {
    for (let y = 0; y < ht; y++) {
      w[x, y] = Q[i];
    }
  }

  return w;
```

```
} // рандом весов

// создание весов
function news(m, n) {
  let mas = [];
  for (let i = 0; i < m; i++) {
    mas[i] = [];
    for (let j = 0; j < n; j++) {
      mas[i][j] = 0.5;
    }
  }
  return mas
}

function getTask(li, idl, test_questionsdouble, test_answers) {
  for (let i = 0; i < li.length; i++) {
    li[j, 0] = test_questionsdouble[j];
  }
  for (let i = 0; i < idl.length; i++) {
    idl[j, 0] = test_answers[j];
  }

  return {
    li: li,
    idl: idl
  }

} // новый пример для нейросетей
```