Чижа Дмитрий Вадимович

**Отчёт по выполнению тестового задания**

# Лента новостей

## Анализ задачи

***Формулировка задачи:***

*«Необходимо создать общую ленту новостей для пользователей с возможностью оценки постов в ленте.*

*Лента должна иметь фильтр по категориям. Любой пользователь может поставить "лайк" или отменить его. Необходимо предусмотреть возможность просмотра списка всех оценивших пост пользователей. Ограничение на размер хранения контента одного поста - 243 байта.*

*Предложите структуру базы данных MySQL, позволяющую реализовать данную задачу.*

*Напишите базовый модуль, реализующий функции вывода новостей, добавление нового поста в ленту, установку и отмену "лайка" на новости».*

Прежде всего разберёмся с фильтром новостей по категориям. Посты и категории добавляются админом с возможностью добавления пользователями при небольшом изменении. Каждая новость может иметь одну или несколько категорий, и категория может иметь новости, связанные с ней.

В постановке задачи у нас явно указано два особых требования – необходимо использовать MySQL и размер одного поста не повышает 243 байта, что исключает использование аудио, видео- и фотоизображений в новостях.

Далее необходимо определиться с выбором подхода к проектированию будущего модуля. Я решил остановиться на **объектно-ориентированном подходе** [1], потому что он позволяет сделать новостную ленту легче поддающейся изменениям (очевидно, что в задании перечислены не все функции, которые она обычно выполняет) без полной переработки.

Следующим шагом будет построение **диаграммы прецедентов** [[1]](#_Библиографический_список)**.** Я выделил двух акторов – **Пользователь** и **Админ** и следующие прецеденты:

1. *Отфильтровать новости по категориям (Главный).*

Акторы: Пользователь.

Описание: Пользователь открывает ленту новостей. Затем открывает меню выбора фильтров и выбирает желаемые категории постов для поиска. После подтверждения выбора система возвращает ленту с новостями только из выбранных категорий.

1. *Оценить пост (Главный).*

Акторы: Пользователь.

Описание: Пользователь открывает ленту новостей. Просматривая её, пользователь находит посты, которые ему особенно нравятся. Пользователь отмечает пост «лайком».

1. *Просмотреть список всех оценивших пост пользователей (Главный).*

Акторы: Пользователь.

Описание: Пользователь открывает ленту новостей. Просматривая её, пользователь находит посты, которые уже оценили другие люди до него. Пользователь открывает список оценивших новость людей и просматривает его. Затем он возвращается к просмотру ленты.

1. Вывести все новости (Главный).

Акторы: Пользователь.

Описание: Пользователь открывает ленту новостей.

1. *Добавить новый пост в ленту (Главный).*

Акторы: Админ.

Описание: Админ решает создать новый пост. Для этого он переходит в режим добавления новости в ленту, выбирает для неё одну или несколько категорий и пишет текст. Далее админ подтверждает своё действие и новость добавляется в общую ленту. Другие пользователи, в том числе и автор, могут эту новость просмотреть и оценить.

1. *Добавить новую категорию (Второстепенный).*

Акторы: Админ.

Описание: Админ решает добавить новую категорию для новостей. Для этого он переходит в режим добавления новой категории, выбирает новости, которые относятся к этой категории. Далее админ подтверждает своё действие, и категория добавляется в общий список. Другие пользователи могут использовать эту категорию при фильтрации новостей.

Расширенное описание прецедентов и саму диаграмму см. в прил. A и B

Далее составим концептуальную модель предметной области. Это необходимо сделать для того, чтобы впоследствии использовать её при построении логической и физической модели БД и диаграммы классов.

Список понятий:

1. Новость (Текст, Дата и время добавления).
2. Пользователь (Никнейм, Флажки прав\*).
3. Категория новостей (Название).

\* - Флажки прав отделяют Пользователей от Админов. Можно было бы вынести админа в отдельное понятие, но пока делать это ради двух прецедентов я не стал. Кроме того, в дальнейшем планируется доверить добавление постов и категорий Пользователями.

**Диаграмма понятий** [[1]](#_Библиографический_список) с комментариями прикреплена к приложению (см. прил. C).

Диаграмму активностей и последовательностей [[1]](#_Библиографический_список) я решил опустить, ввиду простоты разрабатываемого модуля (может быть и зря).

## Проектирование решения

Для проектирования БД построим ERD на основе диаграммы понятий, созданной на этапе анализа (см. прил. C). Одной из альтернатив могло стать хранение данных в MongoDB в формате документов [[2]](#_Библиографический_список), но т. к. в постановке задачи явно указан MySQL будем использовать реляционную модель.

Первым делом подготовим связи «многие ко многим» к записи в БД. Для этого добавим связующие таблицы. Так как модель очень простая, то и нормализация не требуется. Можно вынести права в отдельную сущность, потому что у большинства пользователей они идентичны, но сильного выигрыша на маленьких данных это не даст.

Далее решим, какие типы данных стоит использовать для хранения данных. Именные поля (никнейм пользователя, название категории новостей) будем хранить в «VARCHAR(50)», что более, чем достаточно. Текст новостей будем хранить в «TINYTEXT», поскольку их размер не занимает не более 243 байт. Права пользователей будем хранить в виде целого числа «int».

Итоговый вариант логической (не зависит от СУБД) и физической (с типами данных в MySQL) модели БД находится в приложении (см. прил. D и прил. E).

Следующим шагом определимся, как мы будем связывать БД в MySQL и бизнес-логику на языке PHP. Можно просто использовать запросы «вшитые в код» с помощью расширения «mysqli». Это было быстро и просто, но у программы будет спагетти-архитектура и её пришлось бы полностью переписывать при изменении логики или СУБД.

Другим вариантом является использование ORM-технологий без привязки к конкретной СУБД. Это упрощает поддержку и расширение кода, но могут возникнуть проблемы с производительностью (любой мультитул-инструмент почти всегда хуже своего узкоспециализированного аналога).

Таким образом, каждый прецедент будет обрабатывать отдельный контроллер (паттерн MVC), который будет связан с моделью внутри приложения, которая в свою очередь будет строить запросы к БД, используя сущности созданные ORM.

Интересным моментом ещё была тяга превратить ленту новостей в Singleton, но в данном случае я решил, что лучше будет применить абстрактную фабрику (пусть лента новостей состоит из трёх модулей см. прил. F), т.к. это позволит в будущем прицепить к каждому пользователю свою новостную ленту.

## Реализация решения

Я решил остановиться на **Doctrine-ORM** в составе фреймворка **Symfony4**. Есть ещё несколько аналогов (Laravel, Yii), но я решил остановиться на Symfony по причине его стабильности и модульности. Однако, пусть он и показался мне простым, мне все равно не до конца понятны некоторые детали и сказался недостаток опыта в создании модулей с api.

# Список доменов

## Анализ задачи

***Формулировка задачи:***

*«Имеется таблица пользователей:*

*CREATE TABLE `users` (*

*`id` int(11) NOT NULL AUTO\_INCREMENT,*

*`name` varchar(32) NOT NULL,*

*`gender` tinyint(2) NOT NULL,*

*`email` varchar(1024) NOT NULL,*

*PRIMARY KEY (`id`)*

*) ENGINE=InnoDB;*

*В таблице более 100 млн записей, и она находится под нагрузкой в production (идут запросы на добавление / изменение / удаление).*

*В поле email может быть от одного до нескольких перечисленных через запятую адресов. Может быть пусто.*

*Напишите скрипт, который выведет список представленных в таблице почтовых доменов с количеством пользователей по каждому домену».*

Первым делом я почитал про особенности движка InnoDB и узнал, что он не любит запросы вида: “Select \* from”. Дальше я решил написать пробный запрос наивным алгоритмом и посчитать его асимптотику, потому что моделировать 100 млн записей под нагрузкой показалось для меня непосильным.

Наивный алгоритм:

1. Вытащить у каждого пользователя список email’ов.
2. Запустить по ним курсор.
3. Вычленить домены с помощью “substring\_index”.
4. Если домен уже был добавлен, то кол-во = кол-во + 1, иначе добавить запись (домен, 1)

Так как у нас длина поля с почтовыми ящиками равна 1024, а минимальная длина названия ящика равна 3, то получается примерно 200 ящиков в худшем случае в строке. Тогда получается примерная сложность алгоритма O(n) \* O(1024 \* log(400n) = 10^8 + 1024\*35 ~ O(n) с большой константой, т.к. операции в MySQL нельзя назвать уж очень лёгкими.

Я не очень осведомлён в распределённых системах, но видимо этот алгоритм можно частично распараллелить. Как минимум его первый шаг: можно разбить всех пользователей на 4-8 групп (по кол-ву потоков) и получить 4 – 8 таблиц доменов, которые потом склеить в одну. Таким образом, можно получить улучшение времени работы алгоритма.

## Реализация решения

Никакого другого алгоритма помимо наивного мне придумать не удалось, поэтому я решил реализовать именно его (если есть какой-нибудь алгоритм, отличный от этого, то, пожалуйста, поделитесь).

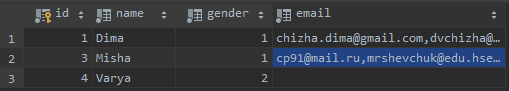
Я разбил его на 1 функцию и 2 процедуры

1. count\_str (str varchar(1024), needle varchar(32)) – функция считает количество подстрок в строку “str”, разделённых “needle”
2. parse\_emails (emailsList varchar(1024)) – процедура, которая разбивает строку на почтовые ящики, вычленяет домены и записывает их в таблицу.
3. extract\_domains () – процедура, которая инициирует выделение доменов из экземпляров сущности «users», запуская курсор по полю «email».

Код запросов находится в приложении (см.прил.G)

## Тестирование

При тестировании был найден очень странный баг, который мне так и не удалось исправить. Дело в том, что вызовы по отдельности “parse\_email” отрабатывают хорошо и, когда в таблице «users» последний экземпляр имеет пустое поле с емейлами, то тоже всё хорошо, но если последний элемент содержит не пустое поле «email», то скрипт считает его дважды:

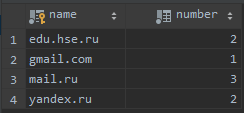


**Рисунок 2.1 Исходные данные (корректно работает)**

SQL-запрос: insert into users (name, gender, email) values  
('Dima', 1, 'chizha.dima@gmail.com,dvchizha@edu.hse.ru,FilaCo159@yandex.ru,chizha\_dmitry@mail.ru,chizha.dima@yandex.ru'),

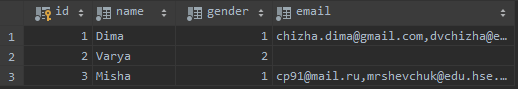
('Misha', 1, 'cp91@mail.ru,mrshevchuk@edu.hse.ru,singer1077@mail.ru'),  
('Varya', 2, '');

Результат:



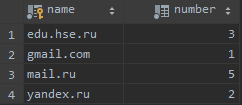
**Рисунок 2.2. Результат корректный**

Поменяем местами Варю и Мишу (не изменяя их email’ы):



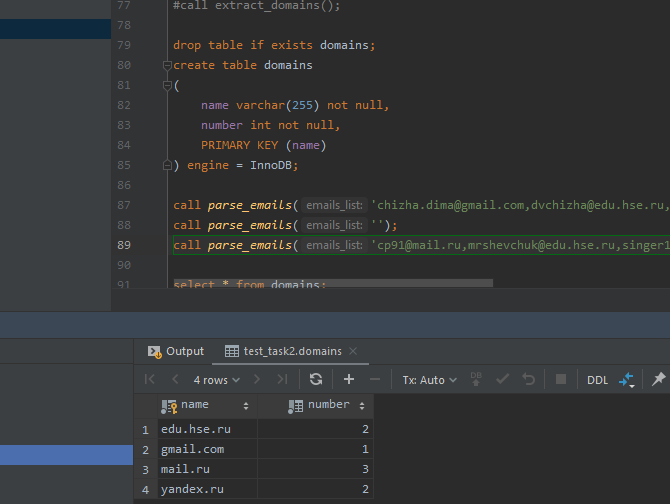
**Рисунок 2.3. Тест с некорректным ответом**

И результат меняется (!):



**Рисунок 2.4. Результат некорректный**

Если вызвать 3 раза подряд parse\_emails, то всё опять нормально:



# Итератор для чтения файла

## Анализ задачи

***Формулировка задачи:***

«*Дан текстовый файл размером 2ГБ. Напишите класс, реализующий интерфейс SeekableIterator, для чтения данного файла».*

SeekableIterator – это итератор с произвольным доступом к перебираемой коллекции. То есть он объявляет метод seek($position), который переносит указатель итератора на элемент на позиции $position.

Размер файла в 2 ГБ как бы намекает на то, что нежелательно загружать весь файл в оперативную память, а также практически накладывает ограничения на функции работы с файлами, например, ftell(), которая может вернуть указатель только в пределах положительной части целого числа (как раз 2 ГБ).

Есть несколько стандартных способов считывания файла:

1. Полностью в строку (данный способ сразу отпадает);
2. Построчно;
3. По блокам фиксированного размера.

Я решил остановиться на последнем способе, потому что он позволяет задать оптимальный размер блоков для разбиения файла, независящих от его содержания, в то время как в файле может быть одна строка во весь файл 2 ГБ, что приводит нас к первому способу.

## Проектирование решения

Итого необходимо реализовать следующий интерфейс:

SeekableIterator extends Iterator {

/\* Методы \*/

abstract public seek ( int $position ) : void

/\* Наследуемые методы \*/

abstract public Iterator::current ( void ) : mixed

abstract public Iterator::key ( void ) : scalar

abstract public Iterator::next ( void ) : void

abstract public Iterator::rewind ( void ) : void

abstract public Iterator::valid ( void ) : bool

}

Сделаем это с помощью стандартных функций работы с дескриптором файла.

## Реализация решения

Итоговую реализацию смотри в папке ./SeekableIterator.

## Тестирование

Для тестирования я воссоздал файл размером около 2 гб и проверил работоспособность итератора. Итератор справился с задачей, но довольно медленно, возможно я не достаточно удачно взял размер блока в 255.

# Библиографический список

1. Шаврин С.М., Лядова Л.Н., Чуприна С.И. Моделирование и проектирование информационных систем.
2. Проектирование новостной ленты в социальных сетях [Электронный ресурс] <https://habr.com/ru/company/dataart/blog/242593/>

# Расширенное описание прецедентов Задания 1

1. *Отфильтровать новости по категориям.*

Акторы: Пользователь.

Краткое описание: Пользователь открывает ленту новостей. Затем открывает меню выбора фильтров и выбирает желаемые категории постов для поиска. После подтверждения выбора система возвращает ленту с новостями только из выбранных категорий.

Триггер: Пользователь открывает ленту новостей и переходит в меню выбора фильтров.

Основной поток:

Таблица A.1 Расширенное описание прецедента "Отфильтровать новости по категориям"

|  |  |
| --- | --- |
| **Действия акторов** | **Отклик системы** |
| 1. Пользователь выбирает необходимые ему категории новостей (E1) | 2. Система отмечает выбранные пользователем категории |
| 3. Пользователь подтверждает окончание выбора категорий | 4. Система выводит список новостей (E2), которые относятся к заданным пользователем категориям. |

Альтернативные потоки:

E1: Категории, которая интересна пользователю, не оказалось в системе. Прецедент завершается.

E2: Новостей по указанным пользователем категориям не оказалось в системе. Прецедент завершается.

1. *Оценить пост*

Акторы: Пользователь.

Краткое описание: Пользователь открывает ленту новостей. Просматривая её, пользователь находит посты, которые ему особенно нравятся. Пользователь отмечает пост «лайком».

Триггер: Пользователь открывает ленту новостей и решает оценить просматриваемый пост.

Основной поток:

Таблица A.2 Расширенное описание прецедента "Оценить пост"

|  |  |
| --- | --- |
| **Действия акторов** | **Отклик системы** |
| 1. Пользователь выбирает пост для оценки и нажимает на кнопку «лайка» | 2. Система помечает новость, как «понравившуюся» (E1). |

Альтернативные потоки:

E1: Если пост уже был помечен как «понравившийся», то «лайк» отменяется.

1. *Просмотреть список всех оценивших пост пользователей.*

Акторы: Пользователь.

Краткое описание: Пользователь открывает ленту новостей. Просматривая её, пользователь находит посты, которые уже оценили другие люди до него. Пользователь открывает список оценивших новость людей и просматривает его. Затем он возвращается к просмотру ленты.

Триггер: Пользователь выбирает пост в ленте и открывает список оценивших новость людей.

Основной поток:

Таблица A.3 Расширенное описание прецедента "Просмотреть список всех оценивших пост пользователей "

|  |  |
| --- | --- |
| **Действия акторов** | **Отклик системы** |
| 1. Пользователь открывает список оценивших новость людей. | 2. Система возвращает список пользователей, которые поставили выбранному посту «лайк» (E1) |

Альтернативные потоки:

E1: Если пост ещё никто не «лайкал», то выводится соответствующее сообщение.

1. *Вывести все новости.*

Акторы: Пользователь.

Краткое описание: Пользователь открывает ленту новостей.

Основной поток:

Таблица A.4 Расширенное описание прецедента "Вывести все новости"

| **Действия акторов** | **Отклик системы** |
| --- | --- |
| 1. Пользователь открывает ленту новостей | 2. Система возвращает последние 50 новостей (E1, E2) |

Альтернативные потоки:

E1: Если новостей ещё нет, то выводится соответствующее сообщение.

E2: Если постов меньше 50, то выводятся все новости.

1. *Добавить новый пост в ленту.*

Акторы: Админ.

Краткое описание: Админ решает создать новый пост. Для этого он переходит в режим добавления новости в ленту, выбирает для неё одну или несколько категорий и пишет текст. Далее админ подтверждает своё действие и новость добавляется в общую ленту. Другие пользователи, в том числе и автор, могут эту новость просмотреть и оценить.

Триггер: Админ выбирает функцию «Добавить новость»

Основной поток:

Таблица A.5 Расширенное описание прецедента "Добавить новость"

|  |  |
| --- | --- |
| **Действия акторов** | **Отклик системы** |
| 1. Админ выбирает категорию(-ии) новости (E1), пишет её текст и подтверждает добавление новости. | 2. Система добавляет новость в общую ленту. |

Альтернативные потоки:

E1: Если необходимой категории для новости ещё нет, то Админу предлагается добавить новую категорию.

1. *Добавить новую категорию*.

Акторы: Админ.

Краткое описание: Админ решает добавить новую категорию для новостей. Для этого он переходит в режим добавления новой категории, выбирает новости, которые относятся к этой категории. Далее админ подтверждает своё действие, и категория добавляется в общий список. Другие пользователи могут использовать эту категорию при фильтрации новостей.

Триггер: Админ выбирает функцию «Добавить категорию»

Основной поток:

Таблица A.6 Расширенное описание прецедента "Добавить категорию"

|  |  |
| --- | --- |
| **Действия акторов** | **Отклик системы** |
| 1. Админ набирает в текстовом поле название категории. | 2. Система предлагает (E1) прикрепить категорию к подходящим новостям |
| 3. Админ прикрепляет новости к новой категории (S1) и подтверждает создание новой категории. | 4. Система добавляет новую категорию в список. |

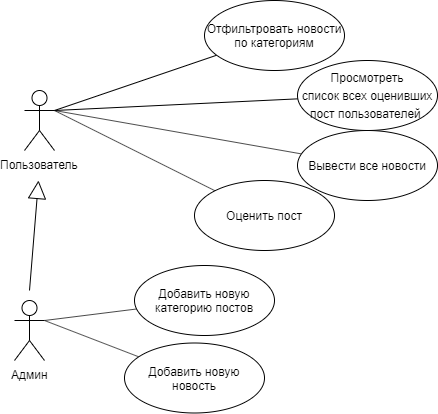
Альтернативные потоки:

E1: Если категория с данным названием уже существует, то Админу предлагается выбрать другое имя.

Подпотоки:

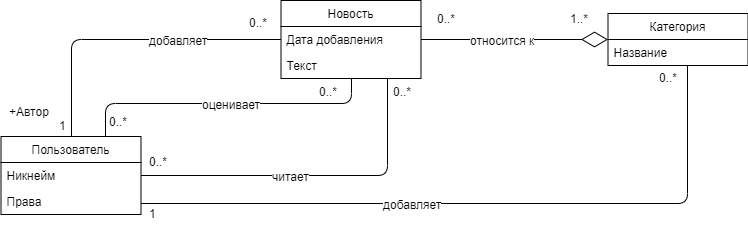
S1: Прикрепление новой новости к категории. Админ выбирает новость в списке и кликает на неё. Система отмечает данный пост, как выбранный.

# Диаграмма прецедентов Задания 1



**Рисунок B.1. Диаграмма прецедентов**

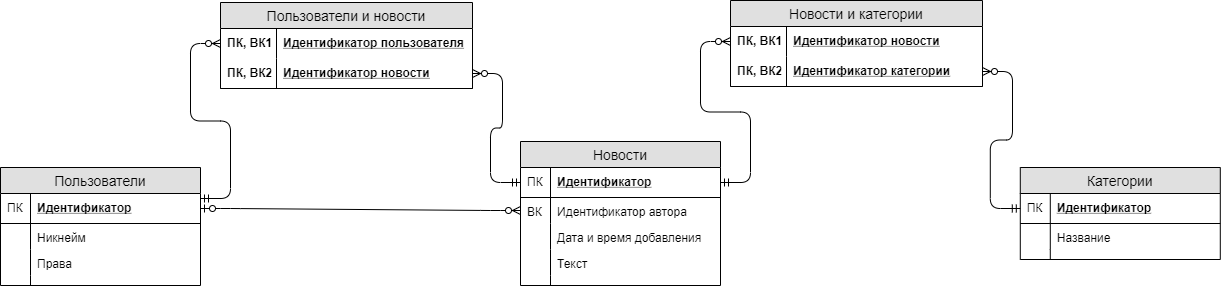
# Диаграмма понятий Задания 1



**Рисунок C.1 Диаграмма понятий**

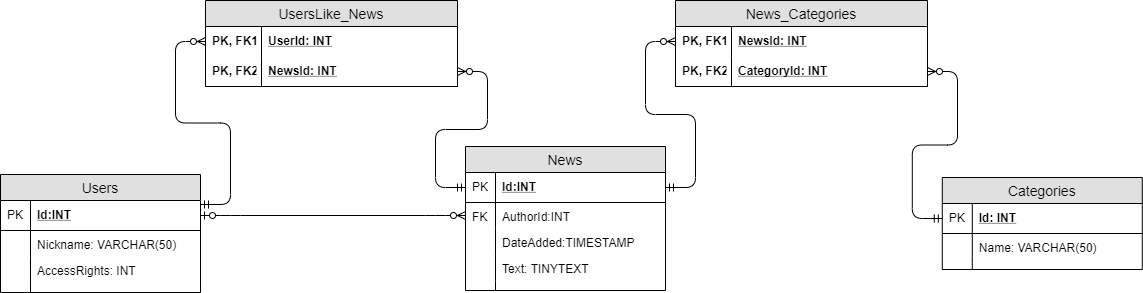
Примечание: «+Автор» у связи «Пользователь - Новость» обозначает роль Пользователя в данной связи.

# Логическая модель БД Задания 1



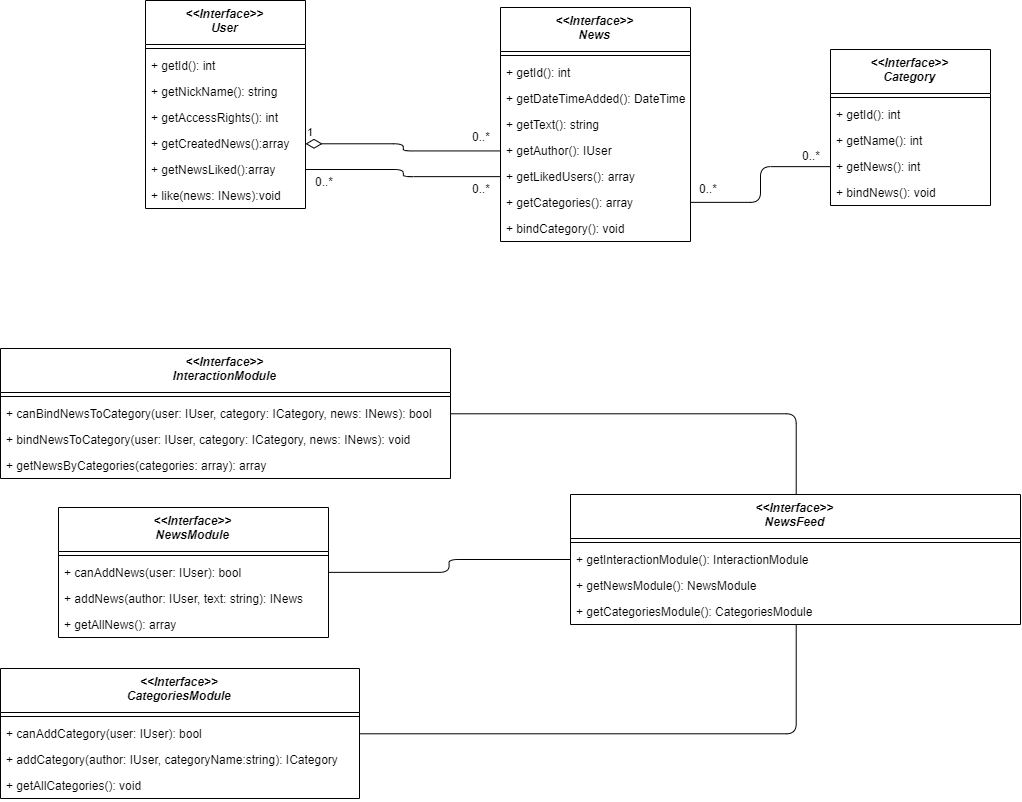
**Рисунок D.1 Логическая модель БД**

# Физическая модель БД Задания 1



**Рисунок E.1 Физическая модель БД**

# Диаграмма классов (неполная)



# Код запросов MySQL (задание 2)

drop procedure if exists *parse\_emails*;  
drop procedure if exists *extract\_domains*;  
drop function if exists *count\_str*;  
  
delimiter //  
create function *count\_str*(haystack VARCHAR(1024), needle VARCHAR(32))  
 returns integer deterministic  
begin  
 return *round*((*char\_length*(haystack) - *char\_length*(*replace*(haystack, needle, ""))) / *char\_length*(needle));  
end//  
  
  
delimiter //  
create procedure *parse\_emails*(emails\_list varchar(1024))  
begin  
 declare len int unsigned;  
 declare cur\_pos int unsigned default 0;  
 declare left\_side\_emails varchar(1024);  
 declare cur\_email varchar(255);  
 declare cur\_domain varchar(255);  
  
 case  
 when  
 *length*(emails\_list) < 1  
 then  
 set len = 0;  
 else  
 set len = *count\_str*(emails\_list, ',') + 1;  
 end case;  
  
 while cur\_pos < len  
 do  
 set left\_side\_emails = *substring\_index*(emails\_list, ',', cur\_pos + 1);  
 set cur\_email = *substring\_index*(left\_side\_emails, ',', -1);  
 set cur\_domain = *substring\_index*(cur\_email, '@', -1);  
  
 case  
 when  
 not *exists*(select \* from domains where name = cur\_domain)  
 then  
 insert into domains (name, number) value (cur\_domain, 1);  
 else  
 update domains set number = number + 1 where name = cur\_domain;  
 end case;  
  
 set cur\_pos = cur\_pos + 1;  
 end while;  
end //  
  
delimiter //  
create procedure *extract\_domains*()  
begin  
 declare done bool default false;  
 declare emailListItem varchar(1024);  
 declare cursorEmailList cursor for select email from users;  
 declare continue handler for not found set done = true;  
  
 drop table if exists domains;  
 create table domains  
 (  
 name varchar(255) not null,  
 number int not null,  
 PRIMARY KEY (name)  
 ) engine = InnoDB;  
  
 open cursorEmailList;  
  
 while done = false  
 do  
 fetch cursorEmailList into emailListItem;  
 call *parse\_emails*(emailListItem);  
 end while;  
  
 close cursorEmailList;  
end //  
  
  
  
call *extract\_domains*();  
  
select \* from domains;

# Диаграмма классов (неполная)