Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

**ПОЯСНИТЕЛЬНАЯ** **ЗАПИСКА**

к курсовому проекту

на тему

**«Веб-приложение для взаимодействия учителей и родителей в школе»**

Студент Ермолович Дмитрий Сергеевич

Гр. 053504

 Ассистент кафедры информатики

Руководитель Давыдчик А.В.

Минск 2023

# **ВВЕДЕНИЕ**

# 

В современном образовательном пространстве сотрудничество между учителями и родителями играет ключевую роль в обеспечении успешности учебного процесса и развитии учеников. Однако, несмотря на значимость этой взаимодействия, оно часто сталкивается с различными вызовами, связанными с ограниченной доступностью времени, общением и координацией. В этом контексте, разработка и внедрение приложения для взаимодействия учителей и родителей в школе представляет собой актуальную и значимую задачу, которая может улучшить образовательный процесс и усилить вовлеченность всех участников.

Данное приложение может стать эффективным инструментом, способствующим облегчению коммуникации и сотрудничества между учителями и родителями. Оно предоставит возможность учителям оперативно и информативно обмениваться информацией о прогрессе и достижениях учеников, а также давать обратную связь по учебным вопросам. Родителям, в свою очередь, будет предоставлена уникальная возможность более активно включаться в учебный процесс своих детей, следить за их успехами, а также общаться с учителями и администрацией школы.

В данной курсовой работе будут рассмотрены основные аспекты проектирования и разработки такого приложения, а также его потенциальные преимущества и вызовы. Мы также рассмотрим современные технологические решения, которые могут быть использованы для создания данного приложения, и практические шаги, необходимые для его успешной реализации.

Цель данного исследования - дать вклад в улучшение коммуникации и сотрудничества между учителями и родителями в образовательных учреждениях, что, в свою очередь, содействует повышению качества образования и успешному развитию учеников.

**1 ОПИСАНИЕ БАЗ ДАННЫХ И СВЯЗЕЙ МЕЖДУ НИМИ**

**1.1 Описание баз данных**

Таблица user содержит два поля логин и пароль. Когда человек заходит в приложение он авторизуется. Пароли и логины должны быть уникальными в целях безопасности и не должны быть нулевыми.

Таблица parent содержит имя родителя и его gmail для связи с ним.

Таблица administrator содержит имя администратора.

Таблица school\_meetings. Иногда учитель хочет пригласить на беседу родителя ученика, для этого он вносит в систему свой id, id родителя и причину встречи.

Таблица complaints. Иногда ученик не доволен учителем, он может составить жлобу на него. Для этого он вносит в систему свой id, id учителя и жалобу.

Таблица student содержит имя студента, id класса в котором он учится, id его учителя, id его родителя, id дневника.

Таблица teacher содержит имя учителя.

Таблица class содержит название класса и количество учеников в нем.

Таблица diary содержит оценки ученика по математике и по английскому.

Таблица high\_school\_graduation содержит id класса, количество людей на выпускном, сколько на выпускной было потрачено денег, когда он будет или был проведен, специальные гости.

Таблица item содержит информацию о продукте, который покупался в школьной столовой. Содержит количество проданной продукции, сколько осталось на складе, моржа.

Таблица Subbotnik содержит информацию о субботнике, который проводит школа. Содержит количество людей и людей, кто был на данном мероприятии.

Таблица Olimpiada содержит информацию о олимпиаде, в которой принимали ученики школы. Содержит имя олимпиады, id ученика и место на олимпиаде.

Таблица competition содержит информацию соревнованиях, в которых школа принимает участие. Содержит имя соревнования, кто принимал участие в олимпиаде, количество человек, место на олимпиаде.

Таблица Graduates содержит информацию о выпускниках школы. Содержит id ученика, университет, в который он поступил, результаты цт, специальность.

**1.2 Связи в базах данных**

Связи делятся на:

1 Многие ко многим.

2 Один ко многим.

3 Один к одному.

Связь "один ко многим" являются наиболее распространенным типом связи. В такого рода связях строка в таблице А может иметь много строк в таблице B. Но строка в таблице B может иметь только одну строку в таблице А.

Связь многие-ко-многим – это связь, при которой множественным записям из одной таблицы (A) могут соответствовать множественные записи из другой (B).

Связи «один к одному» часто используются для получения важных данных, необходимых для ведения бизнеса. Связь "один-к-одному" — это связь между информацией из двух таблиц, когда каждая запись используется в каждой таблице только один раз.

Рассмотрим таблицу *student*, она имеет class\_id, teacher\_id, parent\_id, diary\_id. Таблица student является связанной, a таблицы class, teacher, diary, parents – главные таблицы. Таблица student и class связаны через foreign key class\_id и имеют связь один ко многим. Таблица student и teacher связаны через foreign key teacher\_id и имеют связь один ко многим. Таблица student и parent связаны через foreign key parent\_id и имеют связь один ко многим, потому что могут быть близнецы. Таблица student и diary связаны через foreign key diary\_id и имеют связь один к одному.

Рассмотрим таблицы class и teacher. Они   
связаны связью многие ко многим. Для этого была создана таблица class\_teacher, чтобы обеспечить эту связь.

Рассмотрим таблицу school\_meetings, она имеет parend\_id, teacher\_id. Таблица school\_meetings и parent связаны через foreign key parend \_id и имеют связь один ко многим. Таблица school\_meetings и teacher связаны через foreign key teacher\_id и имеют связь один ко многим.

Рассмотрим таблицу complaints, она имеет student\_id, teacher\_id. Таблица school\_meetings и parent связаны через foreign key student\_id и имеют связь один ко многим. Таблица school\_meetings и teacher связаны через foreign key teacher\_id и имеют связь один ко многим.

Рассмотрим таблицу high\_school\_graduation, она имеет class\_id. Таблица high\_school\_graduation и class связаны через foreign key class \_id и имеют связь один ко многим.

Рассмотрим таблицу Olimpiada, она имеет student\_id. Таблица Olimpiada и student связаны через foreign key student\_id и имеют связь один ко многим.

**1.3 Триггеры**

Триггеры представляют специальный тип хранимой процедуры, которая вызывается автоматически при выполнении определенного действия над таблицей или представлением, в частности, при добавлении, изменении или удалении данных, то есть при выполнении команд INSERT, UPDATE, DELETE.

Триггер insert\_into\_users\_parent, после добавления родителя в таблицу parents, происходит автоматическое создание его аккаунта, то есть создание уникального логина и пароля и запись этой информации в таблицу users.

Триггер bahavior\_check, после добавления студента приводит к правильному формату его поведение, то есть к диапазону [0..5].

Триггер check\_first\_letter, после добавления студента проверяет формат ввод имени и если имя начинается с маленькой буквы, то переводит первую букву в высокий регистр.

Триггер insert\_into\_users\_ teacher, после добавления учителя в таблицу teacher, происходит автоматическое создание его аккаунта, то есть создание уникального логина и пароля и запись этой информации в таблицу users.

Триггер insert\_into\_users\_student, после добавления студента в таблицу student, происходит автоматическое создание его аккаунта, то есть создание уникального логина и пароля и запись этой информации в таблицу users.

**2 РОЛИ И ИХ ФУНКЦИИ**

Администратор.

Администратор имеет следующие функции. Добавить нового учителя, добавить дневник,добавить студента, удалить студента, добавит родитедя, добавить юзера, удалить юзера, выбрать всех юзеров, учителей, студентов, родителей, найти ids по именам, добавить администратора.

На рисунке 1 изображены функции администратора.

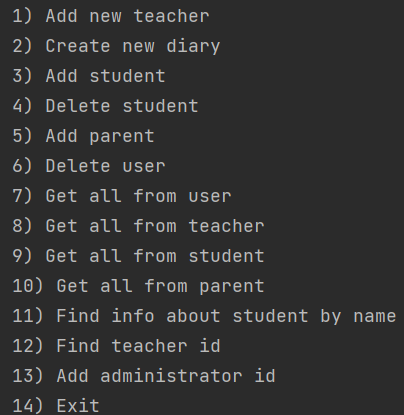


Рисунок 1 - Функции администратора

Студент.

Студент имеет следующие функции. Добавить жалобу, получить оценки свои и при желании любого человека, найти ids любого человека

На рисунке 2 изображены функции студентa.

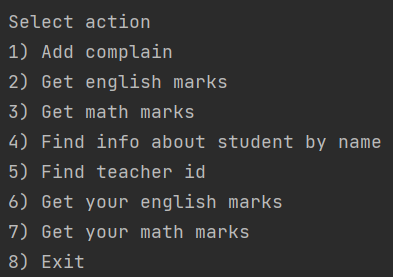


Рисунок 2 - Функции студентa

Учитель.

Поставить оценку по английскому и математике только студентам у которого он преподаёт, найти студента и учителя, у каких учителей преподаёт, вывести всех студентов класса.

На рисунке 3 изображены функции учителя.

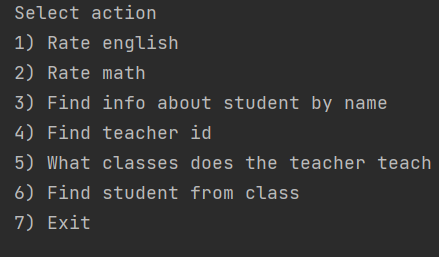


Рисунок 3 - Функции учителя

Родители.

Добавить встречу, получить оценки твоего ребенка или любого, найти учителей и студентов.

На рисунке 4 изображены функции родителя.

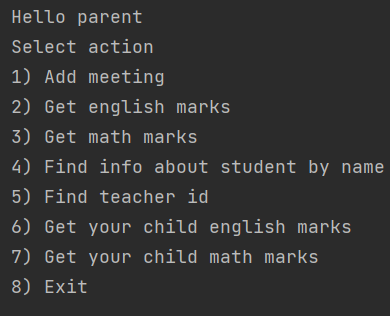


Рисунок 4 - Функции родителя

**3 Нормализация данных и нормальные формы**

Нормальная форма — требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц).

Метод нормальных форм (НФ) состоит в сборе информации о объектах решения задачи в рамках одного отношения и последующей декомпозиции этого отношения на несколько взаимосвязанных отношений на основе процедур нормализации отношений.

Цель нормализации: исключить избыточное дублирование данных, которое является причиной аномалий, возникших при добавлении, редактировании и удалении кортежей(строк таблицы).

**Первая нормальная форма.**

Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

**Вторая нормальная форма.**

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК).

Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

**Третья нормальная форма.**

Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

Нормальная форма Бойса-Кодда (НФБК) (частная форма третьей нормальной формы)

Определение 3НФ не совсем подходит для следующих отношений:

1 Отношение имеет два или более потенциальных ключа.

2 Два и более потенциальных ключа являются составными.

3 Они пересекаются, т.е. имеют хотя бы один общий атрибут.

Для отношений, имеющих один потенциальный ключ (первичный), НФБК является 3НФ.

Отношение находится в НФБК, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта.

Моя база данных была сведена к второй нормальной форме. Вторая нормальная форма (2NF) — это стадия нормализации баз данных, нацеленная на устранение избыточности данных и обеспечение более эффективного хранения информации. Вторая нормальная форма часто является важным шагом в проектировании баз данных, особенно когда имеются составные ключи.

Была выбрана именно вторая нормальная форма, потому что она имеет ряд приемуществ, такие как:

1 Избыточность данных. Одной из ключевых целей нормализации является устранение избыточности данных. Вторая нормальная форма помогает в избежании множественного хранения одной и той же информации.

2 Повышение эффективности обновлений. При использовании 2NF, изменения в данных часто требуются только в одной таблице, что уменьшает риск ошибок при обновлении.

3 Более ясная структура. 2NF способствует более четкой и ясной структуре базы данных, что облегчает понимание и обслуживание системы.

4 Сохранение семантической целостности. Этот уровень нормализации помогает сохранить семантическую целостность данных, так как каждый атрибут в таблице зависит только от ключа этой таблицы.

5 Поддержка индексирования. Использование второй нормальной формы упрощает процесс индексирования таблиц, что может значительно повысить производительность запросов к базе данных.

**4 Этапы проектирование базы данных**

Процесс проектирования базы данных состоит из трех основных этапов: концептуальное, логическое и физическое проектирование.

**4.1 Концептуальное проектирование базы данных**

Концептуальное проектирование базы данных. Процесс создания модели используемой на предприятии информации, не зависящей от любых физических аспектов ее представления.

Первый этап процесса проектирования базы данных называется концептуальным проектированием базы данных. Он заключается в создании концептуальной модели данных для анализируемой части предприятия. Эта модель данных создается на основе информации, записанной в спецификациях требований пользователей. Концептуальное проектирование базы данных абсолютно не зависит от таких подробностей ее реализации, как тип выбранной целевой СУБД, набор создаваемых прикладных программ, используемые языки программирования, тип выбранной вычислительной платформы, а также от любых других особенностей физической реализации.

При разработке концептуальная модель данных постоянно подвергается тестированию и проверке на соответствие требованиям пользователей. Созданная концептуальная модель данных предприятия является источником информации для этапа логического проектирования базы данных.

**4.2 Логическое проектирование базы данных**

Логическое проектирование базы данных. Процесс создания модели используемой на предприятии информации на основе выбранной модели организации данных, но без учета типа целевой СУБД и других физических аспектов реализации.

Второй этап проектирования базы данных называется логическим проектированием базы данных. Его цель состоит в создании логической модели данных для исследуемой части предприятия. Концептуальная модель данных, созданная на предыдущем этапе, уточняется и преобразуется в логическую модель данных. Логическая модель данных учитывает особенности выбранной модели организации данных в целевой СУБД (например, реляционная модель).

Если концептуальная модель данных не зависит от любых физических аспектов реализации, то логическая модель данных создается на основе выбранной модели организации данных целевой СУБД. Иначе говоря, на этом этапе уже должно быть известно, какая СУБД будет использоваться в качестве целевой - реляционная, сетевая, иерархическая или объектно-ориентированная. Однако на этом этапе игнорируются все остальные характеристики выбранной СУБД, например, любые особенности физической организации ее структур хранения данных и построения индексов.

В процессе разработки логическая модель данных постоянно тестируется и проверяется на соответствие требованиям пользователей. Для проверки правильносги логической модели данных используется метод нормализации. Нормализация гарантирует, что отношения, выведенные из существующей модели данных, не будут обладать избыточностью данных, способной вызвать нарушения в процессе обновления данных после их физической реализации. Помимо всего прочего, логическая модель данных должна обеспечивать поддержку всех необходимых пользователям транзакций.

Созданная логическая модель данных является источником информации для этапа физического проектирования и обеспечивает разработчика физической базы данных средствами поиска компромиссов, необходимых для достижения поставленных целей, что очень важно для эффективного проектирования. Логическая модель данных играет также важную роль на этапе эксплуатации и сопровождения уже готовой системы. При правильно организованном сопровождении поддерживаемая в актуальном состоянии модель данных позволяет точно и наглядно представить любые вносимые в базу данных изменения, а также оценить их влияние на прикладные программы и использование данных, уже имеющихся в базе.

**4.3 Физическое проектирование базы данных**

Физическое проектирование базы данных. Процесс подготовки описания реализации базы данных на вторичных запоминающих устройствах; на этом этапе рассматриваются основные отношения, организация файлов и индексов, предназначенных для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты.

Физическое проектирование является третьим и последним этапом создания проекта базы данных, при выполнении которого проектировщик принимает решения о способах реализации разрабатываемой базы данных. Во время предыдущего этапа проектирования была определена логическая структура базы данных (которая описывает отношения и ограничения в рассматриваемой прикладной области). Хотя эта структура не зависит от конкретной целевой СУБД, она создается с учетом выбранной модели хранения данных, например, реляционной, сетевой или иерархической. Однако, приступая к физическому проектированию базы данных, прежде всего необходимо выбрать конкретную целевую СУБД. Поэтому физическое проектирование неразрывно связано с конкретной СУБД. Между логическим и физическим проектированием существует постоянная обратная связь, так как решения, принимаемые на этапе физического проектирования с целью повышения производительности системы, способны повлиять на структуру логической модели данных.

Как правило, основной целью физического проектирования базы данных является описание способа физической реализации логического проекта базы данных. В случае реляционной модели данных под этим подразумевается следующее:

1 Создание набора реляционных таблиц и ограничений для них на основе информации, представленной в глобальной логической модели данных.

2 Определение конкретных структур хранения данных и методов доступа к ним, обеспечивающих оптимальную производительность СУБД;

3 Разработка средств защиты создаваемой системы.

Этапы концептуального и логического проектирования больших систем следует отделять от этапов физического проектирования. На это есть несколько причин.

1 Они связаны с совершенно разными аспектами системы, поскольку отвечают на вопрос, что делать, а не как делать.

2 Они выполняются в разное время, поскольку понять, что надо сделать, следует прежде, чем решить, как это сделать.

3 Они требуют совершенно разных навыков и опыта, поэтому требуют привлечения специалистов различного профиля.

Проектирование базы данных — это итерационный процесс, который имеет свое начало, но не имеет конца и состоит из бесконечного ряда уточнений. Его следует рассматривать прежде всего, как процесс познания. Как только проектировщик приходит к пониманию работы предприятия и смысла обрабатываемых данных, а также выражает это понимание средствами выбранной модели данных, приобретенные знания могут показать, что требуется уточнение и в других частях проекта. Особо важную роль в общем процессе успешного создания системы играет концептуальное и логическое проектирование базы данных. Если на этих этапах не удастся получить полное представление о деятельности предприятия, то задача определения всех необходимых пользовательских представлений или обеспечения защиты базы данных становится чрезмерно сложной или даже неосуществимой. К тому же может оказаться затруднительным определение способов физической реализации или достижения приемлемой производительности системы. С другой стороны, способность адаптироваться к изменениям является одним из признаков удачного проекта базы данных. Поэтому вполне имеет смысл затратить время и энергию, необходимые для подготовки наилучшего возможного проекта.

**5 ИНДЕКСЫ**

Индексы — это структура данных, которая помогает сократить время поиска запрошенных данных. Индексы достигают этого за счет дополнительных затрат на хранение, память и поддержание их в актуальном состоянии (более медленная запись), что позволяет нам пропустить утомительную проверку каждой строки таблицы.

Подобно указателю в конце учебника он помогает вам попасть на нужную страницу. Я не большой поклонник аналогии с книгой, так как сравнение быстро перестает быть корректным, когда мы углубляемся в индексы базы данных, но тем не менее аналогия позволяет отлично понять тему.

Небольшие объемы данных поддаются управлению (подумайте о списке посещаемости для небольшого класса), но когда они становятся больше (подумайте о реестре рождений для большого города), они становятся менее управляемыми. Все, что раньше было быстрым, становится медленным, слишком медленным.

В зависимости от типов баз данных, индексы могут быть очень разными и реализоваться за счет специфических математических механизмов. Но, наиболее частым является древовидный индекс, так как поддерживать такой индекс относительно просто и максимальная скорость поиска в нем составляет логарифм по числу максимального количества дочерних узлом от общего количества записей (плюс минус некоторые технические моменты).

Дерево (древовидный индекс) — это специального вида структура, у которой есть корневая вершина и у каждого узла может быть несколько дочерних узлов. При этом каждый узел встречается только один раз и может иметь всего один родительский узел.

Практически во всех базах данных, существует деление по уникальности:

Уникальный индекс — это такой индекс, у которого все значения встречаются только один раз. Проводя аналогию, когда каждая книга присутствует только в одном экземпляре и никогда названия книг не совпадают.

Неуникальный индекс — это такой индекс, у которого значения могут повторяться. Проводя аналогию, существуют книги с одними и теми же названиями, но разными авторами, или же просто встречаются копии.

Важно отметить, что если для таблицы создается уникальный индекс, то это означает, что при попытке добавить запись со значением, которое уже встречалось, или же изменить значение какой-то записи на существующее, то база данных не позволит сделать такое действие и будет ругаться (выдавать ошибки). В случае же с неуникальным индексом таких проблем нет.

Так же стоит знать, что индексы делятся по количеству входящих в них полей:

Обычные индексы — состоят из одного поля. Здесь, вероятно, все понятно. Обычный каталог страничек.

Составные индексы — строятся по нескольким полям, при этом расположение полей является важным.

**6 КЛАСТЕР БАЗ ДАННЫХ**

Решение для повышения отказоустойчивости и доступности корпоративных информационных систем. Основой решения является программный продукт Microsoft Cluster Service (CS). Отказоустойчивость достигается за счет дублирования всех активных компонентов и встроенной системы мониторинга работоспособности. В сочетании с адаптивной архитектурой серверных компонент R-IT решение может быть эффективно использовано для задач различного уровня сложности.

Основная цель использования кластера – обеспечение высокой доступности базы данных. Сегодня для приложений всё чаще выдвигаются такие бизнес – требования, как обеспечение доступа к данным в режиме 24/7, и недоступность базы данных из-за выхода из строя оборудования или из-за необходимости выполнения операций по обслуживанию сервера часто просто недопустима. Использование кластера серверов баз данных может помочь предотвратить недоступность данных из-за выхода из строя сервера, вызванного сбоем в программном обеспечении, необходимостью выполнения операций по обслуживанию сервера или из-за потери сетевого соединения с сервером. Использование кластеризации позволяет повысить на порядок уровень доступности баз данных MS SQL Server и СУБД Oracle.

**6.1 Область применения**

Решение идеально подходит для построения на его основе высокодоступных корпоративных систем управления, таких как различные конфигурации 1С, MS Dynamics NAV (Navision) и AХ (Axapta), Oracle E-Business Suite и прочих, включая системы собственной разработки, использующие MS SQL Server и СУБД Oracle.

Данное решение также может быть использовано для кластеризации различных серверных приложений и сервисов.

**6.2 Как устроен кластер**

В состав кластера входит два и более узлов (серверов), каждый из которых конфигурируется таким образом, чтобы приложение (в данном случае SQL Server) могло работать на любом из них. При этом само приложение виртуализируется, т.е. становится независимым от какого-либо узла. Обязательным условием является наличие общей для всех узлов системы хранения. Наиболее часто для этого используется архитектура SAN. Основное приложение и все необходимые для его работы ресурсы, такие как файловые ресурсы или сетевое подключение, определяются в общую кластерную группу. В случае недоступности одного из ресурсов кластерной группы управляющее приложение инициирует перевод работы основного приложения и всей кластерной группы на другой узел.

Для мониторинга состояния узлов кластер использует как открытую локальную, так и кластерную сеть. Каждые 5 секунд выполняется проверка LooksAlive (как живой), а каждые 60 секунд – более глубокая проверка IsAlive (действительно живой). В случае неуспешности проверка выполняется еще 5 раз, после чего кластер переведет группу на другой узел, передав другому серверу в монопольное владение все ресурсы данной группы. После того, как группа ресурсов будет передана, экземпляр SQL Server инициирует процедуру стартовой регенерации (recovery), после исполнения recovery для баз данных SQL Server переходит в состояние готовности к работе и обслуживанию запросов пользователей.

Пользовательские приложения должны будут выполнить повторное подключение к серверу баз данных. Приложение, которое соединяется с экземпляром SQL Server, использует виртуальный, а не физический IP - адрес. Фактическая принадлежность виртуального IP - адреса управляется кластером, поэтому приложение никогда не будет знать, на каком узле фактически находится адресуемый экземпляр.

**Преимущества использования:**

1 Предотвращение сбоев.Главным преимуществом от установки серверов баз данных в кластер является исключение длительного простоя в работе приложений, вызванного всевозможными отказами аппаратных средств, которые весьма вероятны для современных серверов, сложность которых постоянно растёт. Часто совсем маленькая проблема в состоянии вывести операционную систему из строя на длительный срок, причём подобные отказы не нуждаются в тщательном расследовании или переустановке компонентов или даже всего сервера, но они бывают достаточно серьезны, чтобы приложение оказалось неработоспособным на недопустимое время. Кластер может помочь в предотвращении многих подобных проблем в работе приложений, поскольку ресурсы приложения могут быть быстро переброшены на другой узел кластера, и часто сделать это можно даже без потери клиентских подключений.

2 Сервисные пакеты.Cервисные пакеты как операционной системы Windows, так SQL Server являются причиной простоя приложений, который может возникнуть во время установки и перезагрузки сервера после инсталляции такого пакета. При использовании кластера администраторы могут установить сервисный пакет практически без простоя приложений, инициируя отказ узла, вследствие которого база данных будет обслуживаться другим узлом, что позволит заняться установкой сервисного пакета на первом узле. В это время база данных будет доступна на втором узле, а экземпляр на первом узле может быть недоступен на время перезагрузок или иных действий.

3 Модернизация.Нередко возникает необходимость увеличения производительности серверов. Это означает необходимость миграции и продолжительного простоя. При использовании кластера миграцию выполнить намного легче и с минимальным временем простоя. В кластер добавляется новый узел, выполняется установка всех необходимых обновлений. Затем с помощью процедуры перехода на резервный ресурс выполняется перенос сервера на новый узел, а старый исключается из кластера. Время простоя ограничивается временем перехода на резервный ресурс, т.е. лишь несколько минут, вместо нескольких часов или даже нескольких суток без использования кластера.

Выполняемые операции

1 Аудит текущей инфраструктуры, выполнение операций для приведения ее в соответствие с требованиями к внедрению кластерной системы.

2 Проектирование и планирование инсталляции в привязке к специфике клиента.

3 Подбор и аппаратное конфигурирование серверного оборудования, в привязке к различному функционалу и планирующейся нагрузке.

4 Сборка и монтаж оборудования на площадке заказчика, установка и настройка кластерного ПО и сопутствующих служб.

5 Кабельная коммутация открытой и кластерной сетей.

6 Настройка коммутационного оборудования.

7 Настройка технологий высокой доступности, отказоустойчивости и резервного копирования.

8 Мониторинг начальной работы системы, выявление и устранение возникающих проблем на этапе опытной эксплуатации.

9 Подготовка документарных сценариев текущего операционного обслуживания и восстановления в случае сбоев.

10 Проведение тренинга по работе с системой для ИТ-персонала.

**7 SQL — ОГРАНИЧЕНИЯ**

Ограничения SQL — это правила, применяемые к столбцам данных таблицы. Они используются, чтобы ограничить типы данных, которые могут храниться в таблице. Это обеспечивает точность и надежность данных в базе данных.

Ограничения могут применяться либо на уровне столбцов, либо на уровне таблицы. Ограничения на уровне столбца применяются только к одному столбцу, тогда как ограничения уровне таблицы применяются ко всей таблице.

Ниже приведены некоторые из наиболее часто используемых ограничений, доступных в SQL.

1 NOT NULL Constraint — столбец не может иметь значение NULL.

2 DEFAULT Constraint — задает значение по умолчанию для столбца, если оно не указано.

3 UNIQUE Constraint — все значения в столбце должны быть разными.

4 PRIMARY Key — уникальная идентификация каждой строки/записи в таблице базы данных.

5 FOREIGN Key — уникально идентифицирует строку/запись в любой другой таблице базы данных.

6 CHECK Constraint — ограничение CHECK обеспечивает, чтобы все значения в столбце удовлетворяли определенным условиям.

7 INDEX — используется для быстрого создания данных базы данных.

Ограничения могут указываться при создании таблицы с помощью оператора CREATE TABLE или вы можете использовать оператор ALTER TABLE для создания ограничений уже после создания таблицы.

Удаление ограничений.

Любое ограничение, которое вы определили, можно удалить с помощью команды ALTER TABLE с параметром DROP CONSTRAINT.

Ограничения целостности.

Ограничения целостности используются для обеспечения точности и согласованности данных в реляционной базе данных. Целостность данных обрабатывается в реляционной базе данных посредством концепции ссылочной целостности.

Существует множество типов ограничений целостности, которые являются частью общей системы Ссылочной целостности (RI): ограничения по первичному ключу, по внешнему ключу, по уникальности и другие ограничения, которые были перечислены выше.

**8 ХРАНИМАЯ ПРОЦЕДУРА**

Хранимая процедура — объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Хранимые процедуры похожи на определяемые пользователем функции (UDF). Основное различие заключается в том, что пользовательские функции можно использовать как и любое другое выражение в SQL запросе, в то время как хранимые процедуры должны быть вызваны с помощью функции CALL.

Хранимые процедуры могут возвращать множества результатов, то есть результаты запроса SELECT. Такие множества результатов могут обрабатываться, используя курсоры, другими хранимыми процедурами, возвращая указатель результирующего множества, либо же приложениями. Хранимые процедуры могут также содержать объявленные переменные для обработки данных и курсоров, которые позволяют организовать цикл по нескольким строкам в таблице. Стандарт SQL предоставляет для работы выражения IF, LOOP, REPEAT, CASE и многие другие. Хранимые процедуры могут принимать переменные, возвращать результаты или изменять переменные и возвращать их, в зависимости от того, где переменная объявлена.

Реализация хранимых процедур варьируется от одной СУБД к другой. Большинство крупных поставщиков баз данных поддерживают их в той или иной форме. В зависимости от СУБД, хранимые процедуры могут быть реализованы на различных языках программирования, таких, как SQL, Java, C или C++. Хранимые процедуры, написанные не на SQL, могут самостоятельно выполнять SQL-запросы, а могут и не выполнять. Всё более широкое использование хранимых процедур привело к появлению процедурных элементов в языке SQL стандарта SQL:1999 и SQL:2003 в части SQL/PSM. Это сделало SQL императивным языком программирования. Большинство СУБД предлагает собственные проприетарные и расширения производителя, сверх SQL/PSM.

**8.1 Реализация хранимых процедур**

Хранимые процедуры обычно создаются с помощью языка SQL и конкретной его реализации в выбранной СУБД. Например, для этих целей в СУБД Microsoft SQL Server существует язык Transact-SQL, в Oracle — PL/SQL, в InterBase и Firebird — PSQL, в PostgreSQL — PL/pgSQL, PL/Tcl, PL/Perl, PL/Python, в IBM DB2 — SQL/PL[en], в Informix — SPL. MySQL достаточно близко следует стандарту SQL:2003, её язык похож на SQL/PL.

В некоторых СУБД возможно использование хранимых процедур, написанных на любом языке программирования, способном создавать независимые исполняемые файлы, например, на C++ или Delphi. В терминологии Microsoft SQL Server такие процедуры называются расширенными хранимыми процедурами и являются просто функциями, содержащимися в Win32-DLL. А, например, в Interbase и Firebird для функций, вызываемых из DLL/SO, определено другое название — UDF (User Defined Function). В MS SQL 2005 появилась возможность написания хранимых процедур на любом языке .NET, а от расширенных хранимых процедур в будущем планируется отказаться. СУБД Oracle, в свою очередь, допускает написание хранимых процедур на языке Java.[1] В IBM DB2 написание хранимых процедур и функций на обычных языках программирования является традиционным способом, поддерживаемым с самого начала, а процедурное расширение SQL было добавлено в эту СУБД только в достаточно поздних версиях, после его включения в стандарт ANSI. Также процедуры на Java и С поддерживает Informix.

В СУБД Oracle хранимые процедуры могут объединяться в так называемые пакеты (англ. packages). Пакет состоит из двух частей — спецификации (англ. package specification), в которой указывается определение хранимой процедуры, и тела (англ. package body), где находится её реализация. Таким образом Oracle позволяет отделить интерфейс программного кода от его реализации.

В СУБД IBM DB2 хранимые процедуры можно объединять в модули.

В СУБД PostgreSQL, помимо хранимых процедур, возможно создавать функции возвращающие тип void.

**8.2 Назначение и преимущества хранимых процедур**

Хранимые процедуры позволяют повысить производительность, расширяют возможности программирования и поддерживают функции безопасности данных.

Вместо хранения часто используемого запроса, клиенты могут ссылаться на соответствующую хранимую процедуру. При вызове хранимой процедуры её содержимое сразу же обрабатывается сервером.

Кроме собственно выполнения запроса, хранимые процедуры позволяют также производить вычисления и манипуляцию данными — изменение, удаление, выполнять DDL-операторы (не во всех СУБД!) и вызывать другие хранимые процедуры, выполнять сложную транзакционную логику. Один-единственный оператор позволяет вызвать сложный сценарий, который содержится в хранимой процедуре, что позволяет избежать пересылки через сеть сотен команд и, в особенности, необходимости передачи больших объёмов данных с клиента на сервер.

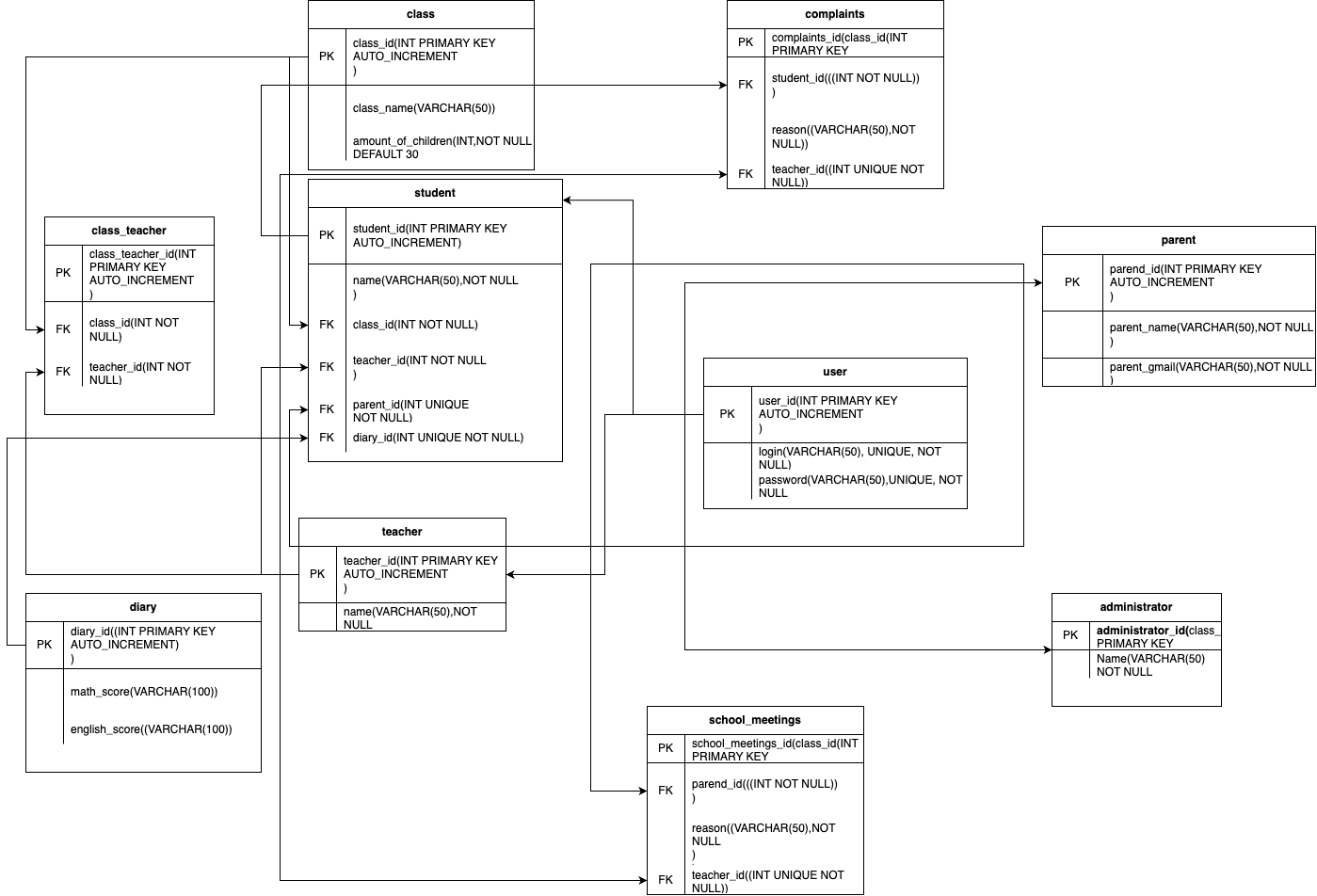
В большинстве СУБД при первом запуске хранимой процедуры она компилируется (выполняется синтаксический анализ и генерируется план доступа к данным). В дальнейшем её обработка осуществляется быстрее. В СУБД Oracle выполняется интерпретация хранимого процедурного кода, сохраняемого в словаре данных. Начиная с версии Oracle 10g поддерживается так называемая естественная компиляция (native compilation) хранимого процедурного кода в Си и затем в машинный код целевой машины, после чего при вызове хранимой процедуры происходит прямое выполнение её скомпилированного объектного кода.

**8.3 Безопасность**

Использование хранимых процедур позволяет ограничить или вообще исключить непосредственный доступ пользователей к таблицам базы данных, оставив пользователям только разрешения на выполнение хранимых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным. Кроме того, некоторые СУБД поддерживают шифрование текста (wrapping) хранимой процедуры.

Эти функции безопасности позволяют изолировать от пользователя структуру базы данных, что обеспечивает целостность и надежность базы.

Снижается вероятность таких действий как «внедрение SQL-кода», поскольку хорошо написанные хранимые процедуры дополнительно проверяют входные параметры перед тем, как передать запрос СУБД.

****

**Запросы**

1 У кого ведет Svetlana Alexandrovna + Запросы на выборку из нескольких таблиц

SELECT amount\_of\_children,class\_name,name

FROM class\_teacher

INNER JOIN class

ON class.class\_id = class\_teacher.class\_id

INNER JOIN teacher

ON teacher.teacher\_id = class\_teacher.teacher\_id

WHERE

name = "Svetlana Alexandrovna"

2 Какие преподаватели ведут у 10А класса

SELECT amount\_of\_children,class\_name,name

FROM class\_teacher

INNER JOIN class

ON class.class\_id = class\_teacher.class\_id

INNER JOIN teacher

ON teacher.teacher\_id = class\_teacher.teacher\_id

WHERE

class\_name = "10A"

3 Люди которые учаться в одном классе

SELECT name, class\_name, amount\_of\_children,behavior

FROM student

INNER JOIN class

ON class.class\_id = student.class\_id

WHERE student.class\_id = 1

4 Люди которые учаться в классе 1 и поведение больше чем два

SELECT name, class\_name, amount\_of\_children,behavior

FROM student

INNER JOIN class

ON class.class\_id = student.class\_id

WHERE student.class\_id = 1 AND behavior>2

5 Найдем среднее, максимальное и минимальное поведение по классам

SELECT class\_name, MAX(behavior) AS max\_behaviour, MIN(behavior) AS min\_behaviour, AVG(behavior) AS avg\_behaviour

FROM student

INNER JOIN class

ON class.class\_id = student.class\_id

GROUP BY class\_name

6 Найти максимальное поведение по классам

SELECT class\_name, MAX(behavior) AS Максимальное\_поведение FROM student

INNER JOIN class ON student.class\_id = class.class\_id

GROUP BY class\_name

HAVING MAX(behavior) < 5

ORDER BY MAX(behavior) DESC

7 Вывести все классы у которых ведут преподы

SELECT teacher.name,class\_id

FROM

teacher LEFT JOIN class\_teacher

ON teacher.teacher\_id = class\_teacher.teacher\_id

8 Каждому преподу поставить все возможные классы

SELECT teacher.name,class.class\_name

FROM

teacher CROSS JOIN class

9 Вывести минимальными с минимальными оценками поведения

SELECT name,behavior

FROM student

WHERE behavior = (SELECT MIN(behavior) FROM student)

10 Нормировать поведение

SELECT name,behavior, (behavior - (SELECT AVG(behavior) FROM student))/(SELECT STD(behavior) FROM student) AS norma

FROM student

11 Вывести все классы, среднее поведение которых больше 3

SELECT class.class\_name

FROM student

INNER JOIN class ON student.class\_id = class.class\_id

WHERE class.class\_id IN (

SELECT class\_id

FROM student

GROUP BY class\_id

HAVING AVG(behavior)>3

)

LIMIT 1

12 Оценка поведения студентов

SELECT name, behavior,

CASE

WHEN behavior>3 THEN 'good'

WHEN behavior<2 THEN 'bad'

ELSE 'medium'

END AS Beh

FROM student

Триггеры и процедуры

1

DELIMITER //

CREATE PROCEDURE add\_user(IN login\_1 VARCHAR(50))

BEGIN

DECLARE password\_1 VARCHAR(50);

SET password\_1 = "123456";

INSERT INTO users(login,password) VALUES (login\_1,password\_1);

END //

2

DELIMITER //

CREATE TRIGGER app\_trig

BEFORE INSERT ON administrator

FOR EACH ROW

BEGIN

SET NEW.name = "DZIMAZ";

END //

3 ADMIN

DELIMITER //

CREATE PROCEDURE add\_parent(IN gmail\_ VARCHAR(50),IN name\_ VARCHAR(50))

BEGIN

INSERT INTO parent(parent\_gmail,parent\_name)

VALUES (gmail\_,name\_);

END //

CALL add\_parent("Hello@gmail.com", "Dver")

5

DELIMITER //

CREATE PROCEDURE add\_teacher(IN name\_ VARCHAR(50))

BEGIN

INSERT INTO teacher(name)

VALUES (name\_);

END //

CALL add\_teacher("Anisimov")

6

DELIMITER //

CREATE PROCEDURE create\_diary()

BEGIN

INSERT INTO diary(english\_score,math\_score)

VALUES ("","");

END //

CALL create\_diary()

7

DELIMITER //

CREATE PROCEDURE add\_math\_score(IN id\_ INT,IgetN math\_score\_ VARCHAR(50))

BEGIN

UPDATE diary

SET math\_score = CONCAT(math\_score," ",math\_score\_)

WHERE diary.diary\_id = id\_;

END //

CALL add\_math\_score(11,"10")

CALL add\_math\_score(11,"9");

8

DELIMITER //

CREATE PROCEDURE add\_user(IN login\_ VARCHAR(50),IN password\_ VARCHAR(50),IN role\_id\_ INT,IN your\_personal\_data\_id\_ INT)

BEGIN

INSERT INTO users(login,password,role\_id,your\_personal\_data\_id)

VALUES

(login\_,password\_,role\_id\_,your\_personal\_data\_id\_);

END //

CALL add\_user("Mikola@gmail.com","1234",1,0)

9

DELIMITER //

CREATE PROCEDURE add\_english\_score(IN id\_ INT,IN english\_score\_ VARCHAR(50))

BEGIN

UPDATE diary

SET english\_score = CONCAT(english\_score," ",english\_score\_)

WHERE diary.diary\_id = id\_;

END //

CALL add\_english\_score(11,"9")

10

DELIMITER //

CREATE PROCEDURE find\_student\_id(IN name\_ VARCHAR(50))

BEGIN

SELECT student.student\_id,student.name FROM student

WHERE name = name\_;

END//

11

DELIMITER //

CREATE PROCEDURE add\_student(IN name\_ VARCHAR(50),IN class\_id\_ INT, teacher\_id\_ INT, IN parent\_id\_ INT,IN diary\_id\_ INT, IN behavior\_ INT)

BEGIN

INSERT INTO student(name, class\_id, teacher\_id, parent\_id, diary\_id,behavior)

VALUES(name\_, class\_id\_, teacher\_id\_, parent\_id\_, diary\_id\_,behavior\_);

END //

CALL add\_student("Xor",2,1,3,11,5)

12

INSERT INTO school\_meetings(parent\_id,reason,teacher\_id)

VALUES(1,"Bad marks",1)

DELIMITER //

CREATE PROCEDURE add\_meeting(IN parent\_id\_ INT,IN reason\_ VARCHAR(50), IN teacher\_id\_ INT)

BEGIN

INSERT INTO school\_meetings(parent\_id,reason,teacher\_id)

VALUES(parent\_id\_,reason\_,teacher\_id\_);

END //

CALL add\_meeting(2,"Simple",2)

13

DELIMITER //

CREATE PROCEDURE get\_user\_data(IN login\_ VARCHAR(50),OUT role\_id\_ INT, OUT password\_ VARCHAR(50), OUT your\_personal\_data\_id\_ VARCHAR(50))

BEGIN

SELECT role\_id, password,your\_personal\_data\_id

INTO role\_id\_,password\_,your\_personal\_data\_id\_

FROM users

WHERE users.login = login\_;

END //

14

DELIMITER //

CREATE PROCEDURE add\_complain(IN student\_id\_ INT,IN reason\_ VARCHAR(50), IN teacher\_id\_ INT)

BEGIN

INSERT INTO complaints(reason,student\_id,teacher\_id)

VALUES(reason\_,student\_id\_,teacher\_id\_);

END //

CALL add\_complain(3,"Simple\_2",3)

15

DELIMITER //

CREATE PROCEDURE find\_teacher\_id(IN name\_ VARCHAR(50))

BEGIN

SELECT teacher.teacher\_id,teacher.name FROM teacher

WHERE teacher.name = name\_;

END

16

DELIMITER //

CREATE PROCEDURE delete\_student(IN student\_id\_ INT)

BEGIN

DELETE FROM student

WHERE student\_id = student\_id\_;

END

17

DELIMITER //

CREATE TRIGGER insert\_into\_users\_parent

AFTER INSERT ON parent

FOR EACH ROW

BEGIN

INSERT INTO users(login,password,role\_id,your\_personal\_data\_id)

VALUES (CONCAT(NEW.parent\_id,"\_","4",NEW.parent\_gmail),CONCAT(NEW.parent\_name,"\_4\_",NEW.parent\_id),4,NEW.parent\_id);

END

18

DELIMITER //

CREATE TRIGGER bahavior\_check

BEFORE INSERT ON student

FOR EACH ROW

BEGIN

IF NEW.behavior>5 THEN SET NEW.behavior = 5; END IF;

IF NEW.behavior<0 THEN SET NEW.behavior = 0; END IF;

END

CALL add\_student("Polyna",2,1,3,12,12)

CALL add\_student("Victoria",2,1,3,13,-2)

19

DELIMITER //

CREATE TRIGGER check\_first\_letter

BEFORE INSERT ON student

FOR EACH ROW

BEGIN

SET NEW.name = CONCAT(UPPER(SUBSTRING(NEW.name,1,1)),SUBSTRING(NEW.name,2));

END

CALL add\_student("angela",2,1,3,14,-2)

20

DELIMITER //

CREATE TRIGGER update\_acad

BEFORE INSERT ON student

FOR EACH ROW

BEGIN

UPDATE academic\_performance

SET score = score+NEW.behavior

WHERE academic\_performance.academic\_performance\_id = 1;

END

CALL add\_student("veniamin",2,1,3,15,2)

21

DELIMITER //

CREATE TRIGGER insert\_into\_users\_teacher

AFTER INSERT ON teacher

FOR EACH ROW

BEGIN

INSERT INTO users(login,password,role\_id,your\_personal\_data\_id)

VALUES (CONCAT(NEW.name,"\_",NEW.teacher\_id,"\_",3,"@gmail.com"),CONCAT(NEW.name,"\_",NEW.teacher\_id),3,NEW.teacher\_id);

END

CALL add\_teacher("Chekanova")

22

DELIMITER //

CREATE TRIGGER insert\_into\_users\_student

AFTER INSERT ON student

FOR EACH ROW

BEGIN

INSERT INTO users(login,password,role\_id,your\_personal\_data\_id)

VALUES (CONCAT(NEW.name,"\_",NEW.student\_id,"\_",1,"@gmail.com"),CONCAT(NEW.name,"\_",NEW.student\_id),1,NEW.student\_id);

END

**DDL скрипты**

CREATE DATABASE new\_data\_basa\_1

CREATE TABLE teacher(

teacher\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL

);

CREATE TABLE class(

class\_id INT PRIMARY KEY AUTO\_INCREMENT,

class\_name VARCHAR(50),

amount\_of\_children INT NOT NULL DEFAULT 30

);

CREATE TABLE class\_teacher(

class\_teacher\_id INT PRIMARY KEY AUTO\_INCREMENT,

class\_id INT NOT NULL,

teacher\_id INT NOT NULL,

FOREIGN KEY (class\_id) REFERENCES class (class\_id),

FOREIGN KEY (teacher\_id) REFERENCES teacher (teacher\_id) );

INSERT INTO teacher(name)

VALUES

("Irina Alexandrovna"),

("Svetlana Alexandrovna"),

("Georgi Semenovich"),

("Elena Alexandrovna");

INSERT INTO class(amount\_of\_children,class\_name)

VALUES

(28,"10A"),

(27,"10B"),

(21,"5V"),

(19,"8A");

INSERT INTO class\_teacher(class\_id,teacher\_id)

VALUES

(1,1),

(1,2),

(1,3),

(2,2),

(3,1),

(4,1);

CREATE TABLE parent (

parent\_id INT PRIMARY KEY AUTO\_INCREMENT,

parent\_name VARCHAR(50) NOT NULL,

parent\_gmail VARCHAR(50) NOT NULL

);

INSERT INTO parent (parent\_name,parent\_gmail)

VALUES

("Dima Ermolovich","dima123@gmail.com"),

("Alex Sneshko","alex007@gmail.com"),

("Tima Dydich","timoxa432@gmail.com"),

("Stes","big\_stas22@gmail.com");

CREATE TABLE diary(

diary\_id INT PRIMARY KEY AUTO\_INCREMENT,

math\_score VARCHAR(100),

english\_score VARCHAR(100)

);

INSERT INTO diary(math\_score,english\_score)

VALUES

("10 9 10","7 7 7"),

("6 2 8","9 7 7 8"),

("5 3 2 1","4 3 2"),

("9 6 8 7 8","9 9"),

("1 6 8","6 3 8");

CREATE TABLE student(

student\_id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(50) NOT NULL,

class\_id INT,

teacher\_id INT,

parent\_id INT,

diary\_id INT UNIQUE,

FOREIGN KEY (class\_id) REFERENCES class (class\_id),

FOREIGN KEY (teacher\_id) REFERENCES teacher (teacher\_id),

FOREIGN KEY (parent\_id) REFERENCES parent (parent\_id),

FOREIGN KEY (diary\_id) REFERENCES diary (diary\_id)

);

INSERT INTO student(name,class\_id,teacher\_id,diary\_id,parent\_id)

VALUES

("a",1,1,1,1),

("b",1,1,2,1),

("c",1,1,3,1);

CREATE TABLE complaints(

complaints\_id INT PRIMARY KEY AUTO\_INCREMENT,

reason VARCHAR(50) NOT NULL,

student\_id INT,

teacher\_id INT,

FOREIGN KEY (teacher\_id) REFERENCES teacher (teacher\_id),

FOREIGN KEY (student\_id) REFERENCES student (student\_id)

);

CREATE TABLE school\_meetings(

school\_meetings\_id INT PRIMARY KEY AUTO\_INCREMENT,

reason VARCHAR(50) NOT NULL,

parent\_id INT,

teacher\_id INT,

FOREIGN KEY (teacher\_id) REFERENCES teacher (teacher\_id),

FOREIGN KEY (parent\_id) REFERENCES parent (parent\_id)

);

INSERT INTO complaints(reason,student\_id,teacher\_id)

VALUES

("she yelled at me",1,1),

("she is bad",2,2);

CREATE TABLE role

(

role\_id INT PRIMARY KEY AUTO\_INCREMENT,

role\_name VARCHAR(50)NOT NULL UNIQUE

);

CREATE TABLE users

(

users\_id INT PRIMARY KEY AUTO\_INCREMENT,

login VARCHAR(50) NOT NULL UNIQUE,

password VARCHAR(50) NOT NULL,

your\_personal\_data\_id INT NOT NULL,

role\_id INT,

FOREIGN KEY(role\_id) REFERENCES role (role\_id) ON DELETE RESTRICT

);

INSERT INTO role(role\_name)

VALUES

("admin"),

("student"),

("teacher"),

("parent");

INSERT INTO users(login,password,role\_id,your\_personal\_data\_id)

VALUES

("dima@gmail.com","123456",1,0);

INSERT INTO parent(parent\_name,parent\_gmail)

VALUES ("Dima","qadsa@gmail");

INSERT INTO school\_meetings(parent\_id,reason,teacher\_id)

VALUES

(1,"Marks",1);

INSERT INTO diary(math\_score,english\_score)

VALUES

("1 3 5","7 7 7"),

("7 4 1","5 7 5 8"),

("3 3 6 2","4 5 2"),

("1 6 7","8 8"),

("4 2 1","3 2 8");

INSERT INTO student(name,class\_id,teacher\_id,diary\_id,parent\_id)

VALUES

("Egor",1,2,5,1),

("Anton",3,3,6,2),

("Nikolay",3,3,7,3),

("Misha",3,3,8,1),

("Nikita",2,1,9,2),

("Ilya",2,1,10,3);

ALTER TABLE student

ADD behavior INT;

ALTER TABLE student

ADD CONSTRAINT diary\_id FOREIGN KEY (diary\_id)

REFERENCES diary (diary\_id)

ON DELETE RESTRICT;

ALTER TABLE student

ADD CONSTRAINT class\_id FOREIGN KEY (class\_id)

REFERENCES class (class\_id)

ON DELETE SET NULL;

ALTER TABLE student

ADD CONSTRAINT parent\_id FOREIGN KEY (parent\_id)

REFERENCES parent (parent\_id)

ON DELETE SET NULL;

ALTER TABLE parent CHANGE parent\_name parent\_name VARCHAR(30) NOT NULL;

ALTER TABLE complaints

ADD CONSTRAINT student\_id FOREIGN KEY (student\_id)

REFERENCES student (student\_id)

ON DELETE SET NULL;