ПЕРЕЧЕНЬ ТЕОРЕТИЧЕСКИХ ВОПРОСОВ

Аделина **3** Божена **3** Рома **4** Софа **4** Эля **3** Сережа **3** Влад **3** Альбина **3** Рита **3**

##### История появления и развития баз данных.

История развития баз данных уходит корнями в**1960-е годы**. В те времена информация собиралась и хранилась в файлах. Каждый файл содержал определенные сведения и для охвата всей предметной области требовалось несколько файлов. *Например,* сведения о товарах хранились в одном файле, а сведения о клиентах - в другом. Информация о приобретении определенных товаров определенными клиентами - в третьем.

**В 1970 году Э. Ф. Кодд** опубликовал статью, которая послужила основой для создания реляционной модели данных. Преимущество такой модели хранения данных заключается в минимальном дублировании данных и исключении некоторых типов ошибок, свойственных другим моделям. Согласно этой модели, данные хранятся в виде *таблиц со столбцами и строками*.

**В 1980-1990-х годах** стали развиваться коммерческие СУБД, такие как Oracle, IBM DB2 и Microsoft SQL Server. Эти системы предлагали надежное хранение и обработку данных и стали широко применяться в бизнес-среде.

**В 1990-х** в ответ на возрастающую потребность в хранении и обработке сложных объектов, появились объектно-ориентированные базы данных (ООБД). Они позволяли хранить объекты с их атрибутами и методами, обеспечивая более естественное представление данных для приложений.

**В 2000-х годах** с появлением больших объемов данных и необходимости их эффективной обработки, стали развиваться нереляционные базы данных, такие как **MongoDB, Cassandra и Redis**. Они предлагают гибкую структуру хранения данных, поддерживая различные модели данных, такие как ключ-значение, столбцовая, документоориентированная и графовая.

##### Понятие информационной системы. Информация и данных. Понятие базы данных, СУБД. Типология баз данных. Примеры

**Информационная система (ИС)** представляет собой систему, которая предназначена для поиска, хранения и обработки информации, и соответствующие ей организационные ресурсы, обеспечивающие и распространяющие информацию.

Информационные системы предназначены для своевременного обеспечения определенной категории людей определенной информацией, т.е. для удовлетворения конкретных информационных потребностей в определенной предметной области.

**Данные** представляют собой информацию, находящуюся в формализованном виде и предназначенную для обработки техническими системами.

Под **информацией** понимается совокупность представляющих интерес фактов, событий или явлений, которые необходимо зарегистрировать и обработать.

*Информация в отличие от данных* — это не все, что мы знаем о предмете, а только то, что нам интересно, что можно хранить, накапливать, применять, передавать и т. д. Например, если составить перечень из двадцати оценок и показать кому-либо, то они будут восприниматься как обыкновенные данные. А если напротив каждой оценки написать фамилии студентов, то это будет восприниматься уже как информация, она будет интересной в данном случае для студентов, получивших оценки по некоторой дисциплине.

**База данных (БД)** - это поименованная совокупность структурированные данных, относящихся к определенной предметной области.

**Система управления базами данных (СУБД)** - это комплекс программных и языковых средств, необходимых для создания баз данных, поддержания их в актуальном состоянии и организации поиска в них необходимой информации.

**Типология БД:**

* **Резидентные**:

Примеры — Redis, Apache Ignite, Tarantool

Сведения хранятся в оперативной памяти. Данные обрабатываются быстро, поэтому резидентные БД популярны там, где нужно обеспечить максимально короткое время отклика.

* **Поисковые**:

Пример — Elastic.

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

* **Реляционные**:

Примеры — MySQL, Oracle DB, PostgreSQL.

Это самый популярный тип БД, в которых информация хранится в виде таблиц. В строках находится описание каждого отдельного свойства объекта, а столбцы нужны для извлечения определённых свойств из строки.

* **Документоориентированные:**

Примеры — CouchDB, Couchbase, MongoDB.

Если в реляционных БД для извлечения данных нужно объединять таблицы, то в этих базах отлично хранится несвязанная информация в больших объёмах. Они поддерживают JSON.

* **Графовые:**

Примеры — OrientDB, Neo4j.

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

##### Понятие модели данных. Понятие модели базы данных. Виды моделей данных. Иерархическая, сетевая и реляционная модели данных.

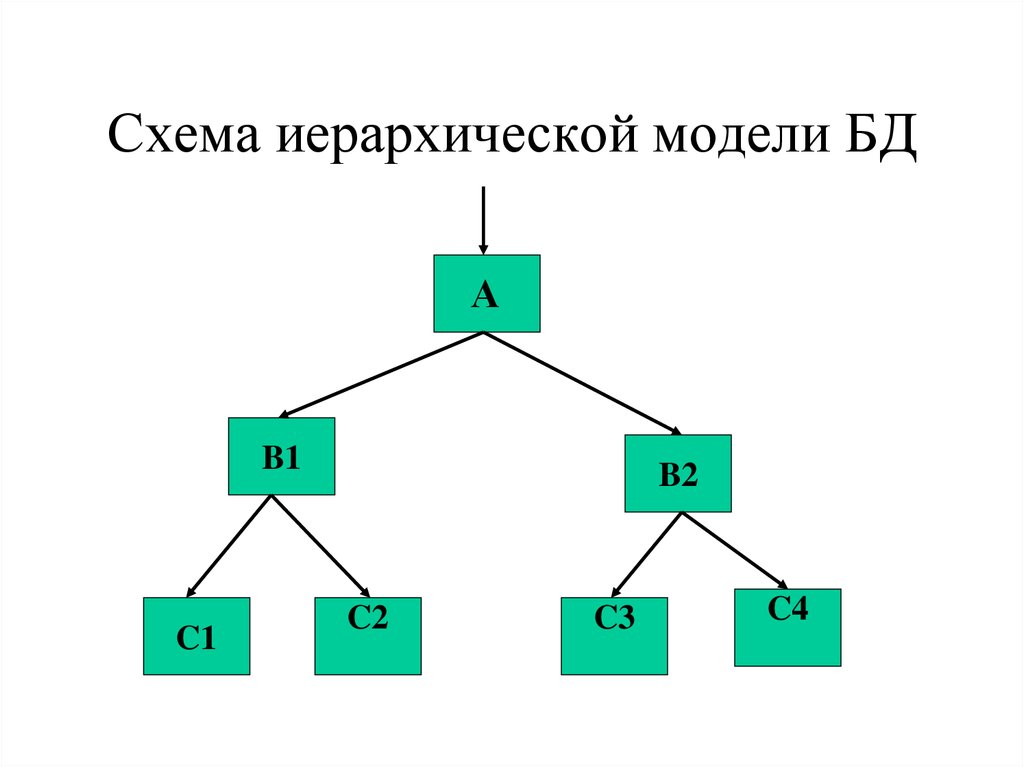
**Модель данных** - способ организации логической структуры хранения данных в базе.

**Модель базы данных** -то же, что и схема базы данных, то есть описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных.

Виды моделей данных :

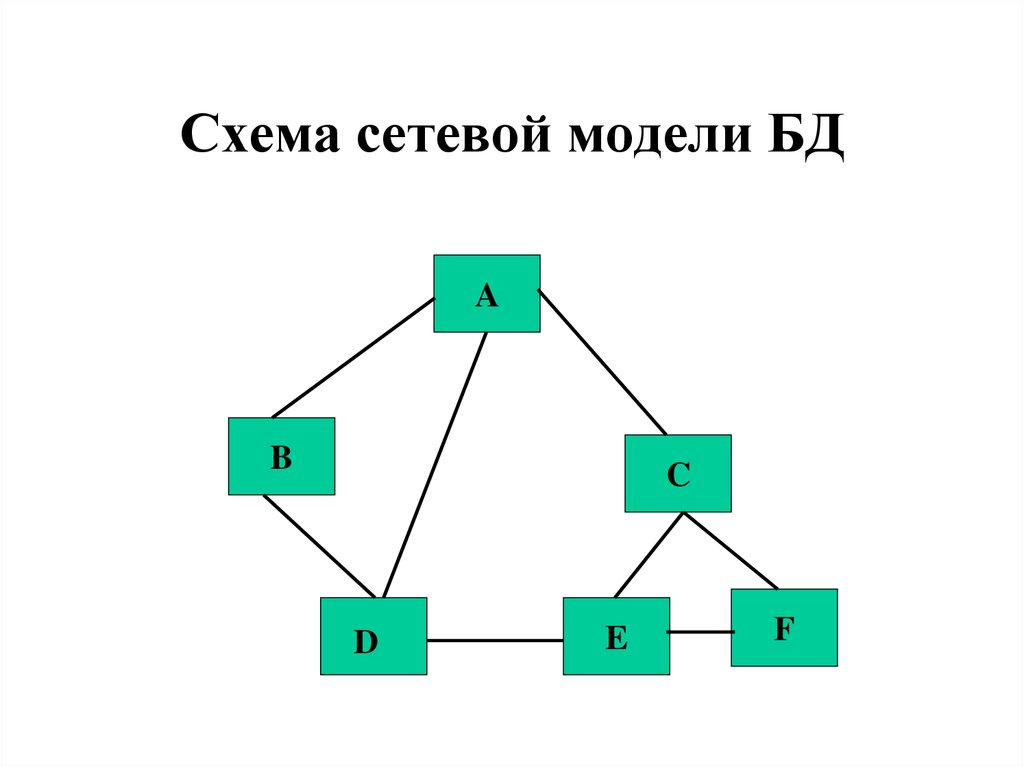
1. **Иерархическая**

Имеет форму дерева , на самом высшем уровне находится только одна величина, которая называется корнем. Это вершина имеет связи с вершинами 2-го уровня, вершины 2-го уровня связаны с вершинами 3-го уровня и т.д. Связи между вершинами одного уровня отсутствуют, т.е. данные неравноправны, они жестко подчинены другим. Доступ к ним возможен только на вертикальной схеме.



2. **Сетевая**

Является расширением иерархического подхода,строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных.



3. **Реляционная**

Прикладная теория построения баз данных , которая является приложением к задачам обработки данных таких разделов математики, как теория множеств и логика первого порядка.



##### Реляционная модель данных. Понятия: домен, кортеж, отношение, реляционная база данных, схема реляционной базы данных, атрибут, запись, null-значение, Примеры.



**Реляционная модель данных** - это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных.

**Значение NULL -**  это специальное значение, которое используется в SQL для обозначения отсутствия данных. Оно отличается от пустой строки или нулевого значения, так как NULL означает отсутствие какого-либо значения в ячейке таблицы.

**Домен**

Домен - тип данных, то есть допустимое множество значений.

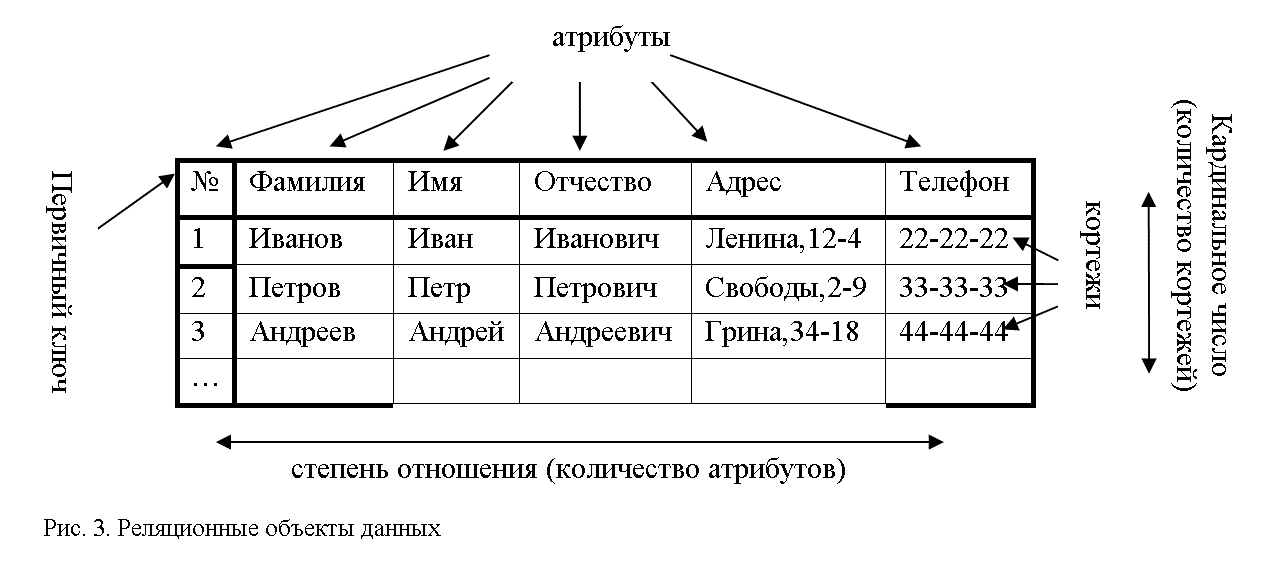
*Свойства:*

* Домен имеет уникальное имя (в пределах базы данных).
* Домен определен на некотором простом типе данных или на другом домене. (например, первичный и внешний ключи)
* Логическое условие, связанное с доменом, представляет собой правило или ограничение, которое говорит, какие значения разрешены для этого типа данных.
* Домен несет определенную смысловую нагрузку.

**Кортеж**

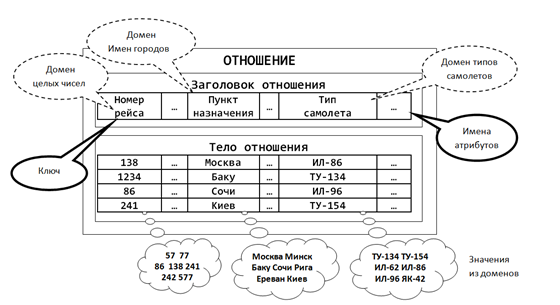
Это упорядоченный набор элементов, каждый из которых принадлежит определенному множеству или, иначе говоря, имеет свой тип. Совокупность однородных по структуре кортежей образует отношение.

Кортеж — строка таблицы.



**Отношение**

Отношение - это множество кортежей (не упорядоченный список), соответствующих одной схеме отношения. Кортежи отношения, будучи переставлены местами, не влияют на содержимое этого отношения.



**Атрибуты**

Представляет свойство, которое описывает некоторую характеристику объекта. Каждый столбец должен хранить один атрибут сущности

**Запись**

Это строка таблицы, содержащая набор значений свойств, размещенный в полях базы данных.

Каждая таблица должна содержать, по крайней мере, одно ключевое поле, содержимое которого уникально для каждой записи в этой таблице. Ключевое поле позволяет однозначно идентифицировать каждую запись в таблице.

**Реляционная база данных**

Тип базы данных, основанный на реляционной модели данных. В реляционной базе данных данные организованы в виде отношений, которые представляются в виде таблиц с рядами (записями) и столбцами (атрибутами).

**Схема реляционной базы данных**

Определяет структуру и описание всех отношений в базе данных, включая их имена, атрибуты, типы данных и другие характеристики.

##### Реляционная модель данных. Первичный, альтернативный и внешний ключи, ссылочная целостность. Стратегии поддержания ссылочной целостности. Примеры.

**Реляционная модель данных** - это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных.

**Первичный ключ** в виде наборов определенных значений, максимально идентифицирует каждую запись, то есть набор определенных признаков, уникальных для каждой записи.

* Primary key не позволяет создавать одинаковых записей (строк) в таблице;
* PK обеспечивают логическую связь между таблицами одной базы данных (для реляционных БД).

**Внешний ключ** обеспечивает однозначную логическую связь, между таблицами одной БД.

*Например*, есть две таблицы А и В. В таблице А (обувь), есть первичный ключ: размер, в таблице В (цвет) должна быть колонка с названием размер. В этой таблице «размер» это и будет внешний ключ для логической связи таблиц В и А.

**Ссылочная целостность**

Необходимое качество реляционной базы данных, заключающееся в отсутствии в любом её отношении внешних ключей, ссылающихся на несуществующие кортежи.

**Альтернативный ключ (Alternate Key)**

Это потенциальный ключ, не ставший первичным. Каждая сущность должна иметь по крайней мере один потенциальный ключ. Многие сущности имеют только один потенциальный ключ. Такой ключ становится первичным.

Например, для сущности служащий (идентификатор служащего, фамилия, имя, отчество) группа атрибутов «фамилия» , «имя» , «отчество» может являться альтернативным ключом (в предположении, что на предприятии не работают полные тезки).

**Для поддержания ссылочной целостности используются 2 стандартных правила:**

* Ограничить (RESTRICT)

Запись автоматически проверяется, ссылаются ли внешние ключи в этой записи на существующие записи в заявленных при описании связанных таблицах. Если выясняется, что операция приведёт к появлению некорректных ссылок, она не выполняется — система возвращает ошибку.

* Каскадировать (CASCADE)

Разрешать выполнение требуемой операции, но внести при этом необходимые изменения в связанные таблицы так, чтобы не допустить нарушения ссылочной целостности между данными.

**Дополнительные правила поддержки ссылочной целостности являются:**

* Установить в NULL (set null)

Разрешить выполнение требуемой операции, однако все некорректные значения внешнего ключа изменить на null – значения

* Установить по умолчанию (set default)

Разрешить значения требуемой операции, однако все некорректные значения внешнего ключа изменить на значения, принятые по умолчанию.

##### Реляционная модель данных. Взаимосвязи вида: «один-к-одному», «один-ко-многим», «многие-ко-многим». Примеры.

**Реляционная модель данных** - это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных.

* Связь **«один-к-одному»**

В этом случае объекту одной сущности можно сопоставить только один объект другой сущности.

*Например*, на некоторых сайтах пользователь может иметь только один блог. То есть возникает отношение один пользователь - один блог.

* Связь **«один-ко-многим»**

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. То есть один блог - много статей.

* Связь **«многие-ко-многим»**

При этом типе связей одна строка из таблицы А может быть связана с множеством строк из таблицы В. В свою очередь одна строка из таблицы В может быть связана с множеством строк из таблицы А.

*Типичный пример* - студенты и курсы: один студент может посещать несколько курсов, и соответственно на один курс могут записаться несколько студентов.

##### Реляционная модель данных. Целостность. Целостность сущностей. Целостность внешних ключей. Примеры.

**Реляционная модель данных** - это способ организации данных в базе данных, основанный на использовании таблиц (отношений), которые состоят из строк (кортежей) и столбцов (атрибутов).

**Целостность данных в реляционной модели** означает, что данные в базе данных должны соответствовать определенным правилам и ограничениям, чтобы обеспечить их правильность и надежность.

**Целостность сущностей** гарантирует, что каждая запись в таблице уникальна и не содержит дубликатов. Это достигается путем определения первичного ключа для каждой таблицы, который уникально идентифицирует каждую запись в таблице.

**Целостность внешних ключей** обеспечивает связь между таблицами в базе данных. Внешний ключ - это атрибут в одной таблице, который ссылается на первичный ключ в другой таблице. Это позволяет связывать данные из разных таблиц и обеспечивает целостность связанных данных. Если запись в основной таблице удаляется или изменяется, то эти изменения автоматически распространяются на связанные записи в других таблицах.

**Примеры** целостности данных в реляционной модели могут включать проверку наличия значения в определенном диапазоне или формате, ограничение на длину строки, проверку уникальности значения в столбце и т.д. Все эти правила и ограничения помогают обеспечить целостность данных и предотвратить ошибки и проблемы при работе с базой данных.

##### Реляционная алгебра. Теоретико-множественные операторы: объединение, пересечение, вычитание, декартово произведение. Примеры.

**Реляционная алгебра** представляет собой набор операторов, использующих отношения в качестве аргументов, и возвращающие отношения в качестве результата.

**Оператор объединения (UNION)**.

UNIONпозволяет объединить две таблицы в одну, удалив все дубликаты. *Например*, если у нас есть таблица "Студенты" с полями "Имя", "Фамилия" и "Группа", и таблица "Преподаватели" с полями "Имя", "Фамилия" и "Предмет", то мы можем объединить эти таблицы с помощью оператора UNION, чтобы получить список всех людей в учебном заведении без дубликатов.

| SELECT expression1, expression2, ... expression\_n  FROM tables  [WHERE conditions]  UNION  SELECT expression1, expression2, ... expression\_n  FROM tables  [WHERE conditions]; |
| --- |
|  |

**Пересечение (INTERSECT).**

INTERSECT позволяет найти общие записи в двух таблицах.

*Например*, если у нас есть таблица "Студенты" с полями "Имя", "Фамилия" и "Группа", и таблица "Экзамены" с полями "Предмет", "Оценка" и "Группа", то мы можем использовать оператор INTERSECT, чтобы найти студентов, которые сдали экзамен по определенному предмету.

| SELECT expression1, expression2, ... expression\_n  FROM tables  [WHERE conditions]  INTERSECT  SELECT expression1, expression2, ... expression\_n  FROM tables  [WHERE conditions]; |
| --- |

**Оператор вычитания (MINUS)**

Используется для удаления записей одной таблицы из другой.

*Например*, если у нас есть таблица "Студенты" с полями "Имя", "Фамилия" и "Группа", и таблица "Отчисленные" с полями "Имя", "Фамилия" и "Группа", то мы можем использовать оператор MINUS, чтобы найти студентов, которые не были отчислены.

| SELECT expression1, expression2, ... expression\_n  FROM tables  [WHERE conditions]  MINUS  SELECT expression1, expression2, ... expression\_n  FROM tables  [WHERE conditions]; |
| --- |

**Декартово произведение (CROSS JOIN)**

Используется для создания таблицы, которая содержит все возможные комбинации строк из двух таблиц.

*Например*, если у нас есть таблица "Студенты" с полями "Имя", "Фамилия" и "Группа", и таблица "Предметы" с полями "Предмет" и "Преподаватель", то мы можем использовать оператор CROSS JOIN, чтобы создать таблицу, которая содержит все возможные комбинации студентов и предметов.

| SELECT l.\*, r.\*  FROM LeftTable l  CROSS JOIN RightTable r |
| --- |

##### Реляционная алгебра. Специальные реляционные операторы: выборка, проекция, соединение, деление. Примеры.

**Выборка (SELECT)**

Оператор, который позволяет выбрать определенные строки из таблицы, удовлетворяющие определенному условию.

*Например*, если у нас есть таблица "Студенты" с полями "Имя", "Фамилия", "Группа" и "Средний балл", то мы можем использовать оператор SELECT, чтобы выбрать всех студентов из определенной группы, у которых средний балл выше заданного значения.

| SELECT expressions  FROM tables [WHERE conditions] |
| --- |

**Проекция (PROJECT)**

Оператор проекции выбирает определенные столбцы (атрибуты) из таблицы, оставляя только те атрибуты, которые вам нужны.

| SELECT column1, column2, ... FROM table\_name; |
| --- |

**Соединение (JOIN)**

Оператор, который позволяет объединить две таблицы по определенному условию.

Например, если у нас есть таблица "Студенты" с полями "Имя", "Фамилия" и "Группа", и таблица "Оценки" с полями "Предмет", "Оценка" и "Группа", то мы можем использовать оператор JOIN, чтобы объединить эти таблицы по полю "Группа".



* JOIN – левая\_таблица JOIN правая\_таблица ON условия\_соединения
* LEFT JOIN – левая\_таблица LEFT JOIN правая\_таблица ON условия\_соединения
* RIGHT JOIN – левая\_таблица RIGHT JOIN правая\_таблица ON условия\_соединения
* FULL JOIN – левая\_таблица FULL JOIN правая\_таблица ON условия\_соединения
* CROSS JOIN – левая\_таблица CROSS JOIN правая\_таблица

**Деление (DIVIDE)** - это оператор, который позволяет найти все значения одного столбца, которые соответствуют определенному набору значений другого столбца. Например, если у нас есть таблица "Студенты" с полями "Имя", "Фамилия" и "Группа", и таблица "Экзамены" с полями "Предмет", "Оценка" и "Группа", то мы можем использовать оператор DIVIDE, чтобы найти все предметы, по которым сдали экзамен все студенты из определенной группы.

**Синтаксис:**

| SELECT Предмет FROM Экзамены WHERE Группа = 'Группа1' AND Предмет NOT IN (  SELECT Предмет  FROM Экзамены  WHERE Группа = 'Группа1'  AND Оценка < 3 ) |
| --- |

Этот запрос найдет все предметы, по которым сдали экзамен все студенты из группы "Группа1", исключая те предметы, по которым были получены оценки ниже 3.

##### Нормализация таблиц. Понятия: функциональной зависимости, декомпозиции отношений, нормализации таблиц. Виды зависимостей. Примеры.

**Нормализация** — это процесс организации данных в базе данных. Это включает создание таблиц и установление связей между этими таблицами в соответствии с правилами, предназначенными как для защиты данных, так и для того, чтобы сделать базу данных более гибкой за счет устранения избыточности и непоследовательной зависимости.

**Функциональные зависимости** — основа нормализации баз данных. Под функциональной зависимостью подразумевается зависимость значения одного поля (столбца) от другого. (по сути связь “один ко многим”)

**Декомпозиция** - это процесс, при котором отношение разлагается на два или более отношения.

**Виды функциональных зависимостей**

- Тривиальная функциональная зависимость

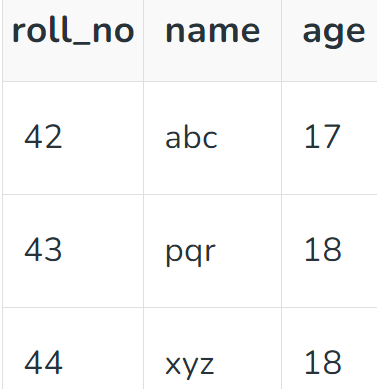
- Нетривиальная функциональная зависимость

- Многозначная функциональная зависимость

- Транзитивная функциональная зависимость

**Тривиальная функциональная зависимость**

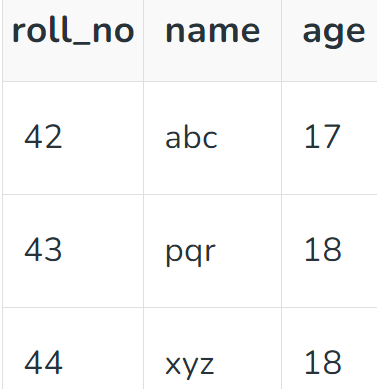
В тривиальной функциональной зависимости зависимый всегда является подмножеством определителя. т. е. если X → Y и Y является подмножеством X, то это называется тривиальной функциональной зависимостью



Здесь {roll\_no, name} → name является тривиальной функциональной зависимостью, поскольку имя зависимого является подмножеством набора определителей {roll\_no, name}. Аналогично, roll\_no → roll\_no также является примером тривиальной функциональной зависимости.

**Нетривиальная функциональная зависимость**

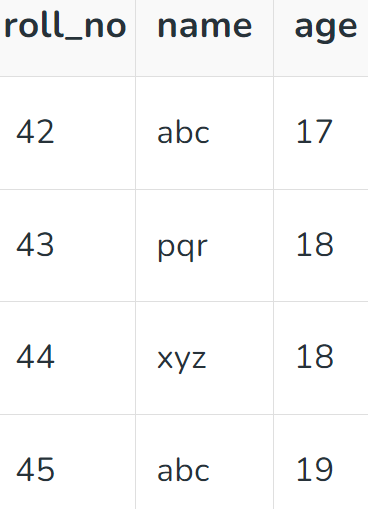
В нетривиальной функциональной зависимости зависимая строго не является подмножеством определителя. т. е. если X → Y и Y не является подмножеством X, то это называется нетривиальной функциональной зависимостью.



Здесь roll\_no → name является нетривиальной функциональной зависимостью, поскольку зависимое имя не является подмножеством определителя roll\_no. Аналогично, {roll\_no, name} → возраст также является нетривиальной функциональной зависимостью, поскольку возраст не является подмножеством {roll\_no, name}

**Многозначная функциональная зависимость**

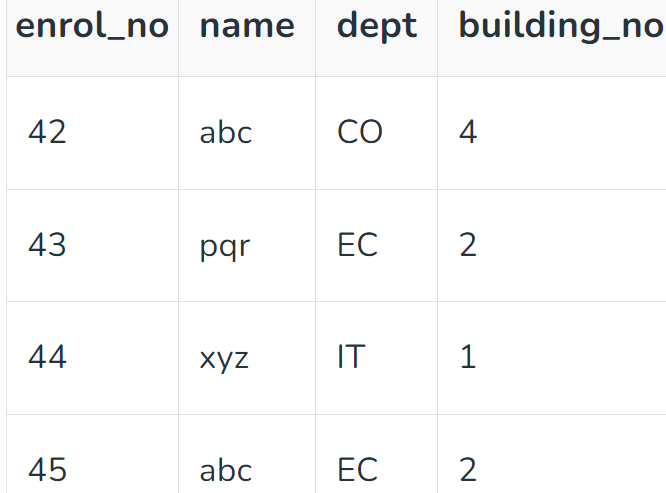
В многозначной функциональной зависимости объекты зависимого набора не зависят друг от друга. т.е. если a → {b, c} и между b и c не существует функциональной зависимости, то это называется многозначной функциональной зависимостью.



Здесь roll\_no → {name, age} является многозначной функциональной зависимостью, поскольку name и age зависимостей не зависят друг от друга (т.е. name → age или age → name не существует !)

**Транзитивная функциональная зависимость**

В транзитивной функциональной зависимости зависимое косвенно зависит от определителя. т. е. если a → b & b → c, то согласно аксиоме транзитивности, a → c. Это транзитивная функциональная зависимость.



Здесь, enroll\_no → dept и dept → building\_no. Следовательно, согласно аксиоме транзитивности, enroll\_no → building\_no является допустимой функциональной зависимостью. Это косвенная функциональная зависимость, отсюда и название транзитивной функциональной зависимости.

##### Нормализация таблиц. Нормальные формы отношений: первая, вторая, третья, Бойса-Кодда, четвертая и пятая. Примеры.

**Нормализация** - это процесс разделения данных по отдельным связанным таблицам, устранаяет избыточность данных , позволяет избежать нарушений целостности данных при их изменении т.е. избежать аномалий.

**1 НФ** должно стать наличие для одного атрибута сущности только одного столбца в таблице .

**2 НФ** предполагает, что каждый столбец, не являющийся ключом, должен зависеть от первичного ключа.

**3 НФ** Предполагает, что каждый столбец не явлющийся ключом должен зависеть только от первичного ключа. Т.е должна отсутствовать транзитивная функциональная зависимость.

Таблицы, у которых первичный ключ простой, и они находятся в третьей нормальной форме, автоматически находятся и в нормальной форме **Бойса-Кодда**.

Отношение находится в **4 НФ** тогда и только тогда, когда оно находится в нормальной форме Бойса-Кодда и не содержит нетривиальных зависимостей.

Отношения находятся в **5 НФ**, если оно находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами.

##### Нормализация таблиц. Алгоритм нормализации таблиц. Пример.

**Нормализация** - это процесс разделения данных по отдельным связанным таблицам, устранаяет избыточность данных , позволяет избежать нарушений целостности данных при их изменении т.е. избежать аномалий.

У нас есть обычная таблица(Рисунок 1) будем её приводить до 3 НФ

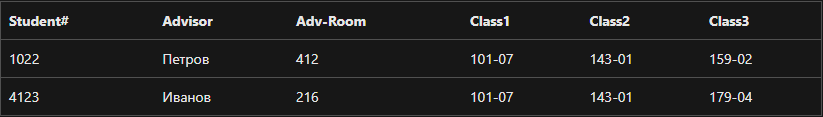


Рисунок 1 - первоначальная таблица

С начала нам нужно привести её к 1 НФ, в данном случае мы будем устранять повторяющиеся группы в таблице.

После приведения к 1 НФ, нам нужно привести её до 2 НФ, мы будем избавляться от лишних данных.

Далее удаление данных не зависящих от ключей, тем самым приводя таблицу к 3 НФ. В 3 НФ у нас получилось 2 таблицы из 1 первоначальной.

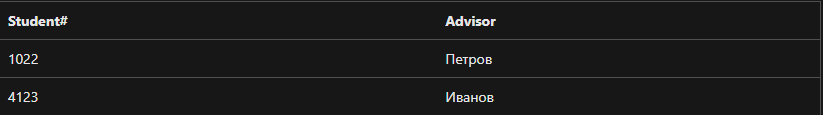


Рисунок 2 - таблица студентов

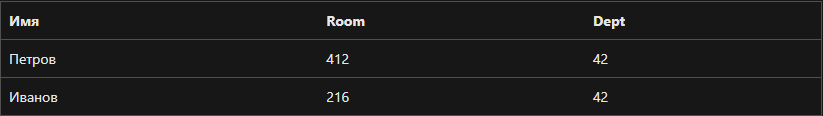


Рисунок 3 - таблица факультета

##### Моделирование данных. Логическая модель данных. Разработка модели с помощью Case средства. Пример.

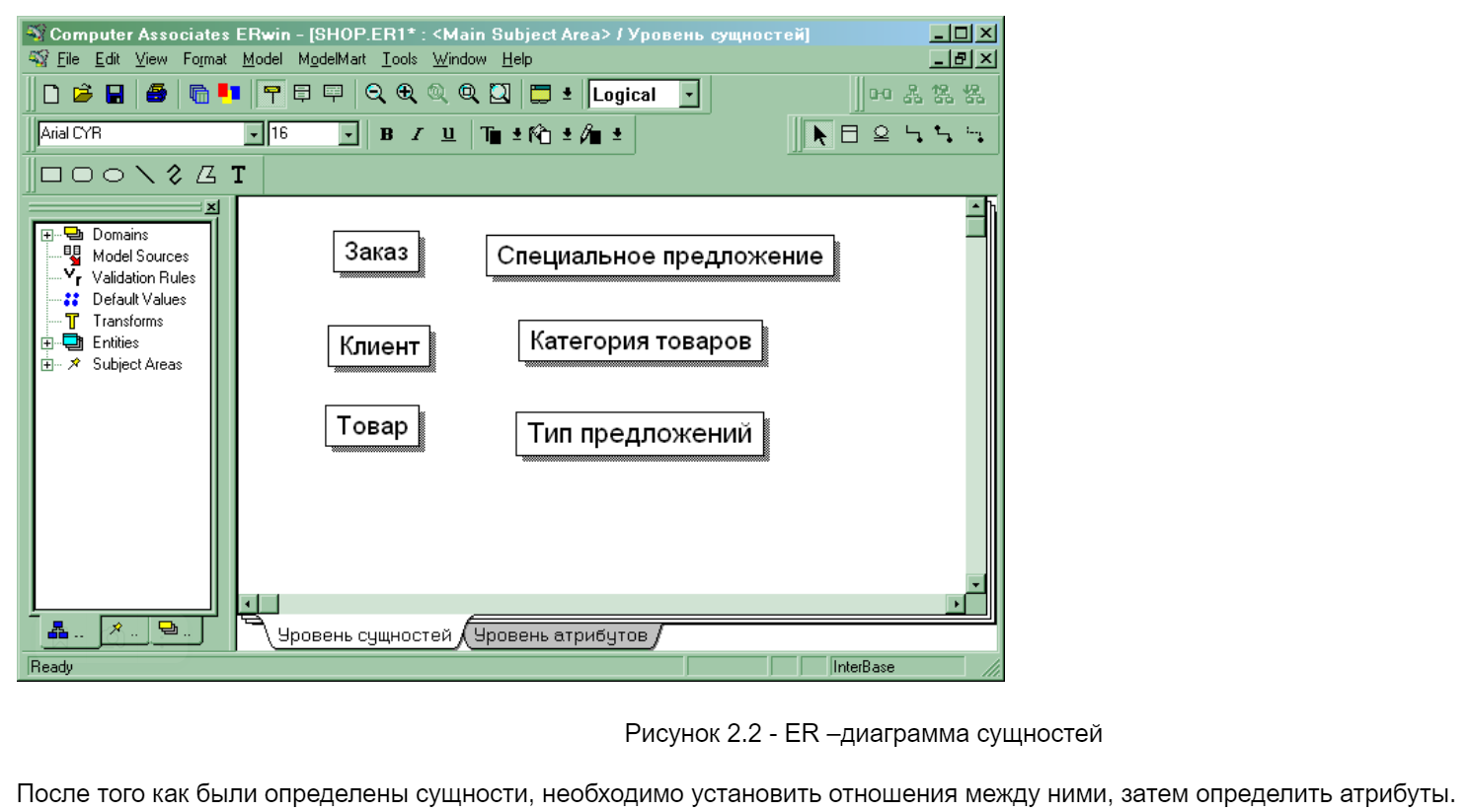
**Моделирование баз данных (БД)** - это процесс создания абстрактной структуры, которая представляет данные, хранимые в БД, и отображает их отношения и связи.

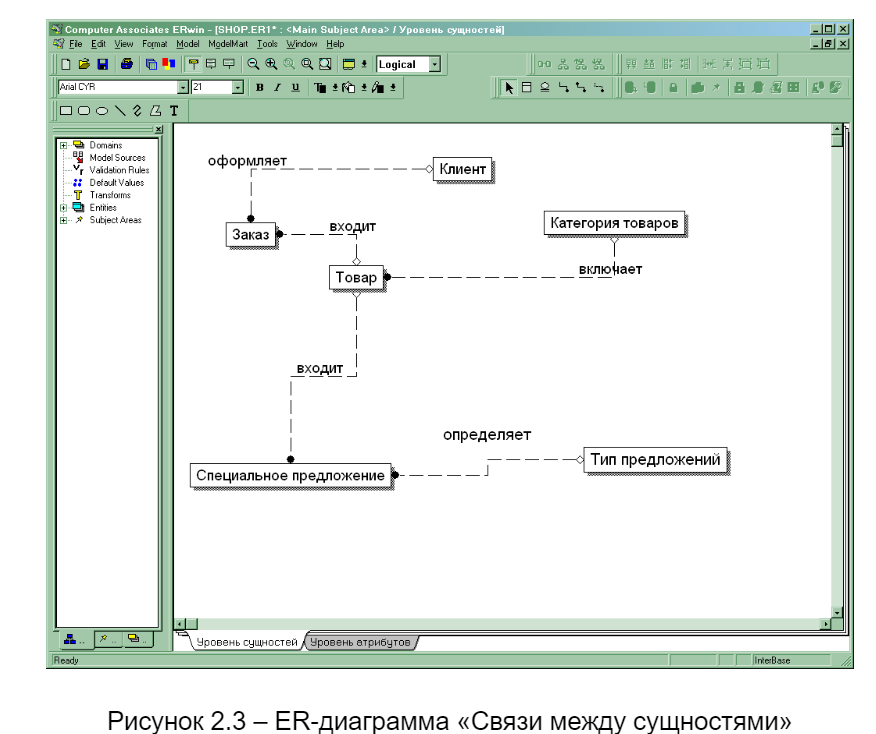
**Логическая модель данных** или логическая схема-это модель данных конкретной предметной области, выраженная независимо от конкретного продукта управления базами данных или технологии хранения (физической модели данных), но в терминах структур данных, таких как реляционные таблицы и столбцы, объектно-ориентированные классы или теги XML.

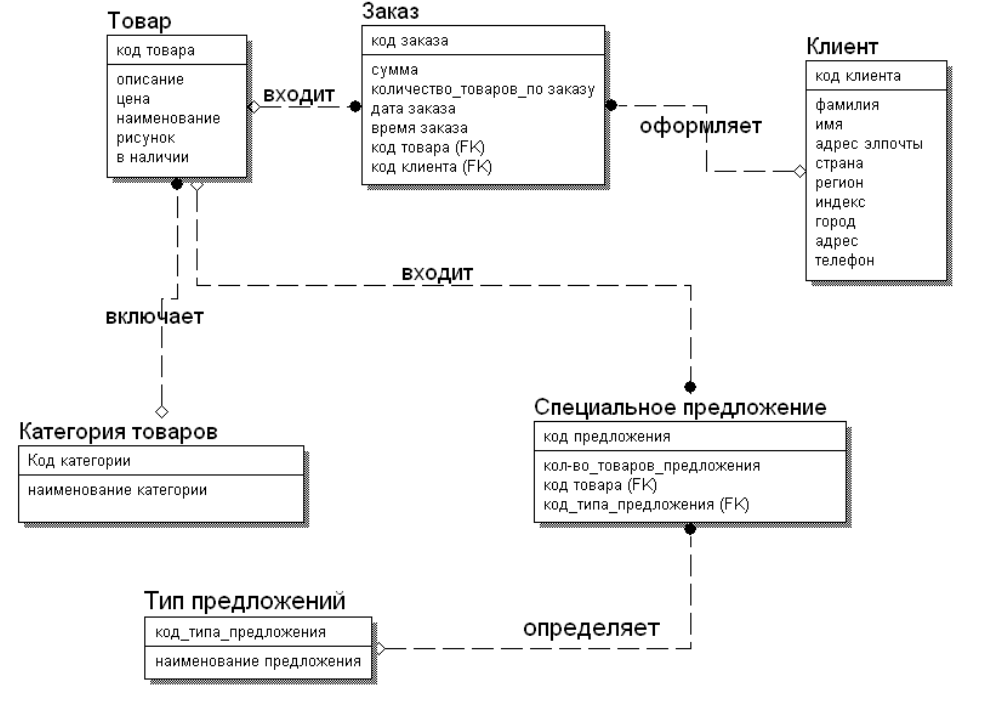
Разница между логической и физической моделью базы данных заключается в том, что логическая база данных используется для сбора информации о бизнесе и его потребностях. В то время как физическая модель базы данных используется для проектирования базы данных.

**Создание логической модели базы данных с помощью case-средства eRwin**

Для построения логической модели данных, прежде всего, необходимо определить набор сущностей рассматриваемой предметной области и задать связи между ними. Затем, определить атрибуты для каждой сущности, установить ключевые атрибуты и задать типы данных для каждого атрибута. При установке связей между сущностями будут определены первичные и вторичные ключевые атрибуты.







##### Моделирование данных. Физическая модель данных. Разработка модели с помощью Case средства. Пример.

**Моделирование баз данных (БД)** - это процесс создания абстрактной структуры, которая представляет данные, хранимые в БД, и отображает их отношения и связи.

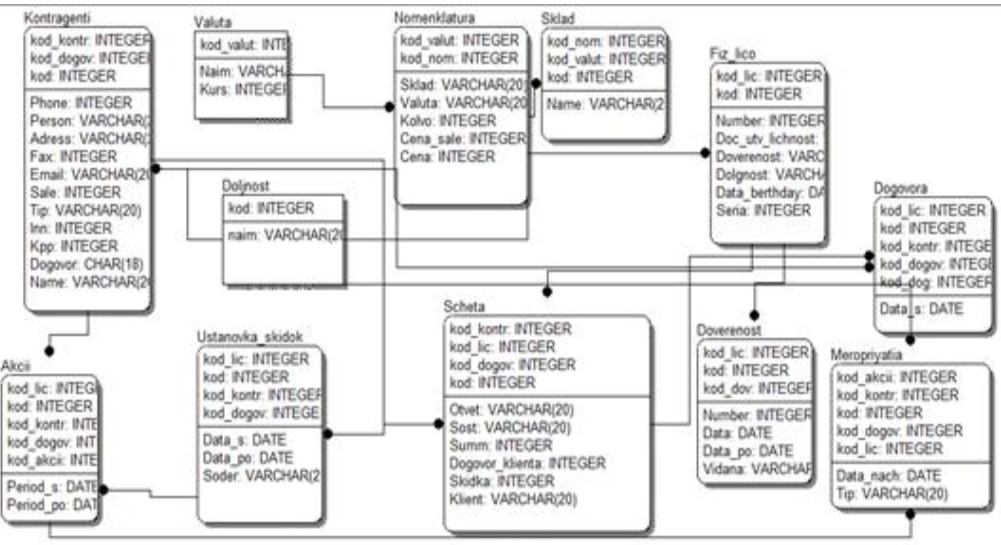
**Физическая модель данных** - модель, определяющая размещение данных на внешних носителях, методы доступа и технику индексирования.

Разница между логической и физической моделью базы данных заключается в том, что логическая база данных используется для сбора информации о бизнесе и его потребностях. В то время как физическая модель базы данных используется для проектирования базы данных.

**Создание физической модели базы данных**

На этапе создания физической модели базы данных информационной подсистемы в Erwin для каждой колонки таблицы базы данных укажем тип данных и возможность пустых значений.

После того как будут выполнены все перечисленные действия, физическая модель приобретет вид:



##### Информационное моделирование данных. Модель данных. Этапы разработки базы данных. Пример.

**Моделирование данных** — это создание визуального представления о всей информационной системе либо ее части.

**Модель данных** — отражает взаимосвязи между объектами.

**1 этап.** Планирование разработки базы данных.

**2 этап.** Определение требований к системе.

**3 этап.** Проектирование концептуальной модели БД. Усилия на этом этапе должны быть направлены на структуризацию данных и выявление взаимосвязей между ними.

**4 этап.** Построение логической модели.

**5 этап.** Построение физической модели.

**6 этап.** Оценка физической модели.

**7 этап.** Реализация БД.

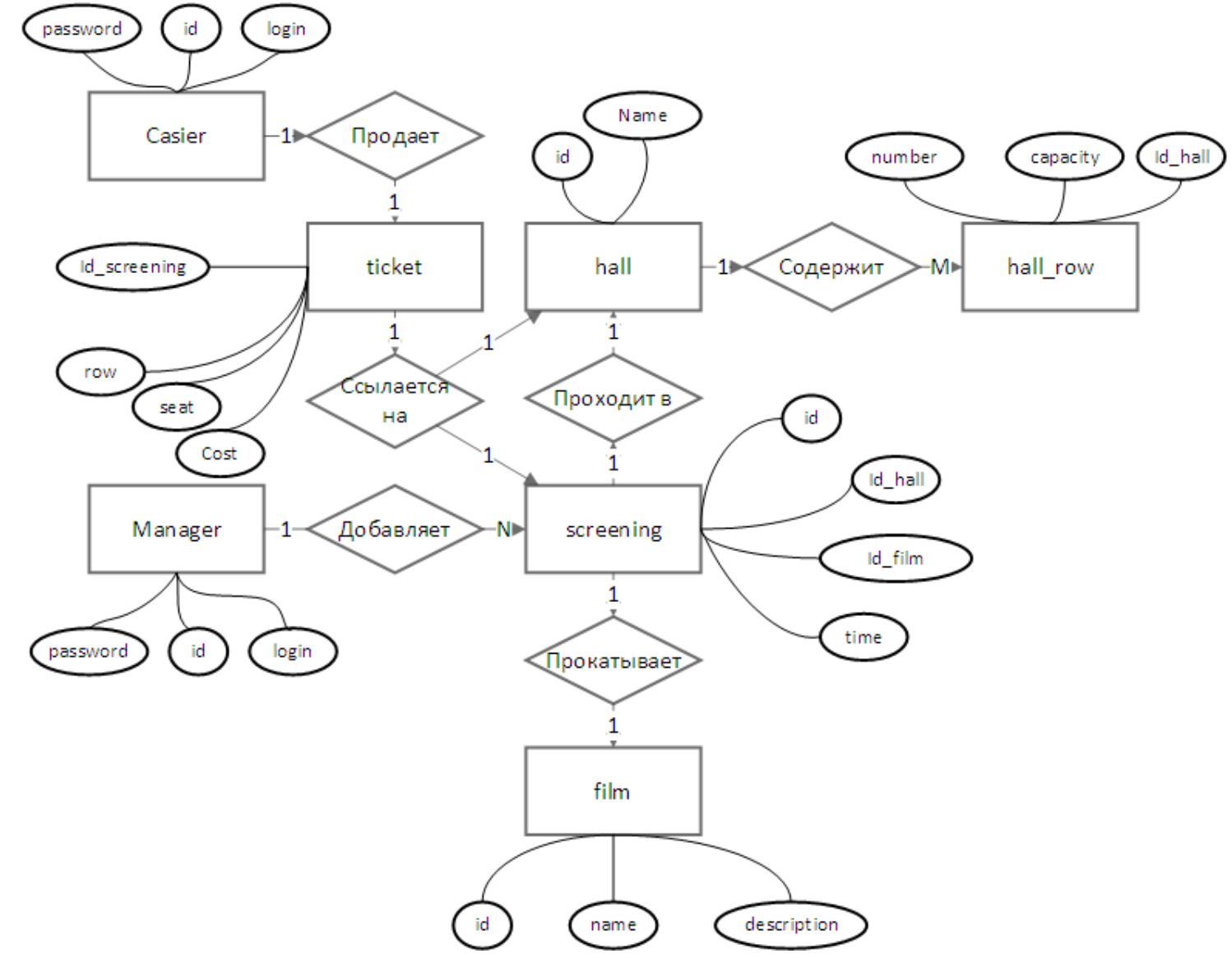
**8 этап.** Тестирование и оптимизация.

**9 этап.** Сопровождение и эксплуатация.

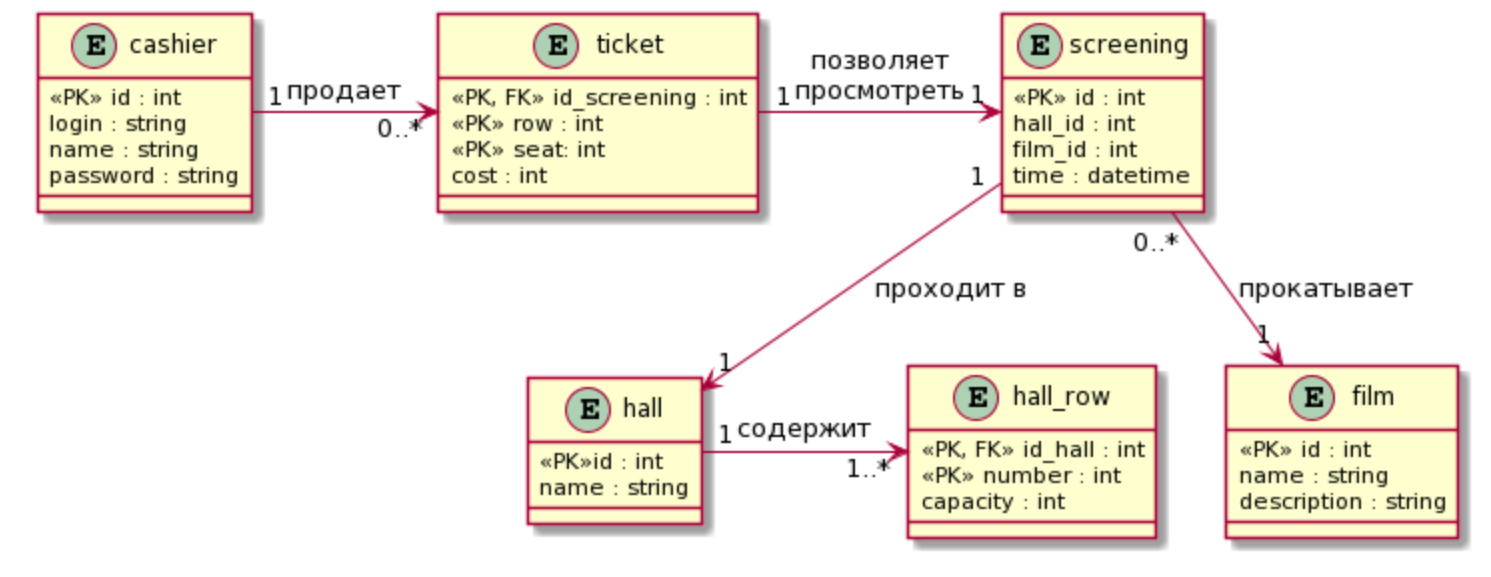
##### Моделирование БД. Нотация Чена и UML. Примеры.

**Моделирование баз данных (БД)** - это процесс создания абстрактной структуры, которая представляет данные, хранимые в БД, и отображает их отношения и связи.

В **нотации Чена** сущность отображается прямоугольником, ее атрибуты — овалами, а отношения между сущностями — ромбами. Нотация Чена - это правила обозначения сущностей, атрибутов связей и тд в виде диаграмм.



**Нотации UML** — это ключевые элементы для моделирования и визуализации процессов.



##### Моделирование БД. Понятие сущности и связи и их виды в модели IDEF1X. Примеры.

**Моделирование баз данных (БД)** - это процесс создания абстрактной структуры, которая представляет данные, хранимые в БД, и отображает их отношения и связи. Одной из популярных методик моделирования БД является IDEF1X, используемая для создания информационных моделей.

В модели IDEF1X сущность (entity) представляет отдельный объект или концепцию, о которой хранится информация в БД. Сущности обычно соответствуют реальным или абстрактным объектам, которые требуется отслеживать или описывать в системе. Каждая сущность имеет атрибуты, которые определяют ее свойства или характеристики.

Связь (relationship) в модели IDEF1X определяет ассоциации и зависимости между сущностями. Она показывает, как сущности взаимодействуют друг с другом или взаимодействуют через атрибуты. Связи могут быть однонаправленными или двунаправленными и иметь различные степени (1:1, 1:М, М:М), отражающие взаимосвязи между сущностями.

Некоторые виды связей в модели IDEF1X включают:  
1) Однородная связь (связь одного типа) - это связь между экземплярами одной сущности.   
2) Связь один-ко-многим (1:М) - это связь между одним экземпляром сущности и несколькими экземплярами другой сущности.  
3) Связь многие-ко-многим (М:М) - это связь, которая позволяет каждому экземпляру одной сущности быть связанным с несколькими экземплярами другой сущности и наоборот.  
**Примеры:**  
1. Банковская система: сущности - клиента, счета; связь один ко многим.  
2. Социальная сеть: сущности - пользователи, сообщения; связь - связь один-ко-многим.

##### Моделирование БД. Алгоритм перехода от модели «сущность-связь» к реляционной БД. Пример.

Вот общий алгоритм для перехода от модели "сущность-связь" к реляционной БД:  
1) Идентификация сущностей: Определите все сущности в модели "сущность-связь". Каждая сущность станет таблицей в реляционной БД.  
2) Идентификация атрибутов: Для каждой сущности определите ее атрибуты. Атрибуты станут столбцами в таблице реляционной БД.  
3) Идентификация связей: Определите связи между сущностями в модели "сущность-связь". Связи могут быть однонаправленными или двунаправленными. Различные виды связей (1:1, 1:М, М:М) будут реализованы разными способами в реляционной БД.  
4) Создание таблиц: Создайте таблицы в реляционной БД для каждой сущности. Каждая таблица будет содержать столбцы для атрибутов сущности.  
5) Определение первичных ключей: Определите первичные ключи для каждой таблицы. Первичный ключ уникально идентифицирует каждую запись в таблице.  
6) Создание внешних ключей: Если в модели "сущность-связь" присутствуют связи, создайте внешние ключи в соответствующих таблицах. Внешний ключ связывает записи в одной таблице с записями в другой таблице.  
7) Установка ограничений целостности: Определите и установите необходимые ограничения целостности, такие как ограничения уникальности, ограничения целостности ссылочной целостности и т.д.  
8) Создание отношений между таблицами: Создайте отношения между таблицами с помощью внешних ключей. Они определяют связи между сущностями в реляционной БД.  
**Пример:**

Предположим, у нас есть модель "сущность-связь" для простой системы заказов, состоящей из трех сущностей: Клиент, Заказ и Товар. Каждый клиент может размещать несколько заказов, и каждый заказ может содержать несколько товаров. Сущности имеют следующие атрибуты:  
1) Клиент (Client): ID, Имя, Адрес  
2) Заказ (Order): ID, Дата, Сумма  
3) Товар (Product): ID, Название, Цена  
Тогда переход к реляционной БД будет выглядеть примерно так:  
1) Создаем таблицу "Client" с колонками: ID (первичный ключ), Имя, Адрес.  
2) Создаем таблицу "Order" с колонками: ID (первичный ключ), Дата, Сумма, ClientID (внешний ключ, ссылается на ID в таблице "Client").  
3) Создаем таблицу "Product" с колонками: ID (первичный ключ), Название, Цена.  
4) Создаем таблицу-связь "OrderProduct" с колонками: OrderID (внешний ключ, ссылается на ID в таблице "Order"), ProductID (внешний ключ, ссылается на ID в таблице "Product").

##### Структурированный язык запросов SQL, история развития, разновидности, особенности.

SQL (Structured Query Language) — стандартизированный язык программирования, используемый для управления реляционными базами данных (РБД). SQL предоставляет набор команд для создания, изменения, удаления и извлечения данных из базы данных.

История развития SQL:

SQL был разработан в IBM Research в конце 1970-х годов. Изначально он был разработан как язык для работы с системой управления базами данных (СУБД) под названием System R. Затем он был расширен и стандартизирован в 1986 году ANSI (American National Standards Institute) и в 1987 году ISO (International Organization for Standardization). Последующие версии стандарта SQL были выпущены в 1992, 1999, 2003, 2006, 2008, 2011, 2016 и 2019 годах.  
Разновидности SQL:

**MySQL**: Открытая реляционная СУБД, часто используется в веб-приложениях. Поддерживает множество расширений и дополнительных функций.

**Oracle**: Мощная коммерческая СУБД, которая обеспечивает широкий набор возможностей для управления данными и обработки транзакций.

**Microsoft SQL Server**: Коммерческая СУБД, разработанная Microsoft. Широко используется в корпоративных окружениях для обработки больших объемов данных и обеспечения безопасности.

Особенности SQL:  
1) Декларативный язык: SQL является декларативным языком, что означает, что вы указываете, что вы хотите получить (что), а не как его получить (как). Вы формулируете запросы, описывающие данные, которые вы хотите получить или изменить, и СУБД самостоятельно определяет оптимальный способ выполнения запроса.  
2) Манипулирование данными: SQL обеспечивает команды для добавления, изменения и удаления данных в базе данных. С помощью SQL можно создавать таблицы, определять связи между ними, вставлять новые данные, обновлять их и удалять.  
3) Запрос данных: SQL позволяет формулировать запросы для извлечения данных из базы данных. Вы можете указать условия, сортировку и объединение таблиц для получения нужных результатов.  
4) Управление структурой базы данных: SQL предоставляет команды для создания и изменения структуры базы данных. Вы можете создавать таблицы, определять индексы, ограничения, представления и другие объекты базы данных.  
5) Транзакции и контроль целостности: SQL поддерживает концепцию транзакций для обеспечения целостности данных. Вы можете группировать операции в транзакции, что позволяет выполнять несколько операций как единое целое и обеспечивать согласованность данных.

##### SQL. Создание файла базы данных. Создание таблиц БД. Открытие, закрытие таблиц базы данных. Другие способы. Примеры.

Новая база данных создается с помощью оператора **SQL CREATE DATABASE,** за которым следует имя создаваемой базы данных. *Например*, для создания новой базы данных под названием MySampleDB в командной строке mysql нужно ввести следующий запрос:

| CREATE DATABASE MySampleDB; |
| --- |

Новые таблицы добавляются в существующую базу данных с помощью оператора **CREATE TABLE SQL**. За оператором CREATE TABLE следует имя создаваемой таблицы, а далее через запятые список имен и определений каждого столбца таблицы:

**CREATE TABLE** имя\_таблицы ( определение имени\_столбца, определение имени\_таблицы ..., **PRIMARY KEY**= (имя\_столбца) ) **ENGINE**= тип\_движка;  
  
В определении столбца ​​задается тип данных, может ли столбец быть NULL, AUTO\_INCREMENT. Оператор CREATE TABLE также позволяет указать столбец (или группу столбцов) в качестве первичного ключа.  
Прежде чем будет создавать таблицу, нужно выбрать базу данных. Это делается с помощью оператора SQL USE:

| USE MySampleDB; |
| --- |

Как открыть таблицу в SQL? Для этого надо открыть SQL Server Management Studio, найти в разделе «Databases» нужную базу и раскрыть ее. Затем в разделе «Tables» выбрать таблицу и правой клавишей мыши вызвать контекстное меню.

Закрытие происходит через команду ……..

##### SQL. Инструкции и имена. Типы данных. Изменение таблицы. Удаление таблицы. Синтаксис операторов. Примеры.

SQL представлен множеством **инструкций**, каждая из которых предписывает СУБД выполнить определенное действие: создать таблицу, извлечь данные, добавить в таблицу новые данные и т. п. Инструкция SQL начинается с команды – ключевого слова, описывающего действие, выполняемое инструкцией. Типичными являются команды: *CREATE (создать), INSERT (добавить), SELECT (выбрать), DELETE (удалить).*  
 Следом за командой указывается одно или несколько **предложений**. Предложение описывает данные, с которыми должна работать инструкция, или уточняет действие, выполняемое инструкцией.  
*Например*  – WHERE (где), FROM (откуда), INTO (куда). Многие предложения в качестве параметров содержат имена таблиц или столбцов; некоторые из них могут содержать дополнительные ключевые слова, константы и выражения.

У каждого объекта в базе данных есть **уникальное имя**. Имена используются в инструкциях SQL и указывают, над каким объектом базы данных инструкция должна выполнить действие. В соответствии со стандартом ANSI/ISO имена в SQL могут содержать от 1 до 18 символов, начинаться с буквы и не должны включать пробелов или специальных символов пунктуации.

  
Рисунок - Типы данных SQL

**ALTER TABLE** — изменение таблицы в SQL

Синтаксис оператора ALTER TABLE выглядит следующим образом:

| ALTER TABLE название\_таблицы [WITH CHECK | WITH NOCHECK] { ADD название\_столбца тип\_данных\_столбца [атрибуты\_столбца] |  DROP COLUMN название\_столбца |  ALTER COLUMN название\_столбца тип\_данных\_столбца [NULL|NOT NULL] |  ADD [CONSTRAINT] определение\_ограничения |  DROP [CONSTRAINT] имя\_ограничения} |
| --- |

Полное удаление данных

Для полного удаления данных, очистки таблицы применяется команда **TRUNCATE TABLE.** Например, очистим таблицу Clients:

| TRUNCATE TABLE Clients; |
| --- |

Удаление таблиц

Для удаления таблицы из БД применяется команда **DROP TABLE**, после которой указывается название удаляемой таблицы. Например, удалим таблицу Clients:

| DROP TABLE Clients; |
| --- |

##### SQL. Объявление ограничений. Использование ограничений для исключения Null-значений. Обеспечение уникальности значений. Именование и удаление ограничений.

**Ограничение (constraints)** — это ограничение типа значений, которое накладывается на один или несколько столбцов таблицы. Это позволяет поддерживать точность и целостность данных в таблице БД.

В SQL существует несколько различных типов ограничений, в том числе:

· NOT NULL  
· PRIMARY KEY  
· UNIQUE  
· DEFAULT  
· FOREIGN KEY  
· CHECK

Ограничение **NOT NULL**

Ограничение NOT NULL указывает, что столбец не может принимать значения NULL.

Если к столбцу применено ограничение NOT NULL, вы не сможете вставить новую строку в таблицу без добавления не-NULL-значения в этот столбец.

**Пример**

Следующая SQL-инструкция создает таблицу persons с четырьмя столбцами, из которых три столбца — id, name и phone — не могут иметь значение NULL.

| CREATE TABLE persons (  id INT NOT NULL,  name VARCHAR(30) NOT NULL,  birth\_date DATE,  phone VARCHAR(15) NOT NULL ); |
| --- |

**Примечание**. Нулевое значение (NULL) — это не ноль(0) и не строка символов нулевой длины (''). NULL означает, что записи нет.

Ограничение **PRIMARY KEY**

Ограничение PRIMARY KEY определяет столбец или набор столбцов, значения которых однозначно идентифицируют строку в таблице. То есть никакие две строки в таблице не могут иметь одинаковое значение первичного ключа. Также нельзя вводить значение NULL в столбец первичного ключа.

**Примечание.** Первичный ключ обычно состоит из одного столбца в таблице, однако несколько столбцов могут составлять первичный ключ. Например, адрес электронной почты сотрудника или присвоенный ID является логическим первичным ключом для таблицы сотрудников.

Ограничение **UNIQUE**

Ограничение UNIQUE означает, что в указанных столбцах обязательно должны быть уникальные значения.

Хотя и ограничение UNIQUE, и ограничение PRIMARY KEY обеспечивают уникальность значений, есть различия.

UNIQUE лучше PRIMARY KEY, когда вы хотите обеспечить уникальность столбца или комбинации столбцов, которые не являются первичным ключом.

Пример

| CREATE TABLE persons (  id INT NOT NULL PRIMARY KEY,  name VARCHAR(30) NOT NULL,  birth\_date DATE,  phone VARCHAR(15) NOT NULL UNIQUE ); |
| --- |

**Примечание**. В одной таблице может быть задано ограничений UNIQUE, но только одно ограничение PRIMARY KEY. Кроме того, в отличие от ограничений PRIMARY KEY, ограничения UNIQUE допускают значения NULL.

Ограничение **DEFAULT**

Ограничение DEFAULT определяет значение по умолчанию для столбцов.

Значение столбца по умолчанию — это некоторое значение, которое будет вставлено в столбец базой данных, если оператор INSERT явно не назначит конкретное значение.

Пример

| CREATE TABLE persons (  id INT NOT NULL PRIMARY KEY,  name VARCHAR(30) NOT NULL,  birth\_date DATE,  phone VARCHAR(15) NOT NULL UNIQUE,  country VARCHAR(30) NOT NULL DEFAULT 'Россия' ); |
| --- |

**Примечание.** Если вы определили столбец таблицы как NOT NULL, но присвоили ему значение по умолчанию, то в операторе INSERT вам не нужно явно присваивать значение этому столбцу, чтобы вставить новую строку в таблицу.

Ограничение **FOREIGN KEY**

**Внешний ключ (foreign key)** — это столбец или комбинация столбцов, которые используются для установления и обеспечения взаимосвязи между данными в двух таблицах.

Ограничение **CHECK**

Ограничение CHECK используется для ограничения значений, которые могут быть помещены в столбец.

**Примечание**. MySQL не поддерживает ограничение CHECK.

**DROP CONSTRAINT**

Команда DROP CONSTRAINT используется для удаления уникального, первичного ключа, внешнего ключа или ограничения проверки.

##### Взаимосвязи между таблицами. Внешние и родительские ключи. Способ установления взаимосвязей. Примеры.

Связи между таблицами в базе данных — основа хранения данных вСУБД..

Связи в базе данных MS SQL позволяютнормализовать БД, настроить отношение между данными таблиц и сделать эффективные выборки данных. Главное — понять, как настраивать и использовать связи между таблицами MS SQL. Это необходимое условие для работы с любой БД.

**Foreign Key**

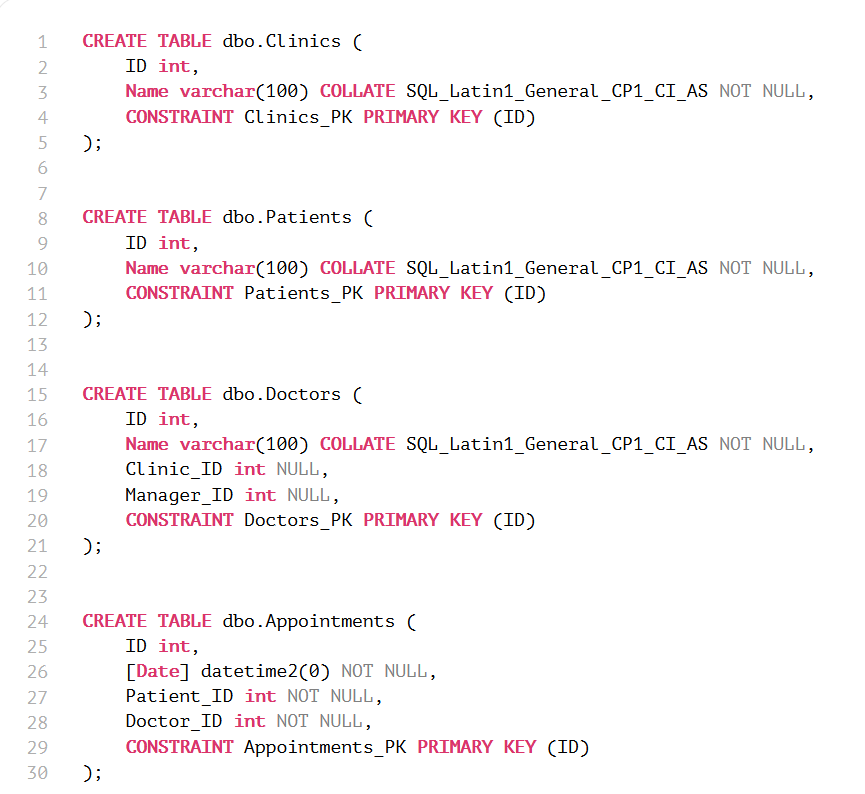
Создание связей MS SQL между таблицами происходит через внешний ключ (foreign key). Данный ключ связывает поле (значение) исходной таблицы с Primary Key внешней таблицы. Через внешний ключ можно не только производить выборку данных, но и контролировать удаление данных в главной таблице:

* NO ACTION — не производит никаких действий;
* SET NULL — зависимые данные установятся в NULL при удалении записи из главной таблицы (primary table);
* CASCADE — удаляются зависимые данные. Опасная операция. В реальной жизни используется редко

**Типы отношений между таблицами**

Используем [Foreign Key](https://sky.pro/media/svyazi-mezhdu-tabliczami-sql/?ysclid=li7btd3av9473951309#Foreign-Key) и [JOINs](https://sky.pro/media/svyazi-mezhdu-tabliczami-sql/?ysclid=li7btd3av9473951309#JOINs) и создадим реальный пример бизнес-задачи. Введем сущности Clinics, Doctors, Patients и Appointments.

* Доктор работает или не работает только в одной клинике.
* У доктора может быть вышестоящий менеджер.
* Пациент может обращаться в разные клиники к разным докторам.



**Отношения «один к одному»**

Используйте данную связь, когда значению из таблицы соответствует только одна запись из внешней таблицы. «Доктор может работать только в одной клинике».

**Отношение «один ко многим»**

Одной записи из таблицы соответствуют несколько записей из внешней. Данный тип связи очень распространен при построении схемы БД.

«Хотя доктор может принадлежать только одной клинике, клиники, в свою очередь, содержат штат докторов».

**Отношение «многие ко многим»**

Организуется через промежуточную таблицу, в которой есть внешние ключи на разные таблицы.

В таблице Appointments есть связь на таблицы Doctors и Patients. Таким образом, организована связь между пациентами и докторами: пациент может посещать нескольких докторов, а доктора — принимать нескольких пациентов.

**Связь с самим собой**

Такой тип связи называется рекурсивным, или иерархическим: связывание строки со строкой из той же таблицы. Полезно при отображении древовидной структуры.

Таблица Doctors содержит колонку Manager, в которой указано, кто из докторов является менеджером текущего доктора. Здесь связь на строку из той же таблицы докторов.

##### SQL. Управление записями. Команды добавления, редактирования и удаления записей. Синтаксис команд. Примеры.

**SQL-запрос на добавление новой записи в таблицу:**

| INSERT INTO users (login, pass) values('TestUser', '123456') |
| --- |

При добавлении записи вначале идёт команда "INSERT INTO", затем название таблицы, в которую мы вставляем запись. Далее идет в круглых скобках названия полей, которые мы хотим заполнить. А затем в круглых скобках после слова "values" начинаем перечислять значения тех полей, которые мы выбрали. После выполнения этого запроса в нашей таблице появится новая запись.

**Для изменения** уже имеющихся строк в таблице применяется команда UPDATE. Вначале идёт команда "UPDATE", затем имя таблицы, а после "SET" мы описываем значения всех полей, которые мы хотим изменить.

| UPDATE имя\_таблицы SET столбец1 = значение1, столбец2 = значение2, ... столбецN = значениеN [FROM выборка AS псевдоним\_выборки] [WHERE условие\_обновления] |
| --- |

**Удаление из базы данных** происходит с помощью команды "DELETE". Функция удаляет не одну строку, а несколько, при этом выбирает для удаления строки по логике функции "SELECT". То есть чтобы удалить данные из базы, необходимо точно определить их.

| DELETE [FROM] имя\_таблицы WHERE условие\_удаления |
| --- |

##### Индексирование. Понятие и виды индексов. Уникальные индексы. Активация индекса. Удаление индекса. Переиндексирование: назначение и команда.

**Индекс** - структура данных, которая помогает СУБД быстрее обнаружить отдельные записи в файле и сократить время выполнения запросов пользователей.

Без индекса SQL-сервер должен сканировать всю таблицу, чтобы найти и вернуть запрошенные данные. Это как с оглавлением в книге: если его нет, то вы долго будете искать нужный вам материал, перелистывая страницу за страницей.

Инструкция **CREATE INDEX** используется для создания индексов в таблицах.

| CREATE INDEX index\_name ON table\_name (column1, column2, ...); |
| --- |

Инструкция **DROP INDEX** используется для удаления индекса в таблице.

| DROP INDEX table\_name.index\_name; |
| --- |

* **Составной индекс**

Такой индекс может содержать более одного столбца. Вы можете включить до 16 столбцов в индекс, но их общая длина ограничена 900 байтами. Как кластеризованный, так и некластеризованные индексы могут быть составными.

* **Уникальный индекс**

Такой индекс обеспечивает уникальность каждого значения в индексируемом столбце. Если индекс составной, то уникальность распространяется на все столбцы индекса, но не на каждый отдельный столбец. К примеру, если вы создадите уникальных индекс на столбцах ИМЯ и ФАМИЛИЯ, то полное имя должно быть уникально, но отдельно возможны дубли в имени или фамилии.

Уникальный индекс автоматически создается когда вы определяете ограничения столбца: первичный ключ или ограничение на уникальность значений

* **Кластеризованный индекс**

Видоизменяет индексируемую таблицу: когда вы создаете кластерный индекс для столбца (или ряда столбцов), SQL-сервер сортирует всю таблицу относительно этого столбца (столбцов). Таким образом, индекс находится внутри таблицы. По аналогии такой тип индекса – книга, которая преобразовалась в словарь в алфавитном порядке.

Обратите внимание, что для каждой таблицы можно создать только один кластеризованный индекс, потому что в данном случае индекс – сама таблица.

Чтобы создать кластеризованный индекс, следует воспользоваться следующим запросом:

| -- Создаем таблицу с нужным количеством колонок  CREATE TABLE [БазаДанных].[Схема].[ИмяТаблицы]   ([Столбец1] int NOT NULL,   [Столбец2] nchar(10) NULL,   [Столбец3] nvarchar(50) NULL);  /\* Создаем индекс с нужным нам названием (например, "Индекс1")   и с нужным количеством колонок в индексе \*/  CREATE CLUSTERED INDEX [Индекс1]  ON [БазаДанных].[Схема].[ИмяТаблицы] ([Столбец1],[Столбец2]); |
| --- |

* **Некластеризованный индекс**.

Используя его, вы не видоизменяете таблицу, не превращаете ее в словарь. Вместо этого вы создаете в таблице совершенно другой объект, который содержит столбцы, выбранные для индексации, и указатель на строки таблицы, содержащие данные.

Таким образом, данный тип индекса похож на оглавление в книге: оно хранится отдельно от основного текста книги и указывает на какой странице содержится интересующая вас информация.

Для каждой таблицы можно создать несколько некластеризованных индексов.

Чтобы создать некластеризованный индекс, следует воспользоваться следующим запросом:

-- Создаем таблицу с нужным количеством колонок   
CREATE TABLE [БазаДанных].[Схема].[ИмяТаблицы]   
 ([Столбец1] int NOT NULL,   
 [Столбец2] nchar(10) NULL,   
 [Столбец3] nvarchar(50) NULL);   
  
/\* Создаем индекс с нужным нам названием (например, "Индекс1")   
 и с нужным количеством колонок в индексе \*/   
CREATE NONCLUSTERED INDEX [Индекс1]  
 ON [БазаДанных].[Схема].[ИмяТаблицы] ([Столбец1],[Столбец2]);

##### Инструкция SQL для создания запросов на выборку. Запрос на выборку данных: выборка данных из одной таблицы или из нескольких таблиц, с условием отбора записей. Примеры.

SELECT SQL — оператор запроса, который возвращает определенный набор данных из базы данных. Список столбцов выборки задается в части оператора, которая называется предложением оператора SELECT.

Для получения данных применяется команда SELECT. В упрощенном виде она имеет следующий синтаксис:

| SELECT список\_столбцов FROM имя\_таблицы |
| --- |

Если нам надо получить данные не по всем, а по каким-то конкретным столбцам, то тогда все эти спецификации столбцов перечисляются через запятую после SELECT:

| SELECT ProductName, Price FROM Products |
| --- |

Для выбора определённых строк таблицы вместе с оператором SELECT уже потребуется ключевое слово WHERE, указывающее на некоторое значение или несколько значений, содержащиеся в интересующих нас строках. Наиболее простые условия задаются при помощи операторов сравнения и равенства (<, >, =), а также ключевого слова IS. Условий может быть несколько, тогда они перечисляются с использованием ключевого слова AND. Запросы для выбора строк имеют следующий синтаксис:

| SELECT ИМЯ\_СТОЛБЦА FROM ИМЯ\_ТАБЛИЦЫ WHERE УСЛОВИЕ |
| --- |

**Примеры использования SELECT SQL**

Если у вас есть таблица employees со столбцами id, name, age, department, salary, вы можете выбрать только имена и возраст всех сотрудников следующим образом:

| SELECT name, age FROM employees; |
| --- |

##### SQL. Поиск и фильтрация данных. Операторы IN, BETWEEN, LIKE, IS NULL. Использование неравенств и логических операторов. Примеры.

Операторы **IN, BETWEEN, LIKE, IS NULL** используются для поиска и фильтрации данных в SQL.

Оператор **IN** позволяет выбрать строки, значения которых соответствуют заданным значениям. **Например:**

| SELECT \* FROM table WHERE column IN (value1, value2, ...); |
| --- |

Оператор **BETWEEN** выбирает строки, значения которых находятся в заданном диапазоне. **Например:**

| SELECT \* FROM table WHERE column BETWEEN value1 AND value2; |
| --- |

Оператор **LIKE** используется для поиска строк, которые содержат определенный шаблон. **Например**:

| SELECT \* FROM table WHERE column LIKE '%value%'; |
| --- |

Оператор **IS NULL** выбирает строки, у которых значение заданного столбца равно NULL. **Например:**

| SELECT \* FROM table WHERE column IS NULL; |
| --- |

Также можно использовать **неравенства** (<, >, <=, >=) и **логические операторы** (AND, OR, NOT) для фильтрации данных. **Например:**

| SELECT \* FROM table WHERE column1 > value1 AND column2 < value2; |
| --- |

Эти операторы позволяют создавать более сложные запросы для поиска и фильтрации данных в SQL.

##### SQL. Поиск и фильтрация данных. Вычисление итоговых значений с помощью агрегатных функций. Предложение GROUP BY и HAVING Примеры.

**Поиск и фильтрация данных в SQL** - это процесс выбора определенных строк из таблицы на основе заданных условий. Для этого используется предложение WHERE, которое определяет условия, по которым нужно выбрать данные. **Например:**

| SELECT column1, column2 FROM table WHERE column1 = 'value'; |
| --- |

**Агрегатная функция** выполняет вычисление на наборе значений и возвращает одиночное значение. Эти функции(SUM, AVG, COUNT, MIN, MAX.), за исключением COUNT(\*), не учитывают значения NULL. Агрегатные функции часто используются в выражении GROUP BY инструкции SELECT.

Все агрегатные функции являются детерминированными, то есть возвращают одну и ту же величину при каждом их вызове на одном и том же наборе входных значений. **Пример использования агрегатной функции:**

| SELECT SUM(column1) FROM table; |
| --- |

Предложение **GROUP BY** позволяет группировать строки по определенному столбцу и применять к каждой группе агрегатную функцию. **Например:**

| SELECT column1, SUM(column2) FROM table GROUP BY column1; |
| --- |

Предложение **HAVING** позволяет фильтровать группы, удовлетворяющие заданному условию. **Например:**

| SELECT column1, SUM(column2) FROM table GROUP BY column1 HAVING SUM(column2) > value; |
| --- |

##### SQL. Сортировка данных. Работа с выражениями. Упорядочение агрегатных групп. Использование ORDER BY с NULL – значениями. Пример

##### **Сортировка данных в SQL** - это процесс упорядочивания результатов запроса по заданному столбцу или нескольким столбцам. Для сортировки используется предложение ORDER BY, которое указывает столбец или столбцы, по которым нужно отсортировать данные, и порядок сортировки (возрастание или убывание). **Например:**

| SELECT column1, column2 FROM table ORDER BY column1 DESC; |
| --- |

**Работа с выражениями в SQL** означает использование математических, логических и строковых операций для получения новых значений на основе имеющихся данных в таблице. Выражения могут быть использованы в различных частях SQL-запросов, таких как SELECT, WHERE, HAVING и других.

Например, можно использовать арифметические операции для вычисления новых значений на основе имеющихся данных, **например:**

| SELECT column1 + column2 AS sum FROM table; |
| --- |

**Упорядочение агрегатных групп** в SQL означает упорядочивание результатов агрегатных функций (например, суммирования, подсчета количества и т.д.) по определенным столбцам таблицы.

**При использовании оператора ORDER BY** в SQL, если в столбце есть NULL-значения, они могут быть отображены в начале или конце результата сортировки, в зависимости от направления сортировки и настроек базы данных.

**(1 пример использования)** если мы хотим отсортировать таблицу по возрастанию значения столбца "age", но в этом столбце есть NULL-значения, то они будут отображаться в начале результата:

| SELECT \* FROM users\_table  ORDER BY age ASC; |
| --- |

Если же мы хотим, чтобы NULL-значения отображались в конце результата, можно использовать операторы IS NULL и IS NOT NULL в комбинации с оператором ORDER BY**(2 пример использования)**:

| SELECT \* FROM users\_table  ORDER BY age IS NULL, age ASC |
| --- |

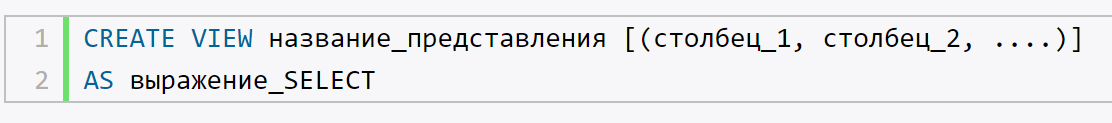
##### SQL. Хранимые объекты БД. Создание, удаление и особенности работы с представлениями. Синтаксис. Пример.

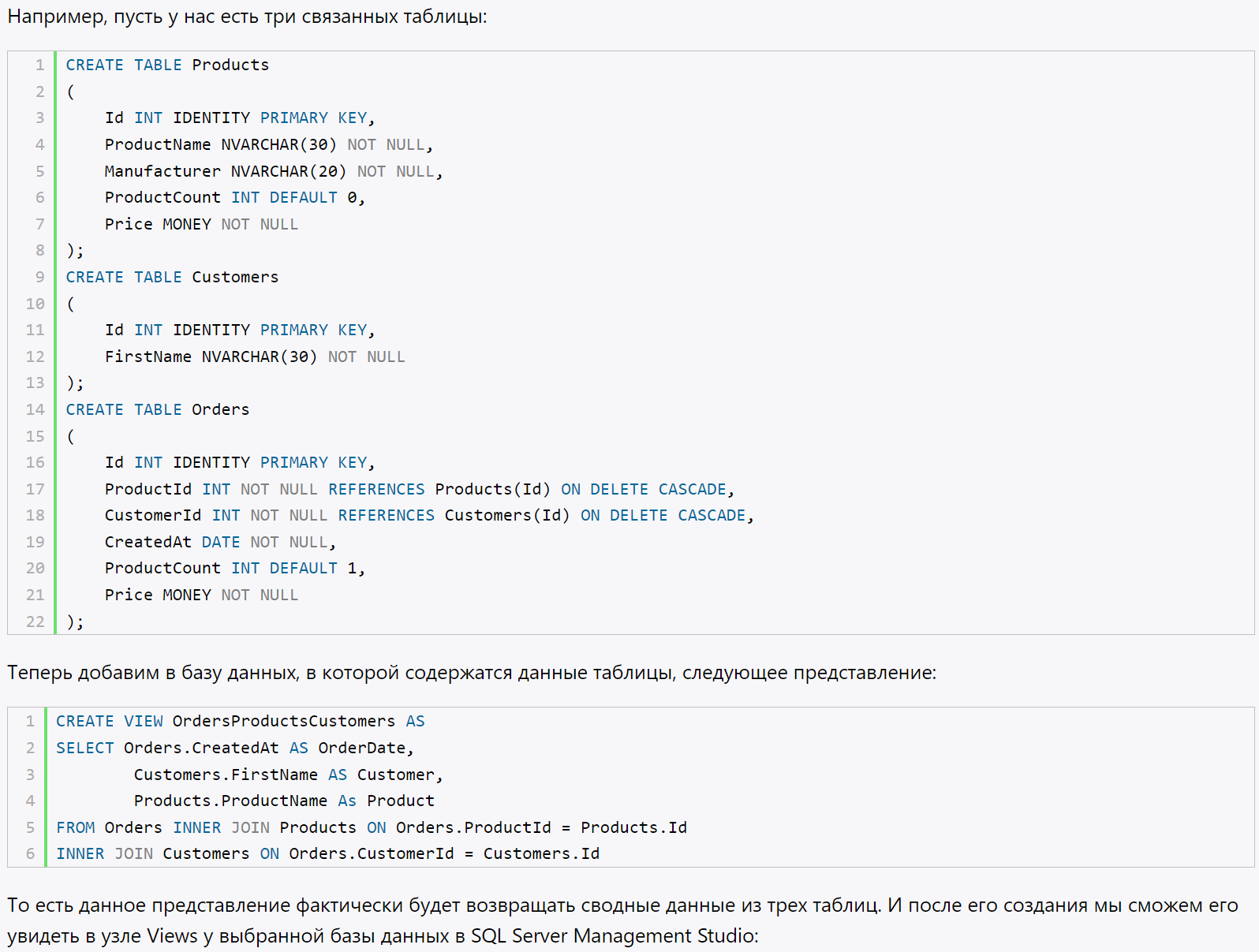
Хранимые объекты БД— основное назначение — предоставить уникальный идентификатор, по которому его можно будет отличить. Кроме того, каждый объект обладает типом.(???)

**Представления** или Views представляют виртуальные таблицы. Но в отличии от обычных стандартных таблиц в базе данных представления содержат запросы, которые динамически извлекают используемые данные.

Представления дают нам ряд преимуществ. Они упрощают комплексные SQL-операции. Они защищают данные, так как представления могут дать доступ к части таблицы, а не ко всей таблице. Представления также позволяют возвращать отформатированные значения из таблиц в нужной и удобной форме.

Для **создания представления** используется команда CREATE VIEW, которая имеет следующую форму:





Для **удаления представления** вызывается команда DROP VIEW:



Также стоит отметить, что при удалении таблиц также следует удалить и представления, которые используют эти таблицы.

**Особенности работы**

Представления дают нам ряд преимуществ. Они упрощают комплексные SQL-операции. Они защищают данные, так как представления могут дать доступ к части таблицы, а не ко всей таблице. Представления также позволяют возвращать отформатированные значения из таблиц в нужной и удобной форме.

При создании представлений следует учитывать, что представления, как и таблицы, должны иметь уникальные имена в рамках той же базы данных.

Также можно создавать представления на основе других представлений. Такие представления еще называют вложенными (nested views).