Лекция 1

База данных как компонента информационной системы

Информационная система – совокупность аппаратных и программных компонент, обеспечивающие выполнение следующих пяти функций относительно информации:

1. Сбор
2. Обработка
3. Хранение
4. Передача
5. Представление

Паттерны проектирования – некие готовые шаблоны, методы по структуированию программного кода, чтобы его можно было эффективно разрабатывать, тестировать и эксплуатировать.

Так как существует много паттернов проектирования, то поэтому логично, что идеальных решений не сущетсвует. Из-за этого часто приходится искать компромиссы в функциональности системы. Однако также приходится искать компромиссы в основных нефункциональных требованиях:

- Производительность
- Надежность
- Безопасность

Примеры противоречий:

1. Чтобы улучшить надежность хранилища, нужно дублировать данные, каналы связи. Но здесь происходит ущерб по производительности (нужно тратить накладные расходы на дублирование).
2. Чтобы улучшить надежность, продублируем узлы обработки, хранения. Но так как у нас теперь больше узлов, значит, сложнее обеспечивать безопасность.
3. Чтобы улучшить безопаность, делаем шифрование данных, аудит, усиливаем контроль доступа. Как результат, получаем ущерб по производительности.

Вывод: любое решение по трем параметрам выше не может быть идеальным. Значит, нужно расставлять приоритеты и искать компромиссы.

Существует также проблема масшабируемости приложений при росте пользователей, так как ресурсы серверов ограничены. Более того, переделывание архитектуры приложений очень затратно по усилиям и времени и противоречит принципам выше.

Помимо этого, нужно обеспечивать обратную совместимость с предыдущими версиями. Итог: очередное противоречие.

Данные (стандарт ISO-2382) – это поддающееся многократной интерпретации представление информации **в формализованном виде** (то есть существуют формальные правила для описания информации), пригодным для передачи, представления или обработки).

EX: Храним информациию о связях студент – группа и преподаватель – группа. Если мы будем в обоих случаях именовать группы строго одинаково (то есть формализовать), то тогда мы сможем узнавать

дополнительные факты: понять у какого студента ведет какой преподаватель, у какого преподавателя наибольшее количество студентов (при том, что эти данные мы не хранили).

Вывод: формализация помогает устанавливать большое множество сложных производных связей (то есть тех, которые мы даже не хранили в первичном виде). Однако еще раз: присутствие этой возможности зависит от формы хранения данных. Также форма хранения данных в значительной степени влияет на функциональные и нефункциональные требования.

Хранение данных в виде файла

Файл – уникальная именованная совокупность данных, к которой можно обращаться и работать с операциями чтения и записи.

Минусы:

- Отсутствие семантики относительно того, что находится в файле.
- Трудно искать нужную информацию в большом файле.
- Нельзя обеспечить консистентность данных. EX: что делать в случае, когда один пользователь читает файл, а другой в это время записывает в него информацию → неактуальность данных при чтении.
- Можно разбить один большой файл на несколько маленьких и при чтении / записи данный файл блокируется клиентом и доступен только ему. Проблемы будут возникать, когда двум клиентам нужно получать доступ к файлу, где сейчас работает другой для него клиент. Как результат, никто не может получить нужную для связи информации и происходит тупик. Другой пример: первый хочет считать из второго файла, второй хочет считать из третьего, третий хочет считать из первого – получили кольцо.
- Дублирование данных: при повторном внесении данных о чем-либо нужно проверять уже существование этих данных дабы не получить их повторение или противоречие → запись становится медленее.
- Зависимость от типов данных. Иногда необходимо увеличить количество символов для поля какого-либо типа данных, но тогда это нужно делать и для всех остальных полей этого типа – ресурсоемко.
- Когда над файлом работает команда, то необходима договоренность о структурах хранения файла.
- Невозможность сделать файловое хранилище, одновременное эффективно для записи и чтения. Сущетсвует два возможных варианта организации файлов: неупорядоченные (вставка O(1), поиск O(n)) и упорядоченные (вставка O(n), поиск O(log(n))).

Задача: нужно уметь разделить (абстрагировать) доступ и работу с данными от операций обработки.

Файл-серверная архитеткура. У клиента есть интерфейс, обработка, доступ к данным. На сервере есть только хранение данных. Я нажал на кнопку в интерфейсе, у меня запустился код обработки, далее выясняется, что мне нужно найти информацию о неком студенте, содержащемся в одном из полей интерфейса, я запрашиваю файл, мне целиком передается файл, я роюсь в этом файле ищу там данные и отда их в обработку, обработчик их преобразует и красиво отображает в интерфейсе.

Файл-сервер		**Клиент-сервер**	
Клиент	Сервер	Клиент	Сервер
Интерфейс Обработка Доступ к данным	Хранение данных	Интерфейс Обработка	Доступ к данным Хранение данных

Клиент-серверная архитектура. Доступ и хранение полностью отнесены на некоторый файл-сервер. Отличие в том, что здесь не я роюсь в этом файле (который итак мог долго передаваться по сети из-за своего большого размера): мы запрашиваем у сервера информацию о данном студенте, у сервера есть компонента, которая знает, как устроен сам файл и где хранятся данные об этом студенте. После поиска нужной информацию сервер сам отдаст готовый результат. *Можно также часть обработки перенести на этот сервер.

Толстый клиент – клиент, на котором оставляется большая часть обработки, иначе если большинство обработки уходит на сервер, то тогда клиент тонкий.

Вывод: нужно сложить понятия доступа и хранения данных и тогда появляется термин база данных как некоторое структуированное хранилище данных, где хранится семантика этих данных и появляется СУБД, обеспечивающее доступ и хранение.

Лекция 2

Виды моделей

Моделирование – процесс перехода от информации к данным.

Международный стандарт **ANSI-SPARC** определяет подход к моделированию данных, выделяя три уровня архитектуры данных в порядке последовательного рассмотрения информации на этих уровнях:

1. **Внешний уровень**. Представление БД с позиции конечного пользователя. Как правило, представление может отличаться от того, что на самом деле хранится в БД. ex: пользователю важен возраст сотрудника, но с точки зрения хра мы будем хранить не возраст, а дату рождения или UNIX-time (количество секунд, прошедших с 1 января 1970 года).
2. **Концептуальный уровень**. На этом уровне строится обобщающее представление БД, то есть, какие данные хранятся в базе и выделяются их сущности, атрибуты и связи между собой. ex: студент числится в группе, преподаватель ведет дисциплину. Помимо этого, также определяются домены, то есть области допустимых значений для атрибутов. ex: для атрибута имя подходят только имена не более 15 символов без цифр из справочника. Иногда домены задаются регулярными выражениями. Также строится семантика данных. ex: можно хранить рейтинг строками: плохо, хорошо, отлично (много памяти), а можно в виде чисел: 0, 1, 2. Тогда получаем семантику данных: 0 – плохо, 1 – хорошо, 2 – отлично. То есть храним одно, но под семантикой имеем в виду другое. Определяются способы безопасности и проверки консистентность (согласованность данных друг с другом, целостность данных, а также внутренняя непротиворечивость.) данных: мы не можем хранить в БД противоречивые друг другу данные или связи. ex: удаление является возможной причиной нецелостности данных.
3. **Внутренний уровень**. Физическое представление БД в компьютере. На этом уровне решается ряд важных вопросов: как оптимизировать хранение данных по памяти, какие типы данных хранить для каждого поля (может отличаться от концептуального уровня), как размещать таблицы БД в файлах, сжатие и шифрование данных (что шифровать, в какой мере, как оптимальнее).

Типы моделей на концептуальном уровне

ER-модель

История: была предложена известным теоретиком данных Питером Ченом в 1976 году.
Суть: основные объекты, с которыми работает, - сущности и связи.

Сущность

Сущность – множество экземпляров, реальных или абстрактных, однотипных объектов предметной области. Сущности делятся на два класса:

- Сильные сущности. Существуют независимо от других сущностей.
- Слабые сущности. Существуют только в связи с другой сущностью (слабой или сильной).

EX: Есть БД для электронного магазина, есть сущности клиент и заказ. Тогда клиент – сильная сущность (может существовать без заказа, ex: сразу после регистрации на сайте), а заказ – слабая (так как заказ не может быть без клиента).

Атрибут

Атрибут (Чен) – это функция, отображающая набор сущностей в набор значений или в декартово произведение набора значений. **Сущность** фактически определяется как совокупность атрибутов. ex: студент имеет следующие атрибуты: возраст, цвет волос, табельный номер. У атрибутов есть домены. EX: возраст – атрибут: возраст отображается в натуральные числа от 0 до 99. **Ключевой атрибут** (ключ сущности) – это группа атрибутов такая, что отображение набора сущностей в соотвествующую группу наборов значений является взаимно-однозначным. ex: табельный номер – ключевой атрибут, так как каждая сущность студента взаимно-однозначно отображается в табельный номер.

Классификация атрибутов:

1. простые и составные
   1.1. простые
   1.2. составные. EX: адрес, в котором указаны город, индекс, улица, дом, корпус
2. обязательные и необязательные
   2.1. обязательные. EX: имя, фамилия, отчество
   2.2. необязательные. EX: номер телефона
3. однозначные и многозначные
   3.1. однозначные.
   3.2. многозначные: для одной и той же сущности принимают несколько значений. EX: номер телефона (у одного человека их может быть несколько)

Связь

Связь – это ассоциация, установленная между несколькими сущностями. Также между сущностями может быть несколько связей. Связи также могут обратимыми.

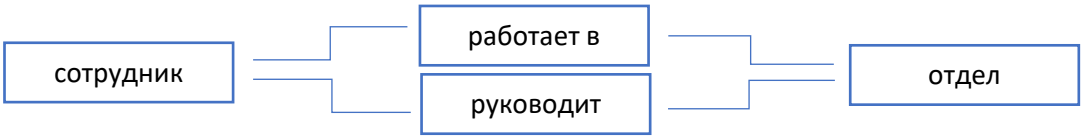
Арность связи – количество сущностей в связи.

Теорема: любая n-арная связь может представлена как совокупность бинарных.

Согласно Чену, все связи должны быть связаны семантикой.

EX: есть сущности сотрудник и отдел и связь “работает в”, получаем связь: сотрудник работает в отделе.

Добавим связь “руководит”: у отдела есть руководитель, он тоже сотрудник, поэтому сотрудник связан связью “руководит” с отделом.



Типы (кардинальность) связей

- **1:1** (один к одному) – связь, в которой каждый экземпляр сущности A связан не более, чем с одним (то есть связь может как быть, но может так и не быть) экземпляром сущности B, а каждый экзампляр сущности B связан не более, чем с одним экземпляром сущности A.
- **1:n** (один ко многим) – связь, в которой каждый экземпляр сущности A может быть связан с 0, 1 или несколькими экземплярами сущности B, а каждый экзампляр сущности B связан не более, чем с одним экземпляром сущности A.
- **n:n** (многие ко многим) - связь, в которой каждый экземпляр сущности A может быть связан с 0, 1 или несколькими экземплярами сущности B, а каждый экземпляр сущности B может быть связан с 0, 1 или несколькими экземплярами сущности A.

В нотациях Чена сначала пишется **модальность**, а затем **кардинальность**.

EX

- 1:1
    - У одного отдела может быть только один руководитель, и один руководитель может руководить только одним отделом.
    - У каждого отдела существует сотрудник, который является руководителем, но не каждый сотрудник является руководителем отдела (такая связь является модальной). Иными словами, у сотрудника может быть 0 или 1 отдел, которыми он руководит, а у отдела может быть только 1 сотрудник, который им руководит.
- 1:n
    - Сотрудник может работать только в 1 отделе, но в 1 отделе может работать более, чем 1 сотрудник. Здесь также присутствует модальность: если есть сотрудник, то он он всегда работает в каком-то отделе, поэтому у сотрудника должен быть отдел, значит, модальность отдела равна 1. Но может быть отдел, в котором нет вообще или есть несколько сотрудников (руководителей не рассматриваем), значит, модальность сотрудников равна 0 или n. На схемах обозначается “вороньей лапкой”.
- n:n
    - Связь “преподаватель читает дисциплину”. У преподавателя более, чем 1 дисциплина. У преподавателя модальность равна 1, то есть у дисциплины должен быть преподаватель, но у преподавателя может и не быть дисциплины (ему дали передохнуть).

У связей также могут быть атрибуты. EX: “Преподаватель ведет дисциплину”: преподаватель – лектор (читает лекции), преподаватель – практик.

Связи могут становиться сущностями и наоборот. EX: есть сущность руководитель, тогда отдел имеет руководителя, сотрудник является руководителем, значит, руководитель станет не связью, а сущностью.

Лекция 3

Определения БД и СУБД

База данных (Конноли и Берг) – это совместно используемый набор логически связанных данных, предназначенный для удовлетворения информационных потребностей организаций.

База данных (Дейт) – это набор постоянно хранимых данных, используемых прикладными системами предприятия.

База данных (Хомоненко и Цыганков) – это совокупность специально организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Рекомендованная литература: Конноли и Берг – толстая, очень крутая подробная книжка, но трудно искать нужную информацию. Дейт – все очень строго и компактно, но иногда тяжело читать, задумываясь над каждой его фразой. Хомоненко и Цыганков – все четко и понятно, объяснятся нестрого, на пальцах, но доступным языком.

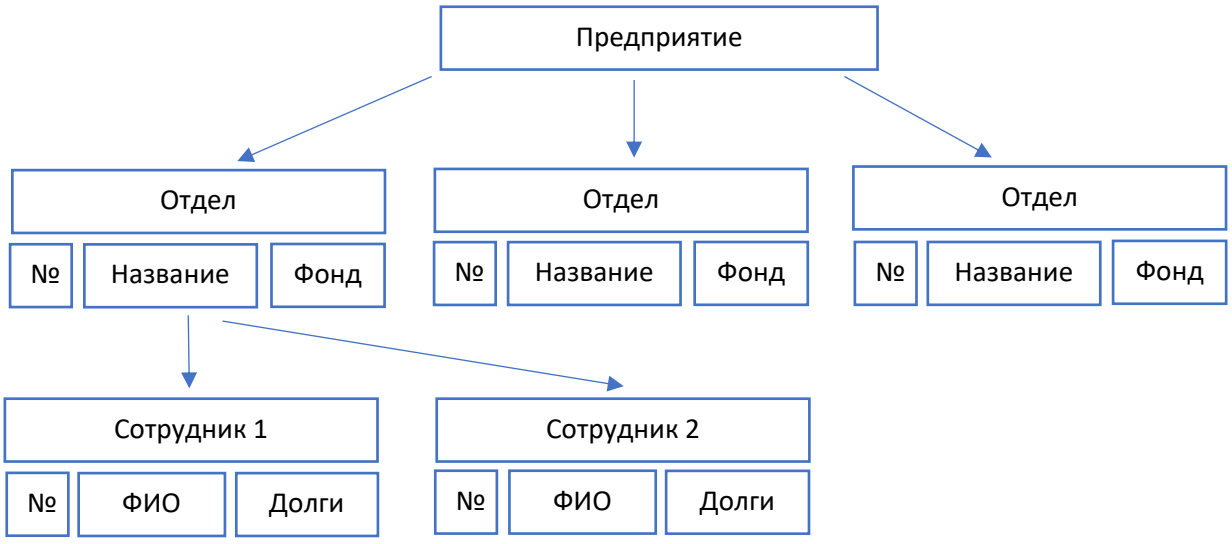
СУБД (Конноли и Берг) – программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать БД, а также осуществлять к ней контролируемый доступ.

СУБД (Хомоненко и Цыганков) – комплекс языковых (могут быть различные языковые диалекты) и программных средств, предназначенный для создания, ведения и совместного использования БД многими пользователями.

Модели данных

Иерархическая (сетевая) модель данных

Концепция: связи между корпоративными объектами – это, как правило, связи подчинения. Иерархическая модель сводится к тому, что мы храним дерево или множество деревьев.



EX: у предприятия есть несколько отделов, каждый отдел – экземпляр сегмента типа “отдел”. В отделе работают сотрудники, каждый сотрудник - экземпляр сегмента типа “сотрудник”.

Поле данных – это минимальная неделимая единица данных.
Сегмент данных – это совокупность полей данных. EX: Сотрудник – это совокупность полей фамилии, имя, отчества, номера телефона и должности.
Экземпляр сегмента – конкретные значения полей. EX: Заполняя данные о человеке, мы получаем конкретный экземпляр сегмента.
Ключ – это значение одного из полей, уникальное для всех экземпляров данного сегмента.

По факту мы храним дерево, состоящее из сегментов. При этом каждое поддерево также является деревом.

Как хранить связи в таком дереве? Потомок хранит номер родителя. EX: у каждого сотрудника хранится идентификатор отдела, в котором он работает.

Плюсы:

- Просто
- Такая связь занимает мало памяти (основное достоинство)

Минусы:

- Проблема возникает, когда мы хотим получить список всех сотрудников данного отдела. Тогда нам нужно будет посмотреть всех сотрудников, чтобы определить, кто из них относится к данному отделу, то есть мы можем только подниматься наверх от сотрудника к отделу, а спускаться – нет. В больших базах данных с уровнями иерархиями 15-20 это весьма долго.
- Целостность данных. После удаления отдела, нужно просмотреть всех сотрудников и тем кто там работал поменять идентификатор. *Альтернатива*: для каждого дерева мы храним полностью всю структуру дерева вниз, то есть для отделов в некоторой структуре храним множество всех его сотрудников. Так можно быстро получать информацию, но любые изменения в данных мы должны контролировать во всех структурах и деревьях.

Иерархическая модель имеет и другой недостаток: любой экземпляр может принадлежать только одному родителю. И при этом могут возникать случаи, когда один сотрудник по факту принадлежит двум родителям в разных деревьях (он работает в отделе и является лицом договора). Если мы реализуем такую логику, то здесь нужно дублировать данные о сотруднике, или делать множественную иерархию, что усложняет целостность данных при их изменении. Как вариант модификации, можно для уникальных объектов хранить ID.

Операция удаления при обычной иерархии происходит легко, но при **множественной иерархии** обеспечивать целостность данных при удалении сложнее: мы удалили отдел, а значит и всех сотрудников, работавших в нем, все ок. Однако удаляя договор, мы не должны вместе с этим удалять и сотрудников. То есть нужно в этих структурах хранить флажки разрешения: что можно удалять, а что нет.

Модель множественной иерархии также называют **сетью**.

Реляционная модель данных

Концепция: значение одного из атрибутов берется из некоторого атрибута другого отношения.

Таблица описывает отношение некоторого множества атрибутов.

Отношение – плоская таблица
Атрибуты отношения – столбцы таблицы
Кортежи (записи) – строки таблицы
Поле – пересечение строки и столбца

Иными словами, мы имеем некоторое множество кортежей, каждый из которых состоит из некоторого множества атрибутов. При этом поле – значение какого-либо атрибута в рамках конкретного кортежа.

В реляционной модели данных, в отличие от иерархической, мы не разделяем хранение данных от хранения связей о том, как связаны эти данные: мы храним и данные, и связи с помощью таблиц (отношений).

Что делать в рамках реляционной модели в случаях, если, например, у сотрудника есть два номера телефона?

- *Вариант 1:* дублировать сотрудника – плохо, так как мы присваиваем два разных ID одному и тому же человеку -> нарушение целостности.
- *Вариант 2:* хранить два телефона в одной ячейке – непонятно, что делать с целостностью атрибутов, так как атрибут “телефон” перестает быть атомарным.
- *Вариант 3:* сделать отдельную таблицу (так обычно и делают) для хранения всевозможных телефонов и между ними ставим таблицу-связку, в которой будут храниться пары: (идентификатор телефона, идентификатор сотрудника). Тогда если у сотрудника 2 телефона, то в этой таблице будут 2 записи. Как недостаток, получаем избыточность с дополнительными таблицами – теперь нужно join’ить таблицы, и поиск занимает больше времени. Чтобы устранить данный недостаток были придуманы постреляционные базы данных.

Постреляционная модель данных

Постреляционная БД по логике такая же, как и реляционная, но снимает запрет на неделимость поля, то есть мы разрешаем полю с помощью перечисления или хранения какой-нибудь структуры данных хранить какие-то не атомарные значения, то есть что-то являющееся совокупностью отдельных значений.

Минусы:

- Как хранить многозначные поля? Можно разделять данные двоеточием, но что делать, если нужно написать двоеточие? Как итог, дополнительные сложности реализации.
- Нельзя проверять БД на целостность данных.

Объектно-реляционная модель данных

Когда пишем код в стиле ООП, то каждый экземпляр класса хранится в БД (аналог кортежа). При поиске или изменении поля у объекта делаем соотвествующий запрос к БД.

JSON – сериализация объекта.
Все базы, которые называются реляционными, по факту являются объектно-реляционными.

Многомерная модель данных

EX: пусть есть данные о продаже: есть товары, которые продавались в разные месяца, в разном объеме, в разных филиалах. Хочется отвечать на запросы: сколько товаров данного типа было продано в этот месяц в этом конкретном филиале. Можно ответить на запрос с помощью GROUP BY, но сама операция GROUP BY является долгой. Идея такая: будет хранить 3-мерную таблицу: (товар x месяц x филиал). В каждой ячейке

храним информацию, сколько товаров данного типа было продано в любой месяц в конкретном филиале. Если нужно добавить еще один атрибут, то мы добавим еще одно измерение.

Концепция: для увеличения скорости ответов на запросы храним данные в многомерном виде. Итог: быстро отвечаем на запросы, но дорого по памяти (вероятно, что будет много значений).

Вывод

Существуют различные модели данных, и у каждой из них есть свои достоинства и недостатки, поэтому нужно искать компромисс. Наиболее компромисной оказывается реляционная модель данных.

Реляционная модель данных (50)

Отношение – двумерная таблица, содержащая данные.
Атрибут – заголовок столбца данной таблицы.
Схема отношения – строка заголовков столбцов (или совокупность атрибутов).
Кортеж (1) – строка таблицы.
Поле (значение атрибута) – значение конкретного атрибута в конкретном кортеже.
Домен – множество допустимых значений отдельного атрибута.
Отношение – некоторое подмножество декартового произведения доменов.
Кортеж (2) – множество конкретных значений атрибутов, где каждое значение принадлежит соотвествующему домену.

Классификация отношений:

1. Именованные и неименованные:
    a. Именованные отношения имеют некоторое уникальное имя
    b. Неимованные отношения не имеют уникального имени.
2. Базовые и производные
    a. Базовое отношение существует независимо.
    b. Производное отношение существует только в связи с некоторыми базовыми.

EX: результат выполнения запроса является неимнованным производным отношением, так как не имеет имени и существует только при наличии базовых таблиц.

Кодд (создатель реляционной модели данных) сформулировал 6 свойств, которыми должна обладать реляционная модель (сейчас это не так):

1. Уникальность имени отношения в реляционной схеме, т.е. каждая таблица должна иметь уникальное имя. Внимание: речь здесь не идет про базы данных.
2. Каждая ячейка содержит только одно неделимое значение.
3. Уникальность имени атрибута (и даже у результатов запроса, так как результат запроса – это тоже отношение) в пределах отношения, т.е. не может быть у двух столбцов одно и то же имя в пределях одного отношения.
4. Все значения атрибута берутся из одного домена (для объектов существует домен “все, что угодно”, где хранится битовая последовательность).
5. В рамках одного отношения каждый кортеж уникален, т.е. не может быть двух одинаковых строк у одной таблицы.

6. Порядки следования атрибутов и кортежей не имеют значения, т.е. столбцы и строки в отношении неупорядоченны.

Виды ключей

Супер-ключ

Супер-ключ – это атрибут или множество атрибутов, единственным образом идентифицирующие кортеж.

EX: Схема отношения также является супер-ключом (так как любой кортеж в рамках одного отношения по Кодду уникален). EX: ID является супер-ключом.

Потенциальный ключ

Потенциальный ключ – это супер-ключ, который не содержит подмножества, также являющего супер-ключом данного отношения.

Иными словами, потенциальный ключ – это минимальный супер-ключ. Очевидно, что на одном отношении можно создать несколько потенциальных ключей.

Первичный ключ

Первичный ключ (primary key, PK) – потенциальный ключ, который выбран для уникальной идентификации кортежа отношения.

Первичный ключ существует только один для данного отношения.

Внешний ключ

Внешний ключ (foreign key, FK) – атрибут или множество атрибутов отношения, которое соотвествует потенциальному ключу некоторого, может быть, того же самого отношения.

У каждого отношения мы можем выделить первичный ключ. С помощью первичного и внешних ключей можно описывать связи между различными отношениями: если мы берем значение некоторого атрибута в отношении из множества значений, являющегося первичным ключом другого отношения, мы обеспечиваем связь между этими отношениями. Иными словами, мы ставим биекцию кортежам из одного отношения в кортежи другого отношения.

Виды связей в реляционной модели

Связь 1:1

Почему мы все равно храним две таблицы, связанные первичным ключом, по отдельности? С точки зрения безопасности, проще всего ограничить доступ на уровне таблицы, значительно сложнее на уровне столбца, и намного сложнее на уровне строки.

Связь 1:n

Есть первичный ключ в первой таблице и он подставляется как значение внешнего ключа в другом отношении. EX: Студент, номер группы.

Связь n:n

Преподаватель, дисциплина. Хранение таблицы связки, куда помещаются пары первичных ключей из данных отношений.

Обеспечение целостности данных в реляционной модели

Выделяют два вида целостности данных: целостность сущностей и целостность связей (ссылочная целостность).

Целостность сущностей

Целостность сущностей обозначает, что в базовом отношении ни один атрибут первичного ключа не может содержать NULL-значений. (после внесения данных вместо NULL может произойти, что у нас будут одинаковые первичные ключи для двух сущностей).

Целостность связей

Целостность связей обозначает, что каждое значение внешнего ключа должно обязательно иметь соотвествующее значение первичного ключа в другом отношении. EX: если студент учится в группе с данным ID, то в этой таблице должна быть группа с этим ID, иначе непонятно, где он учится.

Что делать при удалении данных?

- *Вариант 1:* удалять соотвестующие объекты. EX: Есть студент, у которого расформировали группу, тогда студента мы тоже удаляем. Простой способ.
- *Вариант 2:* каскадное удаление. EX: У компании хранится адрес, у этого адреса хранится город, для городов указана страна. Удаляем страну -> так как больше такого города не существует, то удаляем и город -> удаляем адрес -> удаляем компанию.
- *Вариант 3:* замена на NULL значения. EX: Удалили страну и в таблице городов для городов, которые были в этой стране, поставили NULL. При этом сохранилась связь компания-город.

Лекция 4

Реляционная алгебра

Задать алгебру означает определить следующую тройку: объекты, операции над объектами, законы.

Основные операции

В реляционной алгебры объектами выступают отношения. Реляционная алгебра является замкнутой, то есть любые результаты операций над отношениями являются отношениями. Операции в реляционной алгебре по количеству принимаемых отношений делятся на унарные (1, 2) и бинарные. Основные операции реляционной алгебры:

1. **Проекция**. Результатом проекции является отношение, содержащее вертикальное подмножество исходного отношения, то есть формируется отношение, имеющее лишь часть столбцов.
2. **Выборка** (по некоторому предикату F). Результатом выборки является отношение, содержащее только те кортежи, которые удовлятворяют предикату F (любое суждение, которое может быть построено для кортежей.
3. **Объединение**. Результатом объединения двух отношений R и S определяет новое отношение, включающее кортежи, которые есть в R или S, исключая дубликаты (иначе противоречие по Кодду). Для того, чтобы эта операция была выполнимой, отношения R и S должны быть **совместимы** по объединению, то есть в отношениях R и S должно быть одинаковое количество атрибутов, и эти атрибуты должны быть попарно взяты из одинакового домена (одинаковость имен при этом не требуется). Домены нового отношения при этом сохраняются.
4. **Разность**. Результатом разности двух совместимых отношений R и S являются кортежи, которые есть в R, но нет в S.
5. **Пересечение**. Пересечение двух совместных отношений R и S – это отношение, состоящее из кортежей, которые есть и в R, и в S.
6. **Декартово произведение**. Декартово произведение отношений R и S – отношение, являющееся результатом конкатенации каждого кортежа из R с каждым кортежом из S.
7. **Theta-соединение**. Theta-соединение (по предикату F) определяет отношение, которые содержит кортежи из декартового произведения R и S, удовлетворяющие предикату F, имеющий вид: $F = R.a_i$ theta $S.b_j$, theta принадлежит множеству {=, <, >, <=, >=, …} – множеству операций попарного сравнения кортежей по некоторому атрибуту из R с некоторым атрибутом из S.
8. **Экви-соединение**. Экви-соединение – theta-соединение, где theta является операцией ‘=’.
9. **Натуральное (естественное) соединение**. Натуральное соединение отношений R и S – соединение по эквивалентности, выполненное по всем общим атрибутам, из результатов которого исключается по одному экземпляру каждого общего атрибута. EX: есть две таблицы с общими атрибутами, соединяя их, получаем кортежи, у которых дублируется атрибуты, по которым проводилось сравнение, далее убираем один из столбцов атрибутов.
10. **Левое внешнее соединение**. Левое внешнее соединение отношений R и S – соединение, при котором в результирующее отношение включаются также кортежи отношения R, не имеющие совпадающих значений в общих столбцах отношения S. Фактически, мы дополняем натуральные соединения теми кортежами из R, для которых не было совпадающих значений в S, и тогда для атрибутов, которые были определены на S ставим NULL-значения.
11. **Полусоединение**. Полусоединение отношений R и S – отношение, содержащее кортежи из R, которые входят в соединение отношений R и S. Фактически, берется theta-соединение R и S по

какому-то предикату theta, но выводим только кортежи из R, то есть выводим данные из одной таблицы из тех строк, которые соотвествуют значениям какой-то другой таблицы.

*Хорошая статья про основные операции: https://www.intuit.ru/studies/courses/5/5/lecture/130

SQL:

Следующей целью было создать формальный язык манипулирования данными, который реализовал бы набор операций выше. Таким языком стал простой непроцедурный SQL.

Формальный язык – это четверка (алфавит, синтаксис, семантика, прогматика)
Семантика – какой смысл имеет та или иная конструкция
Прогматика – правило применения: как строить алгоритм на основе конструкций языка

*Две теоремы Гёделя. Процедурные и непроцедурные языки. Prolog

В SQL есть 4 оператора для манипулирования данными: SELECT, INSERT, UPDATE, DELETE. Порядок следования конструкций в операторе SELECT строго определен. Общий синтаксис SELECT:

SELECT [DISTINCT | ALL]{ * | [ColumnExpression[AS Name]][, …]}
FROM TableName[AS Name][{INNER | LEFT OUTER | FULL} JOIN OtherTable [AS Name] ON condition][…]
[WHERE condition]
GROUP BY Columnlist][HAVING condition]
[ORDER BY Columnlist[ASC | DESC]

[ ] – необязательность элемента
{ } – множество элементов
| – или

Порядок работы SQL-интерпретатора:
FROM -> ON -> JOIN -> … -> ON -> JOIN -> WHERE -> GROUP BY -> HAVING -> SELECT (получаем результаты) -> DISTINCT (убираем дубликаты) -> ORDER BY (сортируем результаты)

GROUP BY сортирует значения заданного атрибута и после формирует ColumnExpression.

*Так как оператор FROM выполняется первым, то поэтому проще настраивать безопасность на уровне таблиц.

Пусть даны отношения R и S и атрибуты a_1 и a_2. Как реализовать JOIN? (R JOIN S ON R.a_1 = S.a_2)
Вариант 1: сделать декартово произведение R x S и выбрать нужные строчки. *Время*: $O(n^2)$.
Вариант 2: объединение вложенными циклами. *Время*: $O(n^2)$.

```
Для каждой строки [r] из R
    Для каждой строки [s] из S
        Если ([r], [s], condition)
            Вывести ([r], [s])
```

Вариант 3: объединение слиянием

R.сортировать по a_1
S.сортировать по a_2
Пока не конец R и не конец S

Если $R.a_1 < S.a_2$

R.следующая запись

Если $R.a_1 = S.a_2$

Вывести ([r] + [s])

S.следующая запись

Если $R.a_1 > S.a_2$

S.следующая запись

Время: O(nlogn).

Развитием модели стало внедрение индексов, то есть для атрибутов a_1 и a_2 строятся индексы, вычисляемые как хеш-функции от этих значений, причем такие, чтобы результаты хеш-функций обладали свойством сбалансированного бинарного дерева поиска -> поиск за O(logn) -> ускорение алгоритма объединения слиянием.

Создание вложенных подзапросов

Виды вложенных подзапросов:

- коррелированные
- некоррелированные

EX: получение списка всех ID поставщиков, которые поставляют товары для линейки со значением ‘M’. Такой запрос является некоррелированным, так как вложенному подзапросу ничего не нужно знать о внешнем, он выполняется первым и независимо от внешнего.

```
SELECT VendorID
FROM Product.Vendor
WHERE ProductID in
        (SELECT Product ID FROM Product WHERE ProductLine = ‘M’) // вложенный подзапрос
```

EX: поиск ID всех сотрудников, у которых время пропусков по болезни меньше среднего времени пропуска по болезни для их должности. Такой коррелированный запрос нельзя сделать снизу верх, так как для каждой должности среднее значение будет свое, поэтому взяли первую строку, посмотрели для сотрудника первую должность, выполнил вложенный подзапрос для этой должности, посчитали для них среднее, проверили, перешли к следующей строке.

```
SELECT El.EmployeeID
FROM Employee E1
WHERE SickLeaveHours <
        (SELECT AVG (E2.SickLeaveHours) FROM Employee E2 WHERE EI.Title = E2.title)
```

Лекция 5

Нормализация в реляционной модели

В реляционной модели удалось построить теоретическую модель, которая формальными методами позволяет “улучшать” модели данных, с целью избежания двух основных встречающихся проблем: избыточность и аномалии, которые могут потенциально нарушить целостность данных.

Избыточность

Избыточность не то же самое, что и дублирование

EX: Отношение студент: ФИО – Номер группы
Дублирование: номер группы будет повторяться у всех студентов, которые учатся в одной группе, однако это не избыточное дублирование, так как есть атомарный факт, что данный студент учится в данной группе, то есть повторение значений групп – лишь особенность данного атрибута

EX: Отношение студент: ФИО – Номер группы – Факультет
Избыточное дублирование: номер группы определяет факультет. Получаем, что для одной группы этот факт является уникальным, то есть, что данная группа находится на данном факультете, но для всех остальных – это избыточность, так как мы уже итак знаем, что эта группа находится на данном факультете

Недостаток избыточности

Недостатком избыточности является *провоцирование нарушения целостности данных.*

EX: Можно написать одну и ту же группу на разные факультеты.

Возможное решение проблемы:

Можно хранить факультет для первой группы в списке, а для всех остальных студентов из этой же группы, факультет уже не писать, оставляя пустое значение.

Однако память в этом случае не экономится (физически она полностью резервируется на каждый атрибут), так как поиск происходит за счет сдвига в длину. Кстати именно поэтому память хранится постоянной длины, а не переменной для одного и того же типа.

Кроме того, возможна такая ситуация: при отчислении студента, который единственный в группе хранит название факультета, мы теряем данные об этом факультете во всем отношении. Конечно, можно при удалении такого студента делать проверку на хранение факультета, и если так и есть, то переносить название факультета другому студенту из этой группы, но это будет сложный непроизводительный механизм.

Аномалии и их виды

Аномалия модификаций

Аномалия модификаций – изменение значения одной записи повлечет за собой просмотр всей таблицы и изменения некоторых других записей.

EX: Отношение студент: ФИО – Группа – Факультет. Переносим некоторую группу на другой факультет. Как итог, нам нужно посмотреть в таблице все пары (группа, факультет) с данной группой и везде произвести это изменение.

Аномалия удалений

Аномалия удалений – при удалении записи может пропасть другая информация.

EX: Отношение студент: ФИО – Группа – Факультет. Удаляем студента из таблицы, чтобы перевести его на курс ниже, при этом он должен остаться на том же факультете, но с другим номером группы. Однако при удалении студента, мы можем потерять связь между старой группой и факультетом.

Аномалия добавления

Аномалия добавления – информацию в таблицу нельзя поместить, пока она [информация] не полная или требуется дополнительный просмотр таблицы.

EX: Отношение студент: ФИО – Группа – Факультет. Добавляя запись о студенте и группе, мы должны просмотреть всю таблицу, найти студента с такой же группой, которая есть в новой записи и уже тогда добавлять и факультет для этой группы.

Нормализация

EX: Отношение студент: ФИО – Группа – Форма обучения – Кафедра – Факультет. Данное отношение обладает избыточностью и нарушает все три вида аномалий. Задача: из данного плохого отношения требуется построить хорошее. Этого можно достичь путем нормализации.

Нормализация – процесс преобразования отношения к виду, отвечающему нормальным формам. Основной метод, используемый в нормализации, – **декомпозиция отношений**, то есть разбиение отношения на множество других отношений, связанных с помощью внешних ключей.

В некотором отношении R атрибут Y **функционально зависит от атрибута** X (обозначение: X → Y) тогда и только тогда, когда каждому значению X соотвествует ровно одно значение Y. Атрибут X называется **детерминантом**, а Y – **зависимая часть**. EX: факультет зависит от номера группы (обратное неверно).

Неключевой атрибут – атрибут, не являющийся потенциальным ключом и не входящий ни в один потенциальный ключ.

*Обычно в процессе нормализации X – потенциальный ключ, а Y – неключевой атрибут.

Частично-функциональная зависимость – зависимость неключевого атрибута от части составного потенциального ключа.

Полная функциональная зависимость - зависимость неключевого атрибута от всего составного ключа.

Функциональная зависимость X → Z на некотором отношении R называется **транзитивной**, если существует такое множество атрибутов Y, что есть функциональные зависимости X → Y и одновременно Y → Z.

EX:] ФИО и группа – первичный ключ, тогда факультет находится в частично-функциональной зависимости от ключа, а форма обучения – в полной. Тройка атрибутов (группа, кафедра, факультет) образует транзитивную зависимость.

Кодд ввел первые три нормальные формы и считал, что если модель данных привести к третьей нормальной форме (3НФ), то в данной модели не будет избыточности и аномалий, но вскоре выяснилось, что это не так, поэтому после 3НФ появились форма Бойса-Кодда (БКНФ), 4НФ, 5НФ и 6НФ.

1НФ

Отношение находится в **1НФ**, если все его атрибуты являются простыми (элементарными, неделимыми).

EX: Формально ФИО – три разных атрибута, поэтому не соотвествует 1НФ, но на самом деле на практике удобнее использовать именно ФИО (или полное имя) как один простой атрибут.

2НФ

Отношение находится в **2НФ**, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от первичного ключа.

EX: Имеем три неключевых атрибута: форма обучения, кафедра и факультет. Форма обучения функционально полно зависит от ключа, а кафедра и факультет зависят только частично (только от группы), значит, отношение не находится в 2НФ.

*Если первичный ключ является простым, то если отношение находится в 1НФ, то оно автоматически находится и в 2НФ, так как по сути 2НФ касается только составных ключей.

Чтобы привести модель к 2НФ, делаем декомпозицию отношения.

EX: разобьем отношение на следующие:

- Студент: ФИО – Группа – Форма обучения, PK: (ФИО, группа)
- Группа: Группа – Кафедра – Факультет, PK: (группа)

Оба данных отношения находятся в 2НФ, так как теперь кафедра и факультет полностью зависят от группы.

3НФ

Отношение находится в **3НФ**, если оно находится в 2НФ и все неключевые атрибуты взаимно независимы и полностью зависят от первичного ключа.

Отношение находится в **3НФ**, если оно находится в 2НФ и ни один неключевой атрибут не находится в транзитивной функциональной зависимости от потенциального ключа.

*Если отношение находится в 2НФ и есть только один неключевой атрибут, то это отношение автоматически находится и в 3НФ.

EX:

- Студент: ФИО – Группа – Форма обучения, PK: (ФИО, группа) – 3НФ
- Группа: Группа – Кафедра – Факультет, PK: (группа) – не 3НФ, так как группа → кафедра и группа → факультет, а значит кафедра → факультет

Чтобы решить проблему, делаем декомпозицию второго отношения:

- Студент: ФИО – Группа – Форма обучения, PK: (ФИО, группа)
- Группа: Группа – Кафедра, PK: (группа)
- Кафедра: Кафедра – Факультет, PK: (кафедра)

В итоге мы получили базу данных, удовлетворяющую 3НФ.

БКНФ

EX: Проект: Номер студента – Полное имя – Номер проекта – Роль, PK: (номер студента, номер проекта) или (полное имя, номер проекта), при этом пусть не может быть одинаковых имен в одном проекте.

Меняя номер студента, можем получить ситуацию, в которой относительно одного проекта мы поменяем номер, а относительно другого – нет.

Кодд не рассмотрел случай, когда в одном отношении можно построить два разных потенциальных ключа и при этом части этих ключей окажутся зависимы друг от друга. Чтобы устранить этот случай, придумали БКНФ.

Отношение находится в **БКНФ**, если детерминанты всех зависимостей являются потенциальными ключами.

*Если ключ не составной, то отношение автоматически является БКНФ.

EX: полное имя полностью зависит от номера студента, но при этом номер студента не является потенциальным ключом. Чтобы привести исходное отношение к БКНФ, сделаем декомпозицию отношения:

- Номер студента – Полное имя, PK: (номер студента)
- Номер студента – Номер проекта – Роль, PK: (номер студента, номер проекта)

4НФ

EX: Дисциплина: Дисциплина – Лектор – Практик, PK: (дисциплина). Данное отношение является 3НФ и БНКФ.

*Данное отношение реализует многозначную зависимость, то есть ни лектор, ни практик друг от друга не зависят, но они оба зависят от дисциплины.

Отношение является 4НФ, если оно находится в БКНФ и не содержит нетривиальных многозначных зависимостей.

Чтобы привести отношение к 4НФ, сделаем декомпозицию (*Вариант 1*):

- Дисциплина – Лектор, PK: (Дисциплина)
- Дисциплина – Практик, PK: (Дисциплина)

Вариант 2:

- Дисциплина – Преподаватель – Роль, PK: (Дисциплина, Роль)

5НФ и 6НФ

5НФ и 6НФ сложны для понимания и оперируют связями между отношениями. На практике это формы используют не очень часто. Иногда вместо них используют триггеры.

Резюме

Часто в реальности ограничиваются БКНФ или 3НФ, так как некоторые SQL-запросы работают медленнее на нормализованных моделях данных, чем не нормализованных (особенно JOIN). Именно поэтому применяют **искусственную денормализацию**, то есть вопросы целостности данных переносятся на какие-либо валидаторы (EX: триггеры), но при этом повышаем производительность.

Есть также базы, ориентированные на быстрое чтение, но медленную запись, то есть они хранят нормализованные таблицы и закэшированные JOIN’ы.

Современные СУБД используют профилирование, то есть ставится специальный мониторинг, анализирующий статистику запросов к базе, чтобы определить, какие запросы самые частые или занимают больше всего времени с целью их кэширования.