Мадока ВТ Вики

БД:Теория:Глава 1

(Эта глава основана на материалах Лекции 1: "ИСБД. Введение" и частично Лекции 3: "Нормализация", чтобы заложить фундамент перед созданием БД)

Часть 1: Зачем нужны Базы Данных?

Представьте, что вам нужно хранить информацию о студентах вашей группы: их имена, фамилии, даты рождения, оценки. Как бы вы это сделали без баз данных?

Самый простой вариант — использовать обычные файлы, например, текстовые файлы или таблицы Excel.

```
// Пример хранения в текстовом файле student_data.txt
1;Иванов;Иван;2004-05-15
2;Петров;Петр;2004-03-10
3;Сидорова;Анна;2003-11-20
```

```
// Пример хранения в Excel (упрощенно)
| ID | Фамилия  | Имя  | Дата рождения |
|----|----------|------|---------------|
| 1  | Иванов   | Иван | 2004-05-15    |
| 2  | Петров   | Петр | 2004-03-10    |
| 3  | Сидорова | Анна | 2003-11-20    |
```

На первый взгляд, это работает. Но что если:

1. Структура данных изменится? Допустим, нужно добавить номер группы. Вам придется вручную изменить *все* файлы и *все*программы, которые эти файлы читают. Это негибко.
2. Несколько человек захотят одновременно изменить данные? Например, два куратора одновременно захотят обновить оценки. Как понять, чьи изменения правильные? Как избежать потери данных? Работать многопользовательски с простыми файлами сложно и опасно.
3. Данные повторяются? Если у вас есть отдельный файл для оценок, вам придется дублировать имена студентов. А если студент сменит фамилию? Придется менять ее во *всех* файлах, где она есть. Это избыточность и риск несогласованностиданных (в одном файле фамилия старая, в другом — новая).

Чтобы решить эти проблемы, придумали Базы Данных (БД).

Решение: Хранить *данные* (информацию о студентах, оценках) и *метаданные* (описание структуры данных: какие есть поля, какие у них типы) вместе.

Часть 2: Что такое База Данных и СУБД?

- База Данных (БД) — это, по сути, набор файлов, которые хранят данные, но также содержат описание структуры этих данных (метаданные). Эти файлы управляются специальным программным обеспечением.
- Система Управления Базами Данных (СУБД) — это то самое программное обеспечение, которое управляет базой данных. СУБД отвечает за:

- Хранение данных.
- Предоставление доступа к данным.
- Обеспечение целостности и согласованности данных.
- Поддержку языка для работы с данными (например, SQL).
- Управление одновременным доступом нескольких пользователей.
- Резервное копирование и восстановление.

Примеры СУБД: PostgreSQL (с которой мы будем работать), MySQL, Oracle Database, Microsoft SQL Server, SQLite.

Часть 3: Классификация СУБД

СУБД бывают разными. Их можно классифицировать по нескольким признакам:

1. По степени распределенности:
   - Локальные: Все данные и сама СУБД находятся на одном компьютере.
   - Распределенные: Данные и/или СУБД могут быть разнесены по нескольким компьютерам, объединенным в сеть.
2. По способу доступа к БД:
   - Файл-серверные: Данные лежат на сервере в виде файлов, а сама СУБД (логика обработки) работает на компьютере пользователя (клиенте). Клиент загружает нужные файлы для обработки. *Примеры: MS Access (в некоторых режимах), dBase, FoxPro.* Это не очень эффективно для больших объемов данных и многих пользователей.
   - Клиент-серверные: Данные и основная логика СУБД находятся на сервере. Клиент отправляет запросы на сервер, сервер их обрабатывает и возвращает только результат. Это самая распространенная архитектура для большинства современных СУБД. *Примеры: PostgreSQL, Oracle, MS SQL Server.*
   - Встраиваемые: СУБД является частью другого приложения, как библиотека. Она хранит данные только этого приложения и не требует отдельной установки. *Примеры: SQLite, BerkeleyDB.*
3. По модели данных: (Способ организации данных)
   - Иерархические: Данные организованы в виде дерева. (Сейчас почти не используются).
   - Сетевые: Более сложная структура, похожая на граф. *Частный случай - графовые СУБД (Neo4j, OrientDB)*.
   - Объектно-ориентированные: Данные хранятся как объекты (в смысле ООП). *Пример: InterSystems Caché.*
   - Реляционные (RDBMS): Данные хранятся в виде таблиц (отношений). Это самая популярная модель. *Примеры: PostgreSQL, MySQL, Oracle.*
   - Объектно-реляционные (ORDBMS): Реляционная модель с добавлением объектных возможностей. *Пример: PostgreSQL часто относят и к этому типу.*
   - NoSQL: ("Not Only SQL") - Различные модели, оптимизированные для конкретных задач (ключ-значение, документо-ориентированные, колоночные). *Примеры: Redis, MongoDB, Cassandra.* (Мы их подробно рассматривать не будем).

Мы с вами будем фокусироваться на клиент-серверных реляционных (и объектно-реляционных) СУБД, используя PostgreSQL.

Часть 4: Реляционная модель данных

Основоположником реляционной модели считается Эдгар Кодд (сотрудник IBM, 1960-е - 1970-е). Его идея: любые данные можно представить в виде совокупности отношений.

- Отношение (Relation) — это формальное название для таблицы особого вида в реляционной модели.
- У отношения есть атрибуты (Attributes) — это столбцы таблицы.
- У отношения есть кортежи (Tuples) — это строки таблицы.
- У каждого атрибута есть имя (уникальное в пределах отношения).
- Каждый атрибут определяется доменом.
  - Домен (Domain) — это множество допустимых значений для атрибута (например, домен "целые числа", домен "строки", домен "даты"). Смысл домена в том, что значения из одного домена можно сравнивать между собой.

Основные правила (свойства) отношений:

1. Отношение состоит из заголовка (набор атрибутов) и тела (набор кортежей). Заголовок фиксирован, тело меняется со временем.
2. Каждый кортеж — это набор пар "атрибут-значение", по одной паре для каждого атрибута из заголовка.
3. Значение для каждой пары "атрибут-значение" берется из домена, связанного с этим атрибутом.
4. Нет двух одинаковых кортежей (строк) в отношении. Каждая строка уникальна.
5. Порядок кортежей (строк) не имеет значения.
6. Порядок атрибутов (столбцов) не имеет значения. (Хотя на практике при отображении мы их видим в определенном порядке).
7. Все значения атрибутов атомарны. Это значит, что на пересечении строки и столбца должно быть ровно одно значение из домена (или специальное значение `NULL`). Не может быть списков или других сложных структур внутри одной ячейки (это основа Первой Нормальной Формы, о которой мы поговорим позже).

Пример таблицы (Отношения) `STUDENT`:

ID (integer)	Surname (text)	Name (text)	Birthday (date)	Location (text)
1	Иванов	Василий	1980-12-01	г. Москва
2	Георгиев	Сергей	1992-03-12	г. Санкт-Петербург
3	Васильев	Андрей	1987-10-14	г. Оренбург
7	Романов	Кирилл	1991-12-01	`NULL`

Терминология:

- Переменная отношения / Имя таблицы: `STUDENT`
- Атрибут / Столбец / Поле: `ID`, `Surname`, `Name`, `Birthday`, `Location`
- Кортеж / Строка / Запись: Строка с `ID=1`, строка с `ID=2`, и т.д.
- Заголовок: Набор имен атрибутов (`ID`, `Surname`, `Name`, `Birthday`, `Location`) и их доменов.
- Тело: Набор кортежей (все строки таблицы).
- Степень отношения (Arity/Degree): Число атрибутов (в примере = 5).
- Кардинальное число (Cardinality): Число кортежей (в примере = 4).
- Домен: Для `ID` - целые числа, для `Surname`/`Name`/`Location` - текст, для `Birthday` - даты.

- Значение `NULL`: Означает "значение неизвестно" или "неприменимо". В примере у Кирилла Романова местоположение неизвестно.

Часть 5: Ключи и Целостность

Ключи:

- Потенциальный ключ (Candidate Key): Минимальный набор атрибутов, значения которых уникально определяют кортеж в отношении. Минимальный — значит, что нельзя убрать ни один атрибут из набора без потери уникальности. В таблице `STUDENT` потенциальным ключом является `{ID}`. Возможно, `{Surname, Name, Birthday}` тоже мог бы быть потенциальным ключом, если мы уверены в его уникальности.
- Первичный ключ (Primary Key, PK): Один из потенциальных ключей, выбранный в качестве основного идентификатора кортежей в таблице. Обычно выбирают самый простой и стабильный ключ. В таблице `STUDENT` первичным ключом логично выбрать `{ID}`. Первичный ключ не может содержать `NULL`-значений.
- Внешний ключ (Foreign Key, FK): Набор атрибутов в одном отношении, значения которых должны соответствовать значениям *первичного ключа* в другом (или этом же) отношении. Используется для организации связей между таблицами.

Пример с Внешним Ключом:

Таблица `GROUP`

ID (integer, PK)	Name (text)	class_leader (integer, FK -> STUDENT.ID)
34	P3100	3
37	P3112	2
354	R4230	`NULL`

Таблица `STUDENT` (добавим `gr_id`)

ID (integer, PK)	name (text)	surname (text)	gr_id (integer, FK -> GROUP.ID)
1	Григорий	Иванов	34
2	Григорий	Иванов	34
3	Иван	Сидоров	37

Здесь `gr_id` в таблице `STUDENT` — внешний ключ, ссылающийся на `ID` в таблице `GROUP`. `class_leader` в `GROUP` — внешний ключ, ссылающийся на `ID` в `STUDENT`.

Целостность данных (Data Integrity):

Целостность — это корректность и согласованность данных в любой момент времени. Выделяют три группы правил:

1. Целостность сущностей (Entity Integrity): Ни один атрибут, входящий в состав *первичного ключа*, не может принимать значение `NULL`. Это гарантирует, что каждая строка имеет уникальный идентификатор.
2. Целостность по ссылкам (Referential Integrity): Значение *внешнего ключа* должно либо:
   - Соответствовать значению существующего *первичного ключа* в связанной таблице.
   - Быть полностью `NULL` (если это разрешено для данного внешнего ключа). Это гарантирует, что ссылки между таблицами не ведут "в никуда". Например, нельзя

студенту присвоить `gr_id = 100`, если группы с `ID = 100` не существует в таблице `GROUP`.

3. Целостность, определяемая пользователем (User-defined Integrity): Любые другие бизнес-правила, специфичные для предметной области, которые не покрываются первыми двумя типами. Примеры:
   - Уникальность каких-либо атрибутов (не являющихся первичным ключом).
   - Ограничение на диапазон значений (оценка от 2 до 5).
   - Принадлежность значения определенному набору (пол "М" или "Ж").

Эти правила целостности реализуются в СУБД с помощью ограничений (constraints), о которых мы подробно поговорим при создании таблиц.

Пример нарушения целостности (из слайда с ошибками):

Таблица `STUDENT`

id	name	surname	gr_id	Проблема
1	Григорий	Иванов	34	ОК
2	Иван	Сидоров	34	ОК
2	Петр	Петров	37	Нарушение целостности сущностей (PK): `id=2` повторяется
2	Иван	Сидоров	34	Нарушение целостности сущностей (PK): `id=2` повторяется; Дубликат строки:Полностью совпадает со второй строкой (тоже нарушение)

Таблица `GROUP`

id	name	class_leader	Проблема
'Vasya'	P3100	3	Нарушение домена/типа: `id` должен быть числом (PK), а не строкой
37	P3112	2	ОК
34	R4230	4	Нарушение ссылочной целостности (FK): Студента с `ID=4` нет в `STUDENT`

Часть 6: Язык SQL (Введение)

SQL (Structured Query Language) — структурированный язык запросов. Это *декларативный* язык, то есть мы описываем, *что* хотим получить или сделать, а не *как* это сделать. СУБД сама определяет оптимальный способ выполнения нашего запроса.

SQL используется для:

- Определения данных (Data Definition Language, DDL): Создание, изменение, удаление структуры БД (таблиц, индексов, представлений и т.д.).
  - `CREATE`: Создать объект.
  - `ALTER`: Изменить объект.
  - `DROP`: Удалить объект.
- Манипулирования данными (Data Manipulation Language, DML): Работа с данными внутри таблиц.
  - `SELECT`: Выборка (чтение) данных.
  - `INSERT`: Вставка (добавление) новых данных.
  - `UPDATE`: Обновление (изменение) существующих данных.

    - `DELETE`: Удаление данных.
- Управления доступом к данным (Data Control Language, DCL): Предоставление и отзыв прав доступа пользователям.
    - `GRANT`: Предоставить права.
    - `REVOKE`: Отозвать права.

Мы начнем с DDL для создания таблиц, а затем перейдем к DML для работы с данными.
