

DB LEC 03

isagila

Собрано 14.10.2023 в 19:20



Содержание

1. Лекции	3
1.1. Лекция 23.09.01.	3
1.2. Лекция 23.09.08.	3
1.3. Лекция 23.09.15.	4
1.4. Лекция 23.09.22.	5
1.5. Лекция 23.09.29.	6
1.6. Лекция 23.10.06.	7
1.7. Лекция 23.10.13.	8

1. Лекции

1.1. Лекция 23.09.01.

Информация бывает трех видов

1. Сигнал.

Например, для человека, который не знает какой-либо язык, текст на этом языке будет сигналом: т.е. можно понять, что какая-то информация была передана, но принять (понять) эту информацию нельзя.

2. Знание.

Студент, слушающий лектора и пишущий конспект, получает знания.

3. Данные.

Данные это формализованные знания. В отличие от знаний данные не искажаются в процессе передачи. Если лектор передаст студентам конспект лекции, который он написал сам, то он передаст именно данные.

Def 1.1.1. Данные это поддающиеся многократной интерпретации представления информации в формализованном виде пригодном для передачи, интерпретации и обработки.

Все данные проходят путь вида

Сбор → Обработка → Передача → Хранение → Представление

1.2. Лекция 23.09.08.

Уровни архитектуры данных

1. Внешний.

Решаем проблему представления данных пользователю (как агрегировать и детализировать данные?).

2. Концептуальный.

Занимаемся выделением сущностей, решаем проблемы безопасности (разрешения + ограничения). Определяем семантику (например договариваемся, что кредитный рейтинг будет храниться в виде числа от 1 до 5 и сопоставляем каждому числу некоторое словесное описание, которое записано в документации, но не в самой базе данных).

3. Внутренний.

Решаем проблемы хранения данных на физическом уровне.

Для простоты выделяют три модели данных, которые связаны с некоторыми описанными выше уровнями архитектуры данных.

1. Модель Сущность–Связь (внешний + концептуальный уровни).

2. Логическая (даталогическая модель) модель (все уровни).

3. Физическая модель (концептуальный + внутренний уровни).

Модель Сущность–Связь

Def 1.2.1. Сущность это множество экземпляров (реальных или абстрактных) однотипных объектов предметной области.

Def 1.2.2. Сущность называется сильной, если ее экземпляры могут существовать независимо. Слабые сущности могут существовать только при наличии одного или нескольких экземпляров сильной сущности.

Пример 1.2.3. Пусть у нас есть две сущности: Студент и Группа. Студент будет сильной сущностью, т.к. он может существовать и без группы, а Группа будет слабой сущностью, т.к. она не может существовать без студентов.

Этот пример очень условный: деление сущностей на слабые и сильные зависит от обстоятельств. В некоторых случаях Группа вполне может быть сильной сущностью.

Каждая сущность обладает одним или несколькими атрибутами. Атрибуты делятся на

1. Простые (например дата рождения)

2. Составные (например адрес, если хранить город, улицу, дом и т.п. отдельно)

Замечание 1.2.4. Один и тот же атрибут (например, ФИО) может в разных случаях быть как простым (если его хранить как одну строчку), так и составным (если отдельно хранить фамилию, отдельно имя и отдельно отчество).

Также атрибуты можно делить по другому принципу.

1. Обязательные (например email в некоторых случаях может быть обязательным атрибутом).

2. Необязательные (например отчество может быть необязательным атрибутом).

Помимо этого, атрибуты делятся на

1. Однозначные (например дата рождения, она у каждого ровно одна).

2. Многозначные (например номер телефона, у кого-то может быть несколько номеров телефона).

Для каждого атрибута мы определяем домен, т.е. множество допустимых значений. Доменом может быть перечисление, множество всех натуральных чисел, регулярное выражение и т.д.

Чтобы показать отношения между сущностями, используются связи. На уровне Сущность–Связь они обычно несут семантическую нагрузку, например Студент **принадлежит** Группе. Существует три вида связи

1. Один-к-одному (1:1).

Обычно все связи этого вида это искусственно выделенные атрибуты. Зачем же тогда нужна эта связь? Рассмотрим на примере. Допустим, у нас есть две сущности: Студент и Паспорт. Логично, что у каждого студента один паспорт и у каждого паспорта один студент, т.е. связь вида 1:1. Почему же нельзя добавить паспорт в атрибуты сущности Студент? Так можно сделать, но связь вида 1:1 может быть удобна в целях безопасности: в таком случае все паспорта можно будет хранить отдельно от студентов и (например) дополнительно шифровать (шифровка же всей сущности Студент будет мало того, что излишне затратной, так еще и не очень нужной).

2. Один-ко-многим (1:N).

3. Многие-ко-многим (N:N).

1.3. Лекция 23.09.15.

Def 1.3.1 (по Коннолли и Бегг). База данных это совместно используемый набор логически связанных данных и описание этих данных, предназначенный для удовлетворения информационных потребностей предприятия.

Def 1.3.2 (по Дейту). База данных это набор постоянно хранимых данных, используемых прикладными системами предприятия.

Def 1.3.3 (по Хомоненко). База данных это совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Рассмотрим некоторые модели данных.

Иерархическая модель данных

Эта модель удобная для восприятия человек, т.к. он часто сталкивается с разного рода иерархией в реальном мире. У нас есть некоторые поля данных, которые являются неделимыми атомами, а некоторые объединение этих полей называется сегментом данных. Каждая запись в БД это экземпляр сегмента данных.

К минусам этой модели можно отнести дублирование данных, вследствие чего возникает некоторая сложность поддержки их целостности.

Сетевая модель данных

Если иерархическая модель данных по сути являлась деревом, то сетевая модель представляет собой граф, в котором вершины представляют собой некоторые экземпляры объектов.

Реляционная модель данных

В отличие от иерархической и сетевой модели элементы модели сразу двумерные (т.е. таблицы) и при этом мы не храним связь явно — мы выделяем некоторый атрибут, который наделяем семантикой связи. Это может привести к некоторым проблемам — семантика может быть неправильна трактована, в результате чего запрос к БД либо не выполнится, либо выполнится некорректно. Проблему поддержки целостности данных будем решать нормализацией (об этом будет рассказано в последующих лекциях). Одним из плюсов этой модели является то, что мы можем оценить время работы запроса к БД.

Постреляционная модель данных

Берем реляционную модель и снимаем запрет на неделимость поля, т.е. в качестве атрибута теперь можно хранить массив строк через запятую или сразу JSON с какими-либо данными.

Эта модель была сделана для того, чтобы решить проблему очень большой персонификации данных. Т.е. допустим есть таблица с деталями и у некоторых деталей есть какие-то атрибуты, которых в принципе не может быть у других. Если добавить все возможные атрибуты каждой детали, то получим очень разряженную таблицу почти полностью состоящую из null-значений \Rightarrow проблемы с памятью.

С другой стороны мы усложняем себе поиск по такой базе (сложно проверять детали, которые имеют разный набор атрибутов) и поддержку целостности — поле JSON-данными при желании можно записать некорректное значение (либо нужно жестко валидировать все записываемое в каждую такую ячейку \Rightarrow проблемы со скоростью работы).

Многомерная модель данных

Пусть есть таблица со столбцами «Продавец», «Товар», «Регион», «Квартал» и «Количество». Мы хотим уметь быстро строить сводки по продавцу, товару, региону, кварталу (или по комбинации параметром), находить наибольшее и наименьшее значения и т.п.

Можно представить наши данные в 4ех мерном пространстве, где первые четыре колонки будут задавать координаты, а пятая (количество товара) — значение. Теперь, чтобы получать различные метрики, мы будем рассекать полученный гиперкуб гиперплоскостями, искать в них минимум/максимум и т.п.

Данная модель имеет большие затраты по памяти, но зато с ее помощью можно быстро выполнять запросы. Ее можно использовать в следующем ключе: храним данные в реляционной модели. Ночью строим по ней многомерную модель, а днем пользуемся ей не внося никаких изменений (будем считать, что продажи товаров за день не сильно влияют на глобальную ситуацию \Rightarrow ими можно пренебречь). Следующей ночью заново строим многомерную модель по уже новым данным и так далее. Таким образом в течение дня мы получаем возможность быстро делать разные запросы и узнавать разные метрики.

Объектно-ориентированная модель данных

Похожа на постреляционную модель: мы сериализуем все объекты в системе и храним пары вида `id: [serialized]`. Это усложняет поиск, т.к. нужно десериализовывать данные.

1.4. Лекция 23.09.22.

Def 1.4.1. Схема отношения это строка заголовков.

Def 1.4.2. Одна строка называется кортежем.

Def 1.4.3. Один столбец называется атрибутом. Заголовок столбца определяет его имя.

Def 1.4.4. Домен это множество допустимых значений атрибута.

Def 1.4.5. Степень отношения это количество его атрибутов.

Def 1.4.6. Кардинальность отношения это количество его кортежей.

Def 1.4.7. Отношение это множество упорядоченных кортежей, где каждое значение берется из соответствующего домена.

$$R = \{d_1, \dots, d_n\} \quad d_i \in D_i$$

Свойства отношений (по Кодду)

1. Каждая ячейка содержит одно неделимое значение.
2. Каждый кортеж уникален.
3. Уникальность имени отношения в реляционной схеме.
4. Уникальность имени атрибута в пределах отношения.
5. Значения атрибута берутся из одного и того же домена.
6. Порядок следования атрибутов и кортежей не имеет значения.

Def 1.4.8. Суперключ это атрибут (множество атрибутов), который единственным образом идентифицирует кортеж.

Замечание 1.4.9. Схема отношения всегда является суперключом (см. второе свойство отношений).

Def 1.4.10. Потенциальный ключ это суперключ, который не содержит подмножества, также являющегося суперключом этого отношения.

Def 1.4.11. Первичный ключ это потенциальный ключ, который выбран для идентификации кортежей внутри отношения.

Замечание 1.4.12. Иногда используют технический первичный ключ (например поле `id`), который не несет никакой семантики, а служит лишь для определения уникальности кортежа.

Def 1.4.13. Внешний ключ это атрибут (множество атрибутов) внутри отношения, который соответствует потенциальному ключу некоторого (возможно того же самого) отношения.

Def 1.4.14. Целостность сущностей: в отношении первичный ключ не может содержать `NULL` значения.

Def 1.4.15. Ссылочная целостность: если в отношении существует внешний ключ, то его значение либо соответствует значениям потенциального ключа, либо полностью состоит из `NULL` значений.

1.5. Лекция 23.09.29.

Def 1.5.1. Реляционная алгебра это теоретический язык операций, позволяющий на основе одного или нескольких отношений создавать другие отношения без изменения самих исходных отношений.

Реляционная алгебра замкнута. В ней существуют следующие операции

1. Проекция $\Pi_{a_1 \dots a_n}(R)$

Результатом проекции является новое отношение, содержащее вертикальное подмножество исходного отношения, создаваемое посредством извлечения указанных атрибутов и исключения из результата атрибутов-дубликатов.

2. Выборка $\sigma_{\text{предикат}}(R)$

Результатом выборки является отношение, которое содержит только те кортежи из исходного отношения, которые удовлетворяют заданному условию (предикату).

3. Объединение $R \cup S$

Объединение двух отношений R и S определяет новое отношение, которое включает все кортежи, содержащиеся только в R , все кортежи, содержащиеся только в S и кортежи, содержащиеся и в R , и в S с исключением дубликатов.

Объединять можно не все отношения, а только совместные по объединению отношения. Отношения совместны по объединению, когда они состоят из одинакового количества атрибутов и каждая соответствующая пара атрибутов имеет одинаковый домен.

4. Разность $R - S$

Разность состоит из кортежей, которые есть в R , но отсутствуют в S . Разность двух отношений определена только если они совместны по объединению.

5. Пересечение $R \cap S$

Операция пересечения определяет отношение, содержащее кортежи, находящиеся как в R , так и в S . Пересечение двух отношений определено только если они совместны по объединению.

6. Декартово произведение $R \times S$

Декартово произведение определяет новое отношение, которое является результатом конкатенации каждого кортежа из отношения R с каждым кортежем из отношения S .

7. Тета-соединение $R \bowtie_F S$

Определяет отношение содержащее кортежи из декартового произведения $R \times S$, удовлетворяющие предикату $F = R_{a_i} \Theta S_{b_i}$, где Θ это одна из операций сравнения $\{>, <, =, \dots\}$

8. Экви-соединение

Это тета-соединение, где Θ это «=».

9. Естественное соединение $R \bowtie S$

Соединение по эквивалентности двух отношений, выполненное по всем общим атрибутам, из результатов которого исключили по одному экземпляру каждого атрибута.

10. Левое внешнее соединение $R \Join L S$

Это естественное соединение, при котором в результирующее отношение включаются также кортежи отношения R , не имеющие совпадающих значений в общих атрибутах отношения S .

11. Полусоединение $R \triangleright_F S$

Это отношение, содержащее кортежи R , которые входят в тета-соединение R и S .

12. Деление.

Эту операцию мы рассматривать не будем.

Таким образом общий вид `SELECT` запроса будет следующим

```
SELECT [DISTINCT | ALL] { * | [ColumnExpr [AS NewName]] [, ...]}  
FROM Table [AS NewName]  
[ { INNER | LEFT OUTER | FULL } JOIN Table [AS NewName] [, ...]]  
[WHERE condition]  
[GROUP BY ColumnList [HAVING condition]]  
[ORDER BY ColumnName [ASC | DESC]]
```

Порядок выполнения SQL-запроса будет таким

1. FROM ... ON ... JOIN

2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. DISTINCT
7. ORDER BY

1.6. Лекция 23.10.06.

Соединение двух таблиц можно сделать через два вложенных цикла (обычно медленно), либо через сортировку двух таблиц и проход по ним с помощью двух указателей (как в сортировке слиянием). Второй способ обычно быстрее, кроме случаев, когда в одной из таблиц очень мало записей, а в другой — очень много.

Def 1.6.1. Нормализация это приведение отношения к нормальной форме. Она нужна, чтобы убрать избыточность и аномалии.

Пусть есть «плохое» отношение, в котором есть три атрибута: ФИО, Группа, Образовательная программа (ОП). Сразу видно дублирование: связь группы и образовательной программы хранится несколько раз, также есть избыточность: нам достаточно знать группу, чтобы определить ОП.

Def 1.6.2. Аномалия модификации: изменение значения одной записи повлечет за собой просмотр всей таблицы и изменение некоторых других записей

Пример 1.6.3. Нужно сменить ОП для группы? Придется пройти по всем записям. Это долго, а также можно что-то пропустить.

Def 1.6.4. Аномалия удаления: при удалении записи может пропасть и другая информация.

Пример 1.6.5. Удаляем всех студентов из группы и теряем связь группы с ОП.

Def 1.6.6. Аномалия добавления: информацию в таблицу нельзя поместить пока она не полная или требуется дополнительный просмотр таблицу.

Пример 1.6.7. Добавляем студента, который не знает ОП, а знает только группу.

Def 1.6.8. В отношении R атрибут y функционально зависит от атрибута $x \iff$ каждому x соответствует в точности один y . В этом случае x называется детерминантой, а y — зависимой частью.

Def 1.6.9. Частичная функциональная зависимость это зависимость неключевого атрибута от части составного потенциального ключа.

Пример 1.6.10. ОП частично функционально зависит от составного ключа ФИО-Группа.

Def 1.6.11. Полная функциональная зависимость это зависимость неключевого атрибута от составного потенциального ключа.

Пример 1.6.12. Если добавить в рассматриваем пример атрибут Форма обучения, то он будет полно функционально зависеть от составного ключа ФИО-Группа

Def 1.6.13. Транзитивная функциональная зависимость: два атрибута находятся в транзитивной функциональной зависимости, если существует атрибут (множество атрибутов) такой, что второй атрибут находится в функциональной зависимости от этого атрибута, а сам этот атрибут функционально зависит от первого атрибута.

TODO: в тетради нет четкого определения

Нормальные формы

Нормальные формы «вложены» друг в друга: для того, чтобы отношение находилось в определенной нормальной форме, оно должно находиться в предыдущей нормальной форме и должно выполняться некоторое дополнительное условие. Названия нормальных форм будем сокращать как (например) 1НФ — первая нормальная форма.

Def 1.6.14. Первая нормальная форма: все атрибуты отношения являются простыми (не пытаемся использовать атрибуты как составные).

Def 1.6.15. Вторая нормальная форма: 1НФ и каждый неключевой атрибут функционально полно зависит от первичного ключа.

Def 1.6.16. Третья нормальная форма: 2НФ и все неключевые атрибуты взаимонезависимы и полностью зависят от первичного ключа.

3НФ можно определить иначе:

Def 1.6.17. 2НФ и но один неключевой атрибут не находится в транзитивной функциональной зависимости от потенциального ключа.

Def 1.6.18. Нормальная форма Бойса-Кодда (БКНФ): 3 НФ и детерминанты всех зависимостей являются потенциальными ключами.

Пример 1.6.19. Пусть есть отношение с атрибутами ИСУ, Паспорт, ID Проекта и Роль. ИСУ + ID Проекта и Паспорт + ID Проекта это первичные ключи. Если человек решит сменить паспорт, то он может сменить его относительно одного проекта, а относительно другого — забыть. Для решения этой проблемы следует разбить это отношение на два: ИСУ, ID Проекта, Роль и ИСУ, Паспорт.

Def 1.6.20. Четвертая нормальная форма: БКНФ и отношение не содержит нетривиальных многозначных зависимостей.

Пример 1.6.21. Пусть есть отношение ID Дисциплины, ID Лектора, ID Практика. Если лектор решил уволится, то нужно везде заменить его, но можно ошибиться и поменять не везде. Можно попытаться разбить это отношение на два: ID Дисциплины, ID Лектора и ID Дисциплины, ID Практика, но тогда потеряется связь лектора и практика (а что если некоторые лекторы могут работать только с определенными практиками и наоборот?).

Замечание 1.6.22. Если в отношении два столбца, то оно находится во всех нормальных формах, т.к. его более нельзя разбить

Замечание 1.6.23. Существуют также 5ая и бая нормальные формы, но они редко используются на практике. На практике же, наоборот, иногда прибегают к денормализации для повышения скорости работы.

1.7. Лекция 23.10.13.

Индексы

Можно попробовать отсортировать записи по какому-нибудь атрибуту и использовать бинпоиск. Однако запись в БД нельзя прочитать точечно, поэтому приходится читать блоками, из-за этого бинпоиск не очень эффективен. Также возникают проблемы для строковых атрибутов (сравнение строк не очень быстрое). Помимо этого не очень ясно, как поддерживать отсортированность при добавлении/удалении элементов.

Def 1.7.1. Индекс это структура данных типа ключ-значение, где ключ это результат некой функции от совокупности атрибутов, а значение — адрес хранимой записи.

Пусть индексом будет первичный ключ, и хранить мы его будем в виде сбалансированного дерева. Тогда получаем

Def 1.7.2. Первичный индекс это индекс, построенный по первичному ключу при условии, что исходный файл отсортирован по нему же.

Первичный индекс является плотным, т.е. охватывает все уникальные записи (количество кортежей в отношении равно количеству вершин в дереве индекса).

Def 1.7.3. Индекс кластеризации это индекс, построенный по неключевому полю при условии, что исходный файл отсортирован по нему же.

Пример 1.7.4. В качестве индекса кластеризации будем брать первые 5 символов ФИО и для каждого индекса будем хранить ссылку на первый кортеж в кластере. Теперь по ФИО можно быстро найти кластер, где лежат ФИО с одинаковыми первыми 5-ю символами, а дальше уже найти требуемый кортеж.

Замечание 1.7.5. При использовании индекса кластеризации мы ищем баланс между размером и плотностью индекса: либо индекс большой и плотный (нужно больше памяти), либо индекс маленький и разреженный (нужно меньше памяти).

Def 1.7.6. Вторичный индекс это индекс, построенный по неключевому полю при условии, что исходный файл не отсортирован.

Пример 1.7.7. Почту можно развернуть и кластеризовать.

Замечание 1.7.8. Дерево индекса не обязательно всегда держать сбалансированным, можно считать метрику «разбалансированности» и иногда делать перебалансировку.

Представления

Def 1.7.9. Представление это динамически сформированный результат одной или нескольких реляционных операций, сохраненный в виде нового отношения.

Представления делятся на

1. Материализованные.

На самом деле хранятся в памяти и обновляются раз в некоторое время.

2. Представления замены.

В памяти хранится только запрос для получения этого представления. При получении запроса извне этот сохраненный запрос подставляется во внешний запрос и полученный таким образом запрос выполняется. Такой способ обеспечивает безопасность, но не дает выигрыша в скорости (скорее даже снижает скорость).

Представления бывают обновляемыми (через них можно что-то поменять в исходных отношениях) и нет.

Преимущества представлений

1. Независимость от данных.
2. Повышение защищенности данных.
3. Снижение сложности работы с данными.

Недостатки представлений

1. Ограниченностъ в возможностях обновлений.
2. Структурные ограничения (вынужденность подстраиваться под заданный интерфейс представления).
3. Снижение производительности.