Базы данных

Лекция 6.

Нормализация: 3NF, NFBC.

Версионирование. TCL, ACID и изолированность

МФТИ, 2024

Игорь Шевченко

@igorshvch

I. Нормализация. 3NF, NFBC

Повторение: функциональная зависимость

- Определение функциональной зависимости (по Дж. Дейту):

  Пусть r является отношением, а X и Y — произвольными подмножествами множества атрибутов отношения r. **Тогда Y функционально зависимо от X**, что в символическом виде записывается как

  X → Y

  (читается либо как "X функционально определяет Y", либо как "X стрелка Y") тогда и только тогда, когда каждое значение множества X отношения r связано точно с одним значением множества Y отношения r. Иначе говоря, если два кортежа отношения r совпадают по значению X, они совпадают и по значению Y (⇳ *не существует двух различных кортежей, которые в атрибуте X имеют совпадающие значения, а в атрибуте Y имеют разные значения*).
- Левая часть записи функциональной зависимости называется **детерминантом**, правая – **зависимой частью**.
- **Функциональная зависимость— это связь типа "многие к одному" между двумя множествами атрибутов заданной переменной отношения**

Функциональная зависимость: теорема Хита

- **Теорема Хита**. Пусть R{A, B, C} является отношением, где A, B и C — множества атрибутов. Если R удовлетворяет функциональной зависимости A → B, то R равна соединению ее проекций по атрибутам {A,B} И {A,C} (R = {A,B} ⋈ {A,C}).
- По отношению к таблице выше примем, что A = {Поставщик}, B = {Город}, C = {Продукт}.
- Тогда по теореме Хита один из вариантов декомпозиции исходного отношения следующий: {Поставщик, Город} и {Поставщик, Продукт}.

Функциональная зависимость: теорема Хита

Поставщик	Город	Продукт
ООО "Ромашка"	Москва	Конфеты
ООО "Василек"	Санкт-Петербург	Шоколад
ООО "Ландыш	Казань	Мороженое
ООО "Ромашка"	Москва	Конфеты
ООО "Подорожни	Краснодар	Шоколад
ООО "Подорожни	Краснодар	Шоколад

Функциональная зависимость: теорема Хита

Поставщик	Город	Продукт
ООО "Ромашка"	Москва	Конфеты
ООО "Василек"	Санкт-Петербург	Шоколад
ООО "Ландыш	Казань	Мороженое
ООО "Ромашка"	Москва	Конфеты
ООО "Подорожни	Краснодар	Шоколад
ООО "Подорожни	Краснодар	Шоколад

Очевидно, что данные в приведенной таблице избыточны – информация о поставщиках, городах и продуктах дублируется в различных строках

Функциональная зависимость: теорема Хита

Поставщик	Город	Продукт
ООО "Ромашка"	Москва	Конфеты
ООО "Василек"	Санкт-Петербург	Шоколад
ООО "Ландыш	Казань	Мороженое
ООО "Ромашка"	Москва	Конфеты
ООО "Подорожни	Краснодар	Шоколад
ООО "Подорожни	Краснодар	Шоколад

Вопрос: каким образом мы можем избавиться от дублирования?
Ответ: **теорема Хита (Heath)**

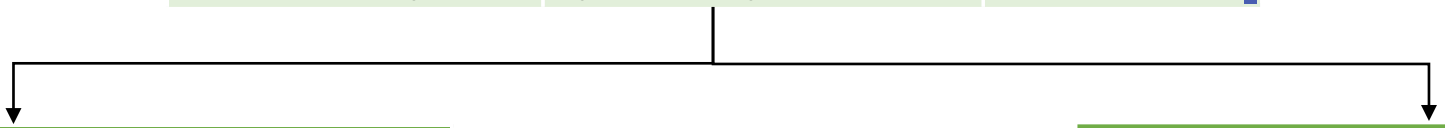
Функциональная зависимость: теорема Хита

- Нам нужны три группы атрибутов и ФЗ между двумя из них:
- Если A → B, то R = {AB} ⋈ {AC}, следовательно если
  - ФЗ: {Поставщик} → {Город}
- То:
  - A – {Поставщик}
  - B – {Город}
  - C – {Продукт}
- Тогда по теореме Хита:
  - Если A → B, то R = {AB} ⋈ {AC} или

  R = {Поставщик, Город} ⋈
  ⋈ {Поставщик, Продукт}

Функциональная зависимость: теорема Хита

Поставщик	Город	Продукт
ООО "Ромашка"	Москва	Конфеты
ООО "Василек"	Санкт-Петербург	Шоколад
ООО "Ландыш	Казань	Мороженое
ООО "Ромашка"	Москва	Конфеты
ООО "Подорожни	Краснодар	Шоколад
ООО "Подорожни	Краснодар	Шоколад



Поставщик	Город
ООО "Ромашка"	Москва
ООО "Василек"	Санкт-Петербург
ООО "Ландыш	Казань
ООО "Подорожни	Краснодар

Поставщик	Продукт
ООО "Ромашка"	Конфеты
ООО "Василек"	Шоколад
ООО "Ландыш	Мороженое
ООО "Подорожни	Шоколад

3NF

- Отношение находится **в третьей нормальной форме** тогда и только тогда, когда оно находится во второй нормальной форме и ни один неключевой атрибут ***не является транзитивно зависимым*** от его первичного ключа (в определении предполагается наличие только одного потенциального ключа, который к тому же является первичным ключом отношения)
- **Неформальное определение**: Отношение находится в 3NF тогда и только тогда, когда каждый кортеж состоит из значения первичного ключа, и множества взаимно независимых атрибутов

3NF: транзитивная зависимость

- Определение: Если для атрибутов X, Y и Z отношения R существуют функциональные зависимости $X \rightarrow Y$, $Y \rightarrow Z$, говорят, что атрибут Z связан транзитивной зависимостью с атрибутом X через атрибут Y (при этом атрибут X не должен функционально зависеть ни от атрибута Y, ни от атрибута Z).

3NF: пример

Поставщик_#	Город	Деталь
ООО "Ромашка	Москва	Саморезы
ООО "Ландыш"	Краснодар	Саморезы
ООО "Василек"	Екатеринбург	Болты
ООО "Роза"	Казань	Гвозди
ООО "Орхидея"	Москва	Саморезы

X – {Поставщик_#} – первичный ключ
Y – {Город}
Z – {Деталь}

{Поставщик_#} → {Город}, {Город} → {Деталь}

{Поставщик_#} ↛ {Город}, {Город} ↛ {Деталь}

3NF: пример

Поставщик_#	Город	Деталь
ООО "Ромашка	Москва	Саморезы
ООО "Ландыш"	Краснодар	Саморезы
ООО "Василек"	Екатеринбург	Болты
ООО "Роза"	Казань	Гвозди
ООО "Орхидея"	Москва	Саморезы

X – {Поставщик_#} – первичный ключ
Y – {Город}
Z – {Деталь}

{Поставщик_#} → {Город}, {Город} → {Деталь}
{Поставщик_#} → {Деталь}

Попробуем избавиться от ФЗ {Поставщик_#} → {Деталь} как и для 2NF, с помощью теоремы Хита:
Если X → Z, то R = {XZ} ⋈ {XY}
R = {Поставщик_#, Деталь} ⋈ {Поставщик_#, Город}

3NF: пример

Поставщик_#	Город	Деталь
ООО "Ромашка	Москва	Саморезы
ООО "Ландыш"	Краснодар	Саморезы
ООО "Василек"	Екатеринбург	Болты
ООО "Роза"	Казань	Гвозди
ООО "Орхидея"	Москва	Саморезы

X – {Поставщик_#} – первичный ключ
Y – {Город}
Z – {Деталь}

{Поставщик_#} → {Город}, {Город} → {Деталь}
{Поставщик_#} → {Деталь}

Попробуем избавиться от ФЗ {Поставщик_#} → {Деталь} как и для 2NF, с помощью теоремы Хита:
Если X → Z, то R = {XZ} ⋈ {XY}
R = {Поставщик_#, Деталь} ⋈ {Поставщик_#, Город}

ТАК ДЕЛАТЬ НЕЛЬЗЯ! (По крайней мере, в общем случае)

3NF: пример

Поставщик_#	Город	Деталь
ООО "Ромашка	Москва	Саморезы
ООО "Ландыш"	Краснодар	Саморезы
ООО "Василек"	Екатеринбург	Болты
ООО "Роза"	Казань	Гвозди
ООО "Орхидея"	Москва	Саморезы

Поставщик_#	Деталь
ООО "Ромашка	Саморезы
ООО "Ландыш"	Саморезы
ООО "Василек"	Болты
ООО "Роза"	Гвозди
ООО "Орхидея"	Саморезы

Поставщик_#	Город
ООО "Ромашка	Москва
ООО "Ландыш"	Краснодар
ООО "Василек"	Екатеринбург
ООО "Роза"	Казань
ООО "Орхидея"	Москва

Мы потеряли функциональную зависимость {Город} → {Деталь}
В реальных БД это может стать причиной проблем при добавлении данных

3NF: пример

Поставщик_#	Город	Деталь
ООО "Ромашка	Москва	Саморезы
ООО "Ландыш"	Краснодар	Саморезы
ООО "Василек"	Екатеринбург	Болты
ООО "Роза"	Казань	Гвозди
ООО "Орхидея"	Москва	Саморезы

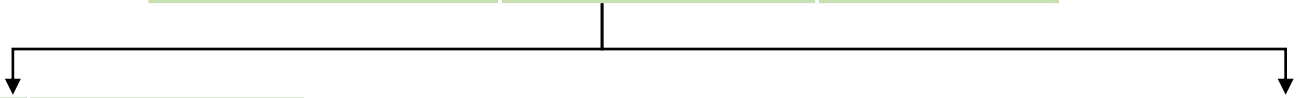
X – {Поставщик_#} – первичный ключ
Y – {Город}
Z – {Деталь}

{Поставщик_#} → {Город}, {Город} → {Деталь}

В данном случае нам желательно провести декомпозицию так, чтобы сохранить «промежуточные» ФЗ в разных отношениях:
R = {Поставщик_#, Город} ⋈ {Город, Деталь}

3NF: пример

Поставщик_#	Город	Деталь
ООО "Ромашка	Москва	Саморезы
ООО "Ландыш"	Краснодар	Саморезы
ООО "Василек"	Екатеринбург	Болты
ООО "Роза"	Казань	Гвозди
ООО "Орхидея"	Москва	Саморезы



Поставщик_#	Город
ООО "Ромашка	Москва
ООО "Ландыш"	Краснодар
ООО "Василек"	Екатеринбург
ООО "Роза"	Казань
ООО "Орхидея"	Москва

Город_#	Деталь
Москва	Саморезы
Краснодар	Саморезы
Екатеринбург	Болты
Казань	Гвозди

BCNF

- BCNF используется при следующих условиях (по К. Дейту):
  - отношение имеет два (или больше) потенциальных ключа
  - эти потенциальные ключи являются составными
  - два или больше потенциальных ключей перекрываются (т.е. имеют по крайней мере один общий атрибут)
- На практике совокупность таких условий встречается не часто, поэтому частно ограничиваются 2NF

BCNF

- Определение:
  - Отношение находится в BCNF тогда и только тогда, когда каждая его *нетривиальная* и *неприводимая слева* функциональная зависимость имеет в качестве своего детерминанта некоторый потенциальный ключ.
    - *Тривиальная ФЗ* – ФЗ между составным потенциальным ключом и его атрибутами
    - *Неприводимость слева* – то же, что минимальная функциональная зависимость
- Неформальное определение:
  - Переменная отношения находится в BCNF тогда и только тогда, когда детерминанты всех ее ФЗ являются потенциальными ключами

BCNF: пример

- Возьмем отношение {Студент, Предмет, Преподаватель}
- Пусть оно удовлетворяет следующим ограничениям:
  - Каждый студент изучает определенный предмет только у одного преподавателя.
  - Каждый преподаватель ведет только один предмет, но каждый предмет ведут не сколько преподавателей
- Из указанного условия следуют ФЗ:
  - {Студент, Предмет} → {Преподаватель}
  - {Преподаватель} → {Предмет}

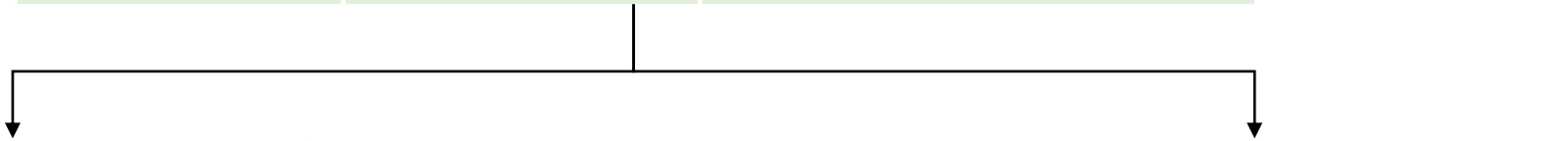
BCNF: пример

Студент	Предмет	Преподаватель
Иванов	Математик	Доц. Смирнов
Иванов	Физика	Проф. Кузнецов
Петров	Математик	Доц. Смирнов
Петров	Физика	Проф. Кузнецов

- Имеющиеся потенциальные ключи:
  - {Студент, Предмет}
  - {Студент, Преподаватель}

BCNF: пример

Студент	Предмет	Преподаватель
Иванов	Математик	Доц. Смирнов
Иванов	Физика	Проф. Кузнецов
Петров	Математик	Доц. Смирнов
Петров	Физика	Проф. Кузнецов



Преподаватель	Студент
Доц. Смирнов	Иванов
Проф. Кузнецов	Иванов
Доц. Смирнов	Петров
Проф. Кузнецов	Петров

Преподаватель	Предмет
Доц. Смирнов	Математика
Проф. Кузнецов	Физика

BCNF: пример

- Однако следует отметить, что все еще существует иная проблема. Суть ее в том, что декомпозиция исходного отношения на проекции ST и TJ позволяет исключить одни аномалии, но приводит к появлению других:
  - Исходная, функциональная зависимость:

    {Студент, Предмет} → {Преподаватель}

    не может быть выведена из той единственной функциональной зависимости, которая присутствует в двух данных проекциях:

    {Преподаватель} → {Предмет}

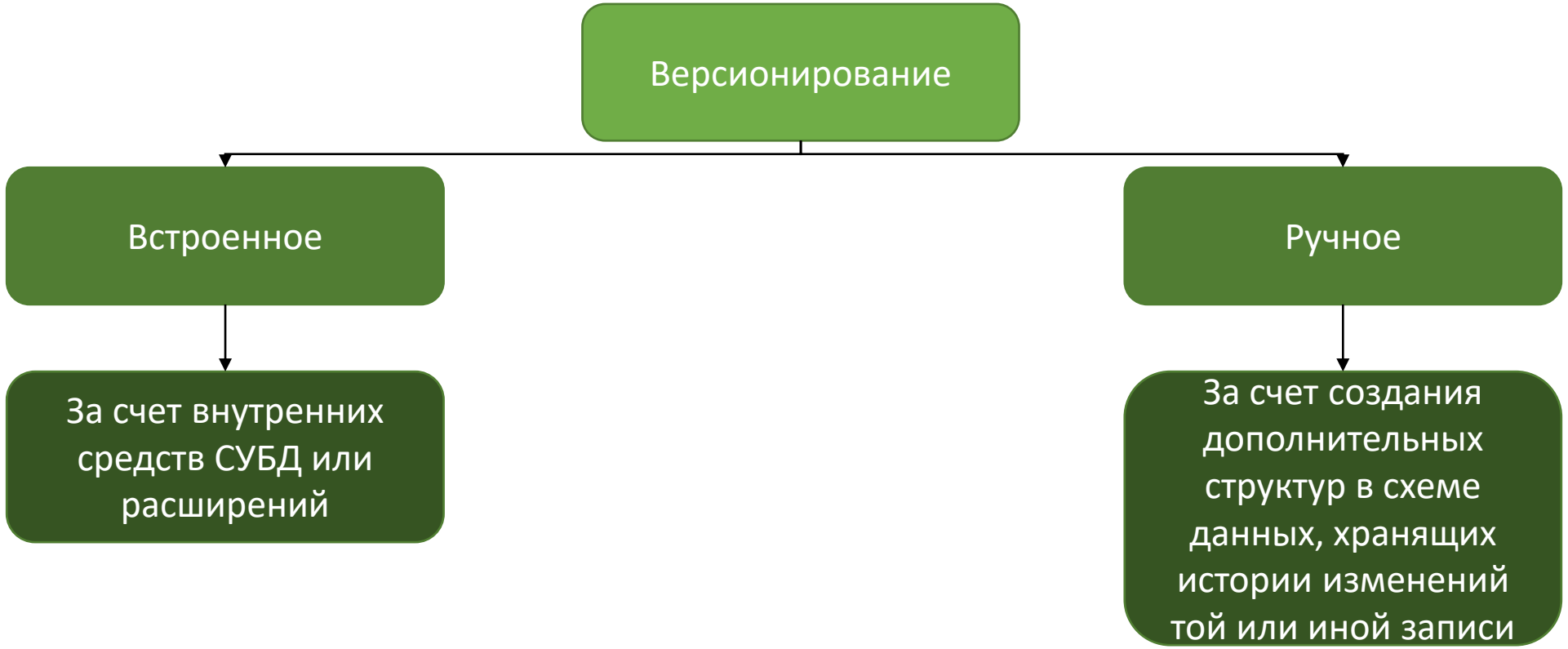
II. Версионирование

Что такое версионирование

- Версионирование - это техника управления изменениями, которая позволяет сохранять, отслеживать и управлять различными версиями данных во времени
- Для чего нужно:
  - Историческое хранение: версионирование позволяет хранить исторические версии записей, что обеспечивает возможность анализа тенденций и восстановления предыдущих состояний данных
  - Аудит и отчетность: поддерживает аудиторские требования, позволяя отслеживать, кто, когда и какие изменения вносил в данные
  - Управление конфликтами: в многопользовательской среде помогает разрешать конфликты, возникающие при одновременном изменении одних и тех же данных разными пользователями.

Что такое версионирование

- Условно версионирование можно разделить на встроенное и ручное



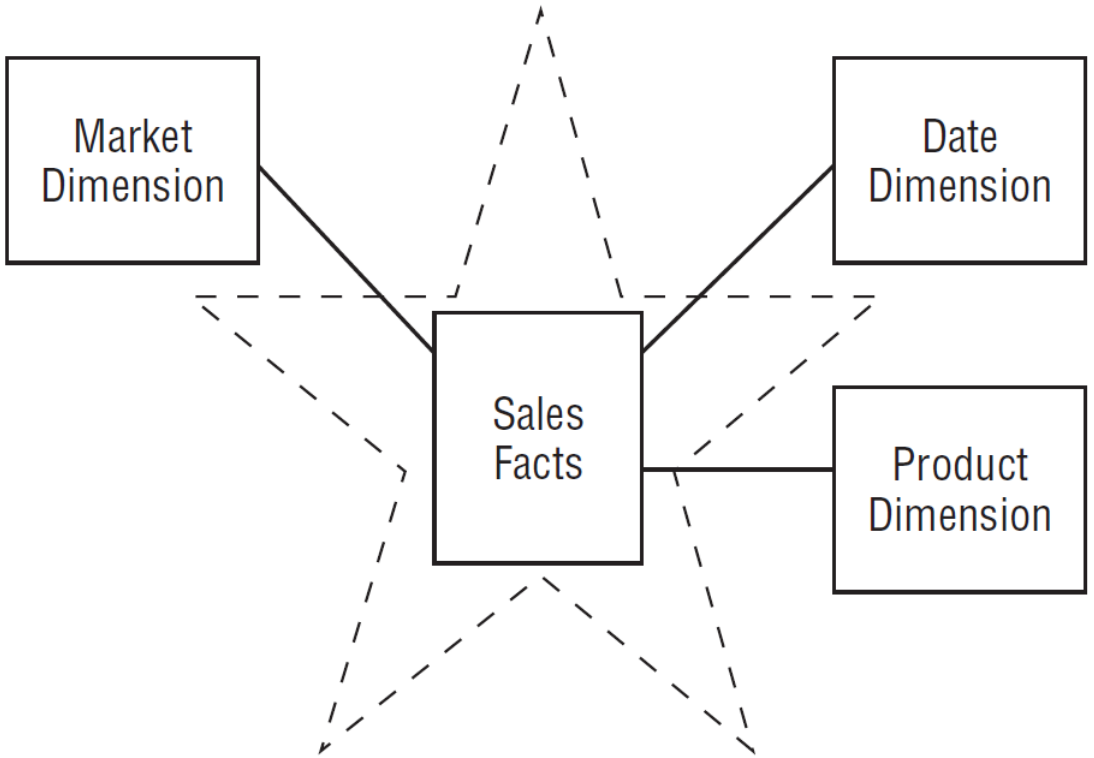
«Ручное» версионирование

- Одна из техник ручного версионирования – Slowly Changing Dimensions (термин – *Ralph Kimball, Margy Ross, The Data Warehouse Toolkit*)
- Термин пришел из концепции проектирования хранилищ данных, разрабатываемой Ральфом Кимбаллом (альтернативная концепция – Уилльям Инмон. Поговорим об этом на отдельной лекции ☺)
- Отношения в хранилище условно делятся на таблицы фактов и таблицы измерений (dimensions).
  - В таблицы фактов записываются данные о фактах «хозяйственной жизни» компании (продажи, поставки, приемка партий товара и т.п.)
  - В таблицы измерений заносятся описательную информацию об объектах или событиях. Каждая строка в таблице измерений представляет одну сущность или один вариант измерения. Таблицы измерений призваны отвечать на вопросы кто, что, где, когда, как и почему «совершил» или поучаствовал в том или ином факте, занесенном в таблицу фактов)

«Ручное» версионирование

- Одна из техник ручного версионирования – Slowly Changing Dimensions (термин – *Ralph Kimball, Margy Ross, The Data Warehouse Toolkit*)
- Термин пришел из концепции проектирования хранилищ данных, разрабатываемой Ральфом Кимбаллом (альтернативная концепция – Уилльям Инмон. Поговорим об этом на отдельной лекции ☺)
- Отношения в хранилище условно делятся на таблицы фактов и таблицы измерений (dimensions).
  - В таблицы фактов записываются данные об измерениях (в смысле подсчета, замера) фактах «хозяйственной жизни» компании (продажи, поставки, приемка партий товара и т.п.)
  - В таблицы измерений (dimensions) заносятся описательную информацию об объектах или событиях. Каждая строка в таблице измерений представляет одну сущность или один вариант измерения. Таблицы измерений призваны отвечать на вопросы кто, что, где, когда, как и почему «совершил» или поучаствовал в том или ином факте, занесенном в таблицу фактов)
- Поскольку таблицы измерений могут меняться с разной периодичность, а также возникает необходимость хранить историю изменений или ранние данные, была предложена техника / концепция Slowly Changing Dimensions

«Ручное» версионирование - измерения



**Ralph Kimball, Margy Ross, The Data Warehouse Toolkit, 2013*

SCD – медленно меняющиеся измерения

SCD – медленно меняющиеся измерения

- Кимбалл выделяет еще два типа SCD (5 и 6)
- Эти типы представляют собой комбинации предыдущих, а также влекут необходимость совершать дополнительные операции при добавлении данных
- Для версионирования данных в «классической» СУБД они менее важны
- На практике чаще всего используют типы 2-4 (из них – чаще тип 2)

SCD 2: пример

EMPLOYEE_NM	POSITION_ID	DEPT_ID	VALID_FROM_DTTM	VALID_TO_DTTM
Николай	21	2	2010-08-11 00:00:00	2016-06-06 23:59:59
Николай	23	3	2015-06-07 00:00:00	5999-01-01 00:00:00
Денис	23	3	2010-08-11 00:00:00	2016-06-01 23:59:59
Борис	26	2	2010-08-11 00:00:00	5999-01-01 00:00:00
Пенни	25	2	2010-08-11 00:00:00	5999-01-01 00:00:00

Здесь: создание новой записи в таблице под каждую версию данных с добавлением полей даты начала и даты конца периода существования версии

SCD 2: пример

- В полях *valid_from_dttm* и *valid_to_dttm* обычно не используются значения NULL. Вместо NULL используется некоторая константа, например, *‘59 99 01 01 00:00:00’* для *valid_to_dttm* , как в примере. Такой подход упрощает написание условий:

```
WHERE day_dt BETWEEN valid_from_dttm AND valid_to_dttm
```

вместо

```
WHERE day_dt >= valid_from_dttm
        AND day_dt <= valid_to_dttm
                OR valid _to_dttm IS NULL)
```

SCD 2: характеристика

- Достоинства:
  - Хранит полную и неограниченную историю версий
  - Удобный и простой доступ к данным необходимого периода
- Недостатки:
  - Провоцирует на избыточность или заведение дополнительных таблиц для хранения изменяемых атрибутов

SCD 3: пример

ID	UPDATE_DTTM	PREV_STATE	CURRENT_STATE
1	11.08.2010 12:58	0	1
2	11.08.2010 12:29	1	1

Здесь: в самой записи содержатся дополнительные поля для предыдущих значений атрибута. При получении новых данных, старые данные перезаписываются текущими значениями.

SCD 3: характеристика

- Достоинства:
  - Небольшой объем данных
  - Простой и быстрый доступ к истории
- Недостатки:
  - Ограниченная история

SCD 4: пример

Таблица с актуальными данными

EMPLOYEE_NM	POSITION_ID	DEPT_ID
Коля	21	2
Денис	23	3
Борис	26	2
Пенни	25	2

Таблица с историей

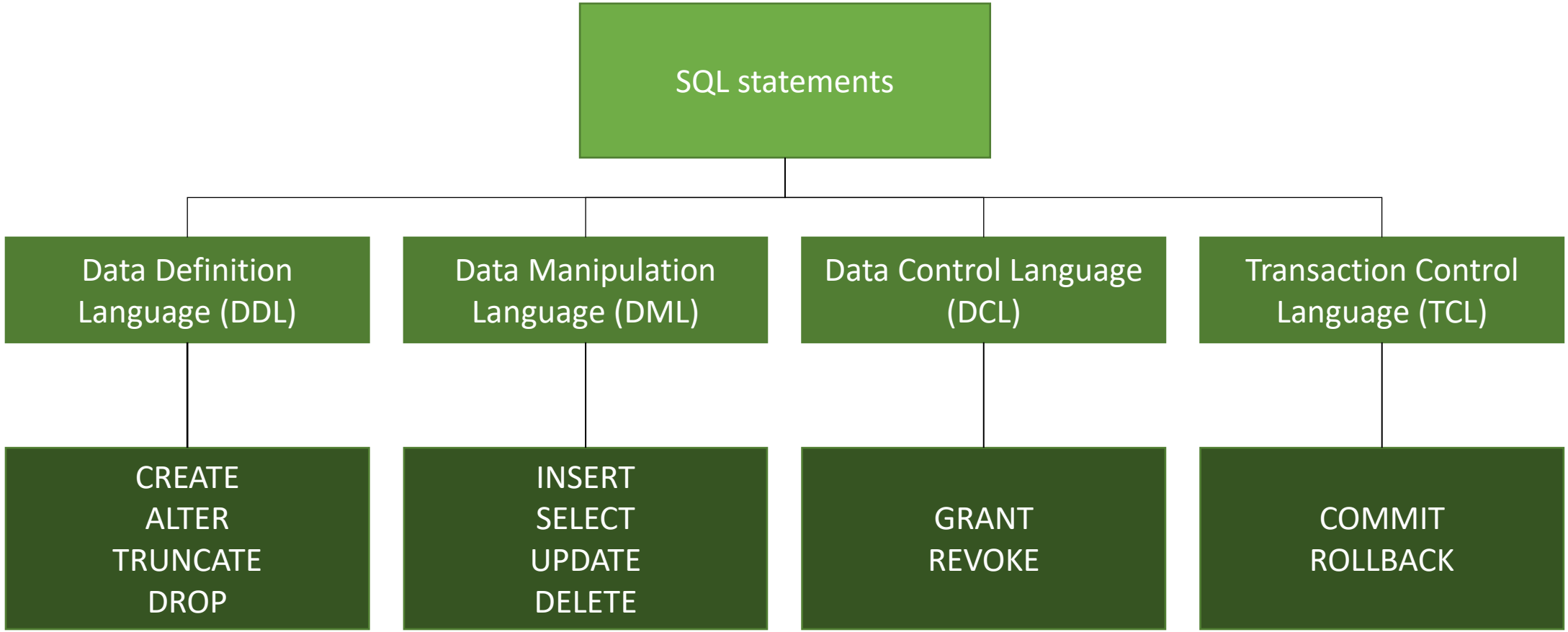
EMPLOYEE_NM	POSITION_ID	DEPT_ID	HISTORY_DTTM
Коля	21	1	11.08.2010 14:12:05
Денис	23	2	19.12.2012 09:54:57
Борис	26	1	09.01.2018 22:22:22

SCD 4: характеристика

- Достоинства:
  - Быстрая работа с текущими версиями
- Недостатки:
  - Разделение единой сущности на разные таблицы

III. TCL, ACID, изолированность

Группы операторов SQL



Что такое транзакция

- Транзакция – группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными, гарантированно переводящая БД из **одного непротиворечивого состояния в другое**.

- Транзакции обеспечивают целостность БД в условиях:
  - Параллельной обработки данных
  - Физических отказов диска
  - Аварийного сбоя электропитания
  - И других…

- Транзакции реляционных СУБД обязательно имеют четыре ключевых свойства. Их обычно сокращают до аббревиатуры ACID

Небольшое отступление – CAP-теорема (теорема Брюера)

- В любой реализации распределённых вычислений невозможно обеспечить более двух из следующих свойств:
  - Согласованность данных (Consistency) – во всех вычислительных узлах в один момент времени данные не противоречат друг другу;
  - Доступность данных (Availability) – любой запрос к распределённой системе завершается корректным откликом, однако без гарантии, что ответы всех узлов системы совпадают;
  - Устойчивость к разделению (Partition tolerance) – расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций.

*Максимизация Consistency и Availability порождает ACID свойства транзакций для РСУБД

ACID

- Транзакции должны удовлетворять следующим свойствам:
  - Atomic – атомарные
    - Выполнены либо все подоперации, либо никакие
  - Consistent – согласованные
    - Каждая успешная транзакция фиксирует только допустимые результаты
  - Isolated – изолированные
    - Параллельные транзакции не влияют на результаты друг друга
  - Durable – долговечные, устойчивые
    - Вне зависимости от сбоев системы результаты успешных транзакций сохранятся в системе

ACID: Atomic – атомарные

- Транзакция должна представлять собой атомарную (неделимую) единицу работы. Должны быть выполнены либо все операции, входящие в транзакцию, либо ни одна из них. Следовательно, в случае невозможности выполнить все операции, все внесённые изменения должны быть отменены:

ACID: Consistent – согласованные

- По завершению транзакции все данные должны остаться в согласованном состоянии. При выполнении транзакции необходимо выполнить все правила реляционной СУБД:
  - Проверки выполнения ограничений (домены, индексы уникальности, внешние ключи, проверки, правила и т.д.)
  - Обновление индексов;
  - Выполнение триггеров
  - И другие
- Но: в ходе выполнения операции согласованность не требуется (вследствие атомарности промежуточная несогласованность остается скрытой)

ACID: Isolated – изолированные

- Изменения в данных, выполняемые в пределах транзакции, должны быть изолированы от всех изменений, выполняемых в других транзакциях, до тех пор, пока транзакция не совершена. Выделяют различные уровни изоляции – для достижения компромисса между степенью распараллеливания работы с БД и строгостью выполнения принципа непротиворечивости:
  - Чем выше уровень изоляции, тем выше степень непротиворечивости данных;
  - Чем выше уровень изоляции, тем ниже степень распараллеливания и тем ниже степень доступности данных.

- В реальных БД полная изолированность не поддерживается!

ACID: Durable – долговечные, устойчивые

- Если транзакия была совершена, её результат должен сохраниться в системе, несмотря на сбой системы. Т.е. если пользователь получил подтверждение об успешности транзакции, гарантируется сохранность результатов
- Если транзакиция не была совершена, её результат может быть полностью отменён вслед за сбоем системы.

ACID: Durable – долговечные, устойчивые

- Если транзакия была совершена, её результат должен сохраниться в системе, несмотря на сбой системы. Т.е. если пользователь получил подтверждение об успешности транзакции, гарантируется сохранность результатов
- Если транзакиция не была совершена, её результат может быть полностью отменён вслед за сбоем системы.

Основные команды TCL

- COMMIT
  - Применяет транзакцию, т.е. сохраняет изменения, произведенные впроцессе выполнения транзакции
- ROLLBACK
  - Откатывает все изменения, произведенные в процессе выполнения транзакции
- SAVEPOINT
  - Создает так называемую точку останова

- Точка останова промежуточный участок в транзакции, на который можно откатиться в случае необходимости
  - Позволяет дробить транзакцию на части
  - Позволяет реализовать «вложенные» транзакции

Основные команды TCL: синтаксис

- BEGIN TRANSACTION transaction_mode [,
  - ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ
  - COMMITTED | READ UNCOMMITTED }
  - READ WRITE | READ ONLY
  - [NOT] DEFERRABLE
- BEGIN / START
  - COMMIT
  - ROLLBACK
- SAVEPOINT name
  - ROLLBACK TO SAVEPOINT name
  - RELEASE SAVEPOINT name

Основные команды TCL: пример

```
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
        INSERT
        INTO table1 VALUES (1);
        SAVEPOINT my_savepoint
        INSERT INTO table1 VALUES (2);
        ROLLBACK TO SAVEPOINT my_savepoint
        INSERT INTO table1 VALUES (3);
COMMIT
```

Проблемы поддержания изолированности

Потерянное обновление

Транзакция 1	Транзакция 2
`UPDATE table_1 SET attr_2 = attr_2 + 20 WHERE attr_1 = 1;`	`UPDATE table_1 SET attr_2 = attr_2 + 25 WHERE attr_1 = 1;`

- Что получим в итоге?
  - 1. attr_2 = attr_2 + 20
  - 2. attr_2 = attr_2 + 25
- Результат однозначно не определен

- Пример из реальной жизни:
  - Вы пополняете свою карту на 5000 рублей
  - Кто то переводит вам 50 рублей
  - Итоговая сумма на вашем счету неизвестна – так как в качестве итоговой может быть выполнена любая из транзакций

«Грязное» чтение

Транзакция 1	Транзакция 2

`UPDATE table_1`
`SET attr_2 = attr_2 + 20`
`WHERE attr_1 = 1;`	

	`SELECT attr_2`
	`FROM table_1`
	`WHERE attr_1 = 1;`

`ROLLBACK WORK;`	

- Пример из реальной жизни:
  - Происходит пополнение счета на 20 единиц, но транзакция не завершена
  - Вы запрашиваете баланс и радуетесь пополнению
  - На самом деле деньги на счет не пришли, т.к. транзакция «откатилась» (ROLLBACK)

Неповторяющееся чтение

Транзакция 1	Транзакция 2
	`SELECT attr_2 FROM table_1 WHERE attr_1 = 1;`
`UPDATE table_1 SET attr_2 = attr_2 + 20 WHERE attr_1 = 1;`	
`COMMIT;`	
	`SELECT attr_2 FROM table_1 WHERE attr_1 = 1;`

- Пример из (почти) реальной жизни:
  - Вы сформировали корзину на сайте и жмете на кнопку «Оплатить»
  - Стоимость товаров на сайте увеличивается
  - С карты списывается бОльшая сумма, чем вы ожидали

Чтение «фантомов»

Транзакция 1	Транзакция 2
	`SELECT sum(attr_2) FROM table_1;`
`INSERT INTO table_1 (attr_1, attr_2) VALUES (15, 20);`	
`COMMIT;`	
	`SELECT sum(attr_2) FROM table_1;`

- Пример из (почти) реальной жизни:
  - Начальник попросил вас составить отчет трат общий по итогам текущего месяца и в разрезе недель того же месяца
  - Вы рассчитываете общую сумму транзакций за месяц
  - Происходит новое списание
  - Вы рассчитываете сумму транзакций в разрезе недель —> учитываете новую транзакцию —> суммы не бьются

Чтение «фантомов»

Транзакция 1	Транзакция 2
	`SELECT sum(attr_2) FROM table_1;`
`INSERT INTO table_1 (attr_1, attr_2) VALUES (15, 20);`	
`COMMIT;`	
	`SELECT sum(attr_2) FROM table_1;`

- Пример из (почти) реальной жизни:
  - Начальник попросил вас составить отчет трат общий по итогам текущего месяца и в разрезе недель того же месяца
  - Вы рассчитываете общую сумму транзакций за месяц
  - Происходит новое списание
  - Вы рассчитываете сумму транзакций в разрезе недель —> учитываете новую транзакцию —> суммы не бьются

Уровни изолированности транзакций

- Read uncommitted (чтение незафиксированных данных)
- Read committed (чтение фиксированных данных)
- Repeatable read (повторяемость чтения)
- Serializable (упорядочиваемость)

Уровни изолированности транзакций

Уровень изоляции	Потерянное обновление	«Грязное» чтение	Неповторяемое чтение	Фантомное чтение
Read uncommited	+	-	-	-
Read committed	+	+	-	-
Repeatable read	+	+	+	-
Serializable	+	+	+	+

“+” – предотвращается
“-” – не предотвращается

Уровни изолированности транзакций Postgres

Уровень изоляции	«Грязное» чтение	Неповторяемое чтение	Фантомное чтение	Аномалия сериализации
Read uncommited	Допускается, но не в Postgres	Возможно	Возможно	Возможно
Read committed	Невозможно	Возможно	Возможно	Возможно
Repeatable read	Невозможно	Невозможно	Допускается, но не в PG	Возможно
Serializable	Невозможно	Невозможно	Невозможно	Невозможно

- «Аномалия сериализиации» - результат успешной фиксации группы транзакций оказывается несогласованным при всевозможных вариантах исполнения этих транзакций по очереди (то есть, в отличие от потерянного обновления, результата как такового не будет вообще)

Read uncommitted: характеристика

- Гарантирует отсутствие потерянных обновлений
- Итоговое значение результат выполнения каждой транзакции
- Возможно считывание незафиксированных изменений
- Данные блокируются на время внесения изменений
- На время чтения данных блокировка отсутствует

- В PostgreSQL можно запросить любой из четырёх уровней изоляции транзакций, однако внутри реализованы только три различных уровня, то есть режим Read Uncommitted в PostgreSQL действует как Read Committed. Причина этого в том, что только так можно сопоставить стандартные уровни изоляции с реализованной в PostgreSQL архитектурой **многоверсионного управления конкурентным доступом (MVCC)**.

Read committed: характеристика

- Второй уровень изоляции
- Используется в большей части СУБД
- Защита от «грязного» чтения
- В этом режиме изменяющая команда (e.g. INSERT, UPDATE) может увидеть несогласованное состояние: она может видеть результаты параллельных команд, изменяющих те же строки, что пытается изменить она, но при этом она не видит результаты этих команд в других строках таблиц. Из-за этого поведения уровень Read Committed не подходит для команд со сложными условиями поиска; однако он вполне пригоден для простых случаев
- В режиме Read Committed каждая команда начинается с нового снимка состояния, который включает результаты всех транзакций, зафиксированных к этому моменту, последующие команды в одной транзакции будут в любом случае видеть эффекты всех параллельных зафиксированных транзакций
- Варианты реализации в СУБД
  - Блокирование читаемых и изменяемых данных
  - Сохранение нескольких версий параллельно изменяемых строк

Repeatable read: характеристика

- Читающая транзакция игнорирует изменения в данных, которые были ей ранее прочитаны
- Никакая транзакция не может изменять данные, читаемые текущей транзакцией, пока чтение не завершено
- Спасает от эффекта неповторяющегося чтения
- Этот уровень отличается от Read Committed тем, что запрос в транзакции данного уровня видит снимок данных на момент начала первого оператора в транзакции (не считая команд управления транзакциями), а не начала текущего оператора. Таким образом,например, последовательные команды SELECT в одной транзакции видят одни и те же данные; они не видят изменений, внесённых и зафиксированных другими транзакциями после начала их текущей транзакции.

Serializable: характеристика

- Транзакции полностью изолированы друг от друга
- Параллельных транзакций как будто бы не существует вовсе
- Транзакции не подвержены эффекту «фантомного чтения»
- На этом уровне моделируется последовательное выполнение всех зафиксированных транзакций, как если бы транзакции выполнялись одна за другой, последовательно, а не параллельно. Фактически этот режим изоляции работает так же, как и Repeatable Read, только он дополнительно отслеживает условия, при которых результат параллельно выполняемых сериализуемых транзакций может не согласовываться с результатом этих же транзакций, выполняемых по очереди.